Towards Fully-Autonomous Ultralight Drones

Mihir Bala

CMU-CS-25-111

April 2025

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Mahadev Satyanarayanan, Chair
David O'Hallaron
Jeff Schneider
Padmanabhan Pillai

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This material is based upon work supported by the U.S. Army Research Office and the U.S. Army Futures Command under Contract No. W519TC-23-C-0003 and by the National Science Foundation under grant number CNS-2106862. The content of the information does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred. This work was done in the CMU Living Edge Lab, which is supported by Intel, Arm, Vodafone, Deutsche Telekom, CableLabs, Crown Castle, InterDigital, Seagate, Microsoft, the VMware University Research Fund, IAI, and the Conklin Kistler family fund.

Any opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the view(s) of their employers or the above funding sources.

Keywords: Autonomous Drones, Robotics, Mobile Computing, Edge Computing

For my dad.

Abstract

Autonomous drones have emerged as an exciting new technology which could revolutionize infrastructure inspection, military reconnaissance, and police surveillance. However, the vast majority of today's platforms are heavy, costly, and difficult to operate. This restricts them from use in many mission settings, such as in densely populated environments, where government regulation forbids autonomous operation of heavy drones near people. Much of this weight comes from the onboard compute resources required for these drones to run the critical computer vision algorithms that provide situational awareness. In this dissertation, I show how autonomy can be induced on lightweight drones using edge computing, offloading high compute jobs to a network-proximal server. I demonstrate how this technique can lead to autonomous aircraft that fly much closer to the FAAs regulatory limits at acceptable performance cost. I also reveal a new operating system designed to unify the disparate landscape of drones under a single, easy-to-program API. I show how this can be leveraged to create heterogeneous collaborative drone swarms on commercial off-the-shelf hardware.

Acknowledgments

I would like to thank my family, especially my mom and dad, for their continued support. I'd also like to thank my Pittsburgh friends: Bhavani Iyer, Brian Singer, Clement Fung, Abby Kroon Fung, Trevor Kann, Madeleine Howell Kann, Eric Zeng, Taro Tsuchiya, Joseph Reeves, Archana Iyer, Keane Lucas, Mark Dong, Alejandro Cuevas, and Amelia Cuevas; my college friends: Matthew Qemo, Evan Arora, Matthew Askar, Jack Dolan, Lindsay Tuttle, Alimul Miah, Jagjot Singh, Tony Muscat, James Sabella, Sneh Bhakta, and Kelton Bursch; my DC friends: Alhan Sayyed, Adaah Sayyed, Lavan Rajan, Kiki Sayyed, and Nirmal Maitra; and lastly, my high school friends: Liam Wilson, David Schindler, Patrick Turiano, Grant Keith, and Alex Sutherland.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Unmanned aerial vehicles, commonly known as drones, are a disruptive technology that have recently seen widespread use. In civilian settings, they enable cheap and safe completion of tasks such as infrastructure inspection, agriculture monitoring, wildfire control, and police surveillance. In military settings, they are a vital tool for advance reconnaissance. For most use cases today, drones are deployed with a human pilot who is in constant control of the aircraft.

In recent years, research efforts have pushed towards drones capable of fully-autonomous flight. The term *fully-autonomous* flight is defined by the National Institute of Science and Technology (NIST) as "pre-programmed flight without a remote human pilot, including mission-specific actions in response to runtime observations" [68]. There are two main benefits to this approach. First, it decreases costs and frees human attention. Second, it allows the practical operation of drone swarms, large groups of aircraft that cooperatively execute tasks. Drone swarms open the door to many missions that could revolutionize several civilian and military sectors [22].

A key driver for fully-autonomous drones is the completion of active vision tasks [6, 93]. Active vision tasks require a drone to react in real time to its current scene interpretation. For instance, it may drop down to lower altitude without human intervention "to take a closer look" before the scene changes. It may then return to its original altitude to continue monitoring the scene. This narrow scope of tasks characterizes many fundamental drone operations like following a target and evading obstacles.

Weight is a fundamental impediment to fully-autonomous drone adoption. Greater intelligence correlates with more powerful (and hence heavier) on-board computing and richer sensing. An on-board GPU, for example, brings with it a long logistical tail: heatsinks, cooling fans, and larger batteries. Increased weight brings with it regulatory challenges for flights over civilian areas. Since 2021, the FAA has pre-authorized flights over people and vehicles by drones with a total weight of less than 250 g [50]. Heavier drones require explicit FAA approval for such flights, conditional upon mitigation measures for collisions and free fall. Even above this limit, regulatory approval for autonomous BVLOS (beyond visual line of sight) flight in urban settings is easier to obtain for lighter drones than for heavier drones. This regulation has proven to be a major obstacle for several civilian projects. In military settings, weight is also a crucial consideration. Heavy aircraft complicate logistics and often require dedicated transportation [73].

Other major limitations to autonomous drone adoption are software portability, accessibility, mission versatility, and unit cost. While there have been attempts to bring all drones under a unified programming ecosystem, it is still the norm for drone companies to develop their own SDKs for their platforms. This makes it difficult to port code written for one ecosystem to another which divides the development community. Existing fully-autonomous drones also require users to have significant flight experience to ensure safe operation, a serious barrier to accessibility. Many lack versatility, and cannot be configured to perform missions outside of a small, manufacturer specified set. Lastly, the unit cost for current autonomous drones is several times higher than manually piloted equivalents. Such prices hurt the economic viability of swarm operations where individual aircraft losses are not only likely but expected.

The core contribution of this work is *SteelEagle*, a hardware-agnostic autonomous drone system which attempts to surmount these obstacles by leveraging edge computing and a new modular drone autonomy stack. Edge computing enables a drone to offload compute-intensive real-time operations over a low-latency, high-bandwidth wireless network to a powerful ground-based server (cloudlet) which is usually located near a cell tower. This reduces the need for heavy on-board computing hardware. In parallel, the SteelEagle operating system is designed to be drone agnostic, developer friendly, and mission centric.

A key consideration of this system is the use of commercial off-the-shelf [49] (COTS) drones and computing / communication payloads. This approach avoids customization of hardware (e.g., drone modifications) and modification of privileged software (e.g., "rooting" a device) which lowers cost and greatly increases accessibility. It also avoids the need for re-certification (e.g., by the FAA or the FCC). However, a COTS approach also introduces new obstacles. Thermal limitations of lightweight COTS communications devices pose latency, frame rate, and quality challenges, and force such systems to intelligently manage communication, computation, and prediction.

1.1 Thesis Statement

In this dissertation, I demonstrate SteelEagle as a capable alternative to existing autonomous drone systems, despite the inherent latency and bandwidth limitations of offloading. I show how it improves over other platforms in the following design categories:

1. **Weight**: the overall weight of the aircraft, including batteries and payload.
2. **Accessibility**: the barrier-for-entry to operate the aircraft.
3. **Versatility**: the diversity of tasks which the system can execute.
4. **Portability**: the ease with which the system can be ported to new hardware.
5. **Cost**: the overall cost of the aircraft, including batteries and payload.

In particular, I claim that:

It is feasible to construct an ultra-light flight platform for autonomous active vision tasks in beyond visual line-of-sight (BVLOS) settings that only uses commercial off-the-shelf (COTS) drones and COTS computing/communication payloads. The most serious obstacle to this goal, namely the weight of computing hardware necessary to achieve autonomy, can be overcome using edge computing. I posit that such a flight platform can emulate the performance of heavier autonomous drones on active vision tasks, despite bandwidth, latency and connectivity challenges.

Why is this thesis important? If this statement were true, then such edge-enabled drones could see widespread adoption for BVLOS missions in urban environments. They would have lower operation costs and increased safety compared to traditional autonomous drones. Tasks such as structure inspection, police surveillance, and traffic monitoring could directly benefit from this. Drone swarms over public infrastructure would become safer.

Why does it not follow trivially from what is already known? Existing work in this space is limited since most current research focuses on improving drone capabilities rather than decreasing their weight. There are no commercial drones under the FAA threshold of 250 g that possess onboard intelligence capable of autonomously executing missions. There are limited drones that are edge-enabled, but these drones are heavy (more than 500 g) and expensive (over $3,000). There has been some academic research on using drones in conjunction with edge computing, with full details provided in Section 3.2. However, these efforts focused on large, heavy drones which used onboard hardware in addition to supplemental edge offload. None identified weight as a dominant design consideration. SteelEagle, by contrast, is designed primarily around reducing weight without compromising capability. It is my belief that this is critical to drive large scale adoption of autonomous drones.

The main contributions of this thesis are as follows:

1. I describe a fully-autonomous flight platform capable of executing active vision tasks on lightweight COTS drones using edge computing. I show why this platform is an improvement over prior work in terms of its *weight*, *accessibility*, *versatility*, *portability*, and *cost*.
2. I provide a measurement study quantifying this platform's performance using a novel benchmarking suite.
3. I show how this platform can be extended to a heterogeneous drone swarm ecosystem.

1.2 Thesis Overview

The remainder of this dissertation is organized as follows:

- In Chapter 2, I provide background on the development history of autonomous drones and summarize related research in this area. I show how SteelEagle builds on this existing research.
- In Chapter 3, I discuss how to connect lightweight COTS drones to the edge. I outline the design challenges and formulate a criteria for choosing an edge communication payload that flies with the drone.
- In Chapter 4, I provide the overall design of SteelEagle including its advantages and dis-

advantages over current systems. I demonstrate a SteelEagle drone performing several autonomous tasks and provide some performance analysis.

- In Chapter 5, I introduce a new edge communication payload that improves on the earlier prototype. I show how this new payload can reduce the OODA (Observe Orient Decide Act) loop of the system, and thus greatly increase autonomous performance.
- In Chapter 6, I describe a family of benchmarks for measuring edge-based and fully-onboard autonomous drone performance on several key tasks. These benchmarks stress the OODA loop of the given test platform and are useful for understanding the impact of high latency and low throughput on edge offloading.
- In Chapter 7, I show how SteelEagle can be deployed on a variety of drone hardware and control schemes by using a driver-based approach. I illustrate how my system adapts to a new drone and lay the groundwork for disconnected operation.
- Finally, in Chapter 8, I conclude the dissertation and explain future work with a summary of contributions.

Chapter 2

Background and Related Work

Autonomous drones are an emerging field in both civilian and military sectors. The investment in Zipline, DJI, and other similar companies, show that there is a lucrative market for urban autonomous drone applications such as same-day drone delivery, automated infrastructure inspection, and programmable aerial surveillance [4, 60]. The rise of companies like Anduril and recent geopolitical events like the War in Ukraine also hint at the wider role that autonomous drones will play in future armed conflicts [24, 114].

Even with this demand, the quest to make smaller, lightweight autonomous drones is ongoing. Weight scales closely with the capability of onboard compute. Improvements in the artificial intelligence algorithms used for drone perception continue to require more computation to run. This combination of factors makes the prospect of creating a drone system with real-time access to such algorithms challenging.

In this chapter, I provide background on current autonomous drone platforms and outline how some of the obstacles impeding their practical deployment can be overcome. In Section 2.1, I briefly describe the history of drone development and regulation. In Section 2.2, I outline the different categories of modern drones and the types of missions they are used for. In Section 2.3, I explain the various problems holding these systems back from widespread adoption. In Section 2.4, I discuss prior research that has attempted to solve these problems, and its limitations.

2.1 The Development of Modern Drones

Research into drone technology started in the 1930s when during the interwar period, British engineers created a radio controlled plane, nicknamed the "Queen Bee", to train their anti-aircraft gunnery [70]. The Queen Bee's name would spawn the colloquial "drone" moniker when referring to radio-controlled aircraft, a reference to worker drones in bee colonies. Over the next several decades, militaries around the world began incorporating drones into their arsenals; first, as pilotless training targets but later, as remotely operated observation and strike aircraft. By the 1990s, drones had become very sophisticated, equipped with multiple onboard sensors which enabled these platforms to conduct aerial reconnaissance at great distances [119]. Despite these advances, one core design tenet remained unchanged: drones were controlled in every respect by a remote human pilot. Such pilots would be referred to as the remote pilot-in-command (RPIC).

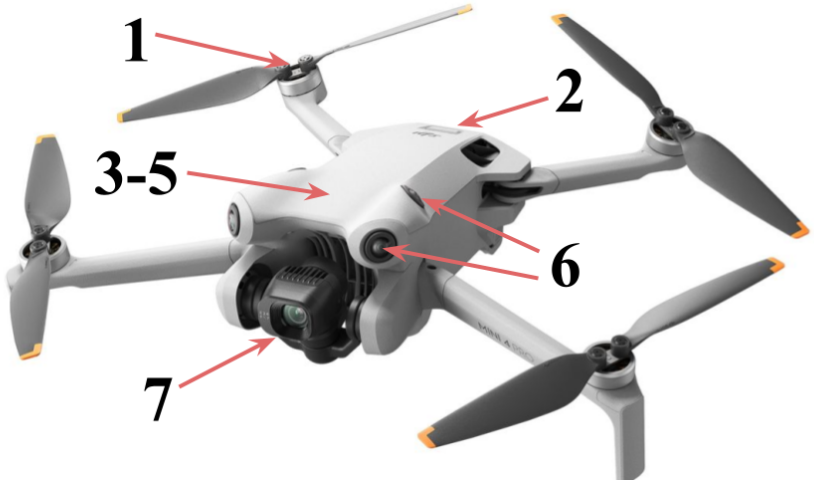
(a) The British-made "Queen Bee", introduced in 1935, was one of the first radio operated aircraft and served as an anti-aircraft practice target [70].

(b) The RQ-4 Global Hawk, introduced in 1998, is a currently operated US Air Force reconnaissance drone with a range of over 14,000 miles [119].

(c) The Skydio 2 platform, released in 2019, provides a small set of semi- and fully-autonomous capabilities like person tracking and obstacle avoidance [110, 111].

(d) The DJI Matrice 600, released in 2016, is a fully-programmable autonomous drone with support for native onboard GPUs. Its huge size limits practical deployment [36].

Figure 2.1: Evolution of Drone Technology



The main components of a modern consumer drone:
1. *rotors,* ***2.*** *hot-swappable battery,* ***3.*** *autopilot,* ***4.*** *sensor module,*
5. *companion computer,* ***6.*** *stereo cameras,* ***7.*** *gimbal camera.*

Figure 2.2: Quadrotor Drone Anatomy [37]

In the mid 2010s, drone technology started to shift away from exclusive manual piloting. The release of commercial autopilots (§2.1.1) like PX4 and Ardupilot enabled the rise of miniaturized (under 2 kg) multirotor drones which could perform limited autonomous flight such as following preset GPS waypoints [12, 43]. Later, this was extended to semi-autonomous visual tracking and autonomous obstacle avoidance in quadrotor offerings like the DJI Phantom 4 and the Skydio 2 [38, 110]. At the same time, growing investment in commercial fully-autonomous drones yielded the first off-the-shelf products. The DJI Matrice series was the most prominent of these, and it offered fully-autonomous capabilities using an onboard embedded computer, the DJI Manifold [36].

While the drone space is diverse in both aircraft size and type, this dissertation will focus on quadrotor drones. Quadrotors are by far the most common drone type and have a number of advantages over fixed-wings and helicopters. Namely, they are affordable, easy to use, and are very stable in multi-directional flight. These make them perfect for tasks involving aerial imagery analysis which are the main focus of this dissertation. For the rest of this document, all mention of "drones" will refer to quadrotor aircraft.

2.1.1 Anatomy of a Drone

Modern drones are made up of several core components. Broadly, these accomplish one of four tasks: *low-level flight control, high-level flight control, power, and sensing*. Figure 2.2 uses the DJI Mini 4 Pro [37], a popular consumer drone, to illustrate these components:

1. **Rotors** (low-level flight control): provide lift to the aircraft. Opposing pairs counter-rotate to provide stability [106]. Rotors are coordinated by the electronic speed controller (ESC), a micro-controller connected to the *Autopilot* module [78].

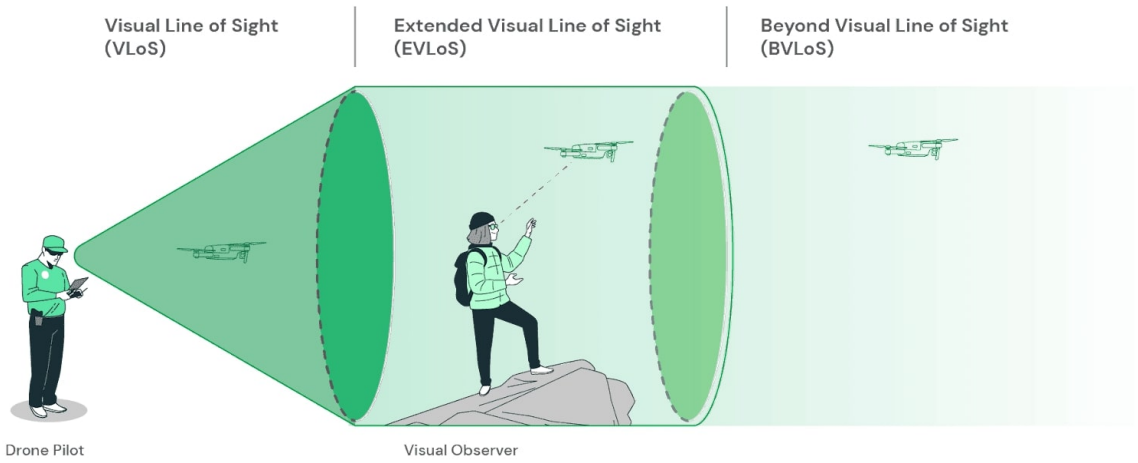


Figure 2.3: Types of Line-of-Sight Operation [54]

2. **Battery** (power): delivers power to all components. In many consumer drones, like the DJI Mini 4 Pro, the battery is a separate part which can be easily swapped out in the field.

3. **Autopilot** (low-level flight control): works with the ESC to execute flight maneuvers. Uses telemetry from the *Sensor Module* to compensate against the wind and maintain a stable hovering position. Acts as a software abstraction layer over all drone sensors and hardware. Also manages the radio connection to the pilot. Popular offerings are PX4 and ArduPilot [12, 43].

4. **Sensor Module** (sensing): provides information about the drone's position and orientation, also known as telemetry. Usually made up of a GPS antenna, an inertial measurement unit (IMU) [75], an altimeter, and a compass.

5. **Companion Computer** (high-level flight control): responsible for high-level autonomous decision-making. Runs computer vision or sensor fusion algorithms, then sends actuation commands to the *Autopilot* based on the outputs.
6. **Stereo Cameras** (sensing): gives a real-time depth map of the drone's surroundings [32]. On the DJI Mini 4 Pro, six overlapping stereo cameras enable full 360-degree obstacle avoidance. Not a feature on all consumer drones but becoming increasingly common.

7. **Gimbal Camera** (sensing): the main camera through which the drone visually senses its surroundings. Able to pitch, yaw, or roll as commanded by the *Autopilot*.

In some cases, drones may be equipped with special equipment like LIDAR, time-of-flight sensors, RTK modules, and cellular modems. These are rare and are usually a feature of purpose-built aircraft for specific mission sets.

2.1.2 Regulation in Civilian Airspace

Drones like the DJI Mini 4 Pro and others are available globally for purchase, generally without the need for a license. This has led to a rapid dissemination of drones to consumers across the world and has spurred government regulators into action. This regulation serves one major purpose: to prevent potential damage to people or property due to in-flight emergencies. In the United States, the Federal Aviation Administration (FAA) introduced the Part 107 regulation in 2016 [52]. This explicitly outlined the classes of UAVs allowed to fly over people, citing that only aircraft under 0.55 lbs or 250 g enjoyed near-unconditional approval [50]. In Europe, the European Union Aviation Safety Agency (EASA) has introduced similar regulation, only permitting flights over uninvolved pedestrians for drones under 250 g [47]. Flights over assemblies of people are not allowed for any weight class. In other countries, drones under 250 g are not regulated at all [117, 118].

In addition to weight regulation for flights over people, most countries also have restrictions on beyond visual line-of-sight (BVLOS) operation of heavy drones. Figure 2.3 shows the different classes of line-of-sight operations. Visual line-of-sight (VLOS) operation requires the drone pilot to be within direct line-of-sight of the drone. Extended visual line-of-sight (EVLOS) operation permits the pilot to lose visual contact with the drone but requires other human spotters, also known as "visual observers" to be within line-of-sight of the drone and in constant radio contact with the pilot. BVLOS operation does not require the pilot or any visual observers to be within line-of-sight of the drone [2]. This makes BVLOS flights dangerous; if a collision is imminent, there is no guarantee that the pilot will be notified in time to stop it. Consequently, regulations are stringent for heavier drones due to their greater damage potential.

2.2 The Current COTS Drone Market

As a result of both regulation and consumer demand, the current commercial-off-the-shelf [49] (COTS) drone market is segmented into three loose categories: fully-autonomous, semi-autonomous, and manually piloted drones. Each of these inhabits a different weight and mission class, and are designed to operate in different regulatory environments. Table 2.1 exhibits the main applications that drones are designed for. Generally, depending on regulation in the mission area, either a fully-autonomous (heavily-regulated) or semi-autonomous (lightly-regulated) drone will be used.

1. **Manually-Piloted Drones**: Manually-piloted drones are designed for hobbyist consumers, usually for the purpose of drone racing or recreational RC flight. They have no onboard compute, are not programmable, and must be manually piloted at all times to function. An example of a manually-piloted drone is the iFlight Cidora. This class of drone is extremely lightweight and very affordable compared to both semi- and fully-autonomous drones. The iFlight Cidora weighs 115 g and costs $295 per unit [92]. Most manually-piloted drones are so light that they avoid regulation. However, they are the least accessible of the three drone classes. This is because they lack any auto-stabilization, and must be flown by an experienced pilot to avoid crashes.
2. **Semi-Autonomous Drones**: Semi-autonomous drones have limited onboard compute and

cannot perform active vision tasks without a human in the loop. They are typically equipped with a commercial autopilot like PX4 which enables autonomous following of GPS waypoints. They are not intended to operate BVLOS and often require a constant link to the RPIC (remote pilot in command). This is because their main use case is aerial photography and videography. For this reason, this class of drones is commonly referred to as *photography drones*. An example of a photography drone is the Parrot Anafi. It is programmable and has no onboard compute, but can follow GPS waypoints and perform limited visual tracking by utilizing compute resources on the RPIC's remote controller. Photography drones are lightweight and affordable compared to fully-autonomous drones, and can usually be flown in dense urban environments with minimal regulatory hurdles. The Parrot Anafi, for instance, weighs 320 g and costs $470 per unit [101]. They are also the most accessible of the three drone classes since they are designed to be flown by non-pilots and are by far the most widely-used of the three classes. They usually feature excellent stabilization, good safety characteristics, and a simple user experience.

3. **Fully-Autonomous Drones**: Fully-autonomous drones have significant compute resources onboard and are able to analyze their own sensor streams in real time. They can perform active vision tasks without human assistance and can operate BVLOS. They also are fully-programmable, outfitted with an onboard computer and a flight control API. An example of a fully-autonomous drone is the DJI Matrice series. This class of drone is typically heavy and expensive. The Matrice 30, for instance, is over 3.5 kg and over $10,000 per unit [39]. Because of their size and weight, fully-autonomous drones are generally limited to use in rural areas away from people or property. They are also highly inaccessible, since their large size and complicated user interfaces often require experienced programmers and mission planners to operate safely.

In reality, most drones share traits from all three of these categories, and seldom fit into one cleanly. This motivates viewing the drone space as a "spectrum of autonomy". Typically, this spectrum is segmented into six levels of increasing automation: levels 0-1 corresponding to manually-piloted, levels 2-3 corresponding to semi-autonomous, and levels 4-5 corresponding to fully-autonomous [30]. This is similar to the levels of automation for self-driving cars [46]. Figure 2.4 shows the capabilities of several drones and where they would lie on this spectrum.

2.3 What is Holding Back Drones?

As presented by Table 2.1, there are many applications where drones have been useful. Yet it is my belief that they still have not lived up to their true potential. Many of the listed tasks, such as aerial surveys, building inspection, and police surveillance, are still done using full manual or human-assisted control in densely populated areas. Yet many if not all of these tasks could benefit greatly from full-autonomy, especially in or around cities. Surveys or inspection flights could be performed daily, without human supervision, with an auto-generated report sent out for engineers to look over. This could greatly reduce costs and prevent critical infrastructure failures, potentially saving lives [42]. Automated police surveillance could free up officers, reduce training overhead, and provide round-the-clock monitoring without any risk of fatigue. So why

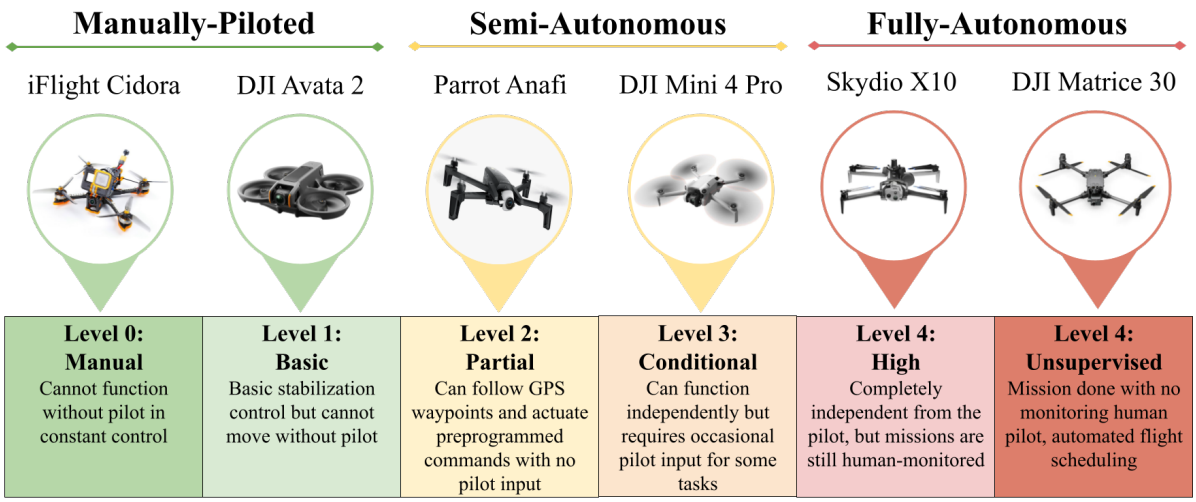
* Precision Agriculture	Drones are very useful for precision agriculture, with uses in monitoring, planting, irrigation, and pollination. They are easily scalable for large crop fields and can cover more area than ground-based solutions [33]. Most drone agriculture solutions use heavy, fully-autonomous platforms since rural areas have less strict regulation.
* Search and Rescue	Aerial vehicles have historically been a vital component in search and rescue. Drones fit nicely into this role, allowing search and rescue teams to scan an area at a much lower altitude than they could with traditional aircraft. They have seen real world use in the wake of the 2010 Haiti earthquake and other natural disasters [131].
Package Delivery	For many years, drones have been touted as the future of last mile package delivery. This is because they can operate with higher cost-efficiency for small size items and can reach areas that are not well-connected by traditional infrastructure [82]. Companies like Zipline have had some success creating a commercial product, using drones to make 1 million deliveries to remote areas [10].
* Structure Inspection	Recently, drones have emerged as a useful tool for structure inspection. They are able to reach inaccessible sections of structures due to their small size and maneuverability. More importantly, they are much safer and efficient than a human inspector. In 2022, the U.S. Bureau of Labor Statistics estimated that 1 in every 5 construction workplace deaths was due to falls [19]. Drones allow viewing of unsafe areas with no personal risk. This has increased the frequency of building inspection checkups, ensuring prompt, safe maintenance on failing infrastructure [5].
* Aerial Surveys	While aerial surveys and 3D scans have been conducted for decades, drones are a new, more cost effective tool for this task [99]. Their ability to provide a high resolution, bird's eye view of an area combined with their flight stability, make them ideal candidates for survey work.
* Aerial Reconnaissance	Modern drones have revolutionized police and military aerial reconnaissance. Their small size allows them to be easily carried to mission areas and deployed on-the-fly when necessary [127]. Both U.S. police and military forces have heavily invested in such platforms [127, 130].
Suicide Aircraft	Suicide drones have been one of the most impactful new weapons in the War in Ukraine. Both Russian and Ukrainian forces have used small quadcopters like the DJI Mavic 3 to deliver bomb payloads [14]. These have been very effective against slow moving targets like tanks [72]. Some fear that this technology could lead to a new breed of terrorist attacks [115].
* Anti-Drone Defense	As the threat of drones has increased, the investment in drone defense systems has surged. Companies like Anduril have tackled this problem by using specialized drones to take down other hostile drones. Their Anvil kinetic interceptor destroys other UAVs by smashing into them at high speeds [8].

* These tasks are within the scope of this work since they do not require additional payload.

Table 2.1: Drone Applications

Drone	Autopilot	Avoidance	Tracking	Programmable	Compute	Weight
iFlight Cidora	None	None	None	No	None	115 g
DJI Avata 2	Yes	Partial	None	No	None	377 g
Parrot Anafi	Yes	None	Assisted	Yes	None	320 g
DJI Mini 4 Pro	Yes	Yes	Assisted	Partially	None	249 g
Skydio X10	Yes	Yes	Yes	Partially	Yes	2110 g
DJI Matrice 30	Yes	Yes	Yes	Yes	Yes	3770 g

Capabilities are colored according to whether they are typically associated with manually-piloted (red), semi-autonomous (yellow), or fully-autonomous (green) drones.



Based on their capabilities, I have placed each drone on the spectrum of autonomy, broken down by level [30]. This placement is subjective, but it gives some reference for the differing levels of automation between commercial drone offerings.

Figure 2.4: The Spectrum of Autonomy [35, 37, 39, 92, 101, 112]

have fully-autonomous platforms failed to see widespread use in these urban applications, when they have been used with great success elsewhere?

The answer is complicated, but I believe that there are five clear problems facing fully-autonomous drones in dense urban settings:

- **Weight**: fully-autonomous aircraft are heavy, as discussed in Section 2.2. This is because they require onboard compute like GPUs to operate which drive up weight. This directly clashes with most international drone regulation which becomes progressively more restrictive over 250 g.
- **Accessibility**: fully-autonomous drones, due to their size and weight, require experienced pilots to operate safely. They demand careful flight planning and program verification before flight, since a loss of aircraft could be hazardous to people and property. This restricts fully-autonomous drones to a small user-base.
- **Versatility**: there is a major trade-off in current fully-autonomous drone products between weight and versatility. Heavy platforms which carry generalized compute hardware like CPUs and GPUs are versatile since they can run most kinds of software natively. On the other hand, lightweight platforms cannot carry generalized compute due to weight constraints, and thus usually carry specialized compute for only one or two mission types.
- **Portability**: drone manufacturers have their own, tightly integrated software stack that does not work with other drone models. This all-or-nothing approach means that a consumer must stick within the hardware ecosystem or be forced to buy an entirely new fleet of drones.
- **Cost**: fully-autonomous drones typically cost around ten times the cost of comparable semi-autonomous drones. This makes them much less economically viable at scale.

If we could solve these five challenges, it is my belief that autonomous drones would see widespread use, even in dense urban environments. At the time of writing this document, no such commercial product or research prototype exists.

2.4 Prior Research on Autonomous Drones

In parallel to commercial efforts, autonomous drone research has surged in recent years. Real-time execution of active vision tasks has been a key driver. Schedl et al proposed an autonomous drone design for classification-driven adaptive search and rescue in densely forested environments [108]. George et al demonstrated a drone inspection system which could localize the drone's camera view onto a 3D model of a target structure in real time [57]. Chen et al showed an efficient drone onboard computation model for visual object tracking [28]. Many other projects have explored similar applications in surveillance, wildlife monitoring, and racing [7, 11, 34, 79, 123].

A growing number of researchers have identified some of the problems outlined in Section 2.3 (*weight*, *accessibility*, *versatility*, *portability*, and *cost*) as important research areas. In this section, I will present projects that attempt to solve these problems. I will also discuss their limitations and why their solutions have not fully addressed the issues I outlined.

2.4.1 Weight

Since as early as 2009, the drone research community has recognized the importance of lightweight aircraft in real world applications [22, 23]. Lightweight (≤ 450 g) autonomous drones have many benefits over their heavier counterparts: they are safer to operate, have simpler transportation logistics, and have a lower noise profile. It was theorized that achieving such lightweight autonomy would enable the vision of cooperative drone swarms [53]. Since the mid 2010s, much work has focused on bringing high-level autonomy to small, lightweight drones. Schmid et al proposed a sub-1 kg aircraft design which could perform unassisted visual navigation [109]. Palossi et al developed visual navigation and human pose estimation software which runs onboard 27 g drones [97, 98]. Müller et al demonstrated a depth-based obstacle avoidance system for 35 g drones [91]. Many of these projects suffer from similar issues of portability; they either use purpose-built aircraft or function on only one type of aircraft. There is little ability to deploy these systems to different drone hardware, a drawback in an ever-changing drone landscape.

2.4.2 Accessibility

In the current drone research space, accessibility is not a first-order consideration. Many projects use custom-built aircraft with no guidance on how to replicate the platform for other researchers. In some cases, these custom drones require low-level electrical engineering experience to ensure safe and stable operation. On the other hand, there has been some effort to increase accessibility for drone programming. Beseda et al presented a mission-oriented control infrastructure for easily managing swarms of autonomous UAVs [16]. Mottola et al proposed a simple but expressive API for drone team control [90]. Tilley et al developed a block programming language that could allow children to program autonomous drones [116]. While this work is useful, these projects do not present a practical integration strategy with real consumer drones, nor do they circumvent the difficulties of custom-built research platforms.

2.4.3 Versatility

Mission versatility is a feature that most heavy fully-autonomous drones already possess, since they carry generalized compute like CPUs and GPUs which can run GPLs (general-purpose programming languages) and inference deep neural networks with relatively low latency. Achieving this versatility on lightweight drones is an enduring challenge. This is because miniaturizing generalized compute hardware is difficult, and weight-optimized compute hardware is inflexible [67]. Still, many have tried to modify lightweight onboard hardware to be able to run heavyweight computer vision algorithms. Albanese et al propose a neural accelerator modification to a Raspberry Pi which could be flown onboard small UAVs [3]. Zhang et al demonstrate an FPGA architecture for drones that can inference neural networks with much lower power demand than GPUs [133]. None of these projects have gained much traction, likely due to their poor practicality and usability.

Corporate research projects have yielded promising results, such as the ModalAI Starling series and the Bitcraze Crazyflie shown in Figure 2.5. These both use custom compute hardware to provide a Linux programming environment in an ultra-lightweight package [17, 86]. Still,

(a) Bitcraze Crazyflie [17] (27 g)

(b) ModalAI Starling [86] (280 g)

Figure 2.5: Consumer Drone Development Platforms

these systems do not truly capture the versatility of CPU/GPU-based systems, and neither fully support the open-source libraries which drive most computer vision research, like OpenCV and PyTorch. This significantly hinders practical use.

2.4.4 Portability

The ability to seamlessly switch underlying drone hardware has only recently become a hot research topic. This is motivated by greater research impact; cross-platform drone work has a much wider reach. One example of such work is BeeCluster, a drone-agnostic swarm orchestration tool with a simplified task programming interface [65]. BeeCluster supports drones that talk using the BeeCluster protocol, and provides a high-level API wrapper over this protocol. As of the time of writing this document, no guidance has been provided by the authors on how to integrate a new drone platform into BeeCluster, but the framework theoretically allows for heterogeneous swarm operation [85].

2.4.5 Cost

Driving down cost has been an enduring goal of the drone research community. Eller et al designed a low-cost autonomous platform for under $225 [45]. Hardy et al used a sub-$1,000 drone to map malaria breeding grounds [63]. Sørensen et al proposed an $800 aircraft for mobile remote sensing [113]. Projects like these aim to make drones more economically viable, especially in regions with less purchasing power. However, none are concerned with the other important challenges outlined, like weight.

Chapter 3

Towards Better Autonomous Drones

The key factors of *weight*, *accessibility*, *versatility*, *portability*, and *cost* all hinder autonomous drone development and deployment, yet no proposed system has solved all of these issues simultaneously. Clearly, significant research effort has been spent on solving these problems; so why does no such system exist? Central to the answer is what I call the *onboard compute trade-off*. The onboard compute trade-off is a property of unmanned aerial platforms that claims:

> Traditional hardware, like CPUs and GPUs, offers the benefit of versatile, software portable, cost-effective compute power at the expense of weight. Specialized hardware offers the benefit of light weight at the expense of versatile, software portable, cost-effective compute power. Thus, in order to reduce weight, *versatility, portability, and cost must be compromised*.

This fact has limited the development of lightweight fully-autonomous drones and it is chiefly responsible for the gap that exists in the drone market today. Unfortunately, even with recent technological advances, it is unclear whether this challenge can be solved. But, there may be ways it can be circumvented.

In this chapter, I will introduce SteelEagle, a drone autonomy system designed to bring intelligence to lightweight, commercial-off-the-shelf (COTS) drones. SteelEagle skirts the need for heavy onboard compute by leveraging edge computing, a computational model that involves offloading work from a less powerful device over a low latency network link to a nearby powerful device which finishes the work and returns the result. In this way, SteelEagle aims to solve the onboard compute trade-off and lower the weight-barrier to drone autonomy. In Section 3.1-3.5, I explain the background and prior research that drove the creation of SteelEagle. In Sections 3.6-3.10, I discuss the many failed prototypes I tested before arriving at an initial working solution.

3.1 The Advent of Edge Computing

Over the past few years, a new computational paradigm has emerged called *edge computing*. Edge computing gives mobile devices access to strong computational resources at low latency by offloading to a network-proximal server [107]. In many ways, this is similar to cloud computing. The theory is that mobile devices will always be resource-poor compared to data centers. By

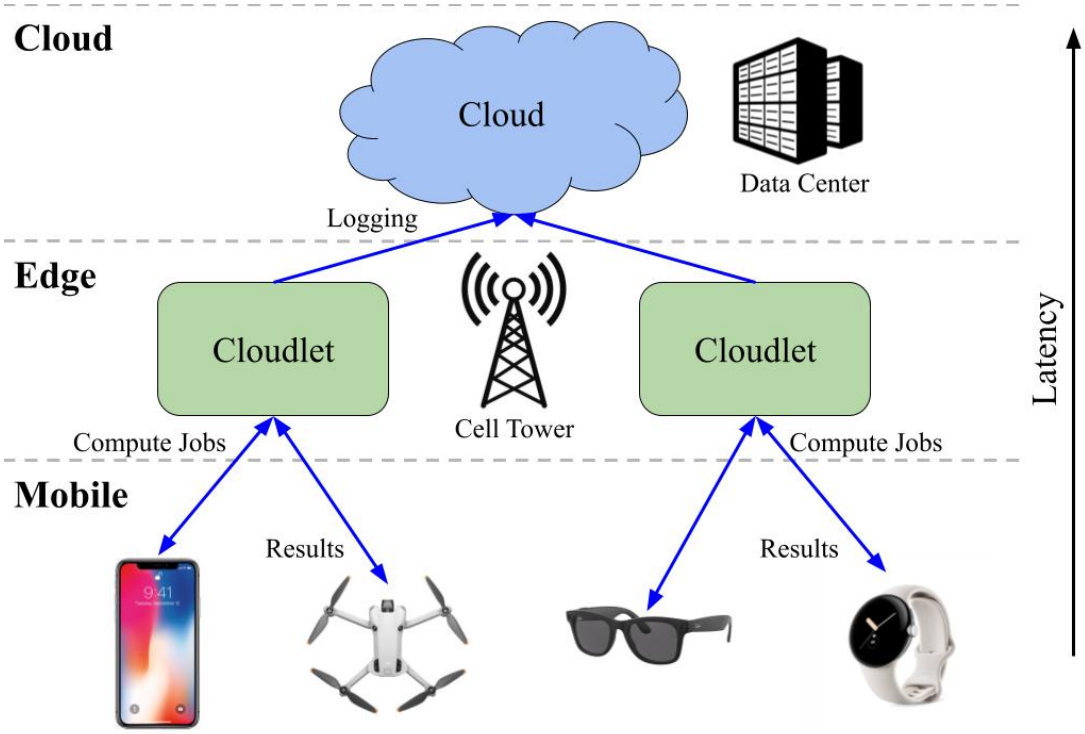


Figure 3.1: Edge Computing Paradigm

sending compute jobs over the network, running on the vast compute of a data center, and getting the result back, mobile devices can mitigate their computational deficiencies.

The insight of edge computing is that some computation is *latency-sensitive*. That is, the faster the result is returned to the user, the better the system performs. This is usually the case for interactive applications, like augmented reality, or for reactive applications, like robotic actuation in response to visual stimuli. In these cases, sending a compute job to the cloud may be too slow. Edge computing positions smaller groups of servers, called cloudlets, physically closer to mobile clients, usually co-located with cell towers. This hugely decreases latency without sacrificing much per-user compute power, since cloudlets, due to their smaller reach, have fewer tenants than cloud servers [25, 41].

3.2 Autonomous Drones and the Edge

Edge computing offers a compelling alternative to the onboard compute trade-off. Instead of miniaturizing compute to fly with a drone, it is much easier to relocate heavyweight compute to the edge so that it is accessible with low latency. This has a number of important advantages over traditional onboard computation paradigms:

- The underlying hardware where compute runs is well-understood and general purpose (server-grade CPUs and GPUs). This hugely increases portability and offers a developer-friendly programming environment.

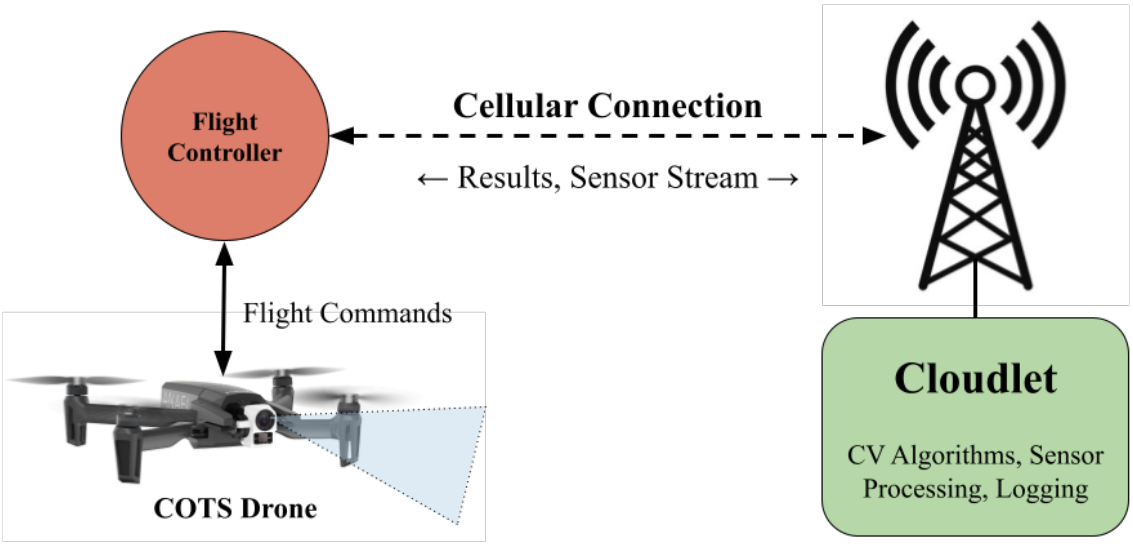


Figure 3.2: SteelEagle Autonomy Model

- The cost of edge-enabled drones is low; there is no longer a need for expensive lightweight compute hardware, only a relatively cheap modem to communicate with the edge. This greatly increases scalability and thus the economic viability of drone swarms.
- Server-grade compute power will always vastly exceed mobile compute power [103]. This opens the door to real-time inference of heavy models like transformers [120].

An edge computing approach is not without its drawbacks. The drone is now fully reliant on its communication link with the cloudlet, meaning it is susceptible to service disruptions and bandwidth constraints. This is no worse than manually-piloted drones, which are also dependent on a communication link to the RPIC. Still, these factors must be accounted for in any successful edge-based drone system.

Edge-enabled drones are not a new idea. For several years, researchers at the intersection of edge computing and robotics have published many influential papers on the topic [13, 15, 58, 121]. In these projects, drones use a ground station or cloudlet in conjunction with other nearby aircraft to offload high compute loads. However, this previous work either focused on the theory of how such a system should be designed or on solutions using custom components. None identified weight as a major constraint, and so they failed to present a practical, fully-functional system for public flight operations.

3.3 SteelEagle: Inducing Autonomy on Lightweight Drones

I introduce *SteelEagle*, a drone-agnostic framework for inducing full autonomy on lightweight drones using edge computing. In contrast to previous work, SteelEagle is built on top of commercial-off-the-shelf (COTS), semi-autonomous, photography drones (§2.2), which are supplemented with a 4G [48] connection to the edge to provide the computation needed for fully-autonomous

operation (Figure 3.2). COTS photography drones are typically cheaper, more accessible, and much lighter than fully-autonomous platforms. With this approach, SteelEagle presents the best of both worlds: lightweight, cheap drones with powerful real time computation capabilities. SteelEagle's backend also supports a plug-and-play approach, allowing developers to swap out underlying drone hardware form beneath its abstraction layer. This makes the system highly portable and versatile, able to adapt to the rapid pace of AI innovation. With respect to the challenges outlined earlier (§2.3), SteelEagle addresses them all:

- **Weight**: SteelEagle is designed to work with lightweight, commercial-off-the-shelf (COTS) photography drones, many of which are under 400 g. These drones typically would be categorized as semi-autonomous, but edge offloading enables them to become fully autonomous.
- **Accessibility**: COTS photography drones are designed to be accessible, since they are marketed to everyday consumers. Thus, they are much easier to work with and safer to fly. In addition, the programming environment and abstractions provided by SteelEagle make it easy to develop new drone applications.
- **Versatility**: SteelEagle's edge backend runs on traditional compute hardware which ensures maximum versatility and full access to popular AI libraries like PyTorch. This encourages rapid development and evolution through preexisting AI tool chains. It also eliminates the need to squeeze models onto constrained drone hardware.
- **Portability**: SteelEagle abstracts away drone-facing hardware and has no restrictions on the drone control stack. This promotes portability and makes SteelEagle drone-agnostic.
- **Cost**: COTS photography drones are some of the most cost-optimized drones on the market, due to their target mass-market audience. As a result, the unit cost for SteelEagle drones can be several orders of magnitude lower than typical fully-autonomous platforms.

As with all edge-based systems, SteelEagle must plan for and adapt to changing network environments. More critically, its performance on common drone tasks like object tracking and obstacle performance must come close to matching that of existing autonomous drones to be useful, despite bandwidth and latency challenges inherent in offloading solutions.

3.4 Design Goals of SteelEagle

Establishing a connection to the edge on any current COTS drone hardware is non-trivial. Doing so on a COTS photography drone is near impossible out-of-the-box. This is the case for the following reasons:

- Photography drones are tightly-integrated, black boxes with no ability to change onboard software or hardware. This is because they are meant to be used by novice pilots and so they simplify the user experience at the cost of customizability.
- Photography drones usually require a constant connection to the pilot via a controller. This not only mandates a human-in-the-loop but also prevents BVLOS operation.
- Photography drones are not designed to carry much payload. They provide no means to power any payload either since their batteries are closed off during flight.

Figure 3.3: Parrot Remote Control Setup [101]

Feature	Description
Weight	320 g
Cost	$470
Flight Time	Up to 25 min, in reality 20 min
Camera	4K photo/video
Video Stream	720p 30fps
Autopilot*	Custom
API	Parrot Olympe / GroundSDK
Connectivity	WiFi

Table 3.1: Parrot Anafi Specifications

These are challenges that any prototype must overcome, regardless of underlying drone platform.

For SteelEagle, drones connect to the edge over public 4G/5G cellular. Cellular has excellent coverage in populated areas, support for device mobility, and favorable penetration characteristics [51]. This ensures good service wherever SteelEagle drones fly, especially in urban or suburban settings. Unfortunately, at the time of writing this dissertation, no lightweight COTS photography drones are equipped with cellular connectivity; they are designed to be operated within visual line-of-sight of a pilot, a task for which WiFi or RC is well-suited. For my prototype to work, I will have to bring cellular connectivity to a COTS drone without modifying its internal components.

3.5 Initial Hardware

For my initial prototype, I chose to work with the Parrot Anafi [101]. The Anafi is a COTS photography drone that weighs 320 g and costs $470. It is equipped with an autopilot that can follow GPS waypoints and perform automatic stabilization. This stabilization involves increasing thrust of rotors to compensate for wind, allowing the drone to maintain its position even in windy conditions. Table 3.1 shows a more detailed specification of the aircraft.

3.5.1 Control Scheme

The Anafi is designed to be controlled by a human pilot using a remote controller with an attached mobile phone (the Parrot Skycontroller) or exclusively using a mobile phone over WiFi. Alternatively, the drone can also be controlled over WiFi with a Python API called Parrot Olympe or an Android API called Parrot GroundSDK. They are based on the ubiquitous MAVLink protocol [83]. Both these APIs grant users high-level flight control and access to telemetry.

Olympe and GroundSDK support two main command types: manual and guided. Manual commands directly actuate the drone. For instance, moving the left stick upwards on the controller would send a manual command to "increase throttle". Guided commands, on the other hand, provide the drone with a target to actuate towards. For example, a "move to GPS location" message is a guided command since the drone self-actuates towards the specified target.

3.5.2 Networking

The drone hosts a WiFi network which supports up to two attached clients. A controller must connect to the WiFi network in order to talk to the drone. This channel may be configured to be either 2.4 GHz or 5 GHz. As reported by Parrot, the connection range of the WiFi channel is around 4 km [101]. From my testing, this can fall to around 0.5 km in areas with high wireless interference.

3.5.3 Video Stream

A 720p 30fps video stream is generated by the drone and transmitted on its WiFi network via the real-time streaming protocol (RTSP) [84]. The settings of this stream are not configurable. It is encoded using an intra-refresh slice-decode scheme which is designed to be resilient to packet loss [31]. I will cover the drone's video stream in more detail in Section 3.8.

3.5.4 Magnetometer

Most drones are equipped with a magnetometer for discerning bearing. The Parrot Anafi has an unusually sensitive magnetometer. If the drone's magnetometer detects sufficient interference (e.g. caused by nearby ferromagnetic material [12]), it refuses to fly any guided commands and cancels any guided actuation. This is for safety; if the drone does not have an accurate understanding of its bearing, it may not actuate correctly towards its target and therefore increase the probability of a crash.

3.6 Achieving Cellular Connectivity

The Parrot Anafi, like most photography drones, only supports WiFi connectivity and mandates a constant connection to the pilot controller. As stated earlier, the goal of SteelEagle is to use this drone without modification and with an entirely COTS payload. Thus, any prototype must operate within these constraints. In order to integrate a cellular connection into the loop, it must

Figure 3.4: Android Phone Control [101]

ensure that from the drone's perspective, it is always connected to what it thinks is a human pilot. One possibility is to mount a router-like device onboard which connects to the drone WiFi and also maintains a cellular connection to the edge. Such a device would have to be light enough to fly but also would need to be able to power itself for the duration of a flight. Ideally, it would also have some internal compute so it can still control the drone during brief network outages.

3.6.1 Mobile Phone

In the COTS mobile device space, one of the best SWaP (size, weight, and power) optimized devices is the mobile phone. Mobile phones have grown immensely powerful in recent years, with some rivaling the power of laptop computers. Furthermore, they are lightweight (usually under 200 g), have a full-featured software development environment, and can power themselves for several hours, even under load. Their connectivity over WiFi and cellular is highly reliable and extensively tested.

In the context of the Parrot Anafi, an Android phone would work well for this purpose, since the Anafi supports an Android API, Parrot GroundSDK. This motivates the following model: an Android phone flies onboard the Parrot Anafi, acting as both a router and a stand-in for the remote controller. By running GroundSDK, the phone will work within the Parrot ecosystem and will therefore maintain the pilot connection that the drone necessitates (Figure 3.4). To save weight, I selected one of the lightest widely-available Android phones on the market for this task: the Google Pixel 4a. The Pixel 4a weights 143 g and has both 4G and 5G connectivity.

Now that I had selected a device, it was time to mount it onboard the drone. For my initial mount, to save weight, I used industrial strength velcro and rubber bands. The drone was able to fly autonomously with the phone onboard and offload its video stream to the edge. However, I observed strange flight characteristics which caused several crashes. Upon further investigation, I determined that the phone was interfering with the magnetometer on the drone. This was causing the drone to lose track of its bearing mid-flight and veer off course.

To solve this magnetic interference, I added a layer of EMI (electro-magnetic interference) shielding between the phone and the drone chassis. The shielding was made of Mu-metal, a nickel-iron ferromagnetic alloy used to absorb electro-magnetic radiation [96]. After testing, I determined that this mostly negated the observed interference. Unfortunately, it also added

Figure 3.5: Android Watch Control [101]

significantly to the payload weight, so much so that it exceeded the maximum takeoff mass (MTOM) of the aircraft. Once an aircraft's payload surpasses MTOM, it cannot fly because the lift it generates is no longer greater than its weight. Even with the lightest Android smartphones available, like the 61 g Unihertz Jelly Pro, the added weight of EMI shielding exceeded the Anafi's MTOM.

As a result of the interference and MTOM restrictions, I was forced to abandon the phone prototype. It was too heavy for the Parrot Anafi hardware. I began searching for an alternative COTS device, much lighter than the Pixel 4a, but with the same broad specifications (self-contained, WiFi and cellular, self-powered, able to run Parrot GroundSDK or Olympe).

3.6.2 Smartwatch

While mobile phones are the best SWaP-optimized devices on the market, smartwatches are by far the lightest, self-contained, commercially available computers. For many years, they were designed to operate in tandem with a phone, acting as a kind of always-visible secondary display. As time progressed, smartwatches grew more independent, incorporating more connectivity options and mounting more internal compute. The fitness community was a driving force behind this transformation. Many runners, for example, wanted a very small, lightweight, wearable device that could track their progress, contact emergency services, and play music on the move. In the late 2010s and early 2020s, this resulted in smartwatch products with cellular connectivity and enough local compute to run basic applications.

In many ways, smartwatches are the perfect device for SteelEagle's use case. They are extremely lightweight (less than 30 g compared to over 100 g for a phone), self-powered, cellular and WiFi-enabled, and able to run a slimmed-down Android (WearOS). For my smartwatch-based prototype, I chose the Samsung Galaxy Watch 4. At the time, this was the lightest cellular-capable Android smartwatch available. The Samsung Galaxy Watch 4 weighs just 25 g.

After porting Parrot GroundSDK to WearOS, I was able to establish a 4G connection from the edge to the watch while autonomously piloting the drone via onboard software (Figure 3.5). Now, all that was left to do was to physically mount the watch on the drone. I used a 3D-printed harness for this task, which clips onto the Parrot Anafi's removable battery. The harness weighs an additional 14 g, bringing the total payload weight to 39 g. Figure 3.6 shows the full assembly

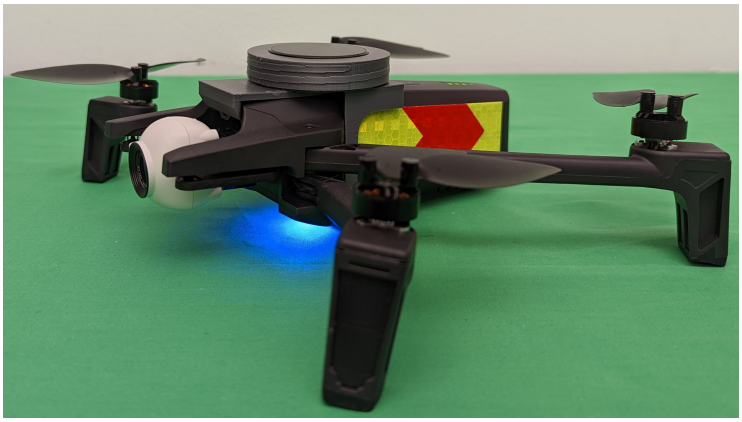


Figure 3.6: Drone with Watch-Harness Payload

mounted on the aircraft. This brings the total take-off weight to 359 g, just 109 g above the 250 g FAA regulation threshold. Through rigorous testing, I determined that the watch payload did not exhibit the same negative flight characteristics as the Pixel 4a payload. It did not create as much EMI, likely because of its smaller size and lower power draw, and thus no shielding was needed. Additionally, the weight of the payload was far below the Anafi's MTOM, even with the added 3D printed mount to securely fasten it onboard.

The Samsung Galaxy Watch 4 finally yielded a successful prototype. The watch payload was able to autonomously fly the drone while maintaining a cellular connection to the edge. However, much work still remained unfinished. In particular, the drone's RTSP video stream would need to be transmitted to the edge in order to run the AI algorithms responsible for true autonomy: object detection, image segmentation, and obstacle avoidance among others. This would prove to be a formidable obstacle that pushed the watch hardware to its limit.

3.7 An Austere Computing Environment

The watch is an austere computing environment with a dual-core 1.18 GHz ARM Cortex-A55 processor, 1.5 GB RAM, and 16 GB flash. Table 3.7 compares its attributes to the Google Pixel 4a and to the Unihertz Jelly Pro, the lightest-available Android smartphone. As wearable hardware, the watch has stringent thermal protection to shut itself down if its temperature approaches hardware limits. In addition, since wearables are meant to be worn, these hardware temperature limits must be much lower than traditional computers to prevent skin burns [66]. Both computing and network transmission cause watch temperature to rise significantly.

This is a major problem, especially since in my design, the watch offloads the drone video stream to the edge. Typically, streaming tasks are both compute and transmission intensive. The act of sending frames too often or decoding frames onboard could cause overheating. In the case of the Samsung Galaxy Watch 4, overheating is detected by the operating system which in turn triggers a thermal shutdown. If this occurs, all currently running applications are suspended and the watch goes into hibernation until it is cool enough to proceed. A thermal shutdown in-flight would turn off the Parrot GroundSDK software running onboard and would thus trigger a

	Samsung Galaxy Watch 4	Unihertz Jelly Pro	Google Pixel 4a
Weight	26 g	61 g	143 g
CPU cores	2	4	8
CPU speed	1.18 GHz	1.45 GHz	2.2 GHz
Memory	1.5 GB	3 GB	6 GB

Figure 3.7: Austerity of Mobile Hardware

pilot disconnection event on the drone. When this occurs, the drone returns automatically to its takeoff GPS location and lands. While this is far from disastrous, it would certainly be a serious handicap. As a result, the watch must be conservative in its network transmission and onboard computation in order to prevent overheating.

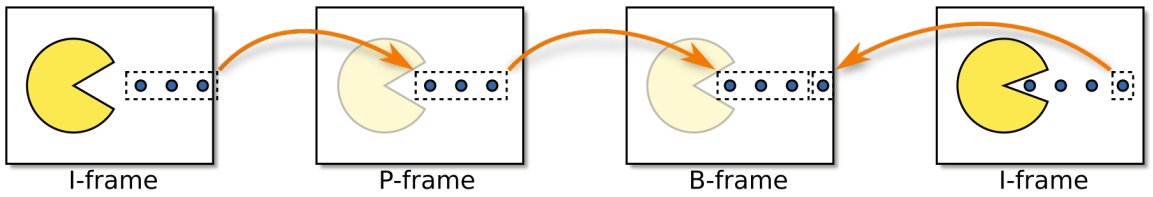
On the other hand, the agility and accuracy of the computer vision algorithms running at the edge are critically dependent on the attributes of the video stream. These algorithms determine the overall performance of the system; they are the "brain" that enables fully-autonomous operation. If the video stream delivered to them is hindered, performance will be directly affected. In order for this system to be effective, the watch must deliver as high fidelity a stream to the edge as possible without compromising its thermal limits. This is a delicate balancing act.

3.8 Anatomy of the Parrot Anafi Video Stream

The Parrot Anafi produces an encoded 720p 30fps video stream. It is produced on the drone's hardware and thus *cannot be modified*. It is generated over the drone's WiFi network and can only be consumed by a single client on the network. Unlike most video streams, this stream uses an *intra-refresh slice-decode* encoding scheme. An intra-refresh slice-decode encoding scheme is a streaming paradigm designed to minimize the visual impact of packet loss.

To understand this special encoding scheme, first consider a normal H.264 video stream. The encoding scheme for these streams involves two types of data: complete frames (I frames) and inter frames (P and B frames). A complete frame is a full photo capture of a moment in time. When a complete frame is transmitted, it typically requires no additional data to decode. The trade-off is that complete frames are bandwidth hungry. An individual 720p complete frame can be several KBs of data. By contrast, inter frames only capture the *change from the previous frame*. They require reference data in order to decode, and are meaningless on their own. Their advantage is that they are very bandwidth efficient, sometimes on the order of a few hundred bytes in size. An H.264 video stream operates by transmitting a complete frame followed by a set amount of inter frames before looping back. This preserves fidelity by smoothing out gaps between complete frames with inter frames but also preserves bandwidth by sparingly transmitting complete frames.

There is one major negative of typical H.264 video streams in the context of mobile devices: they are not resilient to packet loss. On the Internet, which is a mostly wired network, this is not a serious problem. But, over the air, packet loss is frequent. If a complete frame is lost, viewing or



I frames are full frames which are interleaved with partial frames called P frames and B frames. These are frames that contain predicted motion based on previous frames and future frames within the transmission buffer.

Figure 3.8: Video Compression Frame Types (Adapted from [124])

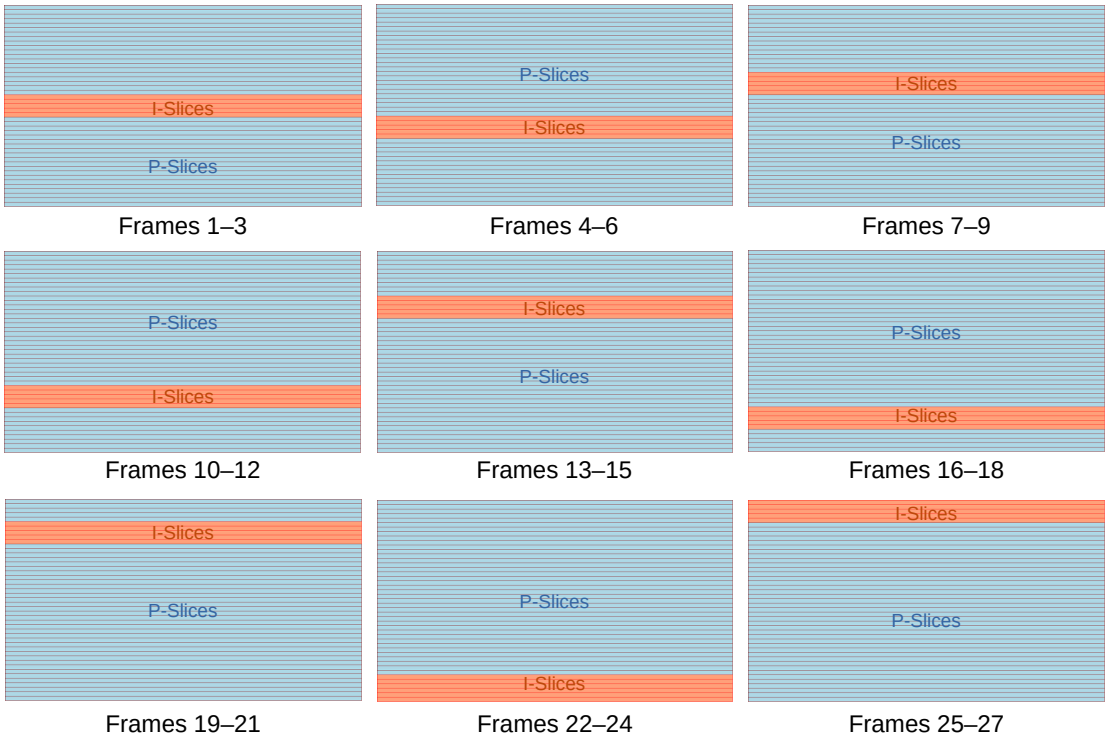
processing of this kind of stream must be skipped until a new complete frame can be broadcast; all following inter frames will have no complete frame to refer to and will be rendered useless. Usually this can take a whole second or more, during which a drone pilot, for example, would not be able to see.

An intra-refresh slice-decode scheme takes a different approach. Instead of complete frames and inter frames, this scheme splits a single frame into *slices*. These slices capture a sliver of data horizontally across a frame. When a frame is transmitted, the encoder allocates one slice to be the "complete slice" (I slice) and the rest to be "inter slices" (P slices). Figure 3.9 shows a breakdown of how each set of frames transmitted by the Anafi are organized. Each inter slice is only dependent upon its section of the frame; they are completely independent of each other.

This has two positive attributes. The first is that this minimizes the impact of packet loss on a single frame. If a single frame is dropped, only one slice of the frame loses its complete slice reference. In this case, that section of the frame is corrupted until a new complete slice is sent. Otherwise, the rest of the frame is decoded normally. The second benefit of this approach is that it keeps bandwidth usage consistent. With a traditional stream, bandwidth usage spikes when complete frames are sent but then plummets when inter frames are sent. In this scheme, since every frame has approximately the same size (one complete slice with the rest being inter slices), bandwidth usage stays somewhat constant. This is useful because other applications can now plan around its tight usage bound.

3.9 Processing The Video Stream on the Watch

There are two obvious methods for offloading the drone video stream from the watch to the edge. The first is to decode the drone stream on the watch and then ship the decoded frames to the edge. This allows the watch to throttle its send rate to account for 4G transmission overheating. The second is to directly offload the stream packets from the drone network to the cloudlet, where they can then be decoded. This involves the least amount of computation power on the watch, but requires a higher transmission rate. I will refer to these as decode-on-device and decode-on-edge respectively.



Each segment of three frames contains one I-slice (shown in orange) and transmits the remaining slices as P-slices (shown in blue). The I-slice that is sent is chosen from the center of the frame going outward. After 30 frames, the process starts from the beginning.

Figure 3.9: Intra-Refresh Slice-Decode Stream

	Decode Time
Samsung Galaxy Watch	55ms
Unihertz Jelly Pro	35ms
Google Pixel 4a	25ms

Red denotes a decode rate that is too slow at 30 FPS.

Table 3.2: Average Decoding Time by Platform

Sleep Interval	Payload Size	
	35 KB	100 KB
33 ms	✗	✗
100 ms	✗	✗
500 ms	✗	✗
800 ms	✗	✗
1000 ms	✔	✔

✗ thermal shutdown

✔ no thermal shutdown

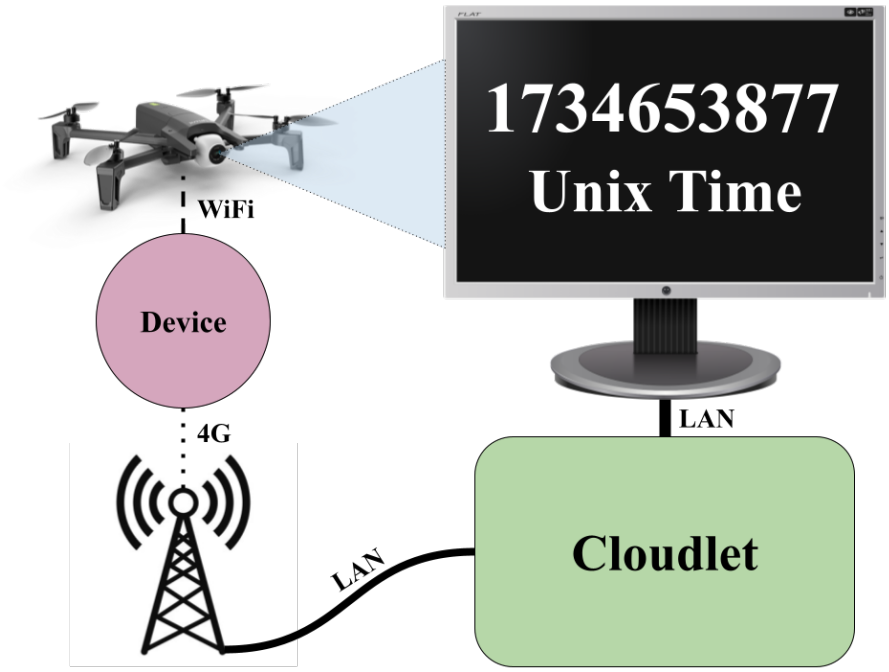
Table 3.3: Effect of Sleeps

3.9.1 Method 1: Decode-on-Device

Decode-on-device involves the watch decoding the drone video stream then sending decoded frames individually to the edge. This allows for some pre-processing on the watch side, such as downscaling or stream throttling for bandwidth saving. Unfortunately, the Anafi's intra-refresh slice-decode stream poses a major hurdle to this method's success: due to its construction, *every frame must be decoded in order to maintain a coherent stream*. This is in stark contrast to a typical stream, which only requires the I frames (usually generated at around 1 fps) to be decoded to maintain coherence. As such, any device consuming the drone stream must decode one frame every 33 ms on average to keep up with the 30 fps output and prevent drifting latency, regardless of cellular transmission throughput. Drifting latency is a phenomenon where latency increases over time due to queueing delay. If a device drops packets to keep up with this strict time bound, it could cause some loss of quality as important I slices are discarded. On most devices, this is not an issue. But on the constrained compute of the Galaxy Watch 4, even relatively light workloads are considered difficult. Table 3.2 shows the time taken to decode a frame of the Anafi's video stream by the watch, the Pixel 4a, and the Unihertz Jelly Pro. As shown, both the watch and the Jelly Pro cannot decode fast enough to maintain stable latency.

3.9.2 Method 2: Decode-on-Edge

Decode-on-edge avoids decoding on device by directly sending the drone stream packets over cellular to the cloudlet, where they can then be decoded. With this method, computation on the device is minimal but network transmission is much more frequent. This is a good trade-off in many cases, but on the watch, heavy transmission workloads may cause overheating. To determine the effect of transmission on the watch thermals, I devised a simple experiment: the



The drone camera captures UNIX time displayed on a monitor that is connected to the cloudlet. It then sends the video stream over the device to the cloudlet. The total latency is the difference between the observed time in the stream versus the actual time the frame was received.

Figure 3.10: Streaming Experiment

watch sends a variable size packet, representing typical stream packet sizes (35KB up to 100KB), over the network with variable sleep times in between sends. There is no video stream or other onboard computation involved, only raw transmission. Table 3.3 shows the results of this experiment. It is clear that the bottleneck on the watch is not the packet size but the send rate. Any send rate above 1 Hz causes thermal shutdown. This means that decode-on-edge streaming is mostly out of the question.

3.10 The Quest for a Working Stream

A successful stream to the cloudlet must have acceptable latency, throughput, and image quality. Without these aspects, the stream will not be usable by AI algorithms running on the edge and therefore cannot contribute to fully-autonomous operation. Specifically, the stream must have:

- **Sustainability**: the watch stream must sustain itself for the duration of a drone flight (at least 20 minutes).
- **Stable Latency**: the watch stream must process frames on average equal to or faster than 33 ms to keep up with the 30 fps stream from the drone.
- **High-Quality Frames**: the watch stream must deliver "quality" images to the cloudlet that lack significant visual artifacts and can thus be used by computer vision algorithms.

	Pass-Through Stream	Decode No Throttle	Decode 3 fps Throttle	Decode 1 fps Throttle
Samsung Galaxy Watch 4	68 s	380 s	565 s	DNO
Unihertz Jelly Pro	DNO	TT	TT	DNO
Google Pixel 4a	DNO	DNO	DNO	DNO

Time in seconds before the device experiences a thermal shutdown. DNO means that the device did not overheat for the duration of the experiment (20 minutes). TT means that the device experienced thermal-related CPU throttling which affected performance.

Table 3.4: Experiment 1: Sustainability

The austere environment on the watch makes achieving such a stream a challenge. The watch cannot decode frames fast enough to maintain stable latency, and thus cannot decode-on-device and send at a throttled framerate. It also cannot transmit consistently enough to decode-on-edge. To discover a path forward, I devised a set of experiments that would better characterize the bottlenecks in the system. By understanding these bottlenecks, an effective solution could be found.

3.10.1 Experimental Setup

For each of the following experiments, the setup is identical. The drone is started so that its RTSP stream and WiFi network are initialized. Then, the client device is connected to the drone's WiFi network and the stream application is launched on the device. The application uses a variable method to stream image data from the drone to a cloudlet. When the frame arrives at the cloudlet, it is saved along with its time of arrival. The drone's camera is pointed at a screen connected directly to the cloudlet which is displaying the current UNIX timestamp. This effectively timestamps each frame produced by the drone, a feature not present in the closed source stream hardware. By comparing the UNIX timestamp in the received image and its time of arrival, the transmission latency can be determined in post processing. The UNIX timestamp can also be used to determine the time at which thermal shutdown occurs. Figure 3.10 shows the experimental setup. For experiment 1 and 2, I provide two other devices, the Google Pixel 4a and the Unihertz Jelly Pro as baselines. The duration of the experiments in all cases is the duration of a drone flight: 20 minutes.

3.10.2 Experiment 1: Sustainability

In this experiment, I timed each device running either decode-on-device §3.9.1 or decode-on-edge §3.9.2. For the decode-on-device runs, I added additional tests with throttling. In these cases, the device decodes at the full frame rate but only sends to the cloudlet at the specified framerate. Table 3.4 shows the results of the experiment. The watch clearly cannot sustain decode-on-edge streaming for any significant duration (about 1 minute), and can sustain non-throttled and 3 fps throttled streaming for up to 9 minutes. However, the 1 fps throttled stream

	Pass-Through Stream	Decode No Throttle	Decode 3 fps Throttle	Decode 1 fps Throttle
Samsung Galaxy Watch 4	DRIFT	DRIFT	DRIFT	DRIFT
Unihertz Jelly Pro	DRIFT	DRIFT	DRIFT	1 s
Google Pixel 4a	DRIFT	1 s	1 s	1 s

Latency in seconds. DRIFT means that the latency increased over the duration of the experiment (20 minutes).

Table 3.5: Experiment 2: Stable Latency

	$c = 1$	$c = 2$	$c = 3$	$c = 4$
$d = 1$	7 s, 80%	7 s, 80%	8s, 90%	8 s, 70%
$d = 2$	7 s, 80%	7 s, 80%	7 s, 90%	7 s, 80%
$d = 3$	1 s, 30%	1 s, 99%	4 s, 60%	4 s, 50%
$d = 4$	$\leq 5\%$	1 s, 50%	1 s, 10%	6 s, 30%

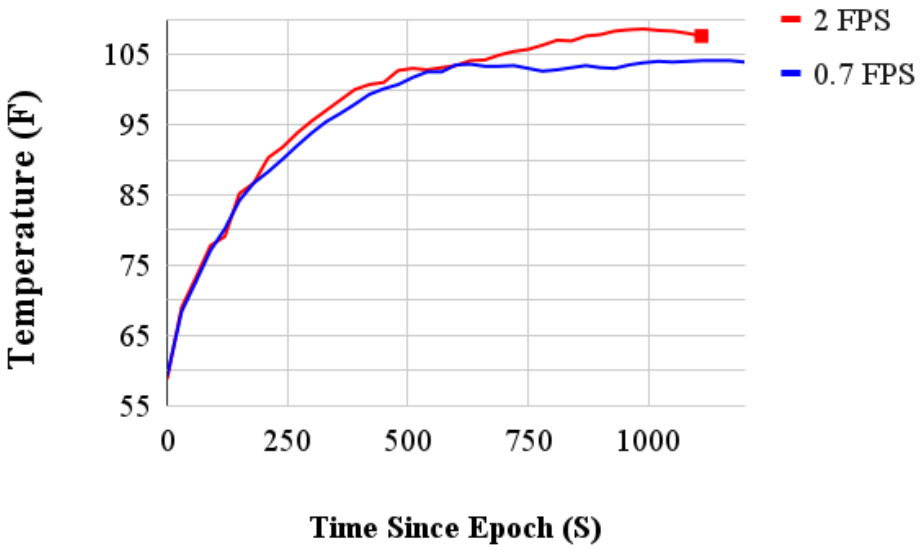
Latency in seconds followed by the approximate percentage of visual-artifact-free frames produced during the test. Combinations off the grid did not produce acceptable results, either due to drifting latency or poor quality. Visual quality is determined subjectively.

Table 3.6: Experiment 3: High-Quality Frames

did not overheat for the duration of the test. This implicates 4G transmission as the main thermal bottleneck and sets an upper-bound for transmission frequency at 1 Hz. The Google Pixel 4a, unsurprisingly, breezed through all four tests. The Jelly Pro had trouble with the higher send rate of some decode-on-device tests but was overall good. This shows that slightly better thermals would drastically improve sustainable performance on the watch.

3.10.3 Experiment 2: Stable Latency

In this experiment, I setup each device running the same methods as in Section 3.10.2. This time, I measured the stream latency over the course of the test. Table 3.5 shows the results of the experiment. No method provided a stream with stable latency on the watch. On the Jelly Pro, only the 1 fps throttled stream did. This implies that for the non-throttled and 3 fps throttled stream, the network transmission affected the decoding performance, slowing it down and causing latency to drift. On the Pixel 4a, all decode-on-device methods provided stable latency. No device successfully performed decode-on-edge streaming without drift. Clearly, a different approach is needed on the watch to guarantee consistent latency.



The red square at the end of the 2 fps temperature curve indicates a thermal shutdown occurred. The 0.7 fps graph continues without overheating.

Figure 3.11: Watch Temperature at Different Throttling Levels

3.10.4 Experiment 3: High-Quality Frames

If the watch decodes the stream as outlined in §3.9.1, it produces very high-quality images with no visual artifacts at the cost of drifting latency. This is not acceptable by the standards outlined earlier in Section 3.10. Even so, room for compromise remains. While not desirable, intentional packet dropping of the video stream prior to decoding is an effective way to reduce latency at the cost of quality, since fewer stream packets reduces the time spent decoding. In this experiment, I determine how much packet dropping is acceptable to preserve quality. I test the watch under the 1 fps throttled decode-on-device workload, decoding chunks of c packets and then subsequently dropping d packets. For example, a $c = 2, d = 2$ scheme would involve decoding 2 out of every 4 packets. Table 3.6 shows the results of the experiment. As d increases, quality drops but latency decreases as the average time to decode shrinks. In the other direction, as c increases, latency generally increases too since the average time to decode grows. At $c = 2, d = 3$, there is a sweet-spot where the watch latency is minimal and the impact on frame quality is also minimal. It is this combination that finally produces a workable stream.

3.10.5 The Stream in Practice

Putting together the results of the three experiments, a working solution emerges. It solves each of the requirements presented in Section 3.10:

- **Sustainability**: *the watch stream must sustain itself for the duration of a drone flight (at least 20 minutes).* The watch throttles sending frames to the cloudlet to 1 Hz which is sustainable for 20 minutes.

- **Stable Latency**: *the watch stream must process frames on average equal to or faster than 33 ms to keep up with the 30 fps stream from the drone.* The watch drops 3 packets for every 2 it decodes which reduces the average decode time enough to keep up with the drone stream.
- **High-Quality Frames**: *the watch stream must deliver "quality" images to the cloudlet that lack significant visual artifacts and can thus be used by computer vision algorithms.* The watch decodes enough packets to maintain a high-quality stream.

Integrated into the larger application, parts of this streaming scheme must be modified to make way for other compute resources. In particular, throttling to 1 fps proves too demanding once other processes are factored in. Once throttling is dropped to 0.7 fps, the stream once again is functional. Figure 3.11 shows the watch temperature over time with a 2 fps and a 0.7 fps throttled stream within the larger application. At 2 fps, the stream experiences thermal shutdown once the watch reaches an external temperature of 108°F. At 0.7 fps, the watch maintains an external temperature under 105°F for 20 minutes without ever experiencing a thermal shutdown.

3.11 Summary

The core insight of SteelEagle is that heavy onboard compute can be avoided by offloading computation from drones over a cellular network to a cloudlet. As a first demonstration of this concept, I successfully connected the Parrot Anafi to a cloudlet via the Galaxy Watch 4. The watch, despite its thermal constraints, can offload the drone's video stream over 4G at 0.7 fps with 1 s latency while simultaneously sending actuation commands. The drone can also operate BVLOS without the need for any connected human pilots. Now, all that is needed is a complementary edge software backend that can provide the drone with the intelligence required for full autonomy.

Chapter 4

The SteelEagle Backend

Fully-autonomous drones are only as effective as the artificial intelligence software and decision-making protocols they use. SteelEagle is no different; its performance is tightly correlated with the sensor processing and control algorithms running on the edge. However, its offload-based design poses additional constraints: all sensor data from the drone is inherently latent and may have suboptimal throughput. Furthermore, cloudlets may be required to serve multiple drones, and may have to deal with changing network conditions that hamper bandwidth.

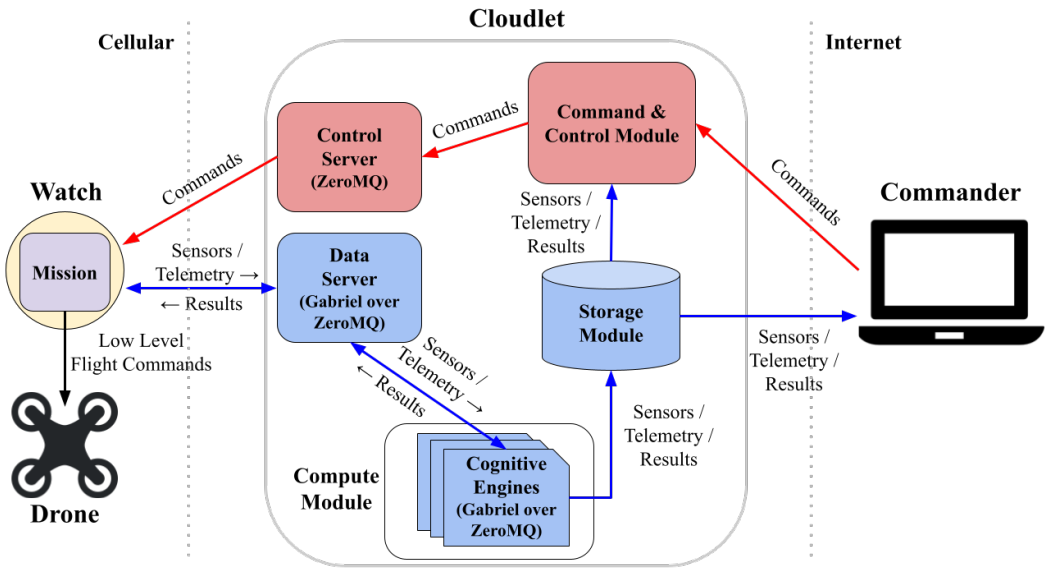
In this chapter, I will discuss my architecture for the SteelEagle backend. I will show how it addresses these unique problems and how it adapts to its network environment. In Section 4.1, I outline the overall design of the system and its interaction with drone clients. In Section 4.2-4.4, I evaluate the system on a few basic tasks to understand its capabilities.

4.1 An Edge Intelligence Framework for Drones

The SteelEagle backend is made up of three main modules: compute, command and control, and storage. To handle transmission between these modules and between the cloudlet and external clients, traffic is split into two channels, the data plane and the control plane. Each channel uses a different communication protocol to suit their specific quality-of-service demands. Figure 4.1 shows the full system architecture. The overall system is characterized by two main data flows: the remote processing flow through the data plane and the remote control flow through the control plane. I will outline both to illustrate how the system operates.

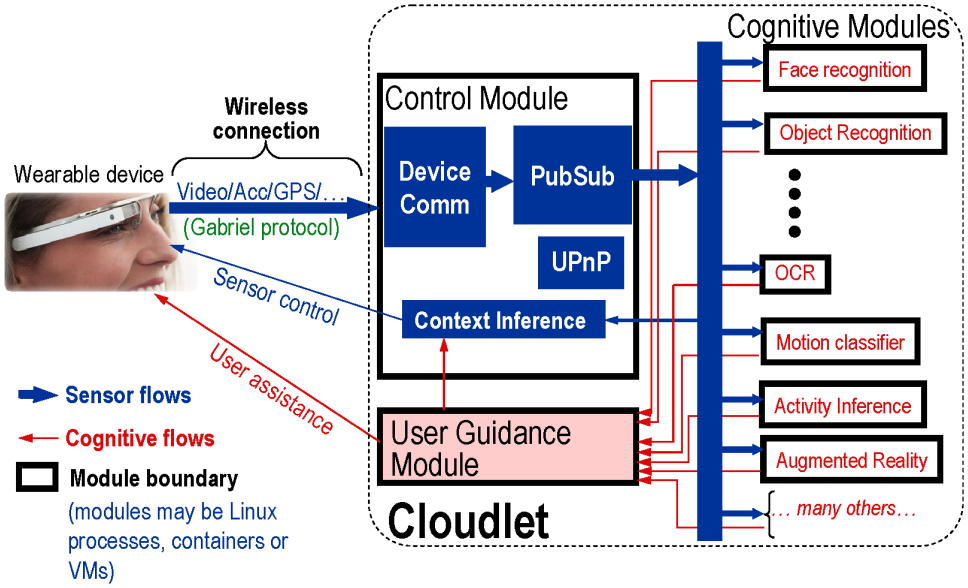
4.1.1 Remote Processing Flow

Perhaps the most important data flow in SteelEagle is the remote processing flow, outlined in blue in Figure 4.1. The role of this flow is straightforward: retrieve sensor data and telemetry from the drone, run the requested algorithm by the current mission, return results to relevant clients, repeat. Yet there are additional considerations that complicate this seemingly simple picture. First, bandwidth is a precious resource for mobile systems, and availability may vary highly based on current network conditions [55]. For this reason, it is important that the stream retrieval portion of the data flow can cope with bandwidth constriction without introducing too



The remote processing flow, which handles the processing of the drone's sensor stream and the delivery of generated results, is highlighted in blue. The remote control flow, which delivers all commander messages and auto-generated commands from the command and control module, is highlighted in red.

Figure 4.1: SteelEagle System Architecture



In the case of SteelEagle, the Wearable Device is replaced with a drone communicating over cellular. The User Guidance VM not only sends processing results back to the drone through the data server (as shown in Figure 4.1), but also stores results in the storage module.

Figure 4.2: Gabriel Cognitive Assistance Model (Adapted from [62])

much latency. Second, many DNNs require vast computational resources to run quickly. Some may not be able to sustain sufficient throughput to keep up with the drone's stream. If the data flow takes on multiple tenants without adapting its inference rate to model throughput, it could lead to queueing delay and thus added latency.

Gabriel and the Cognitive Assistance Model

To address both of these problems, the SteelEagle remote processing flow leverages the Gabriel Cognitive Assistance model [62]. Proposed in 2014, Gabriel is a bandwidth-adaptive edge intelligence system that provides near real-time sensor processing results to mobile clients (Figure 4.2). Gabriel is bandwidth-adaptive and model-throughput-adaptive thanks to its token-based flow control protocol. Token-based flow control is a technique for avoiding queueing delay in a client-producer server-consumer system using data objects called tokens. Tokens are a certificate that signal the server or client to act; if the server has a token, it has client data that needs to be processed and if the client has a token, it must send its most recently produced data to the server. Additionally, it limits the number of outstanding requests which avoids excessive queueing when the client and server operate at different rates.

At system start, a set number of tokens is agreed upon between the server (also called the control VM in Figure 4.2) and client. The server and client exchange these tokens based on their individual processing rates. For instance, if the client's send rate outpaces the server's processing rate, the server will be in possession of a large share of the tokens and vice versa. If no tokens remain on the client side, the client drops the current payload until it sees a new token. If no tokens remain on the server side, it waits to receive new client data. In this model, the client only ever sends the most recently produced payload which prevents the server from receiving highly latent data.

In Gabriel, each distinct sensor processing algorithm is called a "cognitive engine". Gabriel supports running several cognitive engines simultaneously, and manages their input data queues. Each cognitive engine is run in a separate container on the host machine, and is shipped data via an inter-process communication channel. Gabriel's tokens are shared among the engines which effectively bottlenecks the client send rate to the throughput of the fastest engine. On slower engines, queued data is dropped to ensure the latest data available is processed.

SteelEagle Upgrades to Gabriel

Gabriel is designed for mobile clients that have stable connections to the edge over WiFi. Due to this, underlying connections are made via websockets. Websockets are a point-to-point connection medium that are intended for web browser to server links. They are performant but must be modified to properly deal with disconnections. In the case of SteelEagle, cellular links to the backend can be easily disrupted by drone motion or interference. This renders websockets sub-optimal. To fix this, SteelEagle replaces the underlying communication in Gabriel with ZeroMQ sockets that provide auto-reconnection among other quality-of-service guarantees [132]. Other connections within SteelEagle that are not Gabriel-regulated use pure ZeroMQ sockets (Figure 4.1).

Within SteelEagle, the Gabriel control VM is called the *data server* and cognitive engines are managed by an entity called the *compute module* (see Figure 4.1). In most cases, the compute module will spawn a pre-specified set of algorithms demanded by a drone at mission start. In the future, I envision that it may also support runtime configuration of cognitive engines and dynamic scale-out at runtime to adapt to constrained resources or new mission parameters.

Unlike the default Gabriel implementation, SteelEagle supports multiple consumer clients of processing results (generated by the User Guidance VM in Figure 4.2). The main consumer of these results other than the drone is the storage module (see Figure 4.1). The storage module is a database responsible for logging all processed results in addition to the raw sensor and telemetry sent by the drone. This can be read by other monitoring clients such as the command and control module if needed for orchestration purposes.

Supported Cognitive Engines

As a result of its Gabriel-based design, SteelEagle supports a wide range of cognitive engines. To add new engines, users simply need to work within the Gabriel cognitive engine interface and connect to the Gabriel server [62]. The server will automatically configure the requested type of sensor stream and manage timely delivery of data to the engine. For initial evaluation purposes, I have configured two engines useful for drone operations: object detection and obstacle avoidance. I will detail their implementation in Section 4.2.

4.1.2 Remote Control Flow

The remote control flow handles interactions between humans and SteelEagle drones. Within the SteelEagle ecosystem, since the drones are fully-autonomous, there are no conventional human pilots. Instead, humans who interact with SteelEagle drones are referred to as "commanders". Commanders create missions, send missions to aircraft, and monitor telemetry data to ensure continued safe operation. They also have the ability to manually pilot a connected aircraft in case of emergency. A user interface provides continuous feedback on the progress of the mission on a map. It also displays live sensor streams, including video, transmitted by the drone. It includes several buttons to upload missions, take manual control of one or more aircraft, and order one or more aircraft to return home.

Messages sent by commanders are aggregated in the command and control module (see Figure 4.1). Here, messages are relayed to the control server which then sends them to the target aircraft over a direct, unregulated socket connection. This is in contrast to the data server, which uses token-based flow control socket connections. The reason for this difference is that messages sent by commanders, especially emergency commands, are deemed to be high priority and must therefore be forced over the link as quickly as possible without any regard for bandwidth.

The command and control module also acts as an air traffic controller. That is, it manages the airspace inhabited by connected aircraft to ensure there are no mid-air collisions. For instance, it allocates non-intersecting altitude slices to different drones operating in the same mission area to ensure they are flying at different altitudes. The command and control module maintains up-to-date telemetry from all connected drone clients via the storage module and can therefore detect potential crashes before they happen.

Drone Missions

Prior to flight, a mission is planned using a toolchain that starts with Google MyMaps. The flight route can be specified using waypoints, line segments and polygons. Actions such as capturing images, tracking particular objects, or avoiding obstacles can be associated with parts of the route, and can be linked together to form a primitive behavior tree. Behavior trees are a common way of expressing missions in a variety of robotics settings [59]. An offline compilation step, usually performed on a commander computer, transforms the high-level specification into low-level drone-specific runtime actions on the cloudlet and on the drone. In the case of the Parrot Anafi-Galaxy Watch 4 prototype, the compiler output is expressed via Ground SDK API Java classes and packaged into an Android DEX file. A drone simulator can be used for testing and visualization of this output.

In its current form, SteelEagle assumes a one-to-one correspondence of drones and missions. In particular, each drone executes its given mission in isolation, without cooperation. In future, to support swarm operations, this would change. For now, this simple abstraction provides enough for basic autonomous drone operations.

4.2 Evaluation

Given the prototype constraints discussed in Section 3.7 and my proposed backend architecture, I ask the following question: *"Can my flight platform, with severe constraints on both local compute and edge offload, achieve autonomy for active vision?"* Recall, active vision tasks are those that require the drone to react in real time to its surroundings [6, 93]. To answer this question, I conduct experiments with a series of tasks of increasing difficulty that probe and quantify the limits of my flight platform:

- **Task-1:** Detecting a moving object while hovering.
- **Task-2:** Detecting and tracking an object by yawing to keep it in the field of view (FOV) as it moves.
- **Task-3:** Detecting and tracking an object by following it at a fixed leash distance as the object moves.
- **Task-4:** Detecting a moving object from high altitude, and then descending to closely inspect it.
- **Task-5:** Detecting and avoiding static obstacles.

My goal is to perform these tasks from takeoff to landing with no human intervention. Where possible, I test my system against the Parrot Anafi Ai, a semi-autonomous COTS drone with LTE support, limited onboard tracking and obstacle avoidance. It weighs more than twice as much as my flight platform, and costs over five times as much (Figure 4.1). A less-constrained platform can clearly do more, but it may weigh more too. My focus is on whether my 360 g drone-watch platform is too weak or just enough.

The cloudlet used in the following experiments has 36 CPU cores, 128 GB of RAM and an NVIDIA GeForce GTX 1080 Ti GPU. It is capable of using a private CBRS LTE network for

	My Platform	Base Parrot Anafi	Parrot Anafi Ai
Cost	$769	$469	$4,500
Weight	360 g	320 g	898 g
Detection	Yes	No	No
Tracking	Yes	Yes*	Yes*
Avoidance	Yes	No	Yes
Programmable	Yes	Yes	Yes
4G/5G	Yes	No	Yes

* Requires assistance from the pilot for initial object detection.

Table 4.1: Flight Platforms Relevant to my Experiments

	0 g	40 g	% Reduction	60 g	% Reduction
Battery 1	21:04	18:43	11.16%	17:18	17.88%
Battery 2	22:54	19:35	14.48%	18:57	17.25%
Battery 3	17:57	14:23	19.87%	13:00	27.58%
Battery 4	20:00	16:43	16.42%	15:12	24.00%

Table 4.2: Flight Duration by Payload Weight

low end-to-end latency. However, since the Galaxy Watch is not able to connect over CBRS, all presented results are based on public cellular network infrastructure.

4.2.1 Flight Duration

Since my platform consists of mounting external components on an existing drone, a reduction in flight duration is expected. This reduction is important to quantify, as it directly impacts my platform's range and practicality. In Figure 4.2, I show the hover time of the Parrot Anafi with a 0 g, 40 g, and 60 g payload weight. The Galaxy Watch with harness is around a 40 g payload. It is clear that the watch payload reduces battery life by 10% to 20% on average.

4.2.2 Event-to-Detection Latency

The agility of my system is limited by the end-to-end latency of the processing pipeline (Figure 4.1). Events closer in time than this limit may not be resolvable. For example, a surveillance target with a jerky motion will be perceived as moving more smoothly. Large, but brief, deviations from the smoothed path may not be detected. The larger the end-to-end latency, the greater the need for predictive approaches in tracking fast-moving targets. This, in turn, leads to greater likelihood of errors due to mis-prediction.

The base end-to-end latency of the pipeline can by replicating the experiment described in Section 3.10.1 and Figure 3.10. The drone is kept stationary in a lab setting, with its camera pointing at a display attached to the cloudlet. Except for the fact that the drone is not flying, everything else (hardware, software, and network) is identical to Figure 4.1. The cloudlet-connected display shows the current time at millisecond granularity. An image of this times-

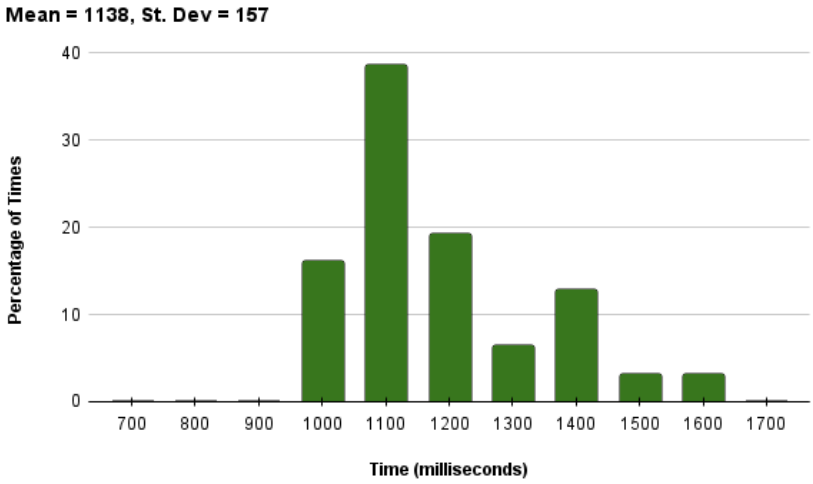


Figure 4.3: Distribution of Detection Latency (Galaxy Watch)

tamp is captured by the drone's camera, transmitted downstream, and recovered at the end of the pipeline. Its difference from current time at recovery gives the end-to-end latency. Figure 4.3 presents my results from 30 samples. The distribution is heavy-tailed, with a mean of 1138 ms and a standard deviation of 157 ms. The high mean and variability arise from jitter in LTE transmission, as well as from processing and scheduling delays on the drone, watch, and cloudlet.

4.2.3 Task-1: Object Detection While Hovering

Task Description

The accuracy of the computer vision pipeline complements its speed. Both are important for active vision. A simple test is the detection of a target on the ground when it moves into the camera's FOV. The problem is harder from higher altitude because objects are smaller and DNNs perform poorly on objects that are just a few pixels in size [69]. Figure 4.4 shows the DJI Robomaster S1 robot [40] used as the detection target in my experiments. It is roughly the size of a small dog, and can be remote-controlled over WiFi by a human driver. It can also be programmed to follow a predefined route, with speed variation in different route segments.

For Task-1, the robot is manually operated on a freeform path that overlaps the FOV of the drone that is hovering at fixed altitude. In postprocessing, I compare ground truth (GT) on each processed frame with the output of the processing pipeline. The object detection DNN was created via transfer learning from SSD-ResNet50 [95] pre-trained on the COCO dataset. The training set was created from drone-captured images of the target shown in Figure 4.4.

Results

I perform this experiment at altitudes of 5 m, 10 m, and 15 m. Accuracy is high at 5 m; a typical result is Figure 4.5(a), where the bounding box indicates detection followed by correct target

432 x 330 x 304 mm
(17 x 13 x 12 in)

Figure 4.4: COTS Target

classification at high confidence (0.98). The person at the top right and the distractor object at the top middle are correctly ignored. At 10 m, accuracy is slightly lower. An example of an erroneous result at 10 m is Figure 4.5(b), which shows a true positive (TP) (the target) at confidence 0.95 at bottom center, and a false positive (FP) (a person misclassified as the target at confidence 0.82) at center left. At 15 m, accuracy suffers significantly. An example of an error at 15 m is Figure 4.5(c). This shows an FP at center right (a person misclassified as the target at confidence 0.86), and also a false negative (FN) (missed target) at center left. Altitudes of 15 m and higher are clearly challenging for this combination of target size, optical system, and processing pipeline.

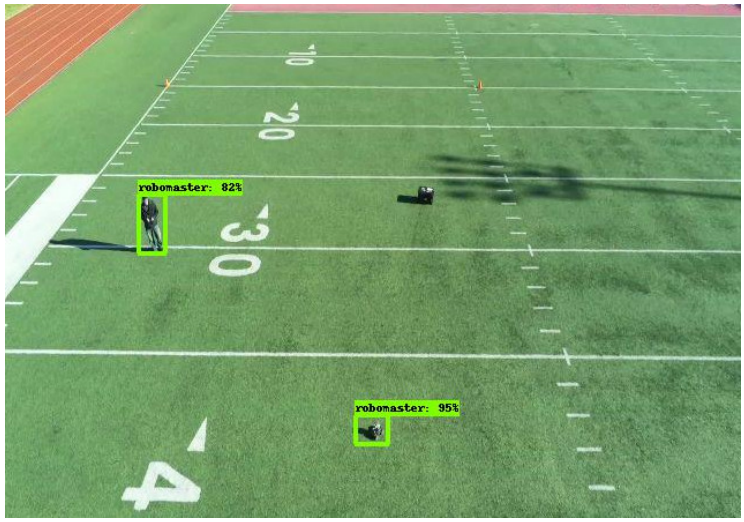
Table 4.3 (a) shows the confusion matrix for Task-1 at a confidence threshold of 0.7. The scoring of images used in this matrix requires explanation. Classic measures of precision and recall address scene classification, where an entire image is correctly or incorrectly classified. In contrast, my setting involves object detection. It is possible for a single image to have both a FP and a TP or FN. Figure 4.5(a) contains a single TP, and no errors. This is scored as a TP in the confusion matrix. Figure 4.5(b) contains both a TP and an FP; this is scored as an FP since errors trump correctness. When there are multiple errors, the worst error determines the score. Figure 4.5(c), for example, contains both an FP and an FN. I view FNs (hurting recall) as more serious errors than FPs (hurting precision), and therefore score the whole image as an FN. These rules preserve the invariant

$$GT_P + GT_N = TP + TN + FP + FN$$

where GT_P and GT_N refer to ground truth positives and negatives, and TN refers to true negatives (no target in image).

At 5 m, a total of 85 frames are processed. The column labeled “Ground Truth” shows that all 85 contain a target instance. The “Detected” columns show that 71 out of the 85 are correctly

(a) Altitude = 5m



(b) Altitude = 10m

(c) Altitude = 15m

Figure 4.5: Task-1 Images

Altitude	Ground Truth		Detected	
			Pos	Neg.
5 m	Pos.	85	71	13
	Neg.	0	1	0
10 m	Pos.	90	55	30
	Neg.	0	5	0
15 m	Pos.	85	24	48
	Neg.	0	13	0

Threshold = 0.7

(a) Detection Results

Altitude	Precision	Recall
5 m	0.99	0.85
10 m	0.92	0.65
15 m	0.65	0.33

(b) Precision and Recall

Table 4.3: Task-1 Results

detected and classified (TPs), but 13 are missed (FNs). At 10 m, all 90 processed frames contain an instance of the target, but only 55 of them are correctly detected (TPs). There are 30 FNs and 5 FPs. At 15 m, accuracy suffers considerably. Out of 85 total processed frames, all are GT-positive. However, only 24 of them are correctly detected (TPs). There are 48 FNs and 13 FPs. Table 4.3 (b) shows the precision and recall resulting from these detection results. These values are excellent at 5 m. Recall is noticeably degraded at 10 m. Both precision and recall suffer at 15 m. These results suggest the importance of active vision. Dropping to a lower altitude could confirm or refute the sighting of an object from higher altitude.

4.2.4 Task-2: Keeping Sight of a Moving Object

Task Description

Processing a frame in Task-1 does not lead to actuation of the drone. In contrast, Task-2 represents a simple form of active vision. After detecting a moving target, the drone yaws to keep the target visible in the frame. There is no forward or backward motion, only rotation to keep the object in the FOV. To find the target in frame, Target speed and motion predictability clearly influence this task. A fast-moving target that unpredictably and frequently changes its path is clearly hard to track. As discussed earlier (§ 4.2.2), the end-to-end latency of processing constrains tracking agility. With the help of the pilot (using the FreeFlight app), the Anafi Ai can also perform this task and thus I use it as a benchmark for my platform.

As shown in Figure 4.6, I set up a rectangular (approximately 20 m x 15 m) course marked by 4 cones. The drone is placed near the center of the rectangle and takes off to a fixed altitude of 2 meters. A 2 m tall by 0.5 m wide foam pillar is placed to occlude the target along the center of the back edge. For each run, the target moves along the right, back, and left edges in a u-shape two times. To test the ability of each platform to reacquire lost targets, I vary the time spent being occluded, starting with no pause, then pausing for 2 seconds behind the pillar, and finally a 5 second pause. Three runs of each delay were recorded. For the Anafi Ai, the pilot must draw a bounding box around the target in order to start tracking; my platform automatically acquires and re-acquires the target. Since my platform is constrained to below 1 fps, we compare results from the two platforms on frames that are one second apart. The Anafi Ai captures video at 30 fps, so I expect its responsiveness in this task to be significantly better.

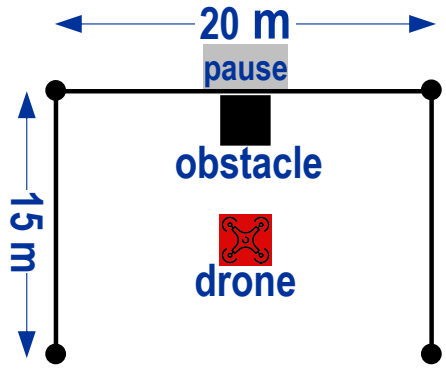


Figure 4.6: Target Occlusion

Algorithm

Edge offload to a powerful cloudlet enables tracking via DNN inferencing on every frame received. This "brute force" approach eliminates the need for predictive heuristics, such as those based on optical flow algorithms. Heuristics are often needed by on-board tracking implementations because the computational demand would otherwise be too high. The brute force approach makes tracking robust with respect to transient occlusions. Flow-based approaches, in contrast, are typically unable to reacquire the target after occlusion ends.

In my system, the cloudlet inferences each frame through an object detection DNN. The highest confidence bounding box is then chosen as the target, and its offset from the center of the frame is calculated in field-of-view degrees. The drone then actuates according to a PID-loop [9] based on the offset error. This simple approach provides a good baseline at low complexity.

Results

For successful tracking, both sensing and actuation are important. If the drone is sluggish in executing a yaw command, even perfect processing may not keep the target in the FOV at all times. Four outcomes are possible for each processed frame:

- the target is visible in the frame ("Success").
- the target is missing in this frame because of slow actuation, but present in the next ("Slow Actuation").
- the target is missing both in this frame and the next. This is scored as a tracking failure ("Fail").
- the target is occluded. This is not included in the frame total and is omitted from the results.

Table 4.4(a) and (b) compare the results for my platform and the Anafi Ai. At 0 s occlusion, my platform performs well, but run 3 experiences some tracking failures. My system's low framerate stream accounts for these failures. None of the frames received by the cloudlet included the target as it traversed the last corner of the pattern. In contrast, the Anafi Ai experiences no failures at 0 s occlusion. At 2 s occlusion, both my platform and the Anafi Ai perform very well.

Occlusion (s)	Run	Total Frames	Success (Target Present)	$\frac{\text{Present}}{\text{Total}}$	Slow Act-uation	Fail
0	1	63	60	91.2% (11.4%)	3	0
	2	62	62		0	0
	3	60	47		0	13
2	1	58	56	98.3% (1.7%)	2	0
	2	58	57		1	0
	3	64	64		0	0
5	1	59	53	90.3% (9.4%)	0	6
	2	53	53		0	0
	3	53	43		0	10

Figures in parentheses are standard deviations.

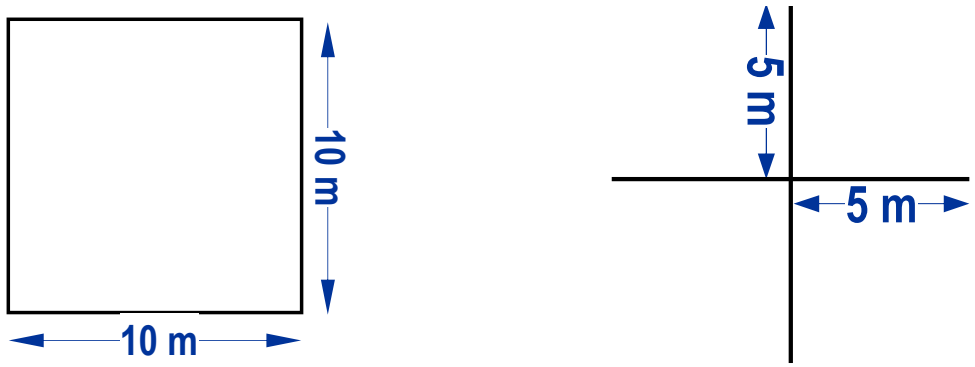
(a) SteelEagle

Occlusion (s)	Run	Total Frames	Success (Target Present)	$\frac{\text{Present}}{\text{Total}}$	Slow Act-uation	Fail
0	1	76	76	100% (0%)	0	0
	2	78	78		0	0
	3	80	80		0	0
2	1	79	79	100% (0%)	0	0
	2	80	80		0	0
	3	86	86		0	0
5	1	78	32	39.4% (1.7%)	0	46
	2	77	29		0	48
	3	81	32		0	49

Figures in parentheses are standard deviations.

(b) Anafi Ai

Table 4.4: Task-2 Results



(a) Square (b) Cross

Figure 4.7: Tracking Patterns

However, my platform experiences a few instances of slow actuation. Both platforms are able to reliably reacquire the target as it reappears from behind the obstacle. At 5 s occlusion, the limitations of the Anafi Ai are exposed. Such a long period of occlusion causes the Ai's optical flow tracking algorithm to become confused, often mistaking the pillar or background objects for the target. my platform's DNN tracking handles the increased occlusion well, with only a modest increase in the number of failures.

4.2.5 Task-3: Following at a Fixed Leash Distance

Task Description

Only limited actuation is needed to keep the target in the drone's FOV in Task-2. More substantial actuation is required for Task-3. After detecting a moving target, the drone moves to keep it at a preset *leash distance.* This compounds latency issues as the drone must now yaw and reposition itself correctly when the target maneuvers quickly. At high target speeds (over 2.5 m/s), this can prove difficult because even small actuation mistakes can result in total loss of visual contact. Although the Anafi Ai can statically track moving objects, it does not have a following feature which makes a direct comparison of its performance infeasible.

I programmed the target to move at a constant speed over flat ground in a specified pattern. I used speeds of 1.5 m/s (slow), 2.5 m/s (medium), and 3.5 m/s (fast). These speeds roughly correspond to a person walking, jogging slowly, and running. Two patterns were used: a square of side 10 m (Figure 4.7(a)), and a cross with four arms of 5 m each (Figure 4.7(b)). The square embodies abrupt change of trajectory after 10 m of straight line travel, while the change of trajectory in the cross occurs after only 5 m. As in Task-1, the drone is initially hovering at fixed altitude. I used altitudes of 5 m and 10 m in my experiments, but omitted 15 m since Section 4.2.3 indicates poor performance at this altitude.

Algorithm

The algorithm used for this experiment is a modified version of the one used in Section 4.2.4. Once the target is detected, the drone tracks it by centering the bounding box in the frame. At the

Speed (m/s)	Run	Total Frames	Success (Target Present)		Slow Actuation	Fail
				$\frac{\text{Present}}{\text{Total}}$		
1.5	1	82	80	95.2% (5%)	1	1
	2	85	76		0	9
	3	77	76		1	0
2.5	1	82	49	55% (8%)	2	31
	2	84	50		4	30
	3	83	38		0	45
3.5	1	87	46	62.7% (9.3%)	2	39
	2	84	60		1	23
	3	83	53		4	26

Figures in parentheses are standard deviations.
Abnormally high LTE latency was observed during the red highlighted run. This is always a possibility when using public cellular infrastructure. During high latency events, the performance of the system suffers considerably.

Table 4.5: Task-3 Results (Altitude = 5 m, Pattern = Square)

same time, it estimates its distance to the target, and moves such that it preserves a preset *leash distance*. This leash distance is chosen before the experiment to adequately frame the target at the specified altitude. If the drone loses sight of the target, it hovers until the object moves back into its FOV. No active search is made by the drone to reacquire a lost target.

Results

I use the same scoring rubric of "Success," "Slow Actuation," and "Fail" as for Task-2. At a target speed of 1.5 m/s, the results for all runs in Table 4.5 confirm successful tracking with only occasional failure. When speed increases to 2.5 m/s, and then to 3.5 m/s, the number of failures increases sharply. This is consistent with the computer vision processing pipeline following real-world scene changes too slowly, due to the very low frame rate (0.7 fps). I show later (§4.4) that increasing frame rate improves tracking. The effects of anomalously high LTE latency are visible in Run 3. This points to the challenge of using public cellular networks which can experience unpredictable changes in bandwidth and latency.

Table 4.6 presents Task-3 results when the pattern used is a cross rather than a square. Comparing the "Fail" columns of Tables 4.5 and 4.6, there is a noticeable decrease in failures at speeds of 2.5 m/s and 3.5 m/s when the pattern is a cross. These results are consistent with the cross being less demanding than the square for tracking.

At an altitude of 10 m, the drone's FOV is increased, but there is a significant drop in precision and recall as shown in Table 4.3 (b). This leads to an increase in the number of tracking failures relative to 5 m, regardless of the target pattern or speed. The effect is most apparent at the slowest speed: the results for 1.5 m/s in Table 4.7 show higher failures than the results at 1.5 m/s in Table 4.5. Similarly, the results for 1.5 m/s in Table 4.8 show higher failures than the results for 1.5 m/s in Table 4.6. These effects persist at higher speeds, but are less obvious. The improvement at 2.5 m/s between Tables 4.5 and 4.7 is due to the high-latency LTE anomaly

Speed (m/s)	Run	Total Frames	Success (Target Present)	$\frac{\text{Present}}{\text{Total}}$	Slow Actuation	Fail
1.5	1	83	83	100% (0%)	0	0
	2	88	88		0	0
	3	78	78		0	0
2.5	1	85	67	88.6% (8.5%)	0	18
	2	80	75		0	5
	3	88	82		1	5
3.5	1	82	82	89% (17%)	0	0
	2	85	59		1	25
	3	86	84		2	0

Figures in parentheses are standard deviations.

Table 4.6: Task-3 Results (Altitude = 5 m, Pattern = Cross)

Speed (m/s)	Run	Total Frames	Success (Target Present)	$\frac{\text{Present}}{\text{Total}}$	Slow Actuation	Fail
1.5	1	83	65	81.6% (12%)	1	17
	2	78	74		0	4
	3	88	63		3	19
2.5	1	82	52	62.3% (4%)	2	39
	2	83	48		1	34
	3	87	57		0	30
3.5	1	88	57	64.8% (10.5%)	1	30
	2	83	45		1	37
	3	89	67		3	19

Figures in parentheses are standard deviations.

Table 4.7: Task-3 Results (Altitude = 10 m, Pattern = Square)

Speed (m/s)	Run	Total Frames	Success (Target Present)	$\frac{\text{Present}}{\text{Total}}$	Slow Act-uation	Fail
1.5	1	80	75	87.3% (16.8%)	0	5
	2	85	85		0	0
	3	85	58		0	27
2.5	1	84	67	86.6% (11.6%)	0	17
	2	81	81		0	0
	3	85	68		0	17
3.5	1	86	62	82.9% (14.1%)	0	24
	2	86	85		1	0
	3	86	67		1	18

Figures in parentheses are standard deviations.

Table 4.8: Task-3 Results (Altitude = 10 m, Pattern = Cross)

mentioned earlier.

4.2.6 Task-4: Close Inspection

Task Description

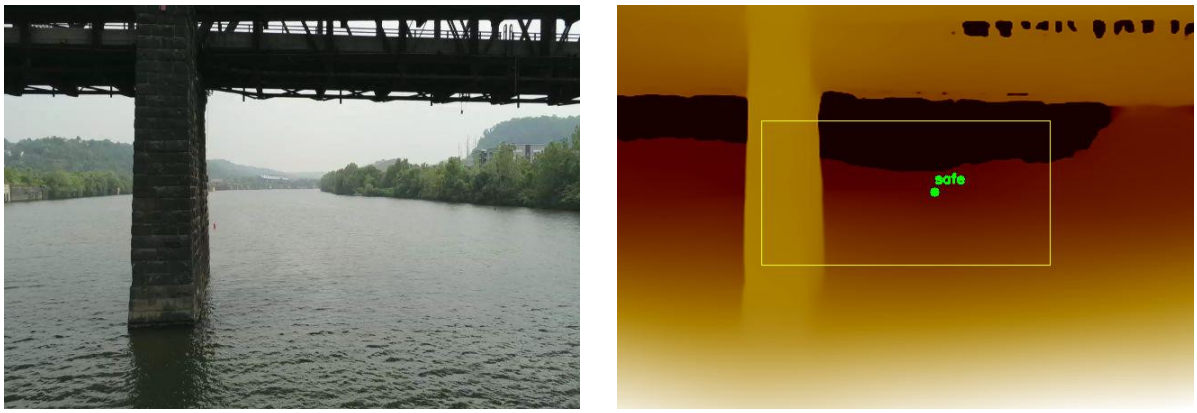
Task-4 corresponds to the classic active vision tactic of "taking a closer look" that was mentioned earlier (§1). It begins with the drone hovering at 15 m. As in Task-1, the target moves in a freeform path that is manually controlled over WiFi. When the target is detected in the FOV of the drone, confirmation at the lower altitude of 5 m is attempted. During the descent, the target is kept in the FOV using yaw and gimbal actuation; the drone's pitch and roll are not modified. If multiple targets are detected at 15 m, only confirmation of the highest-confidence detection is attempted.

Results

The results for Task-4 are shown in Table 4.9. The row labeled "Static 15 m" correspond to the 15 m results from Table 4.3 (b). Relative to that baseline, both precision and recall improve by nearly 30% by "taking a closer look." There is, of course, a cost in time because actuation involves physical motion of the drone to the lower altitude. Further, the increased FOV at higher altitude offers wider coverage. For these reasons, better accuracy at higher altitude will always be valuable. However, when such improvement is not possible due to limitations of the drone's optical system or processing pipeline, autonomously descending to a lower altitude for target confirmation can be effective.

Altitude	Precision	Recall
Close Inspection (15m-5m)	0.94	0.68
Static 15 m (§4.2.3)	0.65	0.33

Table 4.9: Task-4 Results



(a) Raw Input (b) Output of MiDaS

Figure 4.8: Bridge Obstacle Avoidance

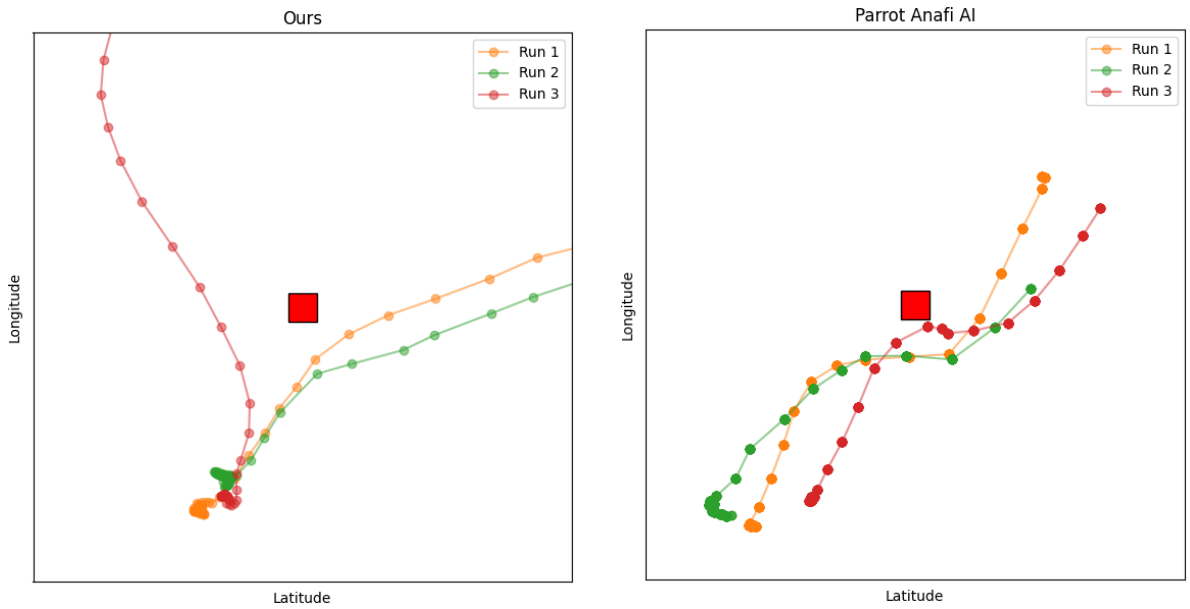
4.2.7 Task-5: Obstacle Avoidance

Task Description

Task-5 compares my platform's monocular obstacle avoidance, using my visual pipeline at 0.7 fps, with the stereoscopic obstacle avoidance of the Anafi Ai using on-board computing at 30 fps. I place the 2 m tall by 0.5 m wide foam pillar used in Task-2 (§4.2.4) directly in the drone's path. The drone is instructed to fly at 1 m/s at a fixed altitude of 1 m directly towards the obstacle. I capture a trace of the drone's flight path across 3 different runs.

Algorithm

Some drones utilize stereo cameras or LIDAR to detect and avoid obstacles. These give accurate depth data (i.e., distance to objects), allowing the drone to map out its environment and to calculate optimal collision-free trajectories. Since my drone is only equipped with a monocular camera, it cannot infer depth via simple geometric methods. I therefore use a DNN-based algorithm called MiDaS [104] to provide relative depth estimates. Using MiDaS on each frame received by the cloudlet, we construct the inverse relative depth map. Based on the rate of change of relative depth across frames, we identify obstacles in the flight path and actuate away from them. Figure 4.8(a) shows an input frame from one of my flights, as the drone approaches the obstacle course. Figure 4.8(b) shows the depth-encoded output of my algorithm on this frame. The drone actuates towards the green dot using a tuned PID-loop [9] to avoid the closest flags. Once past these flags, it re-computes a new safe objective towards which it can fly to avoid the second tier of flags. It repeats these steps until it is past all obstacles. This simple approach to



(a) SteelEagle (b) Anafi Ai

Figure 4.9: Task-5 Results

obstacle avoidance serves as a good experimental baseline.

Results

Figure 4.9(a) plots the flight path of each run for my platform, along with the position of the pillar. Figure 4.9(b) plots the flight paths of the Anafi Ai on the same task. Both platforms successfully avoid the obstacle in all cases, but do so using very different tactics. The low frame rate and high end-to-end latency of my pipeline forces my platform to be very conservative, and to give the obstacle a wide berth. Well past the obstacle, the drone has not yet returned to its original flight path. In contrast, the stereoscopic cameras, high frame rate and low end-to-end processing latency of the Anafi Ai together enable it to be much less conservative in obstacle avoidance. The flight paths cluster more tightly around the obstacle, and the drone soon returns to its original flight path.

4.3 Summary of Results

My evaluation began with a single top-level question. Could autonomous active vision be successfully implemented on a drone with severe constraints on both local compute and offloading? The maximum offloading throughput of 0.7 fps imposed by LTE thermal constraints on the watch (§3.10.5) is a severe bottleneck. The heavy-tailed distribution of end-to-end processing pipeline latency, with a mean of over 1 s (§4.2.2), is another bottleneck. Combined with limitations of onboard processing on the watch and the quality of the drone's optical system, these constraints pose a formidable barrier.

Frame Rate (fps)	Run	Total Frames	Success (Target Present)		Slow Actuation	Fail
				$\frac{\text{Present}}{\text{Total}}$		
3	1	361	283	83.7% (9.8%)	1	77
	2	360	342		1	17
	3	361	280		0	82
0.7	1	87	46	62.7% (9.3%)	2	39
	2	84	60		1	23
	3	83	53		4	26

Speed was 3.5 m/s. Figures in parentheses are standard deviations.

Table 4.10: Increased Frame Rate (Altitude = 5 m, Pattern = Square)

In spite of this barrier, the results presented (§4.2.3 to §4.2.7) show that active vision capabilities such as tracking a moving object and confirming object detection by dropping to lower altitude are feasible at credible target speeds. Even with the current implementation, one can perform useful tasks involving active vision. The range of feasible tasks can be broadened via hardware advancements in the drone and the watch, combined with more sophisticated algorithms that can then be supported. Progress on this front would also enable the superior resources of the cloudlet to be better utilized. Right now, the benefit of the cloudlet is being muted by the low sustainable offloading rate. Even so, thanks to the modular design of SteelEagle, it is still possible to port new AI algorithms with minimum effort into the pipeline which could perform better on this limited stream.

4.4 Benefit of Increased Frame Rate

The results for Task-3 (§4.2.5) make it clear that the drone's frame rate of 0.7 fps is a major limiting factor for tracking. When tracking high speed objects that perform erratic maneuvers, the drone often only has one or two detections to actuate upon. It often does not get feedback on actuation errors until the target has exited the frame entirely. Once that happens, tracking fails and the target is lost.

To quantify the benefits of a higher frame rate, I repeated the most difficult combination of speed and pattern in my Task-3 experiments: 3.5 m/s for a square pattern. Instead of using the watch, I used a ground-based laptop to play exactly the same role. The laptop connects to the drone over WiFi, and connects to the cloudlet via a commercial cellular LTE network. In every respect other than the fact that the laptop is not flying with the drone, the processing pipeline is unchanged from Figure 4.1. The tracking algorithm is also unchanged from that described for Task-3 (§ 4.2.5). Although the laptop experiences the same WiFi, RTSP video stream, and LTE conditions as the watch did, it does not suffer from the same processing or LTE thermal limitations. This enables offloading of video processing to the cloudlet a much higher frame rate. I chose a figure of 3 fps since I believe that this is realistically achievable with slightly better watch hardware. Even the Samsung Galaxy 4 watch can sustain 3 fps for over two minutes if it is pre-cooled with an ice pack. This is the full length of one run of Task-3 (§ 4.2.5).

Platform	Weight (g)	Throughput (fps)
Galaxy Watch 4	26	1
Future Offload Device	<50	3
Pixel 4a	143	5
Dell Latitude 5420 Laptop	2500+	6

Table 4.11: Throughput and Weight by Platform

	Inference (ms)	Effective Throughput (fps)
Object Detection	56.3 (5.9)	17.8
Obstacle Avoidance	124.3 (4.9)	8.1

Standard deviation in parentheses.

Table 4.12: Cloudlet Performance

Table 4.10 presents my results for an altitude of 5 m. For easy comparison, the 3.5 m/s results from Table 4.5 are reproduced below the new results. Increasing the frame rate from 0.7 fps to 3 fps greatly improves tracking — almost a 20% increase in the "Success" column. Even the worst run at 3 fps achieves 77.5% success which is 6.1% better than the best run at 0.7 fps at 71.4% success. This improvement is obtained without modifying my tracking algorithm.

Since frame rate is such an important factor in successful tracking, it is natural to speculate on what might be possible with future hardware advancements. For example, if a future offload device was no heavier than it is today but had the processing power of today's smartphones, how much better could it do tracking? To gain some insight into these speculative questions, I tested the processing pipeline of Figure 4.1 using different offload platforms. Experiments were performed with an identical setup as in Section 4.2.2.

Table 4.11 shows the throughput in fps and the weight of various computing platforms today. The first two rows are the current platform and a theoretical future offload device. The third row is a Pixel 4a smartphone, which is able to sustain 5 fps. At 143 g, it is too heavy for my drone to carry, but its 5x improvement in throughput is very attractive for robust tracking. The fourth row is a Dell Latitude 5420 laptop, which is able to sustain 6 fps; clearly, its 2.5 kg weight is far beyond the payload lift of any ultra-light drone. Since the laptop can decode video and transmit it over WiFi at 30 fps, the bottleneck shifts to the LTE link and the cloudlet's DNN inference time. As Table 4.12 shows, the cloudlet is able to inference at roughly 17.8 fps for object detection (Task-1 to Task-4) and 8.1 fps on obstacle avoidance (Task-5). Thus, an improved offload device on the drone could take better advantage of cloudlet resources. In the next chapter, I explore possible replacement devices for the watch in order to quantify this improvement.

Chapter 5

A Relay Device with Higher Throughput

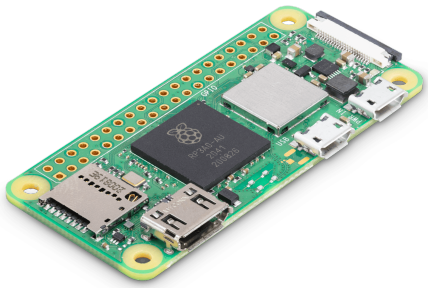
The watch prototype is severely limited in its ability to deliver a high throughput and low latency video stream to the edge, hindering the benefits of edge computing. In order to better take advantage of cloudlet resources, a new offload device is needed which can transmit the video stream without the thermal restrictions of the watch. As discussed in Section 3.6, there are no current Android products other than smart watches that satisfy the stringent weight requirements of the Parrot Anafi platform. The only path forward is to explore other segments of the mobile computing landscape.

In this chapter, I propose a new offload device which fixes some of the shortcomings of the watch prototype. I will show how it improves upon the watch in some ways but fails to match its capabilities in others. In Section 5.1-5.2, I explain the process of choosing a new device. I give a breakdown of its control model and how it differs from the watch. In Section 5.3-5.5, I use an agility analysis framework to compare the overall performance of each prototype.

5.1 Finding a Watch Replacement

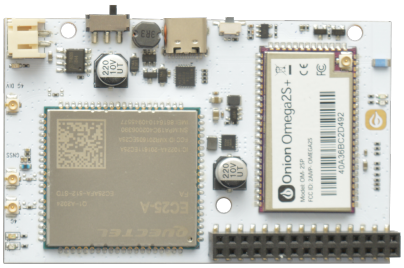
For any device to replace the watch, it must satisfy three main requirements of the SteelEagle pipeline: WiFi connectivity, cellular connectivity, and light weight. The first two requirements, WiFi and cellular connectivity, are relatively easy to find within the mobile device space. Meeting the weight requirement is much harder. The Parrot Anafi was able to fly with the watch payload, around 39 g, but was unable to fly with the Unihertz Jelly Pro payload, around 90 g with EMI shielding included. This gives an approximate weight range for a watch replacement.

Outside of smart watches, the only devices that lie within this weight range are single-board computers (SBCs). SBCs are small form factor computers which live on a board about the size of a credit card. They are widely used for internet of things (IoT) projects because of their small size, low power draw, and connectivity (usually WiFi but sometimes cellular as well). The most popular SBC at the time of writing this dissertation is the Raspberry Pi [105]. The Pi is an Ubuntu SBC available in several sizes, the lightest of which is the Pi Zero 2 W at 16 g (Figure 5.1). The Zero is equipped with a quad-core 1 GHz CPU and 512 MB of RAM. It has built-in WiFi connectivity but must be augmented with a dongle for cellular connectivity. Additionally, it does not come with an integrated battery, so it must be powered by an external



The Pi Zero 2 W is 16 g but lacks cellular connectivity and requires an external power source.

Figure 5.1: Raspberry Pi Zero 2 W [105]

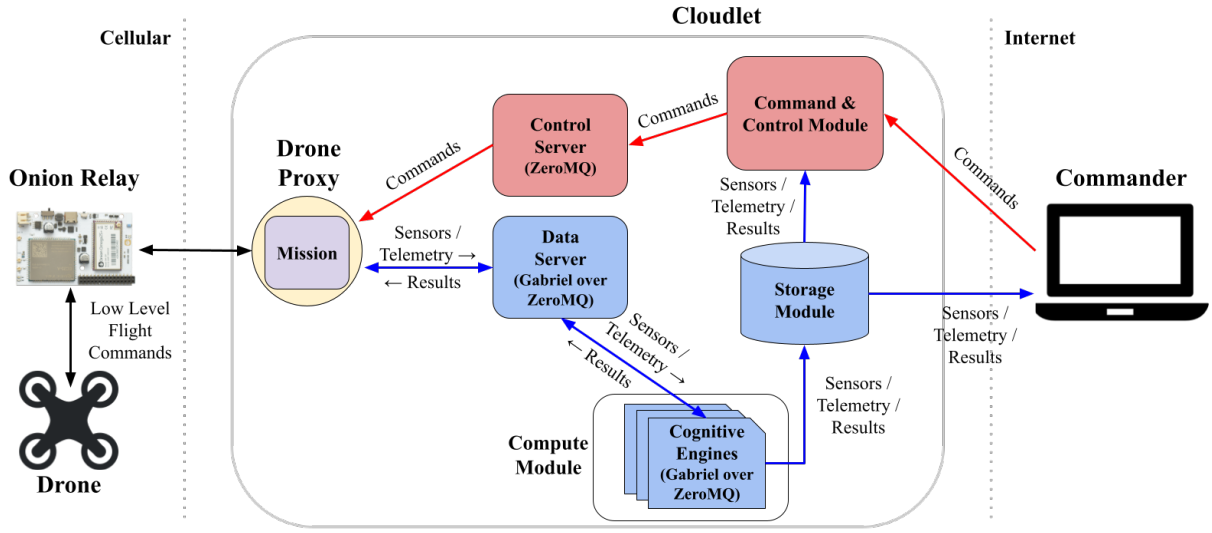


The Onion Omega is 20 g with embedded WiFi and 4G. It requires a 3.3 V, 0.5 A power source.

Figure 5.2: Onion Omega 2 LTE [94]

power pack that provides 1.2 A at 5.5 V. I determined that EMI shielding was not needed since the Pi did not create enough interference to hinder the Anafi's compass. With a power pack, 4G LTE dongle, and mount included, a Pi Zero payload for the Anafi would reach around 80 g. From experimentation, I found this was too heavy as the drone exhibited poor flight characteristics. A lighter alternative was needed.

After an extensive search, I settled on the 20 g Onion Omega 2 LTE (Figure 5.2) [94]. The Omega is another SBC with built-in WiFi and 4G LTE connectivity. It runs OpenWRT Linux and is equipped with a single-core 580 MHz CPU and 128 MB of RAM. This makes it much less powerful, on paper, than the Galaxy Watch 4 (1.18 GHz dual-core CPU, 1.5 GB RAM). It also does not have an integrated battery, and like the Pi, does not create enough interference to warrant EMI shielding. However, unlike the Pi, the Onion only requires 0.5 A at 3.3 V which allows it to be powered by a much smaller, lighter power pack. This means, after adding a mount, power pack, and antennas, the Onion payload is a much slimmer 53 g. Through experimentation, I found this weight to be viable for sustained flight without adverse flight effects.



The Onion acts as a pure relay, ferrying low-level command packets from the drone to the cloudlet. There, they are collected by a "drone proxy" which acts similarly to the watch in Figure 4.1.

Figure 5.3: SteelEagle System Architecture with Onion Omega

5.2 The Onion Omega Payload

The Onion's onboard compute resources are much worse than the Galaxy Watch 4. Its advantage is its thermals; it can run much, much hotter than the watch before overheating. This is mainly because it is not designed to be worn, and thus does not need to worry about burning its wearer. As a result, the Onion can transmit over its 4G link at a high rate without restrictions.

The Onion cannot run a full-featured local application like the watch can, but its ability to use cellular connectivity without constraints suggests a different control paradigm. Rather than controlling the drone locally from the watch and only sending frames to the cloudlet to run computer vision algorithms, the Onion can act as a pure relay, ferrying telemetry and video packets upstream while delivering command packets downstream. In this system, the watch application would migrate to the cloudlet and would communicate with the drone over the Onion (Figure 5.3). This is known as the "thin client" computational paradigm.

This approach has some advantages. First, it theoretically allows the cloudlet to receive the raw video stream from the drone at full frame rate and relatively low latency (compared to the watch). Since the Onion does no video decoding, the latency cost paid is simply the transmission delay from the drone to the Onion over WiFi and from the Onion to the cloudlet over 4G. Second, the mission application lives on the cloudlet and so it can get computer vision results with negligible latency.

On the other hand, this setup has significant drawbacks compared to the watch. The Onion payload is not as robust as the watch payload. It has exposed electronics and requires the use of a flammable LiPo power pack to use, whereas the watch is a consumer device that is easy to charge and is fully weatherproofed. Furthermore, since the Onion acts as a pure relay, it is at the mercy of its cellular connection. It cannot function separately from the cloudlet, while the watch can

still pilot the drone without computer vision assistance. It also cannot adapt the video stream to reduced bandwidth by throttling, and must transmit the whole stream to produce usable frames on the backend.

Despite these problems, the Onion presents an opportunity to establish an upper bound for SteelEagle's performance. Where are the current bottlenecks in the system, and how can they be fixed? What is the best latency and throughput the system can achieve? How *agile* is the system at reacting to new stimuli?

5.3 A Framework for Understanding Agility: the OODA Loop

To answer these questions, I introduce the concept of a drone OODA loop. Originally conceived in the 1950s to characterize man-machine symbiosis in combat aircraft, this concept has since been extended to many other domains [18, 21, 74]. The components of an OODA loop ("Observe", "Orient," "Decide," and "Act") define the stages of any reactive pipeline that involves a human in the loop. I extend this concept from its cyber-human origins to the cyber-physical context of an autonomous drone. Viewing an AI pipeline through the lens of an OODA loop better explains its performance attributes. It can tease apart latency and throughput limitations at fine granularity, thus enabling bottlenecks to be identified and optimized.

An OODA loop's attributes directly limit drone agility. As discussed in Section 4.2.2, throughput limitations may cause closely-spaced real-world events to not be resolvable as separate events. High end-to-end latency may cause slow reactions and punish mis-predictions. SteelEagle's OODA loop includes: (a) on-drone sensing, (b) on-drone pre-processing, (c) transmission to cloudlet, (d) processing on a (possibly multi-tenant) cloudlet, (e) transmission to drone, (f) on-drone post-processing, and (g) drone actuation. I will profile and optimize these components through experimentation.

5.4 Profiling & Optimizing the SteelEagle OODA Loop

For the following experiments, I replicate the setup in Section 4.2.2. The drone is kept stationary in a lab setting and is connected to the cloudlet via the Onion over public cellular. On the backend, I use an identical spec cloudlet as in Section 4.2: 36 CPU cores, 128 GB of RAM, and an NVIDIA GeForce GTX 1080 Ti GPU.

5.4.1 Mapping the OODA Loop

Figure 5.4 maps the end-to-end pipeline to OODA loop components. Components (a), (b) and (c) together map to the "Observe" phase; component (d) maps to its "Orient" and "Decide" phases; and, components (e), (f) and (g) together map to the "Act" phase. Due to closed-source restrictions of the COTS pipeline, some OODA loop components have to be aggregated for purposes of measurement, as shown by Figure 5.5. Total end-to-end latency is given by the sum of these components; total throughput is that of its bottleneck.

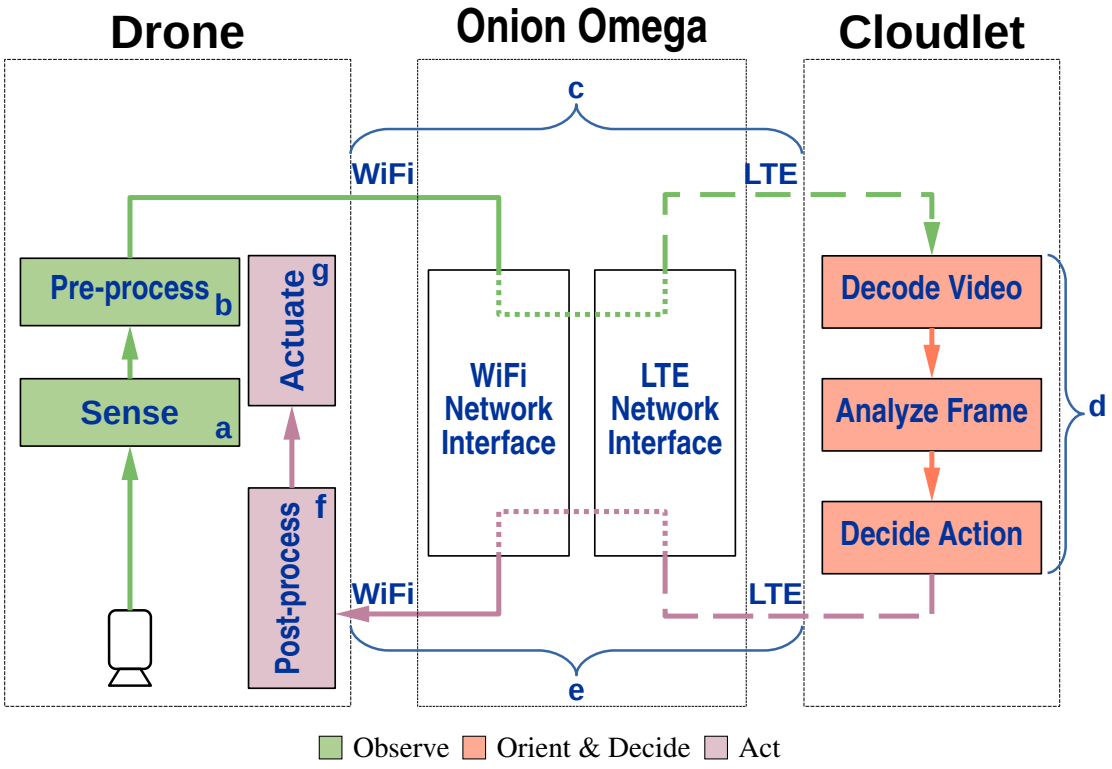
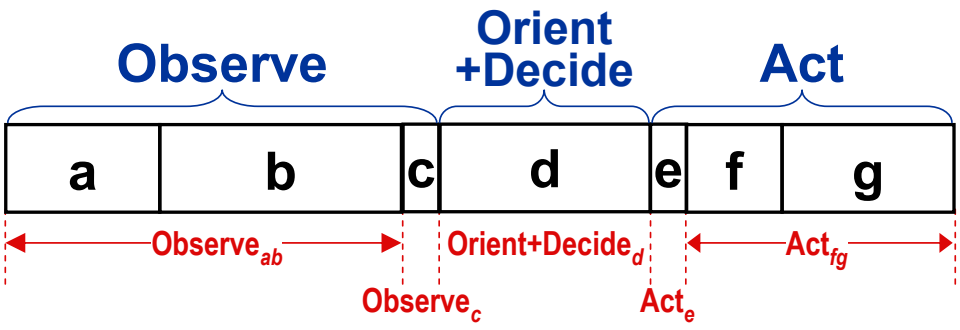


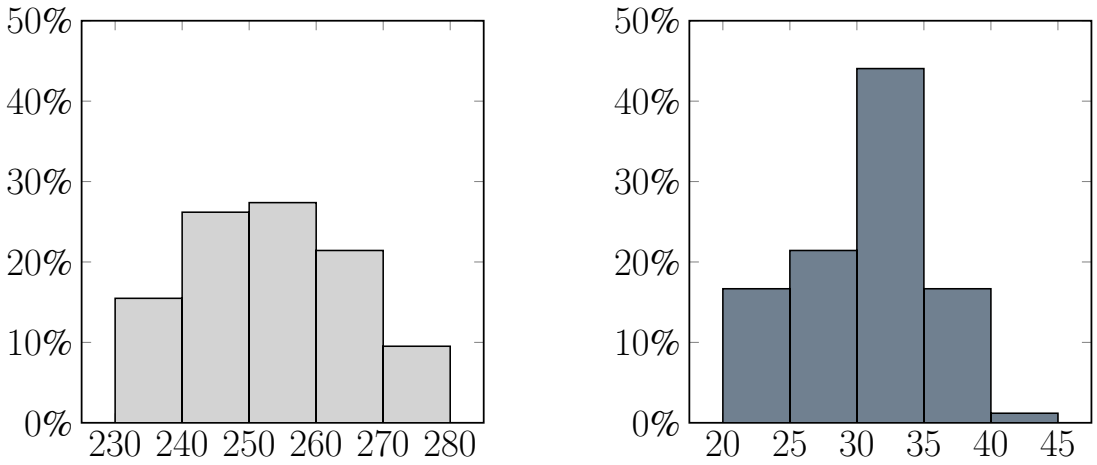
Figure 5.4: Detailed View of the OODA Loop



Only items in red above can be measured.

a = on-drone sensing
b = on-drone pre-processing
c = transmission to cloudlet
d = processing on cloudlet
e = transmission to drone
f = on-drone post-processing
g = drone actuation

Figure 5.5: Measurable Components of Our OODA Loop



Mean: 253 ±12 ms p99: 277 ms
(a) Latency (ms)

Mean: 31 ±5 fps p1: 22 fps
(b) Throughput Jitter (fps)

Figure 5.6: Observe$_{ab}$ Measurements

The earliest point in the pipeline where software instrumentation can be inserted is between the WiFi and 4G interfaces on the Onion Omega. The WiFi part of this pipeline is thus attributed to Observe$_{ab}$ rather than to Observe$_c$. Similarly, on the return path, the WiFi part is attributed to Act$_{fg}$ rather than to Act$_e$. Only 4G transmission is attributed to Observe$_c$ and Act$_e$. The resulting error is likely to be very small since WiFi is much more performant than 4G. I present detailed measurements in §5.4.2 to §5.4.6.

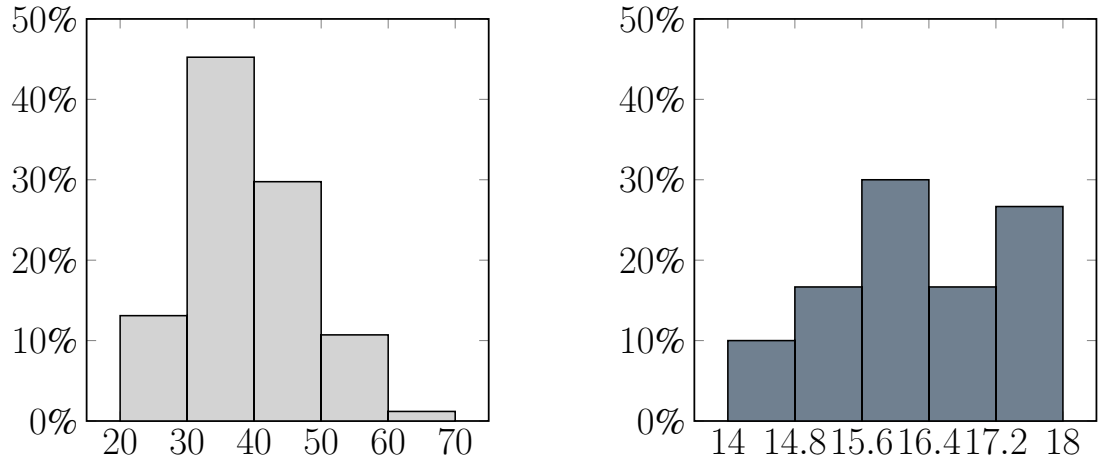
5.4.2 Observe$_{ab}$

As discussed in Section 3.8, the drone's video stream generator is a black box that cannot be modified. This makes attribution of latency costs difficult. It is not possible to insert instrumentation to separate (a) and (b); they merge into an indivisible component.

Figure 5.6 presents my measurements. The latency distribution has a mean of 253 ms, with a standard deviation of 12 ms and a p99 of 277 ms. Throughput, with network jitter factored in, has a mean of 31 fps, with a standard deviation of 5 fps and a p1 of 22 fps. Due to pipelining, throughput can be higher than the reciprocal of latency. The short WiFi Observe$_{ab}$ segment is partly responsible for this observed variation.

5.4.3 Observe$_c$

The wireless network path from drone to cloudlet consists of a very short WiFi segment, transit through the Onion router carried as payload, and then a longer 4G LTE segment to the cloudlet. Figure 5.7 presents the latency and throughput distributions of Observe$_c$. Its latency has a mean of 39 ms, with a standard deviation of 8 ms and a p99 of 60 ms. Instantaneous throughput has a mean of 16.3 Mbps, with a standard deviation of 1 Mbps and a p1 of 14.4 Mbps. Since 720p video at 30 fps only demands an average bit rate of about 6.5 Mbps [1], Observe$_c$ is definitely not the bottleneck.



Mean: 39 ±8 ms p99: 60 ms

(a) Latency (ms)

Mean: 16.3±1 Mbps p1: 14.4Mbps

(b) Throughput (Mbps)

Figure 5.7: Observe$_c$ Measurements

5.4.4 Orient+Decide$_d$

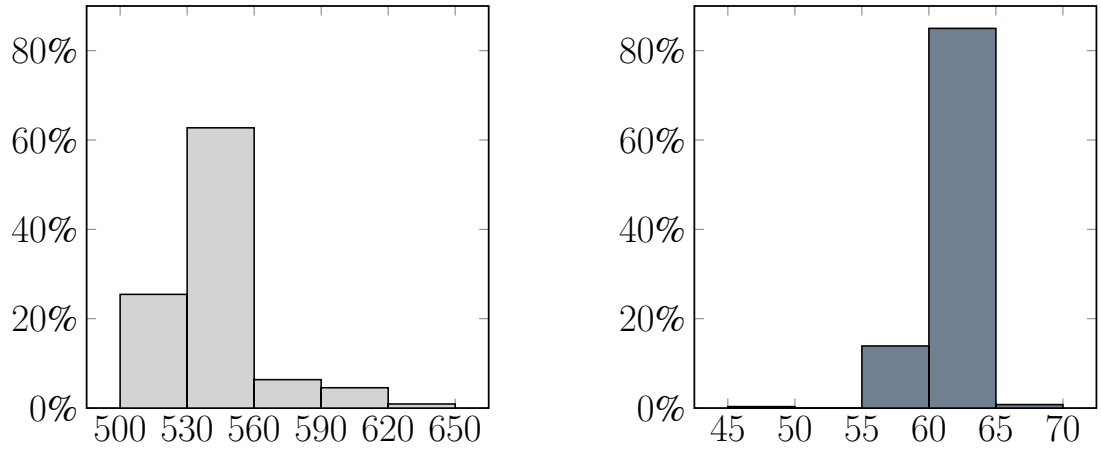
Processing on the cloudlet involves three stages:

- Stage-1: Decoding the UDP packet stream to produce individual frames from H.264 video.
- Stage-2: Application-specific processing of each frame to interpret its contents. For example, this could involve DNN inferencing with a pre-trained model to detect objects of interest currently visible to the drone.
- Stage-3: Application-specific logic to determine salient changes revealed by Stage-2. This early part of Stage-3, together with Stage-1 and Stage-2, constitute the "Orient" part of the OODA loop. The rest of Stage-3 is the "Decide" part. Drone actuation (if any) is determined, and the command to perform this actuation is generated. For example, Stage-2 may show that an object being tracked has moved and the gimbal has to be adjusted to re-center the object in the camera's field of view (FOV).

Stage-2 can be viewed as perception and Stage-3 as cognition. The latency and throughput of Stage-1 constrain the performance of Orient+Decide$_d$ since decoding has to be performed even if Stage-2 and Stage-3 take a negligible amount of time.

Figure 5.8 presents my measurements of Stage-1. The magnitude of the latency, with a mean of 541 m, is surprising. The cloudlet has 36 CPU cores and a powerful GPU whcih should be ample for efficient software decoding of an H.264 video stream, as confirmed by the mean throughput of 62 fps shown in Figure 5.8(b). The high latency observed has no obvious explanation, but it has a large negative impact on the OODA pipeline.

To verify the latency impact of cloudlet hardware on Stage-1, I additionally measured its performance on two different AWS VM configurations. One configuration, "AWS Small," is a `g4dn.xlarge` EC2 VM with 4 vCPUs, 16 GiB of memory and an NVIDIA T4 GPU. The other configuration, "AWS Big," is a `g4dn.16xlarge` EC2 VM with 64 vCPUs, 256 GiB main memory, and an NVIDIA T4 GPU. Figure 5.1 presents the results. Astonishingly, the best



Mean: 541 ±22 ms p99: 620 ms
(a) Latency (ms)

Mean: 62 ±1.5 fps p1: 59 fps
(b) Instant. Throughput (fps)

Figure 5.8: Original FFmpeg-based Stage-1 Performance

Hardware	**vCPUs**	**Memory** (GB)	**Mean** (ms)	**p99** (ms)
Cloudlet	72	128	541 ±22	620
AWS Big	64	256	532 ±13	561
AWS Small	4	16	140 ±16	194

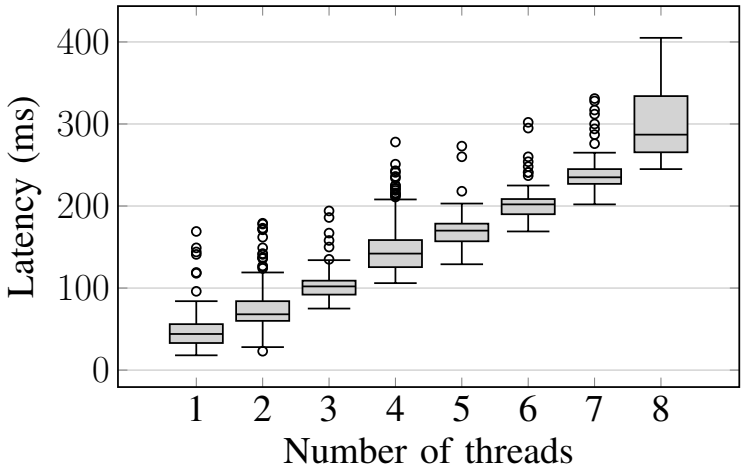
Table 5.1: Stage-1 Latency versus Cloudlet Hardware

performance is obtained on the configuration with the fewest cores (AWS Small). Drilling down deeper into this anomaly, the widely-used open source `ffmpeg` video decoding software is the culprit. Figure 5.9 shows how `ffmpeg` per-frame decoding time varies as a function of number of threads used. As the figure shows, `ffmpeg` demonstrates *negative latency scale-out attributes* — adding more threads makes latency worse. I posit that this suboptimal behavior has not been reported before because `ffmpeg` has not been widely used in latency-critical settings.

By switching to different decoding software [100] tailored for the drone stream, I was able to reduce the latency from a mean of 541 ms in Figure 5.8(a) to a mean of 32 ms in Figure 5.10(a). This has been achieved with a mean throughput of 37 fps (Figure 5.10(b)), which is well above the demand of 31 fps from Observe$_{ab}$. Assuming negligible processing in Stage-2 and Stage-3, Figure 5.10 shows the best-case latency and throughput of Orient+Decide$_{d}$.

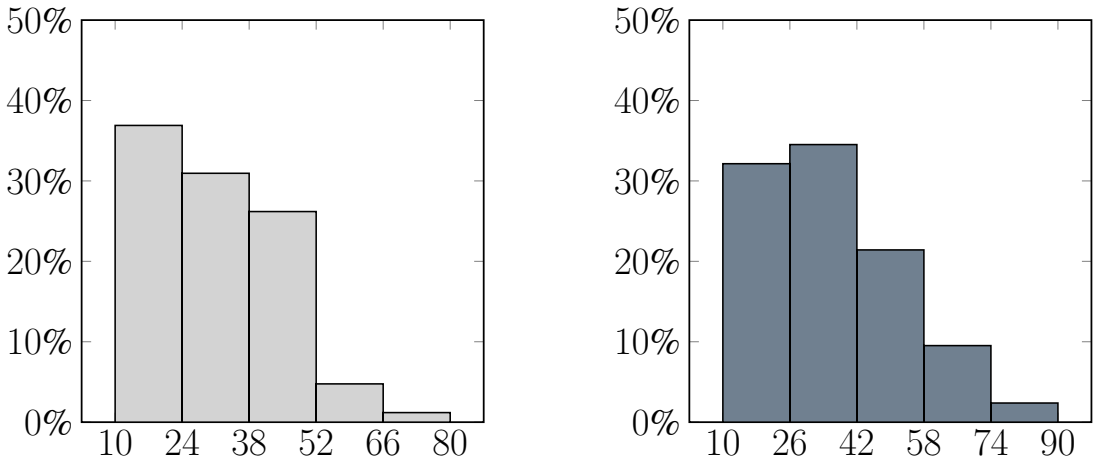
5.4.5 Act$_{e}$

Figure 5.11 presents my measurements of the wireless network path from cloudlet to drone. The latency has a mean of 30 ms, with a standard deviation of 4 ms and a p99 of 49 ms. The throughput has a mean of 28 Mbps, with a standard deviation of 3.7 Mbps and a p1 of 19 Mbps. Since no video is transmitted back to the drone, Act$_{e}$ is not a bottleneck.



Each box extends from the first quartile (Q_1) to the third quartile (Q_3), with a line at the median. Whiskers extend from the box to the farthest data point lying within 1.5x the inter-quartile range ($IQR = Q_3 - Q_1$) from the box. Circles represent outliers.

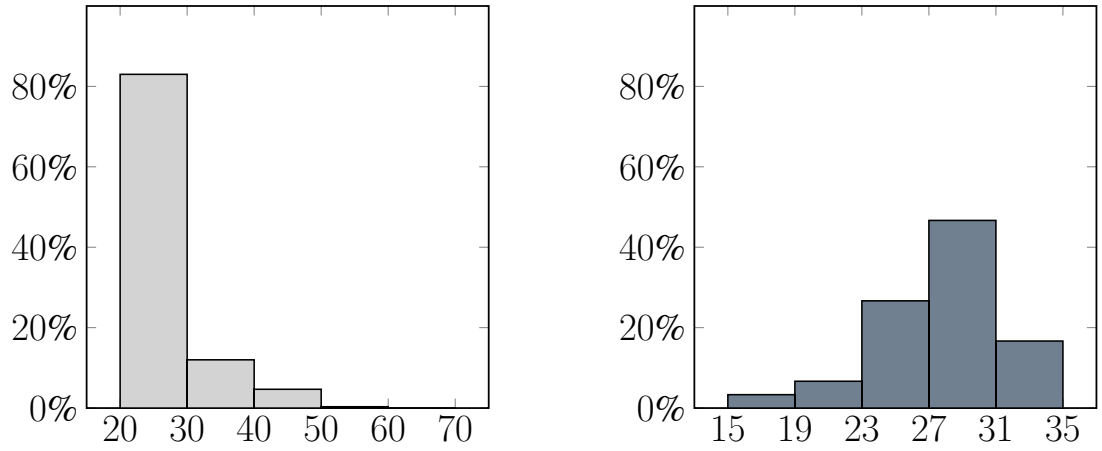
Figure 5.9: Negative Scale-out of FFmpeg Latency



(a) Latency (ms)

(b) Instant. Throughput (fps)

Figure 5.10: Improved Performance With FFmpeg Alternative



Mean: 30 ±4 ms p99: 49 ms
(a) Latency (ms)

Mean: 28±3.7 Mbps p1: 19 Mbps
(b) Throughput (Mbps)

Figure 5.11: Act_e Measurements

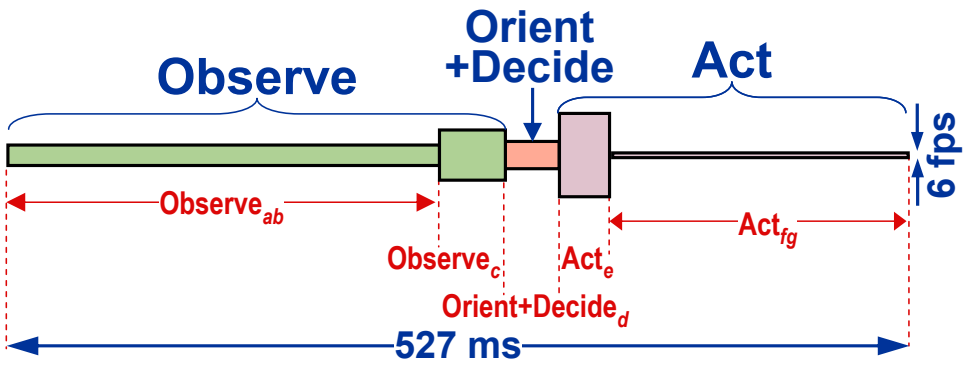
Run	Latency (ms)
1	188
2	170
3	162
4	189
5	155
Mean	173 ±15

Table 5.2: Act_{fg} Latency

5.4.6 Act_{fg}

Black box hardware and software on the drone seamlessly integrate components (f) and (g). All that is externally visible is a set of commands that are accessible via the drone's SDK. The processing of a command and initiation of actuation are integrated into Act_{fg} in Figure 5.5. In this context, latency corresponds to the time difference between the receipt of an actuation command by the drone, and the start of actuation. To measure this difference, I position the stationary drone in front of a display connected to the cloudlet, similar to the setup in Section 4.2.2. The display shows the current timestamp in milliseconds. I send a command to the drone to move its camera gimbal, while recording the display and gimbal using a slow-motion video camera. In post-processing, I manually identify the timestamp of the command and that of the first video frame showing gimbal movement. Act_{fg} latency is the difference between these two timestamps. The slow-motion camera operates at 240 fps, resulting in a frame interval of ˜4 ms. The measurement has an error margin of ˜5 frames, translating to experimental error of ˜20 ms.

Figure 5.2 presents my measurements. The latency distribution has a mean of 173 ms, with a standard deviation of 15 ms. Electromechanical actuation is far slower than processing or network transmission. The experiments do not involve back-to-back actuations without intervening sensing or processing. Hence, throughput is best interpreted as the reciprocal of latency.



This figure uses the same notation as Figure 5.5, and color coding as Figure 5.4. Width is scaled to represent latency, and height to represent throughput. Orient+Decide$_d$ here assumes negligible application-specific processing. For the electromechanical actuation represented by component Act$_{fg}$, the reciprocal of its latency is used as its throughput.

Figure 5.12: OODA Loop Latency & Throughput

5.5 The Full OODA Loop

Using the same notation as Figure 5.5, a visual summary of the measurements reported in §5.4.2 to §5.4.6 is shown in Figure 5.12. This captures the best-case OODA loop, where no time is spent in Stage-2 and Stage-3 of Orient+Decide$_d$. In practice, it is those stages that perform the processing for drone autonomy such as object detection, object tracking, Kalman filtering, and route planning. They also do the processing to generate the commands for drone actuation such as gimbal movement, flight path alteration, or altitude change. The height and width of the resulting Orient+Decide$_d$ component in Figure 5.12 would need to be scaled to include such application-specific processing. In some cases, that component may dominate the entire OODA loop.

In a typical application, many iterations of the OODA loop may involve no actuation, thus eliminating Act$_e$ and Act$_{fg}$. For example, consider a target that is moving in a straight line at constant speed. Successive OODA loops of a drone that is following that target only need to confirm that it remains centered in the FOV. Only abrupt change of motion by the target will stress the OODA loop. Fast reaction is then needed to discover that the target is off-center, and to actuate the gimbal or drone to re-center it before it is lost from the FOV. Figure 5.12 shows that the latency and throughput of Observe$_{ab}$ are the limiting constraint in uneventful settings. It is thus the inherent attributes of the drone, rather than network bandwidth or the cloudlet processing power, that limit this system.

Chapter 6

Benchmarking the SteelEagle Pipeline

While the OODA loop provides a broad understanding of drone reaction time, it does little to measure agility in real-world flight operations. In practice, it is performance on the latter that actually matters. Unfortunately, measuring this performance is difficult because there is no existing metric that defines the units in which to express an answer.

Agility is a complex emergent cyber-physical property that depends both on cyber properties such as latency, throughput, and accuracy of the OODA loop, as well as physical properties such as the drone's size, weight, thrust, lift, drag and moment of inertia. Under benign conditions, a non-agile drone may do as well as an agile one. Only under adversarial conditions does agility become valuable. The cost to achieve this agility may include increased size, weight, and bandwidth/latency demand arising from the need to be faster and more accurate in sensing and actuation. The only way to quantify this complex property is to stress a drone on a precisely-defined task in a reproducible environment, and to use task-level metrics as surrogates for agility. This leads directly to the creation of benchmarks for evaluating autonomous drone agility.

In this chapter, I define two agility benchmarks for measuring drone agility. The first benchmark embodies tracking of an object that moves in an unpredictable manner, with many abrupt changes. The adversarial aspect of this benchmark lies in the existence of an active mobile agent that randomly changes its trajectory. The second benchmark embodies obstacle avoidance in tight spaces. The adversarial aspect of this benchmark lies in the close proximity of obstacles, the need to sense them in real time (e.g., to account for wind effects), and occlusion that prevents full foreknowledge of the optimal flight path. Both benchmarks are parameterized, thereby enabling many levels of difficulty within a common benchmark framework. In Section 6.1, I discuss previous work related to real drone flight benchmarking in reactive scenarios, and how my work improves upon it. In Section 6.4-6.6, I outline my benchmarks for object tracking and obstacle avoidance respectively and the results from my experiments using them.

6.1 Prior Work on Drone Benchmarks

There have been many efforts in the computer vision and machine learning community to create benchmarks for comparing drone performance on specific tasks. These focus exclusively on the accuracy of algorithms such as drone-based object tracking and face recognition, ignoring system

attributes such as agility and end-to-end processing latency. Du et al [44], Li et al [80], Kalra et al [76], and Zhao et al [134] are examples of this genre.

Many drone benchmarks do measure agility but involve only simulated tests. MAVBench [20] is one popular example. It consists of a closed-loop simulator and end-to-end application benchmark suite of five workloads pertaining to scanning, aerial photography, package delivery, 3D Mapping, and Search and Rescue. These workloads lack customization options, and often represent a specific simulated scenario which can only give limited perspective on real performance. Another simulation-based benchmark, FlightBench [129], has agility workloads which provide several levels of difficulty. However, this difficulty is determined arbitrarily by the authors and the obstacle courses are too complex to practically replicate outside of the simulator. Additionally, simulations typically do not fully capture real flight performance, where sensors can experience noise which can influence actuation.

There are live flight drone benchmarks that measure agility, but they are not as common. One example is the disturbance benchmark proposed by Wu et al [126] which uses an indoor course along with a fan to emulate obstacle avoidance in windy conditions. The course is fixed and does not provide guidance for replicating the described experiments. For this reason, while it is useful for evaluating the paper's proposed trajectory planner, it is not as useful for measuring the performance differences between different avoidance methods.

Koubaa et al [77] describe an experimental study that compares on-board drone processing versus offloading to the cloud. The metrics of interest in their work are energy cost, bandwidth demand, and timeliness of results. The last of these metrics is closest to our focus on the agility of drones. However, the experiments described do not include drone actuation in response to real-time observations. They are purely open loop experiments, with timeliness to cloud users being the metric of interest. Further, this work only provides micro-benchmarks to evaluate these metrics. There are no end-to-end benchmarks that include the full OODA pipeline of sensing, processing and drone actuation.

Beyond these experimental efforts is a vast body of published literature on analytical or simulation-based evaluations of algorithms for specific drone tasks. Examples include the work of Chen et al [27], Hayat et al [64], Wang et al [122], and Wu et al [125]. These studies abstract away the physical drone, relying instead on hypothetical cost models of processing and communication. AdaDrone [26] is a slightly more realistic approach that leverages a drone simulator. None of these efforts use real drones, with their intrinsic limitations of weight and maneuverability. In contrast to these prior works, I focus on providing *parameterized* and *reproducible* benchmarking of drones in actual flight.

6.2 Visual Object Tracking Benchmark

In visual object tracking, a drone follows a moving target and tries to keep it centered in its FOV. Many surveillance-related instances of this task, where the target may be adversarial and actively try to escape tracking, occur in law enforcement and military settings. The task is also relevant to wildlife conservation research, where an animal of an endangered species is identified and followed in the wild. It is also used in filmmaking to capture evolving scenes. In such use cases, using an autonomous drone for tracking could reduce attention demand on mission personnel.

6.2.1 Benchmark Requirements

A good benchmark for a task must capture the essential characteristics of successful completion. The size and visual appearance of a target plays an important role in tracking success. An object that is just a few pixels in size from the altitude of the drone will be inherently difficult to detect [69]. Poor contrast with the background, as happens when camouflaged, also contributes to poor detection. Objects that are hard to detect are also hard to track, since actuating to re-center the target in each frame is key to success. The object being tracked and the background on which it moves both need to be specified. Only when these factors and drone optics are held constant will OODA loop performance come to the fore in determining tracking performance.

The benchmark should also be parameterized, so that it is easy to vary the difficulty of the benchmark. The benchmark should only use standardized, off-the-shelf components that can be easily purchased or fabricated. There should be no ambiguity in the experimental setup or interpretation of results, thereby simplifying independent attempts to reproduce published experimental results.

6.2.2 Benchmark Description

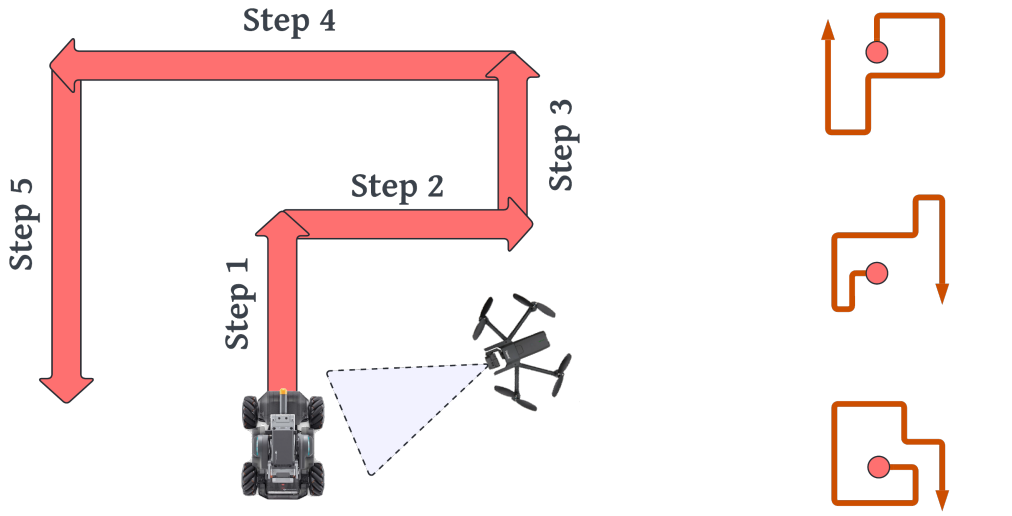
The object followed in my tracking benchmark is a DJI Robomaster S1 robot [40] as in Section 4.2.5. Tracking is done on a level, green background such as a football field. For this combination of target and background, DNN-based object detection from an altitude of 10 m is successful at a confidence level of 0.9 or higher on frames from my drone's video camera.

My benchmark is a random walk with turns in a randomly-chosen cardinal direction at each step. Figure 6.1(a) shows one example with 5 steps, and Figure 6.1(b) shows other examples with more steps. The benchmark has three parameters: the number of steps; the mean length of each step; and the target speed of 1.5 m/s, 2.5 m/s, or 3.5 m/s. The benchmark could be made more complex by making the turns to be at any angle rather than just cardinal directions, and by making step size and target speed non-uniform. All my experiments were conducted across the full range of speeds, using a stepcount of 35 steps and stepsize set to 5 m.

To execute the benchmark, the target is placed in a large open outdoor area. The drone is manually piloted to the desired altitude, and its FOV is adjusted to center the target. Once the drone has locked onto its target, the target is instructed to start its pre-programmed random walk and a timekeeper starts a stopwatch. The experiment continues until one minute has elapsed or the drone loses the target from its FOV. The termination time and the black box footage of the flight are logged for post-flight scoring (§6.2.3).

6.2.3 Benchmark Scoring

In post-processing after a flight, I score the recorded video footage using an automated process. Figures 6.2 to 6.4 show the scoring calculation, using Figure 6.5 as an example. On each frame, a DNN is first used to create a bounding box around the target. With the center of the frame as the origin, the relative distance of the target from the origin is obtained. Using the notation shown in Figure 6.2, the pixel offset vector, $\vec{O}$, gives the L2 distance of the target's centroid from the origin. This is scaled to the diagonal length of the bounding box, $\vec{D}$, to give the centering



(a) Detail for 5-Step Example

(b) Other Examples

Figure 6.1: Parameterized Random Walk

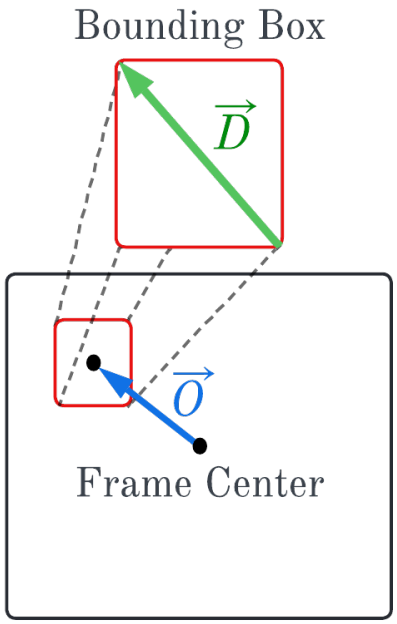


Figure 6.2: $\vec{O}$ & $\vec{D}$

$$c_i = \left\|\vec{O}_i\right\| / \left\|\vec{D}_i\right\| \tag{6.1}$$

$$s_i = 1.1^{-c_i} \tag{6.2}$$

$$s_{\text{avg}} = \frac{\Sigma_i^n s_i}{n} \tag{6.3}$$

Figure 6.3: Calculating Score

$$\left\|\vec{O}\right\| = 0.11,\ \left\|\vec{D}\right\| = 0.03,\ c = 3.67$$

$$s_{10} = 1.1^{-c} = 0.70$$
$$s_{20} = 1.2^{-c} = 0.51$$
$$s_{30} = 1.3^{-c} = 0.38$$

Figure 6.4: Scoring Figure 6.5

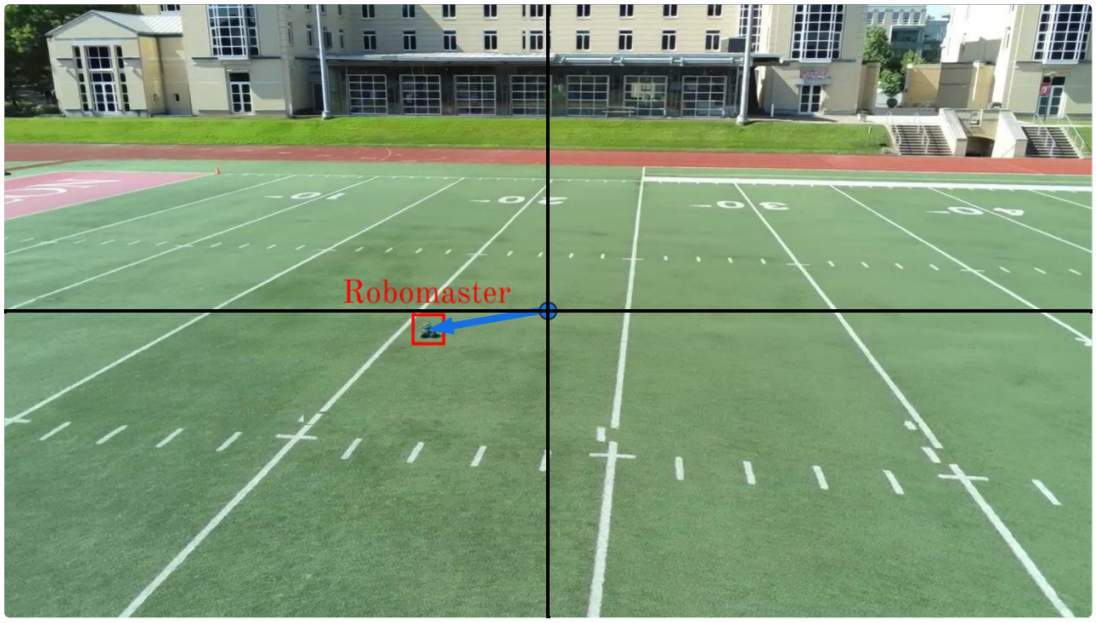


Figure 6.5: Example Frame for Scoring

Model	**Latency** (ms)	**Throughput** (fps)	**mAP**
YOLOv5s	28	25	56.8
YOLOv5m	37	20	64.1
YOLOv5l	42	20	67.3

The inference and throughput were obtained on the cloudlet (§5.4.4). The mean average precision (mAP) is from the YOLO documentation [128].

Table 6.1: YOLOv5 Performance in the SteelEagle Pipeline

ratio c_i (formula 6.1). I then calculate the score of the frame, s_i, by using an inverse exponential, as shown in formula 6.2. The rationale for using an exponential is to super-linearly penalize distance from origin. I use a compounding 10% penalty in reporting my results, leading to the value of 1.1 in formula 6.2. Using s_n to denote a penalty of n%, Figure 6.4 shows the scores for penalties of 10%, 20% and 30% for the example frame in Figure 6.5. A score of zero is awarded when the frame does not contain the target at all, or if the target is too small to be detected in post-processing by a specified model. In my case, this model is YOLOv5x trained on aerial images of the target. This scoring is meant to award high scores to targets that are centered in frame but take up most of the frame real estate.

From the per-frame scores, the entire flight is scored by simple averaging (formula 6.3). The overall score, s_{avg}, lies between 0 and 1, with higher being better. For example, an average score of 0.70 based on s_{10} is achieved when the drone is able to keep the target within about three normalized lengths of the center of the FOV for the entire duration of the flight.

6.2.4 Tracking Algorithm

For this benchmark, I use the tracking algorithm used described in Section 4.2.5. Figure 6.1 shows the latency, throughput and accuracy of the three DNN models that are used for tracking in my system, each trained on the Robomaster target. Even using the slowest of these as Stage-2 of Orient+Decide$_d$ only adds 42 milliseconds of latency to the base value of 527 ms (Figure 5.12). Its throughput of 20 fps is well above that of the bottleneck (Act$_{fg}$). However, there may be situations where load on a multi-tenant cloudlet may need to be reduced, and the smaller models may be valuable for that purpose.

6.2.5 Experimental Setup

I use the Onion-based architecture discussed in Section 5.2 and shown in Figure 5.3. I also use the same cloudlet as in Section 5.4, with 36 CPU cores, 128 GB of RAM, and an NVIDIA GeForce GTX 1080 Ti GPU. For the underlying drone hardware, I use a different version of the Parrot Anafi, the Anafi USA. This variant is slightly larger and heavier (550 g vs. 360 g), but maintains the same SDK and flight control structure. Its larger size allows it to carry the Onion Omega payload for longer without battery swaps.

6.3 Visual Object Tracking: Results

The basic question I ask about tracking is as follows:

- *How well does my platform follow a target that makes random, rapid changes in direction?*

As Figure 6.6 shows, my platform is able to track the target on my benchmark even at the fastest speed (3.5 m/s) without ever completely losing it. However, as the scores show, the target is off-center in some frames at all speeds. As target speed decreases, the score achieved shows a modest improvement. The results shown here are based on the best model for each speed. This dependence is explored further in §6.3.1.

Since humans are the standard against which AI systems are measured, I ask how a human pilot does under identical conditions. To explore this, I manually pilot the drone through the benchmark for the same parameters. I am an experienced pilot on the Parrot Anafi platform with over a dozen hours of flight time. The OODA loop is now cyber-human: I uses the drone's live video stream to manually fly it. Figure 6.7 shows how well I scored on the benchmark. Comparing Figures 6.6 and 6.7, the autonomous drone and I are comparable at 1.5 m/s, but at higher speeds the autonomous drone outperforms me. I cannot actuate fast enough to keep up with the rapidly shifting target.

6.3.1 Impact of Model Accuracy

Since multiple DNN models are available to use in tracking (Figure 6.1), I ask the following question:

- *Does the use of a better model improve tracking?*

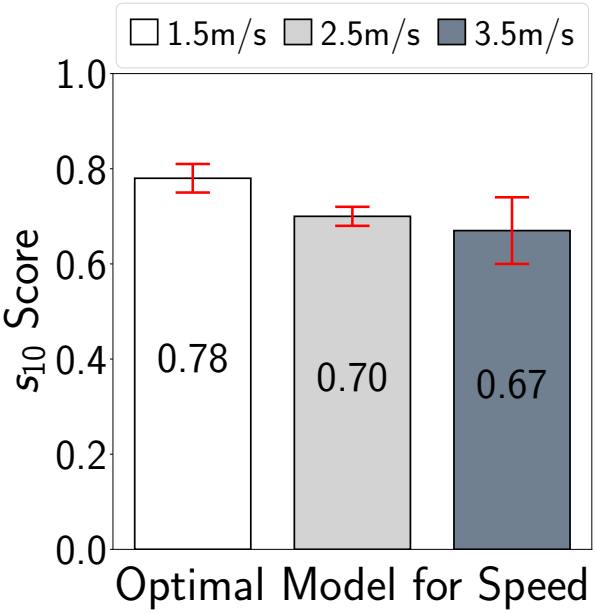


Figure 6.6: Baseline Scores

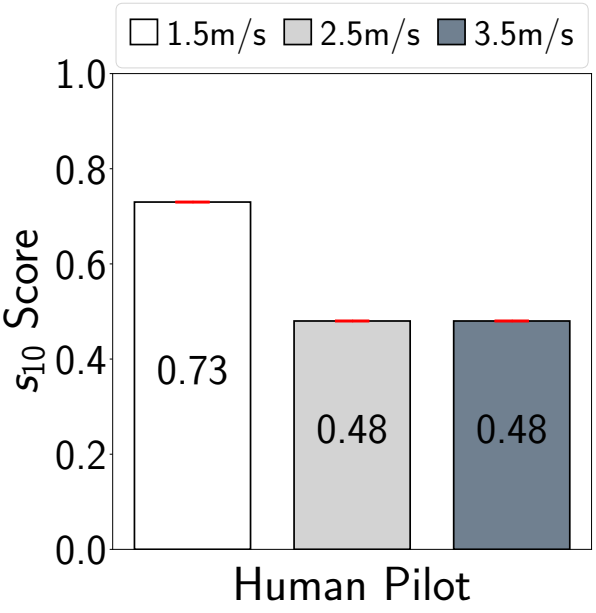


Figure 6.7: Human Pilot

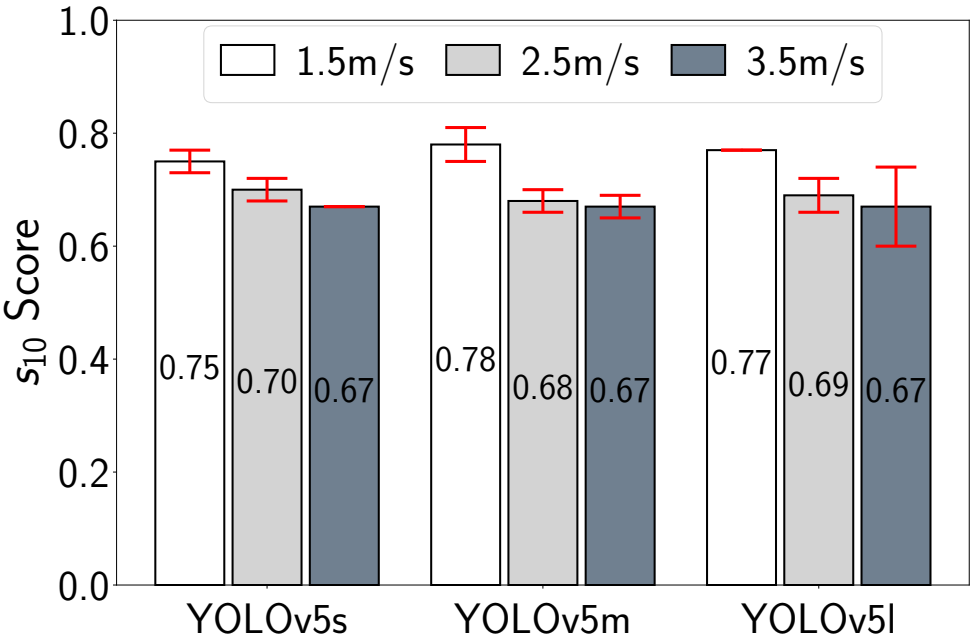


Figure 6.8: Impact of YOLO Model on Tracking Benchmark

Figure 6.8 presents my results. For any given speed, there is little difference across models. The increased cloudlet load of a more accurate model does not pay off. However, it should be noted that this observation may only be true for this specific tracking benchmark. As described in §6.2.2, the benchmark is defined as being conducted in an open area free of clutter. If I were to create a different tracking benchmark that embodies extensive clutter (such as that of a busy street filled with moving cars, bicycles, and pedestrians, along with static objects such as parked cars and trees), the results may be quite different. In that case, the improved accuracy of the larger models may prove decisive. The creation of such a benchmark could be a subject of future work.

6.3.2 Impact of Latency & Throughput

As in the case of obstacle avoidance (§6.5.2), I ask:

- *What is the impact of latency or throughput degradation of the* OODA *loop on benchmark score?*

Figure 6.9(a) shows what happens when additional latency of 250 ms and 500 ms are added to the OODA loop. For all target speeds and models, there is a noticeable drop in benchmark score. The drop is worse at higher speeds. This is directly attributable to the inability of the more sluggish OODA loop to keep the target centered in the FOV. At higher speeds, the drone often loses sight of the target early in the tracking. This results in a zero score for the remaining frames of that experiment, and hence an overall low average score.

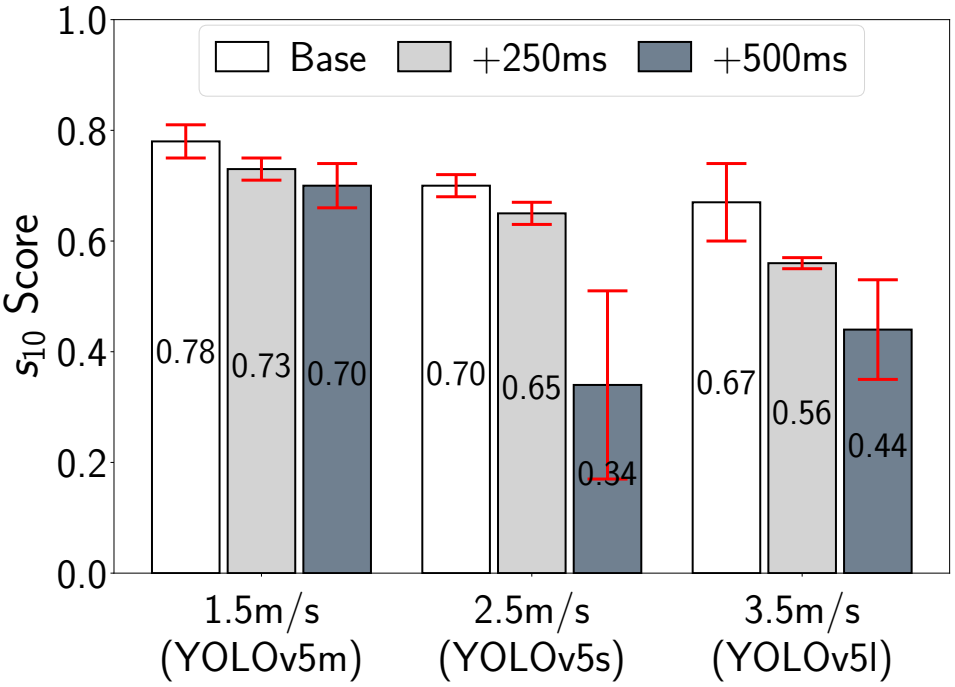
Figure 6.9(b) shows the same trend when OODA loop throughput is artificially throttled to 3 fps or 1 fps. At all target speeds and for all models, there is a noticeable drop in benchmark score. This drop is greater at higher speeds.

The results in Figures 6.9(a) and (b) confirm that both end-to-end latency and bottleneck throughput are important independent factors in determining tracking ability. Optimizing one at the cost of the other, as occurs when using strategies such as batching of operations, is unlikely to be beneficial.

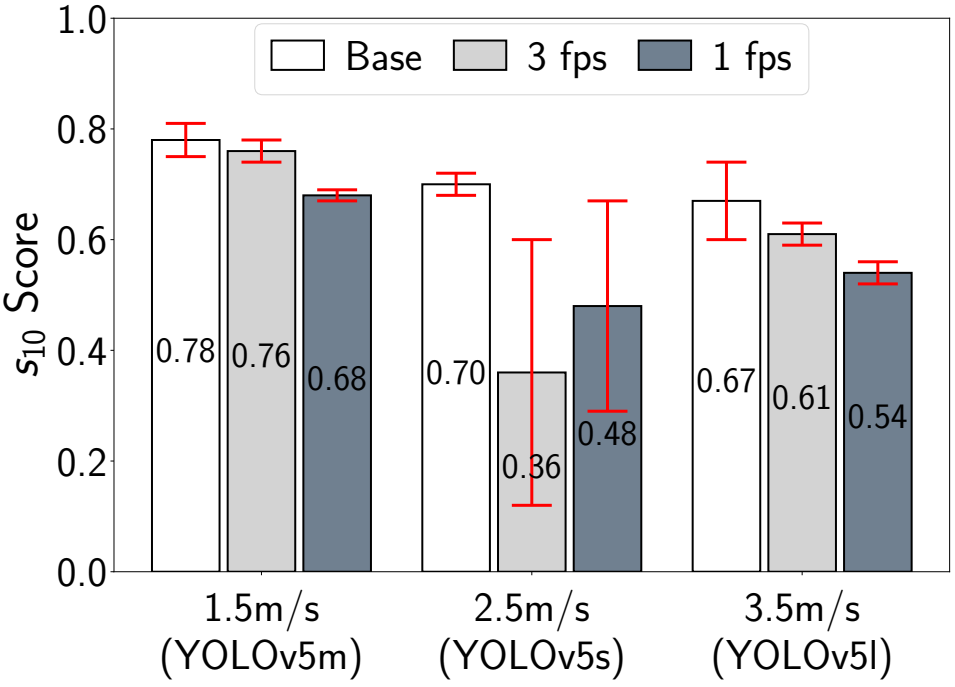
6.4 Obstacle Avoidance Benchmark

Obstacle avoidance is vital for drone flights at low altitude (up to a few hundred feet) in urban or forested settings. Otherwise, being restricted to only high altitude flight impairs visual detections and hides many details. The most dangerous obstacles are typically trees, lightposts, and telephone poles, which can easily reach altitudes usually used by drones. Being relatively thin, they are difficult to detect from afar.

Efficient avoidance of such obstacles is a challenge. Since drone flight is limited by battery life (typically on the order of 30-50 minutes), bypassing obstacles without wasting too much flight time is important. If flight is too slow or avoidance maneuvers are too convoluted, mission performance will be impaired. At the same time, reckless flight could be catastrophic. Striking the right balance between safety and speed for the given flight conditions is essential. Since effective but rapid avoidance of obstacles is a valuable capability in a drone, this task is a good candidate for an agility benchmark.



(a) Additional Latency



(b) Reduced Throughput

Figure 6.9: Impact of Latency and Throughput on Tracking

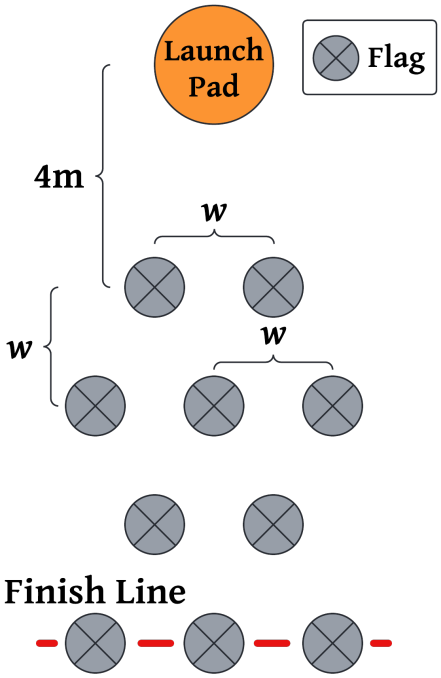


Figure 6.10: Obstacle Course Layout

6.4.1 Benchmark Requirements

The benchmark requirements for avoidance are similar to those described for tracking (§6.2.1). Parameterization that controls the difficulty of the task is valuable. Use of standardized, off-the-shelf components and careful attention to reproducibility of results are important. Capturing the essential difficulty of the task is vital.

6.4.2 Benchmark Description

To emulate tall and skinny obstacles such as lightposts, I use 1.8 m long drone racing flags. The flags are arranged in a precisely-defined slalom pattern (Figure 6.10), that forces a drone to sequentially evade several obstacles. The difficulty of the course is controlled by a spacing parameter, w, that determines the separation of the flags along both the azimuth and range axes. A smaller value of w defines a more difficult course because of higher density of obstacles in both directions. All my experiments were conducted with 4 tiers of obstacles. Adding more tiers to make the obstacle course longer, while preserving obstacle spacing, would add further difficulty to the course. The drone's goal is to navigate the course as fast as possible, without touching the flags or striking a flagpole. The metric of interest is the transit time through the course.

To execute this benchmark, the obstacle course is set up as in Figure 6.10. The drone is placed on a pad that is centered 4 m in front of the leading line of flags. A human spotter stands a safe distance behind the drone, and a human timekeeper is positioned along the finish line. The remote pilot-in-command (RPIC) continuously monitors the video stream from the drone, and

Model	Latency (ms)	Throughput (fps)
MiDaS Small	61	10
DPT Hybrid	105	8
DPT Large	132	7

These numbers were obtained on the cloudlet described in §5.4.4

Table 6.2: Inference Speeds of MiDaS DNN Models

stands ready to wrest back manual control if the drone's autonomous flight software appears to be getting it into trouble. Such a manually aborted flight is scored as "Did Not Finish (DNF)." A flight in which the drone touches a flag or pole is also scored as DNF.

The drone takes off and then hovers at an altitude of 1 m. It is then directed to autonomously move to a destination beyond the obstacle course. When forward motion begins, the spotter visually signals the timekeeper to start a stopwatch. The spotter then follows the drone through the course, warning the RPIC of imminent collision, if any. Such a warning aborts the experiment without damage to the drone. On a successful flight, timing is stopped as soon as the trailing end of the drone crosses the finish line. We deem an obstacle spacing, w, as viable if the drone successfully flies through the course on at least 80% of its attempts. The average time of these successful flights at the smallest viable w is the figure of merit. For small values of w, a more agile drone can fly more aggressively and therefore requires less time to complete the benchmark. However, at higher values of w, agility may not be important because the obstacle course easier.

6.4.3 Benchmark Scoring

The average time, t_{avg}, for multiple experimental runs at the lowest viable w is a raw measure of agility. However, this needs to be normalized with respect to attributes other than agility. For example, a drone whose top speed is low relative to other drones may be penalized unfairly when measuring agility. A low value of t_{avg} in that case is not due to a poor OODA loop, but simply reflects the "brute force" attribute of top speed. The normalization is performed by removing flags from the course and conducting a control experiment at top speed. We denote the average time for multiple runs of the control experiment as t_{min}. The score, S_w, is then given by:

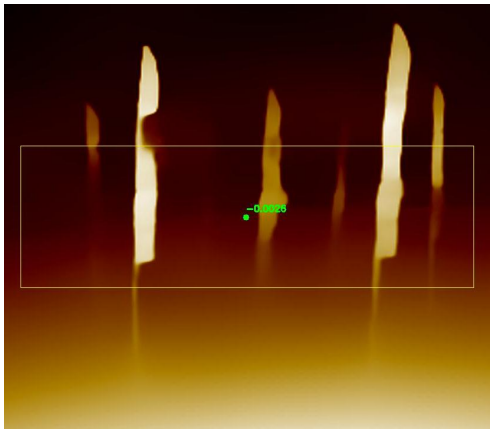
$$S_w = \frac{t_{\text{min}}}{t_{\text{avg}}} \tag{6.4}$$

Scores for this benchmark thus lie in the interval $0 < S_w \leq 1$ where $S_w = 1$ is the best possible score. A score of 0 is awarded when a drone cannot achieve a successful completion rate of at least 80% of the runs for the given value of w.

6.4.4 Depth Sensing

I use the same MiDaS-based avoidance algorithm described in Section 4.2.7. For this task, the OODA loop determines the speed and accuracy with which the drone can acquire fresh frames, execute the MiDaS algorithm on them, calculate new headings for safety, and perform actuations

(a) Raw Input



(b) Output of MiDaS

Figure 6.11: MiDaS Running on the Benchmark Course

towards those headings. In the context of §5.4.4, MiDaS represents Stage-2 of cloudlet processing. Stage-3 is the processing of MiDaS output to determine a new heading, and generating the actuation command. Figure 6.11 shows the algorithm running on a setup benchmark course.

I explore three DNN variants of MiDaS that vary in accuracy and speed: *MiDaS Small, DPT Hybrid*, and *DPT Large*. MiDaS Small prioritizes throughput and inference latency at the cost of lower accuracy. DPT Hybrid strikes a compromise between speed and accuracy. DPT Large prioritizes accuracy above all else. Figure 6.2 shows the inference latency and throughput of these three models on my cloudlet. I explore the use of all three in my experiments.

6.4.5 Experimental Setup

The experimental setup for this benchmark is identical to that in Section 6.2.5.

6.5 Obstacle Avoidance: Results

The most basic questions in my evaluation are as follows:

- *What is the smallest value of w for which the drone can successfully complete the benchmark?*
- *At that w, how fast is benchmark completion?*
- *As w is increased, how much faster is the drone able to complete the benchmark?*

Initial experiments showed that 2 m is the smallest value of w that meets my criterion for successful benchmark completion (i.e., at least 80% of the flights are successful). Using the scoring criterion described in §6.4.3, Figure 6.12 shows how well the drone did for w set to 2 m, 2.5 m, and 3.0 m. The results shown are the mean of 5 runs of each experiment. For each value of w, the drone results were obtained with the choice of MiDaS model that gave the best results. These choices were DPT Large for w =2 m, and MiDaS Small for $w = 2.5$ m and

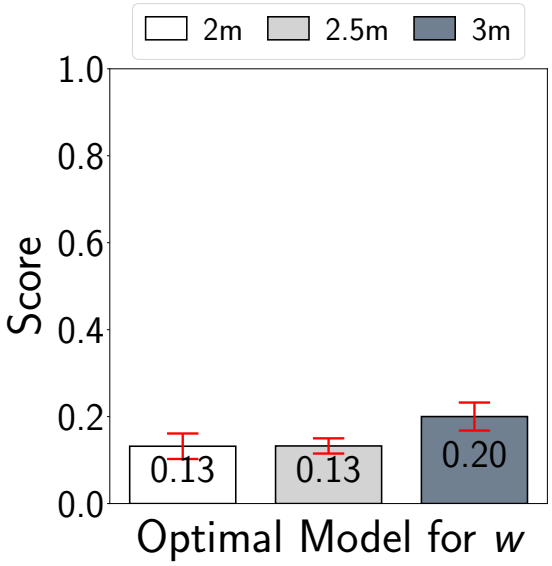


Figure 6.12: Baseline Scores

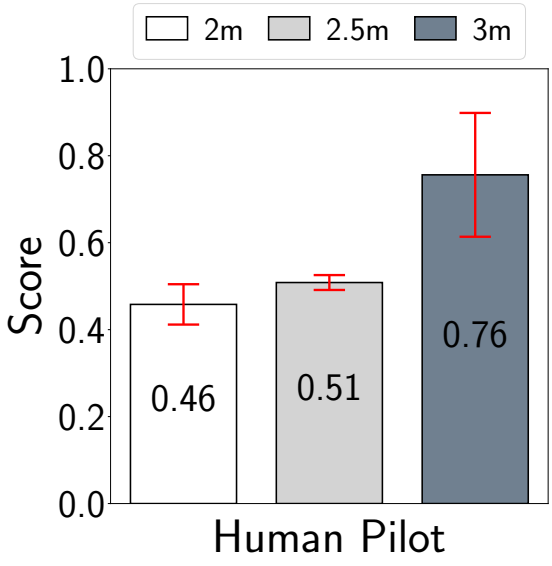


Figure 6.13: Human Pilot

$w = 3$ m. The scores of 0.13 to 0.2 show that the drone suffers almost an order of magnitude slowdown when avoiding obstacles, relative to its unimpeded traversal of the course. This is the price of having to execute the OODA loop shown in Figure 5.12, with the additional Stage-2 and Stage-3 processing for depth estimation described in §6.4.4. As w is increased from 2 m to 3 m, Figure 6.12 shows the score improving from 0.13 to 0.2. This confirms my expectation that less challenging courses are faster to traverse.

As in the case of obstacle avoidance, I ask how well an experienced human pilot performs under identical conditions. To explore this, I again piloted the drone through the benchmark. Of course, a human also has foreknowledge of the obstacle course and can subconsciously leverage that knowledge in planning the drone's flight. In contrast, the autonomous drone is purely reactive — what it sees right now is all that it knows. This is a limitation of the benchmark, since it cannot tease apart the effects of agility versus better path planning.

Figure 6.13 presents my results. Relative to unimpeded traversal of the course, the scores of 0.76 to 0.46 show that even I suffer a slowdown. However, the slowdown is much smaller than that suffered by the autonomous drone in Figure 6.12. Clearly, when it comes to avoidance, my drone system is not yet as good as a human pilot. This is in contrast to tracking (Figures 6.6 and 6.7), where the human was bested by the autonomous drone. I conjecture that at least part of this difference is attributable to the fact that the obstacle course is static, and hence subconscious pre-planning by the human helps in navigating it. In contrast, the human is no better than the drone in anticipating random turns made by the target. At higher speeds, raw reaction speed (i.e., the OODA loop) is all that matters, and the autonomous drone proves to be better in this regard.

6.5.1 Impact of Model Accuracy

The availability of the different MiDaS models shown in Table 6.2 leads to the question:

- *Is accuracy or speed more important?*

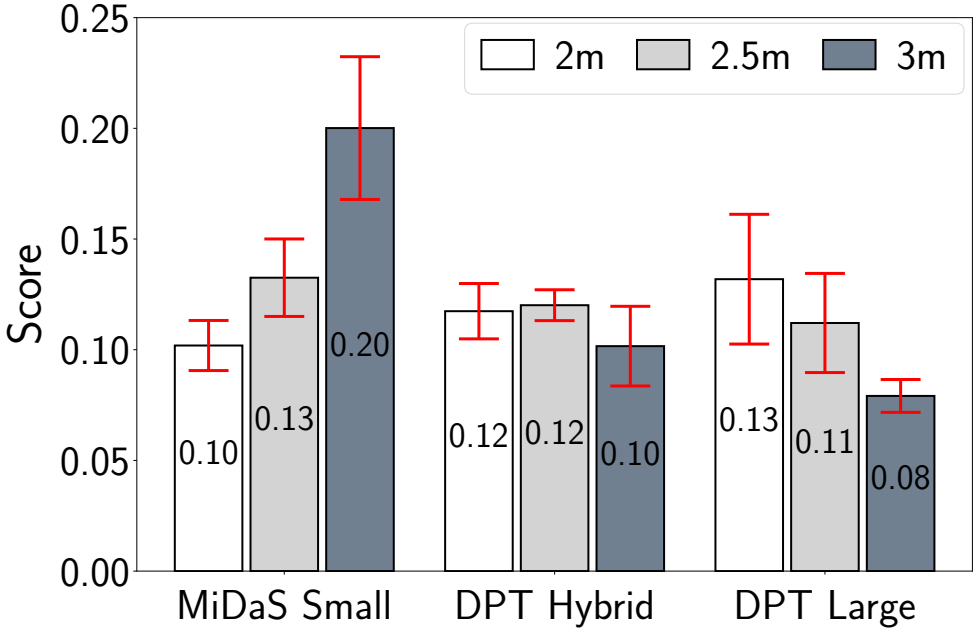


Figure 6.14: Impact of MiDaS Model on Avoidance Benchmark

My experiments indicate that there is no simple answer to this question. The results in Figure 6.12 were obtained using the best MiDaS model for each value of w. MiDaS Small performs the best on the 3 m course but worst on the 2 m course. DPT Large is the inverse, performing best on the 2 m course and worst on the 3 m course. DPT Hybrid stays consistently in the middle for all three courses. The fact that different models had to be used in each case to obtain the best results indicates that there is no single "best" model.

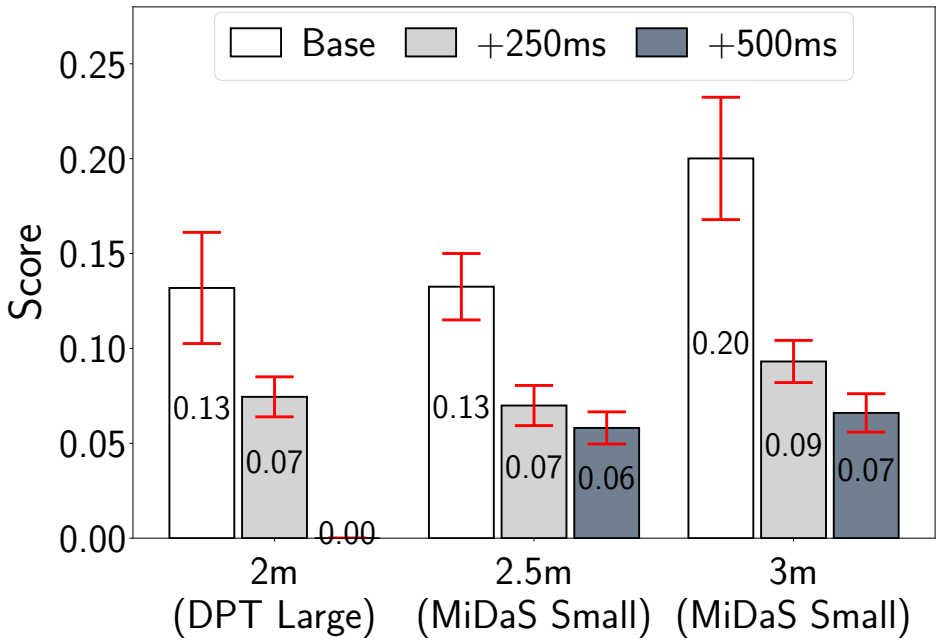
I conducted a set of experiments to better understand this tradeoff space. The results in Figure 6.14 show the score achieved on the benchmark for each model and value of w. Since MiDaS Small focuses on throughput and low latency over accuracy, a drone that uses it is able to sustain a higher maximum speed than one using DPT Large. At w =3 m, the course is sufficiently easy that the increased risk of collision is small. The higher accuracy of a better model is not useful. However, at w =2 m, the increased likelihood of collisions makes higher model accuracy worthwhile. Now, DPT Large attains the highest score. For a tight course, it is difficult to travel at high speed without collisions. Hence, a model that can navigate the gaps better gets a higher score.

6.5.2 Impact of Latency & Throughput

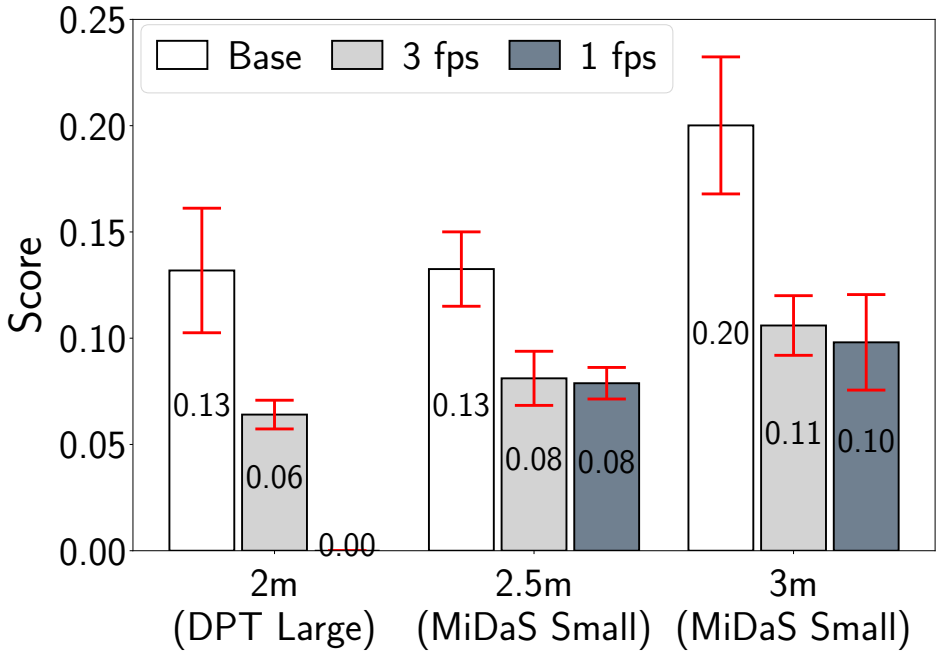
The results presented so far reflect best-case conditions. In practice, the wireless network or the cloudlet may suffer degradation due to multi-tenancy. This leads to the question:

- *What is the impact of latency or throughput degradation of the* OODA *loop on benchmark score?*

The baseline scores reflect what is achievable with an OODA loop whose latency is the sum of three components:



(a) Additional Latency



(b) Reduced Throughput

Figure 6.15: Impact of Latency and Throughput on Avoidance

Architecture	Efficiency (GFLOP / J)
CPU (Core i7)	1.14
FPGA (Xilinx LX760)	3.62
GPU (NVIDIA GTX285)	6.78
GPU (AMD R5870)	9.87
ASIC	50.73

Source: Table 4 in Chung et al [29]

Figure 6.16: Matrix Multiplication Kernel Implementations

- a lower bound of 527 ms (Figure 5.12).
- the latency of the relevant MiDaS model (Table 6.2).
- a small additional overhead (<1 ms) for Stage-3 processing in the Decide part of the OODA loop.

This total end-to-end latency is on the order of 600–650 ms. For OODA loop iterations that involve drone actuation, the bottleneck throughput is the smaller of 6 fps for Act_{fg} (§5.4.6) and the throughput of the MiDaS model (Table 6.2). This is effectively 6 fps, regardless of model. If no drone actuation is involved, the bottleneck throughput becomes that of the MiDaS model. Since even MiDaS Small has lower throughput than Observe_{ab}, Observe_{c}, or best-case Orient+Decide_{d}, throughput is always in the 6–10 fps range.

Figure 6.15(a) shows how benchmark score drops as latency is artificially added to the OODA loop. Even 250 ms of additional latency (i.e., a roughly 35–40% increase from baseline) causes benchmark score to drop to nearly half its baseline value, for all values of w and regardless of MiDaS model. If 500 ms of latency is added, the score drops further. For the most challenging course ($w = 2$ m), the score drops to zero because not even 80% of the flights are successful. These results are consistent with a long-standing design principle of deeply-immersive closed-loop interactive systems: *increased latency is deadly, even if throughput remains good.*

Figure 6.15(b) shows how benchmark score drops as the throughput of the OODA loop is artificially reduced from its baseline value. Reducing throughput to 3 fps causes benchmark score to drop to nearly 40–50% of its baseline value. A further drop is observed when throughput is reduced to 1 fps. For $w = 2$ m, the benchmark score drops to zero. These results confirm that latency is not the sole determinant of task performance in an OODA loop — throughput also matters.

6.6 Value of On-board Drone Intelligence

Edge computing allows use of compute resources that are far larger and heavier than could be carried by an ultralight drone. In the context of AI, this translates to *generality* and *versatility*. Purely through software development on the cloudlet, it is easy to re-purpose the drone for new tasks that were not anticipated earlier.

The drone marketplace, however, is moving in the opposite direction. Drone vendors are constantly identifying specific new functionality to add to drones. There is a well-understood

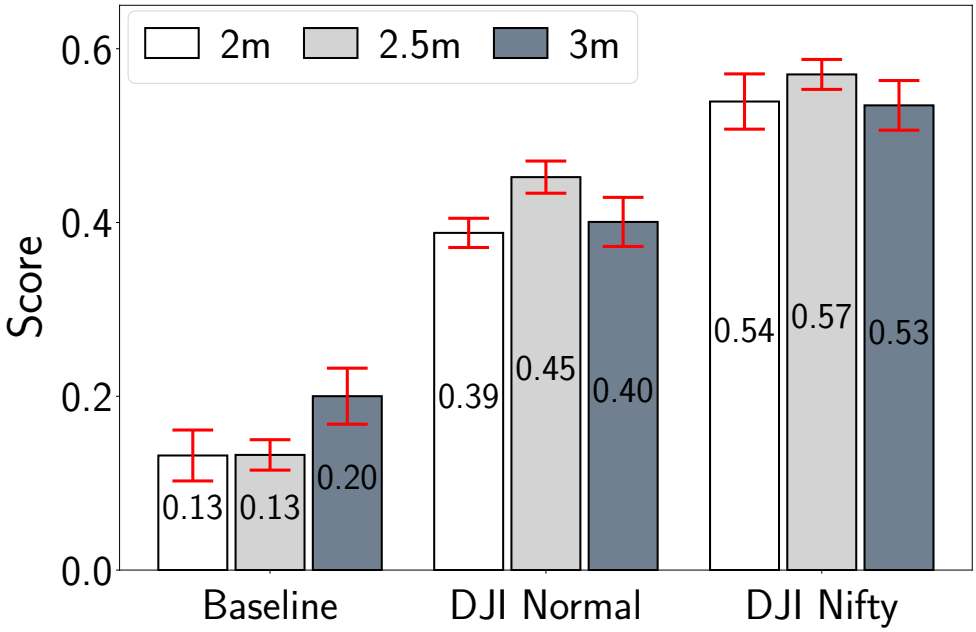


Figure 6.17: On-Board Obstacle Avoidance

tradeoff between generality, energy-efficiency, development cost, and weight/size that applies in this context. As Figure 6.16 from Chung et al [29] shows, an ASIC is by far the most energy efficient alternative for a given functionality. It is also likely to have the lowest weight. But, it takes much longer to develop, is much more expensive to create than pure software, and only provides fixed functionality.

To quantify the value of on-board intelligence, I re-ran my obstacle avoidance benchmark using a DJI Mini 4 Pro drone, seen in Figure 2.2. This consumer photography drone weighs 249 g, and is equipped with 6 stereo cameras and an onboard obstacle avoidance system. It has two modes, Normal and Nifty. Normal prioritizes safe flight while Nifty attempts to pass obstacles as quickly as possible. The obstacle avoidance feature of this drone is intended as a form of "pilot assist." The RPIC flies the drone without worrying about obstacles. The drone's builtin capability performs all the necessary sensing and actuation needed to avoid collisions.

Figure 6.17 presents the scores obtained by the DJI Mini 4 Pro on my obstacle avoidance benchmark. The baseline results for my platform from Figure 6.12 are also shown for comparison. For all values of w, the DJI Mini 4 Pro is at a clear advantage. This the direct result of additional sensors and a much faster OODA loop that avoids edge offload.

These results suggest that even when using edge offload, there is very clear value in taking advantage of on-board capabilities when they are available. The "pilot assist" approach to obstacle avoidance implemented by the DJI Mini 4 Pro could equally well be used by cloudlet-based software to control the drone's flight path. Only the macro components of that flight path would incur the overhead of the OODA loop from drone to cloudlet. The micro components of the flight path that maneuver the drone around obstacles would only be subject to its much tighter on-board OODA loop.

Complementing on-board fixed functionality with edge offload provides extensibility and

versatility. The DJI Mini 4 Pro, for example, is unable to do general-purpose object detection even though it can detect people and vehicles. It cannot detect the Robomaster target, and is hence unable to execute my tracking benchmark. Edge offload could remedy that limitation, thereby increasing the versatility of the drone.

6.7 Summary

The parameterizable agility benchmarks presented in this chapter are the first reproducible method to compare autonomous drone performance in real flight. SteelEagle's scores, especially on the avoidance benchmark, show that there is vast room for improvement in the platform OODA loop. However, its good performance on tracking demonstrates the usefulness of the platform, outperforming the human pilot analog. The success of the DJI Mini 4 Pro on avoidance further suggests that onboard compute cannot be entirely ignored; for some tasks, it bests an edge computing approach. This motivates a system evolution of SteelEagle, which can not only adapt to varied drone hardware but also use available resources to its advantage. In the next chapter, I create a design sketch of such a system, and explore how to make it truly drone agnostic.

Chapter 7

Drone Agnostic Flight Operations

Portability is one of the many absent features of fully-autonomous drones today (§2.3). Most consumer drone manufacturers lock their products in to a closed-source, proprietary SDK which cannot be used on other platforms. This is an economic decision; drone companies make profits by selling drone hardware, and so there is little incentive to build companion software which could also be used with competing aircraft.

From its inception, SteelEagle was positioned as a drone agnostic system, able to support many different classes of drone hardware. Theoretically, this could allow users to press disparate drones into service to fulfill demand, or operate heterogeneous drone swarms using aircraft with complementary capabilities. Thus far, I have only demonstrated the system on the Parrot Anafi and the Parrot Anafi USA, which share the same control paradigm.

In this chapter, I show how SteelEagle can adapt to different drone hardware. In Section 7.1, I discuss how disparate platforms can be made to work within its ecosystem, and the types of drone control schemes which can be supported. In Section 7.5, I walk through the process of integrating a new drone and run it through my benchmark suite (§6.2, §6.4).

7.1 Need for a Hardware-Adaptive Software Architecture

A vital component of SteelEagle is the companion application software which links the edge with the underlying drone hardware and runs mission logic. For the watch prototype, this was an Android application running on-device. For the Onion prototype, this was an application running on the cloudlet that talked to the drone over the network. In both cases, this application served the same role: relay video and telemetry from the drone to the edge, run missions, and send commands to the drone using its SDK.

From the perspective of the backend, the companion application, denoted by the "Mission" label, is abstracted behind the data and control servers. This is clear in both the Onion (Figure 5.3) and watch (Figure 4.1) architecture diagrams; everything to the right of the data and control server is identical. There is a shared protocol for sending commands on the control plane and for receiving data on the data plane but there are no assumptions made about the makeup of the application. In this way, the companion application represents a kind of drone abstraction layer that enables the drones it interacts with to plug into SteelEagle.

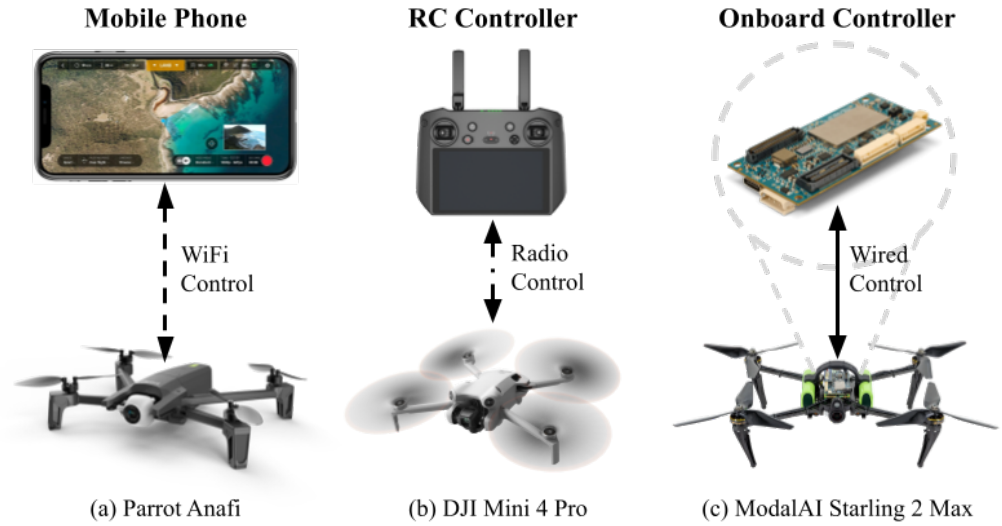


Figure 7.1: Drone Control Schemes

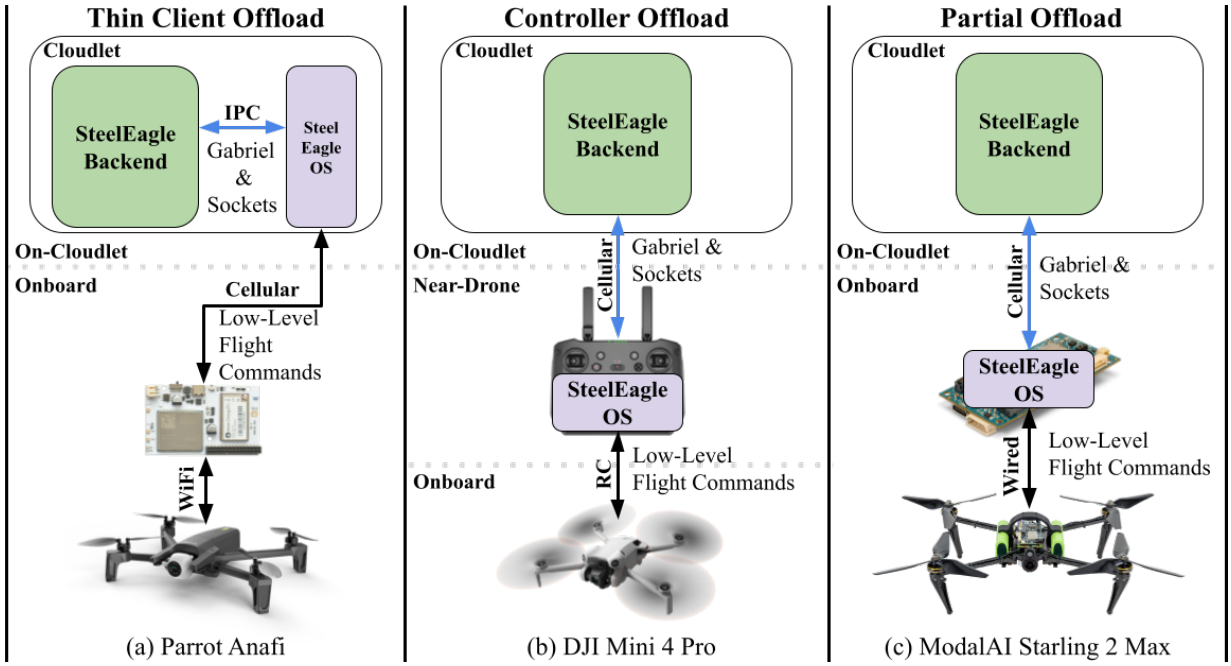
The design of this abstraction layer is difficult. Drones not only have different flight properties, SDKs, and sensors, but also often have divergent control methods, as shown in Figure 7.1. For instance, some drones may require a radio controller to be connected at all times, even when autonomous commands are sent by an onboard application. Others necessitate a pilot to take-off manually before autonomous flight can begin. These safeguards can rarely be circumvented without serious modification of the drone hardware, a non-starter for a project focused on accessibility and easy portability.

In addition, the current Parrot Anafi prototype with the Onion payload lacks onboard compute, but this is not the case for all drones. Some have stereo cameras and onboard obstacle avoidance like the Parrot Anafi Ai. A few may even have generalized computation resources like GPUs which could run lightweight versions of the models on the cloudlet. These drones could even support disconnected operation.

The only way to reconcile these differences is to design around them. Ideally, the companion application should be itself highly portable, able to work with many different drones, each using a completely distinct control model. It should take advantage of onboard compute and support disconnected operation. All the while, it must maintain a unified interface for the backend, allowing these drones to gain the benefits of edge-based autonomy.

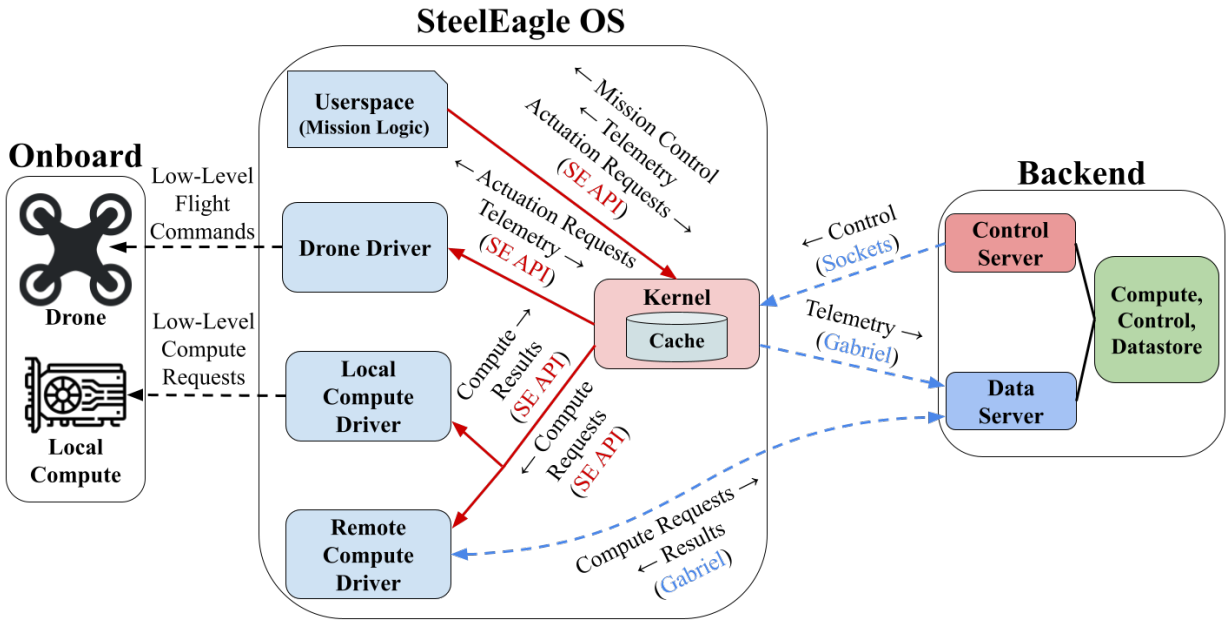
7.2 The SteelEagle Operating System

To address the above design constraints, it is helpful to view the companion application as the *SteelEagle Operating System*. As with other popular operating systems like Linux, the SteelEagle Operating System (SteelEagle OS) must manage varied underlying hardware (drone control schemes) and run user code (missions). It also has to ensure safe operation via monitoring, interrupts, and permissions. This further motivates separating such an operating system into logical units like the kernel, userspace, memory, and drivers.



The SteelEagle OS can live on the cloudlet, on an RC controller, or on an onboard device. This supports the majority of COTS drone control schemes on the market today.

Figure 7.2: SteelEagle OS Placement for Different Control Schemes



Dashed connections represent external links, as seen in Figure 7.2. All SteelEagle OS modules are assumed to be collocated. The SteelEagle API (shown as SE API) is described in detail in Section 7.3.

Figure 7.3: Architecture of the SteelEagle OS

The SteelEagle OS architecture, shown in Figure 7.3, is organized in a hierarchical structure, with the kernel acting as the command center, and the userspace and drivers using the kernel for system calls. SteelEagle OS, as illustrated in Figure 7.2, can be configured in one of three modes. For a thin client drone (Figure 7.2(a)), SteelEagle OS can run on the cloudlet as a proxy with the drone driver talking to the aircraft over cellular. For a RC controller-dependent drone (Figure 7.2(b)), SteelEagle OS can run on the RC controller, using native RC to connect to the drone and cellular to communicate with the cloudlet. For a drone with sufficient compute resources onboard (Figure 7.2(c)), SteelEagle OS can be deployed onboard, sending commands directly to the drone autopilot over a wired connection while using an onboard modem to communicate with the cloudlet.

All communication between SteelEagle OS modules is done via a unified system call API called the SteelEagle API (shown as the SE API in Figure 7.3) which will be further discussed in Section 7.3. This is a near-identical structure to other popular operating systems today. Each driver is designed to interface with one of the pieces of hardware within the overall system. For example, the drone driver is solely responsible for handling all interactions with the physical aircraft and autopilot. There are no constraints on how modules are written, only that it follows the schema of the SteelEagle API. This promotes maximum portability and gives SteelEagle OS the widest possible compatibility umbrella.

There are three primary data flows within SteelEagle OS: the manual control flow (Section 7.2.1), the autonomous control flow (Section 7.2.2), and the computation flow (Section 7.2.3). I will step through each flow to show how modules interact with each other.

7.2.1 Manual Control Flow

The manual control flow is responsible for giving a remote commander manual control over a drone in case of emergency or in case human assistance is needed. When a commander requests manual control of a drone, a control message is sent from the commander client through the backend over the control server to the SteelEagle OS instance, where it arrives in the kernel module. The kernel is responsible for maintaining this link, and if this link is over the air, its reliability is directly tied to the reliability of the underlying network.

Once the manual control request arrives at the kernel, it immediately revokes actuation permission from the userspace and prepares for further instructions. Actuation permission gives a module authority to control the drone. The kernel is the only module that always has this permission and it decides when to grant or revoke it for other modules. Any actuation requests received from non-permitted sources are ignored. In this case, if actuation requests arrive from the backend, they are promptly converted into SteelEagle API messages by the kernel, relayed to the drone driver, and executed. Meanwhile, the kernel continues sending telemetry via the data server to the backend which is eventually displayed to the commander client.

7.2.2 Autonomous Control Flow

The autonomous control flow is responsible for running autonomous missions within SteelEagle OS. When a commander sends a mission to a drone, a control message containing compiled mission logic is sent from the commander client through the backend over the control to the

kernel. Upon receiving the mission, the kernel sends it to the userspace module and grants the userspace actuation permission. The sole task of the userspace is to configure, monitor, and run autonomous missions. Once the userspace has the mission, it configures all computation relevant to the mission through the SteelEagle API (Section 7.3). For instance, if the mission involves tracking a human target, the userspace may provision an object detector trained for the specified class of human. These messages are sent to the kernel where they are then passed on to the local and remote compute drivers. All compute results will eventually be stored in the kernel cache which can be read by the userspace on demand while it has actuation permission.

As soon as the appropriate computation requests have been acknowledged, the userspace starts the mission. During its lifetime, the mission may actuate the drone by sending SteelEagle API actuation requests, or query computation results by sending SteelEagle API computation requests. All messages are sent to the kernel where they are then delivered to the intended recipient. Fundamentally, this design mirrors the design of the userspace in modern operating systems. Since user code is not trusted, all system-level calls must be validated by the kernel before they are executed. At mission end, the userspace notifies the kernel which then revokes actuation permission and signals an end of mission to the commander.

7.2.3 Computation Flow

The computation flow is responsible for configuring compute resources and delivering computation results to consumers like the userspace. Computation configuration messages are generated by the userspace during mission setup. These messages, in the case of object detection, contain the model type, camera sensor identification, target class, and confidence score. This will change depending on the task. For example, an obstacle avoidance task may request a MiDaS avoidance model which would be specified with just the model type and camera sensor identification. Regardless, these configuration messages are delivered to the local and remote compute drivers.

After the local and remote compute drivers receive a computation configuration message, they set up the corresponding model, if they have access to it. For the local compute driver, this could be a pruned, less accurate model which can run on constrained mobile hardware. For the remote compute driver, it will request the full size model to be provisioned by the SteelEagle backend. By default, SteelEagle OS attempts to set up appropriate computation resources both locally and remotely. This gives the drone a local fallback model for disconnected operation in case remote resources are inaccessible. As the mission is executed, frames are constantly ferried from the drone driver to the compute drivers via the kernel. There frames are then inferenced using the configured compute, and results are cached on the kernel's datastore. Compute requests sent from the userspace to the kernel can then read these results, indexed by model type.

7.3 SteelEagle API Design

The SteelEagle API forms the backbone of all communication within SteelEagle OS. It is, at its core, a list of synchronous and asynchronous remote procedure calls (RPCs) that modules can invoke on each other. For example, the userspace may make an RPC actuation request to the kernel to move the drone to a GPS location. This request contains the move to GPS actuation

Type	Message	Response	Payload
Telemetry	get_telemetry	telemetry_frame	Empty
Telemetry	telemetry_frame	None	Aircraft telemetry
Telemetry	get_sensor	sensor_frame	Sensor ID
Telemetry	sensor_frame	None	Sensor content
Actuation	actuation_request	actuation_ack	Actuation type[†]
Actuation	actuation_ack	None	Success or failure
Compute	compute_configuration	compute_ack	Model, sensor, class, confidence
Compute	compute_request	compute_result	Model, sensor, class, confidence
Compute	compute_result	None	Inference result
Mission	mission_start	mission_ack	SteelEagle mission
Mission	mission_stop	mission_ack	Empty
Mission	mission_ack	None	Success or failure

[†]Actuation requests roughly follow the format of the MAVLink API [83]. This is the canonical way in which quadrotor drones are controlled.

Table 7.1: High-Level Overview of the SteelEagle API

type, followed by relevant parameters. In this instance, the parameters would include the target latitude, longitude, and altitude. The actuation request API closely follows the MAVLink API in its overall structure, but drivers can implement the API using a custom SDK of choice [83]. Once a request is sent from the userspace to the kernel, it is then forwarded to the drone driver which completes the action and returns an acknowledgment. In this way, modules are easily separable; so long as they communicate using the SteelEagle API, they can be written in any format or language.

RPC calls within the SteelEagle API are divided into four categories: telemetry, actuation, compute, and mission. Telemetry messages are for sending sensor data and the video stream from the drone driver to other customer modules. Actuation messages are for requesting actuation on the drone. Mission control messages relay directives to the userspace from the commander. Compute messages request computation from local or remote engines. Since SteelEagle API RPCs are based on the Protocol Buffer data packaging system, more message types can be added later without breaking compatibility [102]. Table 7.1 shows a high-level overview of the SteelEagle API and its associated messages.

7.4 Implementation Considerations

As mentioned earlier, portability and flexibility are first class design considerations for SteelEagle OS. For this reason, all OS modules are deployed as containers which can run on a variety of host operating systems. On Linux, Windows, or Mac based hardware, setup is trivial. This covers a broad range of devices, including mobile single-board computers like the Raspberry Pi and most consumer computers. In some cases, the host operating system may not support

Figure 7.4: ModalAI Starling 2 Max [87]

containers. As a result, SteelEagle OS cannot be set up in its container configuration. Instead, it must be rewritten and optimized for the new environment while maintaining all external abstractions. This is not ideal as it promotes fragmentation of the code base when confronted with incompatible devices. However, this is a necessary concession to ensure SteelEagle can function in almost every case.

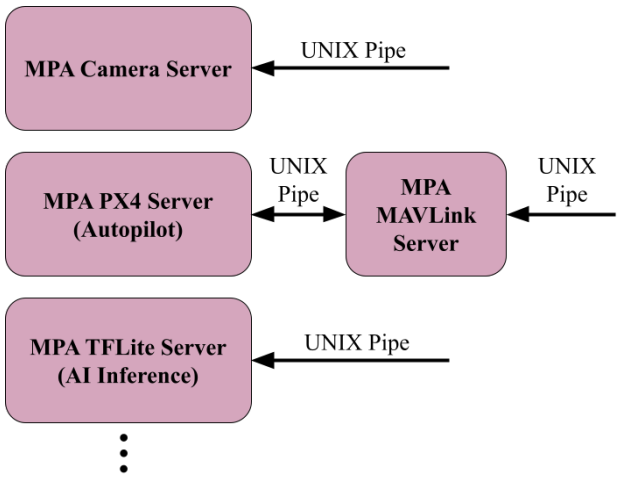
If a drone has sensing or actuation capabilities beyond the scope of the SteelEagle API, SteelEagle OS supports freeform RPC calls which can be forwarded from customized mission code to the drone driver. This is similar to `ioctl(2)` in Linux which gives the operating system flexibility when dealing with specialized drivers [81]. Using this system, SteelEagle OS can take advantage of exotic drones without needing to change the SteelEagle API. The API can always be extended later if a critical mass of compatible drones requires this RPC feature. In the case of custom ASICs or local computation, the local compute driver can be written to support any associated API. This is where the language agnosticism of each module becomes advantageous; modules can be written in completely distinct ways as long as they follow their RPC interface.

7.5 Integrating a New Drone into SteelEagle

The only true test of SteelEagle's drone agnostic design is to successfully integrate a new drone into the system. In this section, I will walk through the integration of the ModalAI Starling 2 Max [87], a drone with completely distinct hardware from the Parrot Anafi series. I will show where development effort needs to be spent and where my design simplifies this task.

7.5.1 Hardware and Software Overview

The Starling 2 Max, shown in Figure 7.4, is a quadcopter with a rich sensor suite and significant onboard compute. It is equipped with a downward-facing fisheye visual inertial odometry (VIO) camera for GPS-denied localization, front-facing stereo cameras for obstacle avoidance, and a front-facing RGB camera for perception. Providing onboard intelligence is the VOXL 2 chip,



A small selection of the servers avaiable within the default MPA configuration. Each server is started as a UNIX service and can be accessed via UNIX pipes.

Figure 7.5: ModalAI Pipe Architecture (MPA) (Adapted from [89])

a collaboration with Qualcomm to build a SWaP (size, weight, and power) optimized robotics-focused single-board computer. The VOXL 2 sports a smartphone caliber CPU in the QRB5165, and hosts a proprietary GPU which can inference weight-optimized models in real time [88]. It also has an embedded 4G modem which can communicate with the cloudlet out-of-the-box.

The VOXL runs an Ubuntu [56] distribution and uses standard UNIX services to run its in-flight processes. These services communicate with each other using the ModalAI Pipe Architecture (MPA), an IPC mechanism built on UNIX pipes [89]. Figure 7.5 shows a simplified diagram of the MPA. For instance, to send commands to the autopilot, an autopilot service, called the MAVLink service, must first be started. Then, commands can be fed over a UNIX pipe to the service and the drone will actuate.

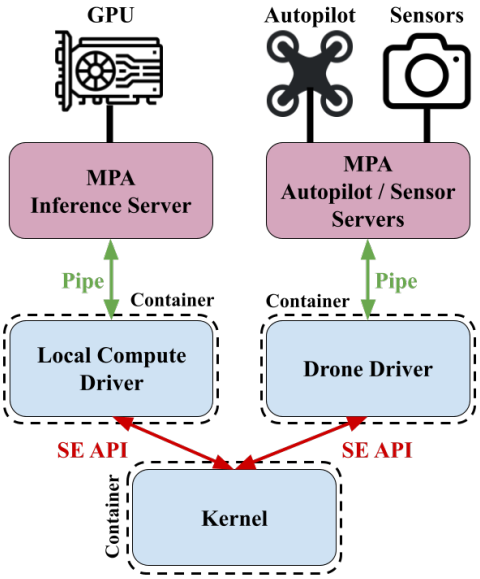
The VOXL 2 is equipped with PX4 autopilot software which handles all low level flight control [43]. PX4 is a well-known open source autopilot, and it is compatible with the ubiquitous MAVLink protocol [83].

7.5.2 Development and Deployment Experience

There are two main aspects that differentiate the software environment of the Starling from the Parrot Anafi series. First, it has onboard compute and thus requires a compute driver. Second, its control scheme depends on pure MAVLink rather than Parrot Olympe. Thus, a new drone driver and local compute driver must be created. Once this is done, all components of SteelEagle OS can be deployed onboard as Docker containers. Figure 7.6 shows the integrated system diagram.

Drone Driver

As mentioned above, MAVLink is the canonical protocol used to communicate with PX4 autopilots. The MAVLink API is very similar to the Parrot Olympe API; this is not surprising since



Red arrows show communication using the SteelEagle API. Green arrows show communication using VOXL MPA pipes [89].

Figure 7.6: VOXL Integration into SteelEagle OS

Olympe is a modified MAVLink wrapper. To convert the existing Olympe driver into a MAVLink driver, only slight modifications are needed.

Local Compute Driver

The VOXL 2 has a Tensorflow inference server that can be automatically started on boot. The inference server consumes frames over an MPA pipe and sends out results over a different MPA pipe. A compatible compute driver only needs to transform SteelEagle API compute requests into MPA data to send over the pipe and vice versa. The MPA API is most easily compatible with C/C++ while the rest of SteelEagle OS is written in Python. Thanks to its modular language agnostic design, this is not an issue, and the local compute driver can be written in a different language than the rest of the pipeline without compromising abstraction layers.

7.6 Validation

I have successfully tested the ModalAI Starling 2 Max flying autonomously via SteelEagle both in simulation and in real flight. This proof of concept could be extended to drones running ArduPilot [12] or other drone families like DJI.

Chapter 8

Conclusion and Future Work

In the previous chapters, I have shown that SteelEagle is an effective solution to the onboard compute tradeoff. By taking advantage of edge computing, it can induce full autonomy on lightweight COTS drones without demanding heavy onboard compute. I showed how to connect COTS drones to the edge using off-the-shelf relays, and how to design an edge backend that could support autonomous operation. I also evaluated the system on a set of benchmarks to show its capabilities. Lastly, I demonstrated how SteelEagle could be easily ported to new drone hardware. In this chapter, I will summarize the major contributions of this dissertation and suggest future research directions.

8.1 Summary of Contributions

8.1.1 Active Vision with Ultralight Drones

Current autonomous drones have seen limited use in urban settings because their heavy weight exceeds government regulatory limits. SteelEagle presents a way to achieve full autonomy on lightweight drones within these limits by offloading computation normally done on heavy onboard resources to a nearby cloudlet via a cellular connection. I demonstrated how such a connection can be established on consumer drones and I showed that an initial proof-of-concept system was able to execute active vision tasks. My lightest prototype had a takeoff weight of just 360 g, only 110 g over the FAA regulatory cutoff, which is much closer than traditional autonomous drones that typically weigh over 1 kg. I expect that future improvements in drone and mobile hardware may drive this number down considerably.

8.1.2 Autonomy on COTS Hardware

Most drone research tends to focus on custom-built hardware that is tailor made for its intended task. SteelEagle, by contrast, is designed to work with commercial off-the-shelf (COTS) components to promote easy deployment. Critically, my prototypes show that it is possible to induce full autonomy on COTS drones that have no cellular connection or compute by mounting a lightweight COTS relay device on board which acts as an intermediary between the drone and

the cloudlet. To my knowledge, this has never been done before. This control paradigm has the potential to be disruptive, possibly introducing a new market segment of lightweight, COTS edge-based UAVs.

8.1.3 An Open-Source Edge Pipeline for Drone Intelligence

While edge-based drone intelligence is not new, many solutions either have a narrow range of compatibility or are closed-source. SteelEagle represents the first open-source edge intelligence framework that is designed to support a wide range of AI backends. Its cognitive engine approach, borrowed from Gabriel [62], creates a plug-and-play development environment which can easily integrate new AI models into the processing pipeline and ride the wave of AI innovation in algorithms and hardware accelerators.

8.1.4 Benchmarks for Autonomous Drone Agility

Existing benchmarks for drone agility often leverage simulated environments to encourage accessibility and reproducibility. However, in my experience, simulations struggle to capture the flight dynamics of real aircraft. My benchmark suite attempts to find a middle ground by providing a reproducible and accessible setup while testing drones in real flight. To the best of my knowledge, it is the first such benchmark suite of its kind. Over time, more benchmarks can be added to the suite and scoring can be refined. Additionally, as more drones are incorporated into SteelEagle, each can be scored on the suite. The benchmarks are parameterized so that future versions can easily increase in difficulty as drone technology improves.

8.1.5 Hardware Agnostic Operations

Today, the drone space is fragmented with dozens of custom SDKs and autopilots, each used for a specific set of aircraft. Without a unified development environment, it is impossible to port code written for one class of drone to another. SteelEagle attempts to solve this by introducing SteelEagle OS, an operating system designed to integrate varied drone SDKs under a single unified API. The architecture of SteelEagle OS is also built to support as wide a range of control schemes as possible, including those that include a local RC controller. This system, if seen through to its full potential, could be the primary development tool for mission-centric drone programming within the research community and in industry practice.

8.2 Future Work

8.2.1 Other Robotic Platforms

Quadcopters are not the only types of robotic platforms that can benefit from edge computing. Rovers, fixed-wing aircraft, helicopters, sea vehicles, and more could all leverage edge computing in a similar way to SteelEagle drones. MAVLink, the ubiquitous UAV control protocol, is actually designed to work with many of these platforms, and sees actual use in these settings

today. SteelEagle could replicate the success of MAVLink, and could open the door for multi-domain robotic cooperation between compatible systems.

8.2.2 Drone Swarms

The majority of this dissertation is concerned with testing the performance of a single drone. However, most drone research is now headed towards drone swarms. With drone swarms, dozens of aircraft, much like a swarm of bees, collaborate towards a shared goal. The idea is that such aircraft would be cheaper and less capable than larger platforms, but would perform equivalently in aggregate. This perfectly fits the ethos of SteelEagle which seeks to connect small, cheap drones to powerful compute without increasing cost or weight. Thus the logical next step is to make drone swarms a first-class citizen of SteelEagle. In the future, this could mean heterogeneous swarms of cheap, insect-like aircraft coordinated by one or several cloudlets for building inspection, surveillance, or weather monitoring.

8.2.3 Expressive Mission Specifications

In order to achieve collaboration between swarm aircraft, changes must be made to the SteelEagle mission specification. Currently, it is a static script that the drone follows until completion. While this can support some dynamic action, it lacks the structure to cleanly integrate collaborative behavior. One option is to shift from a static flight script to a finite state machine, where behaviors are linked together by a pre-defined transition function. This could eventually turn into an independent domain-specific language which, when integrated with the SteelEagle API, could be a cross-platform coding tool that spans many drone control paradigms.

8.2.4 On-Demand Edge Deployment

For real flight operations, especially in adversarial environments, it is rarely the case that compute resources are pre-provisioned for missions. Instead, they must be allocated on demand to support the current task. In its present configuration, the SteelEagle backend has no mechanism for dynamic deployment; it must be set up on a machine manually and then drones must be notified of its address within the network for the system to function. In the future, this startup routine could become more dynamic. For example, when a drone requests computation, a central authority could find the closest cloudlet and provision a SteelEagle backend for the drone. It could also respond to heightened demand through server replication, or migrate cloudlet sessions across servers as drones are in flight [61].

8.2.5 Leveraging Onboard Compute

As shown by Section 6.6, onboard compute can sometimes outperform edge-based solutions. In these situations, the SteelEagle OS local compute driver is responsible for taking over computation duties from the remote compute driver. However, there are times when the separation between what should be done onboard versus on the cloudlet is not cleanly separated. In particular, prior work has shown the benefits of a collaborative approach in which local computation informs

when data is promising enough to be shipped to the cloudlet for further analysis [71, 121]. This approach, known as "offload shaping" can save bandwidth and energy. Currently, SteelEagle OS cannot use offload shaping, but it can be extended to support it.

8.2.6 Transient Disconnected Operation

A major weakness of SteelEagle is its dependence on its connection to the edge. If network disconnection occurs, SteelEagle OS is designed to safely return the drone to its launch point. This is the only possible course of action for thin client drones. Even so, for more capable clients, limited disconnected operation could be possible. One can imagine a drone with marginal onboard compute that could try to seek out the last location where it had service, or alternatively continue its mission with degraded performance until reconnection occurs. Such functionality could be critical for real world deployments of SteelEagle, especially in adversarial settings, where consistent connectivity cannot be expected.

8.3 Closing Thoughts

SteelEagle presents the first open-source attempt at bringing full autonomy to lightweight, COTS drone platforms. It promotes portability with its hardware-agnostic modular design, and is positioned well to take advantage of future research. My hope is that its accessibility could make it a "Linux for drones", the default platform that drone autonomy developers the world over deploy their projects on. Such democratization could herald a new era of drone innovation, unbound by the constraints of proprietary SDKs while preserving public safety consistent with existing government regulation.

Bibliography

[1] Adobe. What is video bitrate? https://www.adobe.com/creativecloud/video/discover/bitrate.html. Last accessed on June 23, 2024.

[2] Alan Perlman. BVLOS: The Future of Commercial Drone Operations. *UAV Coach*, 2024. Last accessed on November 4, 2024.

[3] Andrea Albanese, Matteo Nardello, and Davide Brunelli. Low-power deep learning edge computing platform for resource constrained lightweight compact uavs. *Sustainable Computing: Informatics and Systems*, 34:100725, 2022.

[4] Alex Konrad. Drone Delivery Startup Zipline Boosts Evaluation to $4.2 Billion. *Forbes*, 2023. Last accessed on November 3, 2024.

[5] Alex Netto. How Are Drones Used for Infrastructure Inspection? *Skydio*, 2023. Last accessed on November 4, 2024.

[6] J. Aloimonos, I. Weiss, and A. Bandyopadhyay. Active vision. *International Journal of Computer Vision*, 1:333–356, 1988.

[7] Bilal Hazim Younus Alsalam, Kye Morton, Duncan Campbell, and Felipe Gonzalez. Autonomous uav with vision based on-board decision making for remote sensing and precision agriculture. In *2017 IEEE Aerospace Conference*, 2017. doi: 10.1109/AERO.2017.7943593.

[8] Anduril. Anduril Anvil. https://www.anduril.com/hardware/anvil/. Last accessed on November 7, 2024.

[9] Kiam Heong Ang, G. Chong, and Yun Li. Pid control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, 2005. doi: 10.1109/TCST.2005.847331.

[10] Anne Field. Zipline's Milestone: 1 Million Commercial Drone Deliveries. *Forbes*, 2024. Last accessed on November 4, 2024.

[11] Ludovic Apvrille, Tullio Tanzi, and Jean-Luc Dugelay. Autonomous drones for assisting rescue services within the context of natural disasters. In *2014 XXXIth URSI General Assembly and Scientific Symposium (URSI GASS)*, pages 1–4. IEEE, 2014.

[12] Ardupilot. Ardupilot. https://ardupilot.org/. Last accessed on November 3, 2024.

[13] Godwin Asaamoning, Paulo Mendes, Denis Rosário, and Eduardo Cerqueira. Drone swarms as networked control systems by integration of networking and computing. *Sensors*, 21(8):2642, 2021.

[14] BBC. How are 'kamikaze' drones being used by Russia and Ukraine? *BBC*, 2023. Last accessed on November 12, 2024.

[15] Lorenzo Bertizzolo, Salvatore D'Oro, Ludovico Ferranti, Leonardo Bonati, Emrecan Demirors, Zhangyu Guan, Tommaso Melodia, and Scott Pudlewski. Swarmcontrol: An automated distributed control framework for self-optimizing drone networks. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 1768–1777, 2020. doi: 10.1109/INFOCOM41043.2020.9155231.

[16] Juan A. Besada, Ana M. Bernardos, Luca Bergesio, Diego Vaquero, Iván Campaña, and José R. Casar. Drones-as-a-service: A management architecture to provide mission planning, resource brokerage and operation support for fleets of drones. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 931–936, 2019. doi: 10.1109/PERCOMW.2019.8730838.

[17] Bitcraze. Crazyflie 2.1. https://www.bitcraze.io/products/old-products/crazyflie-2-1/. Last accessed on December 7, 2024.

[18] Leslie M Blaha. Interactive OODA processes for operational joint human-machine intelligence. In *NATO IST-160 Specialist's Meeting: Big Data and Military Decision Making*, pages 3–1. NATO, 2018.

[19] BLS. A look at falls, slips, and trips in the construction industry. *Bureau of Labor Statistics*, 2024. Last accessed on November 11, 2024.

[20] 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, 2018.

[21] John R. Boyd. Patterns of Conflict. https://www.ausairpower.net/JRB/poc.pdf, 1986. Slides from a presentation by John R. Boyd, Last accessed October 18, 2024.

[22] Axel Bürkle and Sandro Leuchter. Development of micro uav swarms. In *Autonome Mobile Systeme 2009: 21. Fachgespräch Karlsruhe, 3./4. Dezember 2009*, pages 217–224. Springer, 2009.

[23] Axel Bürkle, Florian Segor, and Matthias Kollmann. Towards autonomous micro uav swarms. *Journal of intelligent & robotic systems*, 61:339–353, 2011.

[24] Cate Tarr. Defense tech start-up Anduril Industries raises $1.5 billion, now valued at $14 billion. *CNBC*, 2024. Last accessed on November 3, 2024.

[25] Batyr Charyyev, Engin Arslan, and Mehmet Hadi Gunes. Latency comparison of cloud datacenters and edge servers. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[26] Haowei Chen, Liekang Zeng, Xiaoxi Zhang, and Xu Chen. AdaDrone: Quality of Navigation Based Neural Adaptive Scheduling for Edge-Assisted Drones. In *Proceedings of 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, 2022.

[27] Jienan Chen, Siyu Chen, Siyu Luo, Qi Wang, Bin Cao, and Xiaoqian Li. An intelligent

task offloading algorithm (iTOA) for UAV edge computing network. *Digital Communications and Networks*, 6:433–443, 2020.

[28] Peng Chen, Yuanjie Dang, Ronghua Liang, Wei Zhu, and Xiaofei He. Real-time object tracking on a drone with multi-inertial sensing data. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):131–139, 2018. doi: 10.1109/TITS.2017.2750091.

[29] Eric Chung, Peter A. Milder, James C. Hoe, and Ken Mai. Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs? In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010.

[30] CloudFactory. Breaking Down The Levels of Drone Autonomy. https://www.cloudfactory.com/blog/levels-of-drone-autonomy. Last accessed on December 9, 2024.

[31] Cloudinary. Slice-Based Encoding. https://cloudinary.com/glossary/slice-based-encoding. Last accessed on December 11, 2024.

[32] Robert Collins. Stereo Algorithms. https://www.cse.psu.edu/ rtc12/CSE486/lecture09.pdf. Last accessed on November 19, 2024.

[33] Croptracker. Drone technology in agriculture. https://www.croptracker.com/blog/drone-technology-in-agriculture.html. Last accessed on October 29, 2024.

[34] Arne Devos, Emad Ebeid, and Poramate Manoonpong. Development of autonomous drones for adaptive obstacle avoidance in real world environments. In *2018 21st Euromicro conference on digital system design (DSD)*, pages 707–710. IEEE, 2018.

[35] DJI. DJI Avata 2. https://www.dji.com/avata-2, . Last accessed on December 19, 2024.

[36] DJI. DJI Matrice. https://enterprise.dronenerds.com/commercial-drone-platforms/matrice-series-2/matrice-600-series/, . Last accessed on November 3, 2024.

[37] DJI. DJI Mini 4 Pro. https://www.dji.com/mini-4-pro, . Last accessed on November 19, 2024.

[38] DJI. DJI Phantom 4 Pro V2.0. https://www.dji.com/phantom-4-pro-v2, . Last accessed on November 3, 2024.

[39] DJI. DJI Matrice 30 Series. https://enterprise.dji.com/matrice-30, . Last accessed on November 4, 2024.

[40] DJI. Robomaster S1. https://www.dji.com/robomaster-s1. Last Accessed November 14, 2022.

[41] Koustabh Dolui and Soumya Kanti Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[42] Sattar Dorafshan and Marc Maguire. Bridge inspection: Human performance, unmanned aerial systems and automation. *Journal of Civil Structural Health Monitoring*, 8:443–476, 2018.

[43] Dronecode. PX4. https://px4.io/. Last accessed on January 2, 2025.

[44] Dawei Du, Yuankai Qi, Hongyang Yu, Yifan Yang, Kaiwen Duan, Guorong Li, Weigang Zhang, Qingming Huang, and Qi Tian. The Unmanned Aerial Vehicle Benchmark: Object Detection and Tracking. https://arxiv.org/abs/1804.00518, 2018. Last accessed on July 1, 2024.

[45] Luke Eller, Théo Guérin, Baichuan Huang, Garrett Warren, Sophie Yang, Josh Roy, and Stefanie Tellex. Advanced autonomy on a low-cost educational drone platform. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1032–1039, 2019. doi: 10.1109/IROS40897.2019.8967595.

[46] Environmental Protection Agency. Self-driving Vehicles. https://www.epa.gov/greenvehicles/self-driving-vehicles. Last accessed on December 9, 2024.

[47] European Aviation Safety Agency. Open Category - Low Risk - Civil Drones. https://www.easa.europa.eu/en/domains/drones-air-mobility/operating-drone/open-category-low-risk-civil-drones. Last accessed on November 1, 2024.

[48] European Telecommunications Standards Institute. 4th Generation (LTE). https://www.etsi.org/technologies/mobile/4G. Last accessed on December 10, 2024.

[49] Federal Aquisitions Regulatory Council. Definitions. https://www.acquisition.gov/far/2.101. Last accessed on December 10, 2024.

[50] Federal Aviation Admin. Operations Over People General Overview. https://www.faa.gov/uas/commercial_operators/operations_over_people, January 2021.

[51] Federal Communications Commission. Mobile LTE Coverage Map. https://www.fcc.gov/BroadbandData/MobileMaps/mobile-map. Last accessed on December 11, 2024.

[52] FlexAir. Drone regulation history in the us: A comprehensive overview. https://www.flexairco.com/news/2023/flex-air-news/drone-regulation-history-in-the-us-a-comprehensive-overview/. Last accessed on October 29, 2024.

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

[54] Flytbase. Scaling BVLOS Operations through Robust Connectivity Autonomy Infrastructure. https://www.flytbase.com/blog/bvlos-drone-operations. Last accessed on December 19, 2024.

[55] G.H. Forman and J. Zahorjan. The challenges of mobile computing. *Computer*, 27(4): 38–47, 1994. doi: 10.1109/2.274999.

[56] Linux Foundation. Ubuntu. https://ubuntu.com/. Last accessed on April 21, 2025.

[57] Shilpa George, Junjue Wang, Mihir Bala, Thomas Eiszler, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards drone-sourced live video analytics for the construction industry. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, pages 3–8, 2019.

[58] Mirmojtaba Gharibi, Raouf Boutaba, and Steven L Waslander. Internet of drones. *IEEE Access*, 4:1148–1162, 2016.

[59] Razan Ghzouli, Thorsten Berger, Einar Broch Johnsen, Andrzej Wasowski, and Swaib Dragule. Behavior trees and state machines in robotics applications. *IEEE Transactions on Software Engineering*, 49(9):4243–4267, 2023. doi: 10.1109/TSE.2023.3269081.

[60] Grandview Research. Commercial Drone Market Size and Outlook. https://www.grandviewresearch.com/horizon/outlook/commercial-drone-market-size/global. Last accessed on November 3, 2024.

[61] Kiryong Ha, Padmanabhan Pillai, Wolfgang Richter, Yoshihisa Abe, and Mahadev Satyanarayanan. Just-in-time provisioning for cyber foraging. pages 153–166, 06 2013. doi: 10.1145/2462456.2464451.

[62] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards wearable cognitive assistance. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, page 68–81, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327930. doi: 10.1145/2594368.2594383. URL `https://doi.org/10.1145/2594368.2594383`.

[63] Andy Hardy, Makame Makame, Dónall Cross, Silas Majambere, and Mwinyi Msellem. Using low-cost drones to map malaria vector habitats. *Parasites & vectors*, 10:1–13, 2017.

[64] Samira Hayat, Roland Jung, Hermann Hellwagner, Christian Bettstetter, Driton Emini, and Dominik Schnieders. Edge computing in 5g for drone navigation: What to offload? *IEEE Robotics and Automation Letters*, 6(2), 2021. doi: 10.1109/LRA.2021.3062319.

[65] Songtao He, Favyen Bastani, Arjun Balasingam, Karthik Gopalakrishna, Ziwen Jiang, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. Beecluster: drone orchestration via predictive optimization. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 299–311, 2020.

[66] Mark Hepokoski, Allen Curran, Timothy Viola, and Alex Ockfen. Thermal acceptability limits for wearable electronic devices. In *2021 37th Semiconductor Thermal Measurement, Modeling Management Symposium (SEMI-THERM)*, pages 16–19, 2021.

[67] Yunxiang Hu, Yuhao Liu, and Zhuoyuan Liu. A survey on convolutional neural network accelerators: Gpu, fpga and asic. In *2022 14th International Conference on Computer Research and Development (ICCRD)*, pages 100–107, 2022. doi: 10.1109/ICCRD54409.2022.9730377.

[68] Hui-Min Huang. Autonomy Levels for Unmanned Systems (ALFUS) Framework, Volume I: Terminology, Version 2.0. Technical Report NIST Special Publication 1011-I-2.0, National Institute of Standards and Technology, October 2008.

[69] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In *Proceedings*

of IEEE Computer Vision and Pattern Recognition (CVPR), 2017.

[70] Imperial War Museum. A Brief History of Drones. https://www.iwm.org.uk/history/a-brief-history-of-drones. Last accessed on November 3, 2024.

[71] Roger Iyengar, Qifei Dong, Chanh Nguyen, Padmanabhan Pillai, and Mahadev Satyanarayanan. Offload shaping for wearable cognitive assistance. In *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, pages 183–189, 2023. doi: 10.1109/EDGE60047.2023.00037.

[72] Jack Detsch. Ukraine's Cheap Drones Are Decimating Russia's Tanks. *Foreign Policy*, 2024. Last accessed on November 12, 2024.

[73] Joe Saballa. US Army Testing Pocket-Sized Drones for Covert Reconnaissance. *Defense Post*, 2024. Last accessed on November 12, 2024.

[74] James Johnson. Automating the OODA loop in the age of intelligent machines: reaffirming the role of humans in command-and-control decision-making in the digital age. *Defence Studies*, 23(1):43–67, 2023.

[75] JOUAV. A Complete Guide to Inertial Measurement Unit (IMU). https://www.jouav.com/blog/inertial-measurement-unit.html. Last accessed on November 19, 2024.

[76] I. Kalra, M. Singh, S. Nagpal, R. Singh, M. Vatsa, and P. B. Sujit. DroneSURF: Benchmark Dataset for Drone-based Face Recognition. In *Proceedings of the 14th IEEE International Conference on Automatic Face & Gesture Recognition*, Lille, France, 2019. doi: 10.1109/FG.2019.8756593.

[77] Anis Koubaa, Adel Ammar, Mahmoud Alahdab, Anas Kanhouch, and Ahmad Taher Azar. DeepBrain: Experimental Evaluation of Cloud-Based Computation Offloading and Edge Computing in the Internet-of-Drones for Deep Learning Applications. *Sensors*, 20(18), 2020. doi: 10.3390/s20185240. https://www.mdpi.com/1424-8220/20/18/5240.

[78] Lauren Nagle. What is an Electronic Speed Controller How Does an ESC Work. *TYTO Robotics*, 2023. Last accessed on November 19, 2024.

[79] 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. *Robotics and Autonomous Systems*, 133:103621, 2020.

[80] Siyi Li and Dit-Yan Yeung. Visual object tracking for unmanned aerial vehicles: A benchmark and new motion models. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2017.

[81] Linux. ioctl(2) - linux manual page. https://man7.org/linux/man-pages/man2/ioctl.2.html. Last accessed pn February 11, 2025.

[82] Matthew Urwin. Is Drone Delivery on the Horizon? https://builtin.com/articles/drone-delivery. Last accessed on November 4, 2024.

[83] MAVLink. MAVLink Developer Guide. https://mavlink.io/en/. Last accessed on December 11, 2024.

[84] Microsoft. Real-Time Streaming Protocol. https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rtsp/d9c6f2be-6778-4531-971d-2d295b568d52. Last accessed on December 10, 2024.

[85] MIT. BeeCluster. http://beecluster.csail.mit.edu/. Last accessed on December 7, 2024.

[86] ModalAI. Starling 2. https://www.modalai.com/products/starling-2, . Last accessed on December 7, 2024.

[87] ModalAI. Starling 2 max. https://www.modalai.com/products/starling-2-max?variant=48172375310640, . Last accessed pn April 21, 2025.

[88] ModalAI. Voxl 2. https://www.modalai.com/products/voxl-2?variant=39914779836467, . Last accessed pn April 21, 2025.

[89] ModalAI. Voxl sdk. https://docs.modalai.com/voxl-sdk/, . Last accessed pn February 11, 2025.

[90] Luca Mottola, Mattia Moretta, Kamin Whitehouse, and Carlo Ghezzi. Team-level programming of drone sensor networks. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 177–190, 2014.

[91] Hanna Müller, Vlad Niculescu, Tommaso Polonelli, Michele Magno, and Luca Benini. Robust and efficient depth-based obstacle avoidance for autonomous miniaturized uavs. *IEEE Transactions on Robotics*, 2023.

[92] MyFPV. iFlight Cidora. https://www.myfpvstore.com/ready-to-fly-drones/plug-n-play/iflight-cidora-sl5-6s-fpv-drone-pnp/. Last accessed on November 4, 2024.

[93] Dimitri Ognibene and Yiannis Demiris. Towards active event recognition. In *The 23rd International Joint Conference on Artificial Intelligence, IJCAI-13*, August 2013.

[94] Onion. Onion omega 2 lte. https://onion.io/store/omega2-lte-na/. Last accessed pn January 10, 2025.

[95] OpenVINO. ssd_resnet50_v1_fpn_coco. `https://docs.openvino.ai/2021.1/omz_models_public_ssd_resnet50_v1_fpn_coco_ssd_resnet50_v1_fpn_coco.html`. Last Accessed on November 15, 2022.

[96] Jonathan Tersur Orasugh and Suprakas Sinha Ray. Functional and structural facts of effective electromagnetic interference shielding materials: A review. *ACS Omega*, 8(9): 8134–8158, 2023. doi: 10.1021/acsomega.2c05815. URL `https://doi.org/10.1021/acsomega.2c05815`.

[97] Daniele Palossi, Francesco Conti, and Luca Benini. An open source and open hardware deep learning-powered visual navigation engine for autonomous nano-uavs. In *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 604–611. IEEE, 2019.

[98] Daniele Palossi, Nicky Zimmerman, Alessio Burrello, Francesco Conti, Hanna Müller, Luca Maria Gambardella, Luca Benini, Alessandro Giusti, and Jérôme Guzzi. Fully onboard ai-powered human-drone pose estimation on ultralow-power autonomous flying nano-uavs. *IEEE Internet of Things Journal*, 9(3):1913–1929, 2021.

[99] PAPA. History of Aerial Photography. *Professional Aerial Photographers Association*, 2024. Last accessed on November 11, 2024.

[100] Parrot. PDrAW – Parrot Drones Awesome Video Viewer. https://github.com/Parrot-Developers/pdraw. Last accessed June 19, 2024.

[101] Parrot. Parrot Anafi. https://www.parrot.com/us/drones/anafi. Last accessed on November 4, 2024.

[102] Protobuf. Protocol buffers documentation. https://protobuf.dev/. Last accessed pn February 11, 2025.

[103] Han Qi and Abdullah Gani. Research on mobile cloud computing: Review, trend and perspectives. In *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, pages 195–202, 2012. doi: 10.1109/DICTAP.2012.6215350.

[104] Rene Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-Shot Cross-Dataset Transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3):1623–1637, 2022.

[105] Raspberry Pi. Raspberry pi. https://www.raspberrypi.com/. Last accessed pn January 10, 2025.

[106] Rhett Allain. How Do Drones Fly? Physics, of Course! *WIRED*, 2017. Last accessed on November 19, 2024.

[107] Mahadev Satyanarayanan. The Emergence of Edge Computing. *IEEE Computer*, 50(1), 2017.

[108] D. C. Schedl, I. Kurmi, and O. Bimber. An autonomous drone for search and rescue in forests using airborne optical sectioning. *Science Robotics*, 6(55):eabg1188, 2021. doi: 10.1126/scirobotics.abg1188.

[109] Korbinian Schmid, Philipp Lutz, Teodor Tomić, Elmar Mair, and Heiko Hirschmüller. Autonomous vision-based micro air vehicle for indoor and outdoor navigation. *Journal of Field Robotics*, 31(4):537–570, 2014.

[110] Skydio. Skydio 2 Plus. https://www.skydio.com/skydio-2-plus-enterprise, . Last accessed on November 3, 2024.

[111] Skydio. All About Skydio 2+ for Enterprise. https://www.skydio.com/blog/skydio-2-plus-enterprise-drone, . Last accessed on December 19, 2024.

[112] Skydio. Skydio X10. https://www.skydio.com/x10, . Last accessed on December 19, 2024.

[113] Lars Yndal Sørensen, Lars Toft Jacobsen, and John Paulin Hansen. Low cost and flexible uav deployment of sensors. *Sensors*, 17(1):154, 2017.

[114] Stacie Pettyjohn. Drones Are Transforming the Battlefield in Ukraine but in an Evolutionary Fashion. *Center for a New American Security*, 2024. Last accessed on November 3, 2024.

[115] Thomas Pledger. The Role of Drones in Future Terrorist Attacks. *Association of the United States Army*, 2021. Last accessed on November 12, 2024.

[116] Eric Tilley and Jeff Gray. Dronely: A visual block programming language for the control of drones. In *Proceedings of the SouthEast Conference*, pages 208–211, 2017.

[117] UAV Coach. Drone Laws in China. https://uavcoach.com/drone-laws-in-china/, . Last accessed on November 1, 2024.

[118] UAV Coach. Drone Laws in India. https://uavcoach.com/drone-laws-in-india/, . Last accessed on November 1, 2024.

[119] US Air Force. RQ-4 Global Hawk. https://www.af.mil/About-Us/Fact-Sheets/Display/Article/104516/rq-4-global-hawk/. Last accessed on November 3, 2024.

[120] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

[121] Jingjing Wang, Chunxiao Jiang, Zhu Han, Yong Ren, Robert G. Maunder, and Lajos Hanzo. Taking drones to the next level: Cooperative distributed unmanned-aerial-vehicular networks for small and mini drones. *IEEE Vehicular Technology Magazine*, 12 (3):73–82, 2017. doi: 10.1109/MVT.2016.2645481.

[122] Rui Wang, Yong Cao, Adeeb Noor, Thamer A Alamoudi, and Redhwan Nour. Agent-enabled task offloading in UAV-aided mobile edge computing. *Computer Communications*, 149:324–331, 2020.

[123] Sean Ward, Jordon Hensler, Bilal Alsalam, and Luis Felipe Gonzalez. Autonomous uavs wildlife detection using thermal imaging, predictive navigation and computer vision. In *2016 IEEE Aerospace Conference*, 2016. doi: 10.1109/AERO.2016.7500671.

[124] Wikipedia. Video compression picture types. https://tinyurl.com/2s8rkshz. Last accessed on March 12, 2025.

[125] Gaoxiang Wu, Yiming Miao, Yu Zhang, and Ahmed Barnawi. Energy efficient for UAV-enabled mobile edge computing networks: Intelligent task prediction and offloading. *Computer Communications*, 150:556–562, 2020.

[126] Yuwei Wu, Ziming Ding, Chao Xu, and Fei Gao. External forces resilient safe motion planning for quadrotor. *IEEE Robotics and Automation Letters*, 6(4):8506–8513, 2021.

[127] Wyatt Olson. Drones, new squad vehicle are key to 'hunter-killer' Army platoons. *Stars and Stripes*, 2024. Last accessed on November 12, 2024.

[128] YOLO. Yolov5. https://github.com/ultralytics/yolov5. Last accessed on July 1, 2024.

[129] Shu-Ang Yu, Chao Yu, Feng Gao, Yi Wu, and Yu Wang. Flightbench: A comprehensive benchmark of spatial planning methods for quadrotors. *arXiv preprint arXiv:2406.05687*, 2024.

[130] Zacc Dukowitz. Police Drones: A Guide to How Law Enforcement Uses Drones in Its Work. *UAVCoach*, 2024. Last accessed on November 11, 2024.

[131] Zacc Dukowitz. Search and Rescue Drones: A Guide to How SAR Teams Use Drones in Their Work. *UAVCoach*, 2024. Last accessed on November 4, 2024.

[132] ZeroMQ. ZeroMQ. https://zeromq.org/. Last accessed on December 27, 2024.

[133] Zhichao Zhang, M. A. Parvez Mahmud, and Abbas Z. Kouzani. Fitnn: A low-resource fpga-based cnn accelerator for drones. *IEEE Internet of Things Journal*, 9(21):21357–21369, 2022. doi: 10.1109/JIOT.2022.3179016.

[134] Xin Zhao, Shiyu Hu, Yipei Wang, Renshu Li, Kun Liu, Jiadong Li, Jing Zhang, Yimin Hu, Rongshuai Liu, Haibin Ling, and Yin Li. BioDrone: A Bionic Drone-Based Single Object Tracking Benchmark for Robust Vision. *International Journal of Computer Vision*, 132: 1659–1684, 2024.