Virtual Reality for Drones: Assessing Physical Actuation via Idealized Sensor Feedback

Hunter Rhoades

CMU-CS-26-116

May 2026

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

Thesis Committee:
Mahadev Satyanarayanan, Chair
Padmanabhan Pillai
Mihir Bala

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science.

Keywords: drones, autonomy, computer vision, agility, benchmarking, digital twin, sensor twin, simulation, artificial sensing, idealized sensing

Dedicated to the Father, my wife, and my family, for their unconditional love.

Abstract

Computer vision-enabled autonomous drone flight creates challenges with reproducibility and behavior assessment due to inherent variability in the models that provide cognitive function to the vehicle. These challenges are most visible when attempting to validate control logic in the automation pipeline and when benchmarking for the purpose of making objective assessments on agility and maneuverability. In this work we begin by exploring the current state of simulation technology and its particular applications in the development and fielding of autonomous drone systems. This exploration exposes a sudden jump from simulation-heavy prototyping to end-to-end full system testing involving physical flight that increases the difficulty of validating correctness and incurs unnecessary risk.

We present the concept of a SensorTwin, a novel application of simulation technology aimed at facilitating behavior validation and performance benchmarking for vehicle-automation pipeline systems. We provide a concrete implementation of a SensorTwin built over the SteelEagle Autonomous Drone system and demonstrate proof of concept by evaluating the performance of the SensorTwin-augmented system against existing cognitive task evaluations. We show how idealizing the visual feed and computer vision components of a real system contributes to both benchmarking and validation when applied to a system during actual, physical flight.

We also introduce a novel set of tracking agility evaluation tests, referred to as the Autonomous Tracking Agility Benchmark, with the intent of enabling future relative comparison across vehicle platforms, autonomy pipelines, and cognitive function implementations. An initial dataset for the benchmark suite is produced through physical flight augmented by the SensorTwin implementation and analyzed to provide objective and subjective insights on agility assessment specific to the automated drone setting.

SensorTwin Demo Video: https://drive.google.com/drive/folders/1ep9udk48KGw6JmN3L3ZwftNdeGHhQ-bR?usp=sharing

Acknowledgements

Thank you to my committee, Dr. Mahadev Satyanarayanan, Dr. Padmanabhan Pillai, and Dr. Mihir Bala, whose guidance and patience, both on this endeavor and during my larger time as part of the lab, has been immensely formative. Special thanks are owed to Thomas Eislzer Jr. for his mentorship during my time in the lab, and his encouraging company and capable assistance during many hours of drone flight. Additional thanks are given to Aditya Chanana and Xiangliang Chen for their help throughout the process.

For my wife, Rachel, your love is a blessing and words can never communicate the depth of my gratitude for your support in this season of our lives. I am thankful for the endless encouragement and wisdom of my parents, Frank and Jane Rhoades, without whom I would never have gotten to this point. I am also thankful for the sound advice provided over the duration of my time at this institution by my academic advisor, Dr. David Eckhardt.

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 of the above funding sources.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

"agile: marked by ready ability to move with quick easy grace." – Webster's Dictionary [1]

Maneuvering capability represents a clear and common limiter on what may be accomplished across the manifold applications of unmanned drone systems. Historic assessments of aerial platforms focus heavily on agility metrics that measure the extent of what hardware under direct human control can accomplish. While the raw hardware capabilities of drones remain important for defining the theoretic space of what a given vehicle can feasibly accomplish, active human piloting is itself limited by the span of control and cognitive load incurred when manually operating a drone. The introduction of intelligence to unmanned platforms represents an effort to expand beyond such boundaries by offloading cognitive demand to automated platforms that require reduced human supervision. This stride incurs its own complications, however. While the limitations of a platform operated by a human pilot in real-time are relatively clear, the same claim cannot be made for platforms operated entirely via autonomy pipelines. Unlike skilled pilots who can interpret ambiguous imagery and sensor feedback and intuitively respond with an appropriate maneuver with the appropriate proportion of the controlled platform's actuation capabilities, autonomous pipelines respond within the parameters of their in-built control logic.

Computer vision, driven by advances in the AI/ML world, makes more complex autonomy possible. But in attempting to complete complex tasks without the emergency switch of real-time human intervention, several significant concerns emerge. Do the autonomy pipeline and underlying models have the capability to sense and process sufficient pertinent data to enable automated decision-making? Given sufficient information is the automated decision-making both correct and appropriately scaled? Having converted real or near-real-time sensor data into an actionable decision or requirement, can the drone's hardware produce the flight characteristics required to carry out the directive given by the autonomy pipeline? In the best case if the answer to any of these questions is "no," the platform fails the assigned task. In the worst case, a vehicle collides with another entity, possibly another aircraft, a structure, or a person. This creates a very real onus on the developers of such systems to verify that they perform safely across a range of systems, itself a non-trivial challenge.

Part of this verification on autonomous drones includes the testing of control behavior while executing cognitive functions. In real-world systems, statistical modeling based on image capture that drives many of the cognitive functions used in automation produces inherently non-deterministic results. Though architecture and available compute resources place real constraints on the quality of model available to an autonomous vehicle, a number of challenges are consistent regardless of price point and system design philosophy. Compared to the text-based input most people use to interact with foundation models today, images are complex and

computers struggle with tasks that humans perform subconsciously, like identifying figures and objects, at different distances under different lighting conditions and from different angles. Even the relative location of an object within an image, centered, in a corner, out of focus, can create problems that result in totally different model outputs if not properly accounted for during model training. These problems are further compounded by the potential for artifacts, anomalies resulting from issues in processing and decoding that distort or mar a picture, that humans easily ignore. Naturally then it comes as no surprise that computer vision systems experience high variability when operating outside of a controlled environment (and sometimes, even within one). Flight behavior and task performance quality on the same vehicle, running the same model, performed in the same location but under different weather and lighting conditions can demonstrate significant non-determinism, making reproducible assessment difficult.

While eliminating such variability during real operation is infeasible, removing such problems from the verification equation during the development and assessment processes would reduce the complexity of confirming correct control behavior and performing relative comparison. Any such attempt would require a foundation in reproducibility and determinism, suggesting potential for application of simulation to the problem space. As will be discussed later, the body of work surrounding application of digital simulation to physical system development and optimization provides an excellent step-off point from which to attempt addressing the above-described problems.

1.2 Overview

This thesis addresses the points motivated above via the application of simulation technology in a novel fashion. Concretely, we seek to remove, or at least heavily reduce, the variability and non-determinism introduced by vision models when in physical operating scenarios short of actual real-world deployment by an intended end-user. Such a result would simplify the problem of performing accurate relative comparison across a number of system variables, such as vehicle model, autopilot type, or even configurable actuation settings, while also easing the problem of inducing desired behaviors on a physical, automated drone to verify control logic correctness and perform response tuning. We begin with a discussion of the current spaces surrounding simulation and drone automation to expose the gap in simulation application present between early development and production-grade fully fielded systems.

After defining the target problem space, we introduce the concept of a SensorTwin, a novel derivation from existing simulation concepts that isolates a physical drone's actuation characteristics and its supporting automation pipeline's actuation directives with real-time purely simulated sensor data. SensorTwin, however, goes beyond simply interposing on a vehicle's imagery stream by also targeting the vision models that enable complex autonomous function. SensorTwin combines the simulated inputs with idealized variants of vision models to produce deterministic, idealized results that enable the system to respond with effectively perfect knowledge about the environment it is observing and operating in. This deterministic but realistic analog provides immense value in the development and debugging of higher-level mission and control logic, especially in cases that are too difficult, too dangerous, or too rare to be recreated in the real world. The proposal raises the obvious challenge of synchronization: if a physical entity is to respond in a useful way to simulated feedback, the production of said simulated feedback must be dependent on the state of the physical entity in real-time. We solve this with a

concrete implementation of a physical-virtual binding that facilitates a bi-directional flow of data. This data flow maintains the fidelity of the simulated space while simultaneously producing output that drives accurate actuation of the physical vehicle in support of an automated task.

Having created a functional SensorTwin, we show proof of concept by recreating a set of experiments performed by Bala et al. [2], holding constant the vehicle but substituting out real vision models and physical target objects via SensorTwin, and providing a relative comparison of results produced. With viability established, we show a practical application by introducing a suite of agility tests, the Autonomous Tracking Agility Benchmark. Using flight data collected by operating an autonomous drone augmented with SensorTwin, we analyze the performance of the autonomous system on the suite and provide the performance results to create a foundation for future relative comparison across the system variables mentioned at the beginning of this section.

1.3 Contributions

As part of this work, we make the following contributions:

- Introduction of the SensorTwin concept, a modification of existing digital twin concepts that reduces variability from sensor input and computer vision output
- Implementation of a concrete SensorTwin proof of concept that addresses physical-digital synchronization to enable accurate physical actuation based on simulation space state
- A discussion on agility in the autonomous drone setting consisting of:
    - Recreation of and relative comparison against results produced by Bala et al. [2] using a drone augmented with SensorTwin
    - Development of the Autonomous Tracking Agility Benchmark, a suite of tests evaluating tracking agility characteristics, and initial results for the suite

1.4 Organization

Chapter 2 provides primers and discussions for background and related work. We briefly examine drone autonomy and how computer vision contributes to developing increasingly capable mechanisms, then discuss prevalent simulation techniques and their usage in automated drone development. We provide a short overview of SteelEagle, an autonomous framework used as an enabler for this work, then complete the chapter with a survey of related efforts. Chapter 3 explains each of the implementation artifacts and their individual contributions to the above-described motivations. Chapter 4 begins with an attempt to convert the nebulous concept of drone agility into something more quantifiable by defining a set of measurement criteria. We then demonstrate proof-of-concept of the SensorTwin idea by benchmarking our implementation against the work of Bala et al. [2]. Chapter 5 continues the agility performance thread by introducing a novel tracking agility benchmarking suite. We explain the traces contained within the overall benchmark suite and their individual relations to autonomous platform agility. Finally, we provide a set of performance measurements achieved by our testing platform on the introduced suite. The conclusion is presented in Chapter 6 and includes an outline for future works that extend the effort of assessing drone agility and furthering drone autonomy pipelines.

Chapter 2

Background & Related Work

2.1 Drone Automation

Early definitions of drone automation established single-axis spectrums, categorized based upon the level of human involvement required for operation [3]. More recent attempts to stratify the concept of drone autonomy introduce multiple spectrums representing different operational phases [4] or consideration factors [5]. The ALFUS work released by NIST in 2008 defines autonomous flight as "pre-programmed flight without a remote human pilot, including mission specific actions in response to runtime observations" [5]. Current FAA regulations heavily restrict the autonomous operation of drones and were designed primarily for human-in-the-loop operation, placing constraints on operational range and simultaneous vehicle operation while requiring a human pilot be readily available and able to take manual remote control of the vehicle [6]. Drafted regulatory changes suggest a near-term loosening of some of these requirements while imposing technical requirements on systems used for Beyond Visual Line-of-Sight (BVLOS) operations, with the proposed requirement for detect-and-avoid systems of particular relevance [7].

Computer vision dramatically increases the level of autonomy available to a platform, with sophisticated functions such as detection and avoidance benefitting heavily from active vision. However, these higher-level functions condition on first having access to autonomous actuation of the vehicle in question. An autopilot stack, such as PX4, ArduPilot, or Betaflight handles both moment-to-moment profile stability and pitch, roll, yaw, and elevation control when directed to move [8]. To achieve both functions, the autopilot's Proportional-Integral-Derivative (PID) controller converts actuation demands into physical motor rates. This component is configurable and tuned based on maximal values and gain rates. These values determine how much and how quickly a vehicle ramps velocity – rotational, lateral, or vertical – in response to actuation input. The goal of tuning then is to produce a system that feels responsive but avoids overreacting based on the magnitude of input given.

Modern computer vision leverages trained statistical models to perform semantic tasks, like detection or avoidance, given image-based input. In the automated drone context, a platform's cameras, often in the visible spectrum, are the simplest and most common producer of input for supporting models. Higher-end or specially tailored platforms may include additional, more capable sensor types. Generically, autonomous vehicles feed vision models with data collected by their on-board sensing and imaging suite and receive in return actuation directives that shape the behavior of the vehicle without need for active human piloting. Figure 2.1 below depicts an example flow of what a visual avoidance task looks like at each stage of the pipeline.

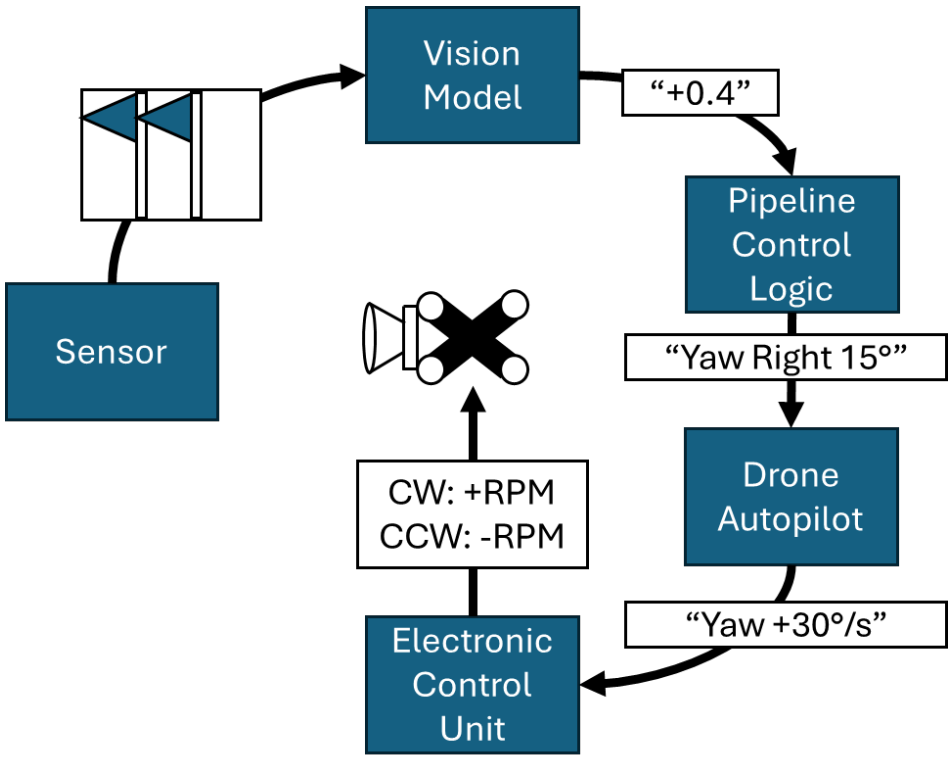


Figure 2.1. Autonomous Actuation Cycle

In a detection task, a vision model analyzes an image for particular objects and their relative locations within the frame. The quality of model training dramatically impacts the performance of the final object detection engine during operation and influences how susceptible the engine is to the conditional variances mentioned in Section 1.1. Object tracking requires actuating to keep a detected object in frame, thus variance in a detection model directly impacts automated capability. In an avoidance task, a vision model attempts to perceive the surrounding environment based on available sensor data. Sensors like radar, LiDAR, and stereo cameras produce absolute distance estimates and can employ more sophisticated algorithms. Smaller, cheaper drones (like the ones utilized in this work) frequently lack such capabilities though, constraining them to a monocular vision-based approach. When limited to a monocular camera, depth estimation becomes the central (difficult) problem and severely constrains the magnitude of actuation considered safe.

2.2 Applied Simulation & Autonomous Drone Development

Traditional software engineering involves layers of abstraction that enable software developers to ignore most hardware concerns, confident that hardware will "just work" for any written software. Embedded systems demand greater consideration of hardware capabilities, limitations, and control mechanisms to produce useful physical actuation. However, hardware is slower to

modify and arduous to integrate compared to entirely software components, motivating multiple forms of simulation application.

2.2.1 Software-in-the-Loop Testing

Software-in-the-loop (SIL or SITL) testing defers hardware integration in favor of a fully simulated environment. Typical SITL testing may target an entire system or only certain subsets but is chiefly characterized by the complete absence of real hardware [9]. Even rudimentary digital implementations can provide enormous value early in the development process of a larger project because of the error-prone nature of software development. Assuming a minimal degree of simulator fidelity, control algorithms that fail to behave in simulation assuredly will not function properly in the physical world. SITL testing facilitates development in several ways:

- Test setup and reset time is reduced, taking only as long as simulation startup.
- Control software becomes testable before hardware prototypes exist.
- Simulation time can be scaled faster or slower because every component is virtualized.

2.2.2 Hardware-in-the-Loop Testing

Hardware-in-the-loop (HIL or HITL) testing addresses the intermediary step by introducing individual hardware subcomponents in lieu of jumping directly to full physical end-to-end testing. In HIL testing, a physical component's control unit connects to a simulation of the larger system, called the plant. The control unit pushes signals to the plant which emulates state changes, driving further signals from the tested hardware off the simulated state and control logic in place. [9] HIL testing has two primary benefits:

- Problems not apparent in simulation can be tested for and corrected without incurring significant risk to the overall system.
- (1) can begin as soon as physical control units are available – there is no requirement to wait for the entire, potentially costly, hardware system.

Importantly, HIL testing primarily isolates singular control units and evaluates their behavior in response to controlled simulation state changes. It is not full physical testing, and only rarely are multiple hardware systems tested simultaneously due to the difficulties in synchronization and the increase in pre-requisite simulation complexity [9].

2.2.3 Digital Twins

Unlike SITL and HIL testing, digital twins support fielded systems. At their core, digital twins are complex models of physical systems – such as a full wind turbine field, including every system within each individual turbine and the network between them [10]. They require a real-time data stream piped from their physical counterpart. Aggregate analysis of this data enables

digital exploration, real-time optimization, and failure prediction without having to experiment on physical entities empirically. Siemens Electronic Works Amberg (EWA), a highly automated modern electronics factory, demonstrated the potential in digital twins by cutting electronic controller production cycle times. They first used the twin to identify a series of bottlenecks spanning multiple production lines and then evaluated alternative setups in the digital twin before reworking the physical companion to match [11]. Similarly in spirit to SITL and HIL testing, digital twins exist on the premise that making exploratory changes in simulated systems are safer, cheaper, and faster than making changes in physical ones. Due to their innate complexity however, digital twins are difficult to develop and field, a point we will revisit in Section 2.4.

2.2.4 Simulation in Development Pipelines

While current regulations are fairly prohibitive of autonomous operation, waivers of certain segments of the regulation may be approved if would-be autonomous vehicle operators can demonstrate sufficient mitigation of risk. Because drones pose a risk to other aircraft in the airspace and people underneath, successful waiver applications detail control, avoidance, fault recovery, and safe flight termination mechanisms. The technical complexity involved with producing and validating these functions means phased-testing is essentially required.

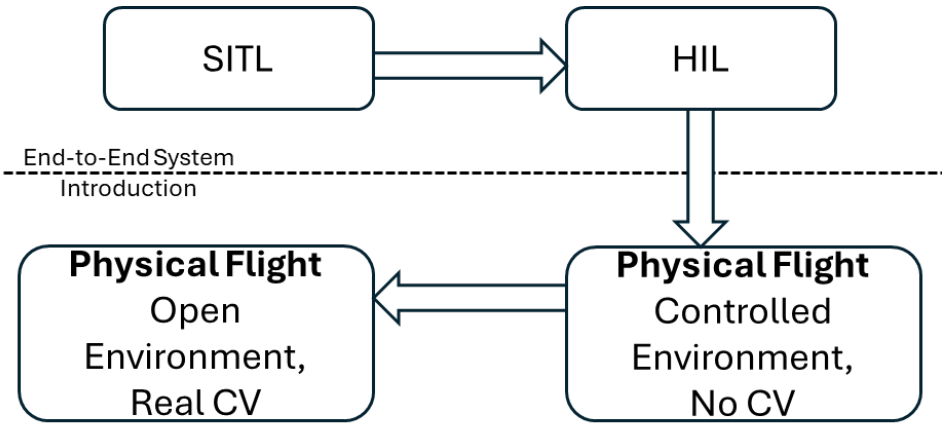


Figure 2.2. Automated Drone Development Pipeline [9]

Jiang et al. detail the industry-standard progression for automated drone development shown in Figure 2.2 above [9]. SITL testing validates core modules and software prototypes before HIL testing validates fault detection and recovery mechanisms on individual components. Jiang et al. also identify a critical gap between the capabilities of traditional simulation tools and the data requirements of modern automation pipelines. This gap becomes obvious in the transition from HIL testing to physical flight in a controlled environment, where end-to-end systems with real vision models are first deployed. Jiang et al. observe the potential in using synthetic data to evaluate system performance on edge cases and low probability events but do so only in the context of digital twins. In Chapter 4, we explore how synthetic data generated in

real-time enables the pairing of a physical vehicle with idealized vision models to remove variability from the automation pipeline when real computer vision function is not required.

2.3 SteelEagle Autonomous Drone Framework

SteelEagle [12, 13] is a mission-centric drone automation framework that seeks to achieve platform agnosticism by providing a unified control framework. Core to the design of SteelEagle is the avoidance of assumptions about supported platform capabilities. Aimed primarily at commercial-off-the-shelf (COTS) ultra-light drones, the system is architected to convert minimally capable vehicles into fully autonomous platforms capable of reducing cognitive burden on their operators while bringing a level of intelligence and sophistication totally at odds with their price point. To this end, SteelEagle champions the idea of offloading to the edge via linkage to a cloudlet, a small, relatively portable, server cluster, operating sufficiently powerful computer vision models to provide small, cheap, expendable drones the faculties required for truly autonomous flight.

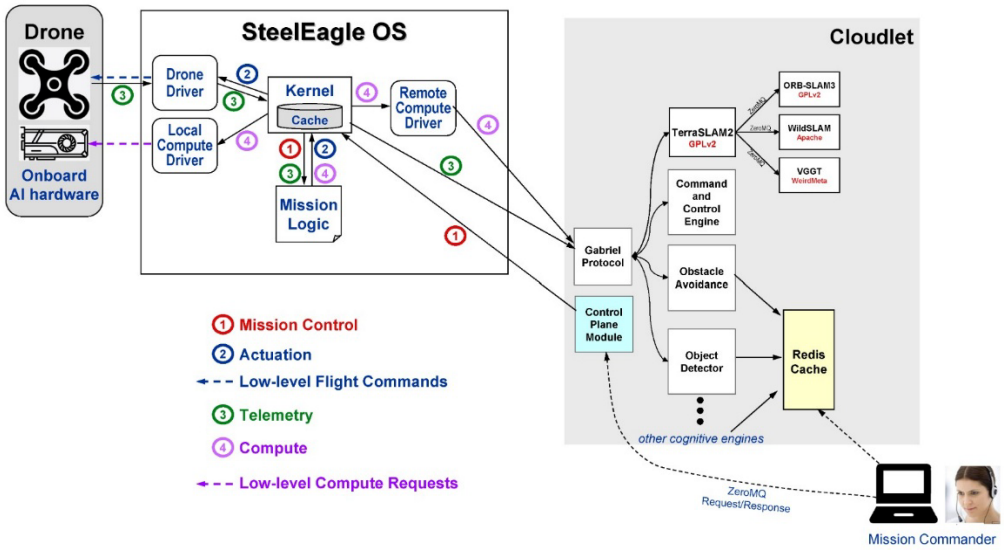


Figure 2.3. SteelEagle Autonomous Framework Architecture [13]

The framework, depicted by Figure 2.3, is implemented in three-tiers: drone-side, SteelEagle OS, and backend. Drone-side covers the vehicle hardware and sensors that ship with the drone or are fastened on after-market. SteelEagle OS interfaces directly with the drone's native control system and sensor suite and wraps the hardware with a standardized software control interface. It includes the drone driver – the control interface described above – the kernel, which provides a common communication interface for both data and control planes, and a mission controller that enacts semantic tasks through the driver. The backend is a cloudlet machine that houses the vision models, which the system refers to as cognitive engines, that perform object detection, object avoidance, and other higher-level semantic tasks.

Discussed in the next chapter, the architecture of SteelEagle lends perfectly to the introduction of a SensorTwin. Because vision model result production is an opaque process to

the vehicle, a SensorTwin process generating artificial results is indistinguishable from a live vision model inferencing on the drone's sensor data or image stream. Synchronizing a SensorTwin's artificially generated computational results with the live data stream of an active drone poses a challenge, but one which we demonstrate is not insurmountable.

2.4 Related Work

While the field examining agility or maneuverability of human-operated rotary platforms (both manned helicopters and remote piloted UAVs of the style used in this work) [14, 15], exploration into the topic of drone agility specifically in an automated setting is limited. 2017 work by Fotouhi et al. [16] examines a series of metrics relevant to automated operation within an urban context in support of Internet-of-Things (IOT) application aerial relays. Work in 2023 [17] and 2024 [2] by Bala et al. explores active vision-based autonomous flight and discuss in-depth actuation requirements and their implications on agility. In particular, their 2024 work, *The OODA Loop of Cloudlet-based Autonomous Drones*, is foundational to this endeavor, providing the experiment design and results for comparative analysis used as the basis for the SensorTwin proof of concept covered in Section 4.2.

Literature for drone-centric digital twin systems is predominantly conceptual, discussing desired properties or behaviors [18] and suggested architectures [18, 19]. Despite this, several concrete implementations of varied complexity exist. Grigoropoulos and Lalis produce a digital twin that functions in both SITL and HIL testing setups and facilitates detection of hardware performance irregularities. In their design, the digital companion operates with a several second lag period behind the physical vehicle to avoid synchronization issues from network jitter [20]. For the context of identifying faulty component behavior this delay is sensible, but because we intend to drive real-time actuation of the physical vehicle from the simulated space, we are forced to synchronize the physical and digital entities as close to real-time as possible. The work of McClellan et al. in the space of fixed-wing autonomy details a layered double-digital-twin system that uses a lower-level digital twin to provide real-time flight dynamic estimation to a higher-level digital twin responsible for flight control manipulations [21]. Ghosh et al. provide a digital twin implementation for the generation of synthetic signals data in support of automated machine tooling for smart manufacturing [22]. This work is conceptually closest to what we attempt with SensorTwin, but differs in that synthetic data generated is used in model training without time-sensitivity rather than for real-time manipulation of the augmented entity.

Chapter 3

Extending the Digital Twin Concept

3.1 Overview

There is immense value in simulating a complex system faithfully. As described in Section 2.2, SITL and HIL testing use simulators to enable rapid prototyping and behavior validation before a full end-to-end system exists. Once into steady state operations, deployed systems require lifecycle maintenance and often present opportunities for optimization as new technology becomes available. Digital twins, also touched on in Section 2.2, involve more intricate simulations aimed at faithfully recreating complex physical entities in their entirety. Such digital replicas utilize real-time data streams to shadow their physical companions. This enables digital exploration of system modifications and provides insights into subsystem operation from the corpuses of data built while the simulation is active. These two applications of simulation live on either end of the spectrum of system maturity, leaving space for other useful applications of virtualization, one such example we introduce in detail in this chapter. Figure 3.1 below shows the pipeline discussed in Section 2.2.4 adapted for the capabilities provided by SensorTwin.

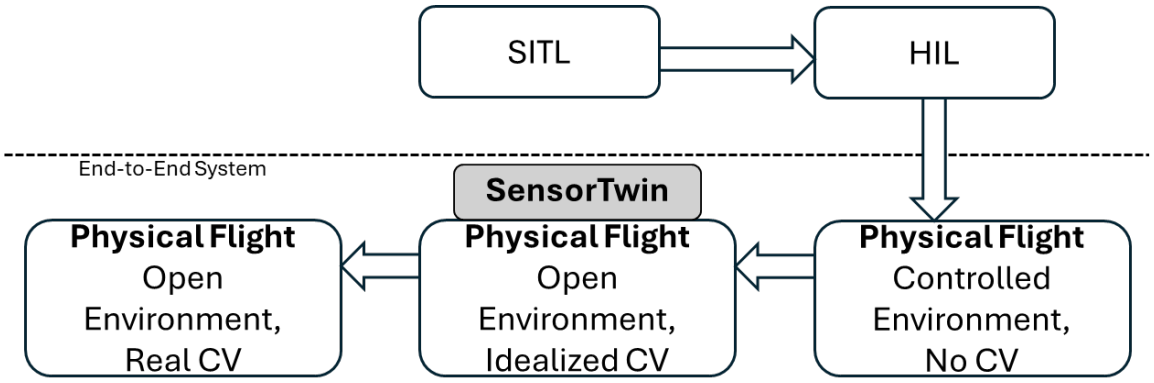


Figure 3.1. SensorTwin in the Automated System Development Pipeline

We begin by outlining the impetus behind the development of the SensorTwin concept, briefly discussing the preceding work of DigitalDrone and the limitations motivating SensorTwin in Section 3.2. We then explore the SensorTwin concept relative to the larger body of work on digital twins, summarizing notable differences and departures from existing concepts in Section 3.3. The chapter concludes in Section 3.4 with a dive into the implementation details responsible for bringing SensorTwin from the conceptual realm to reality, split into three parts. Section 3.4.1 covers the modifications and extensions made to pre-existing components in the SteelEagle

ecosystem. Section 3.4.2 explains an idealized vision engine created as part of this work for the proof-of-concept shown in Chapter 4. Section 3.4.3 details the implementation of the Virtual Reality Interface, which provides the core functionality of SensorTwin.

3.2 Digital Drone – A Prelude

SensorTwin's conceptualization followed the development of DigitalDrone, a full SITL drone that used kinematic simulation for the purposes of control API behavior testing and mission validation. DigitalDrone provided an idealized platform to the SteelEagle ecosystem – isolating out the concerns of arbitrary connection loss, end-to-end latency, and hardware irregularity intrinsic to the distributed nature of autonomous drone flight. DigitalDrone consisted of two components, a simulated drone that maintained position and pose state via kinematic updates, and a driver that implemented the SteelEagle control API and wrapped the underlying simulated object. Problematically, each simulated drone encapsulated an independent simulated world and lacked a mechanism for rendering imagery, real or simulated, tied to the drone's reported location. While kernel modules further upstream still enabled a DigitalDrone to operate concurrently with other vehicles, the lack of any kind of image stream limited the simulated vehicle. While this was a useful test mechanism for basic flight, it could not simulate complex mission logic or exercise vision models. For instance, it could fly patrol patterns but could not simulate tracking because it could not ever provide input that would produce a detection.

3.3 SensorTwin – The Concept

From the gaps revealed by the work on DigitalDrone, SensorTwin was conceptualized. Software simulation of the vehicle alone was useful for examining the control API but limited in testing dynamic mission logic that required real-time feedback from a computer vision engine. What was really needed to bridge the capability gap was a means of providing vehicles with idealized sensor feedback. Drawing inspiration from the broader idea of digital twins, SensorTwin critically differs in the sense that while a traditional digital twin attempts only to faithfully mimic the behavior of its physical companion through a real-time reflection of the physical entity's sensor data, SensorTwin produces artificial artifacts that are used to generate behavior changes in the primary physical system in real-time.

The SensorTwin concept makes several important contributions to the space of autonomous vehicle development. As pointed out in Section 2.2, current trends in autonomous flight system development lend to early phase transitions where risk is well-mitigated by a gradual introduction of hardware components following initial validation in purely software simulated environments. This pattern of gradually stepped progression breaks down once flight with physical vehicles needs to move from a controlled environment to an open one – despite the vehicle and augmenting vision models representing a complex system themselves. SensorTwin's first contribution to the space is to reduce the size of this leap of faith. SensorTwin guarantees deterministic sensor feedback for a given set of state conditions and deterministic vision model output for a given input. By idealizing these portions of the system's operating flow, it becomes possible to remove the variability of real sensors, network connections, and vision models while

preserving the exercise of higher-level control logic during actual flight. This is especially impactful when validating the control logic for emergency procedures or rare events. Real historic data for these events is inherently uncommon and reproducing them in a physical environment is often impractical or dangerous. With SensorTwin, the stimulus to the system is entirely simulated, allowing a platform to "see" the desired case virtually and execute the corresponding control logic, but without any of the risks, such as collision, experienced in a real emergency. Figure 3.2 below depicts the autonomy pipeline discussed in Section 2.1 when augmented with SensorTwin.

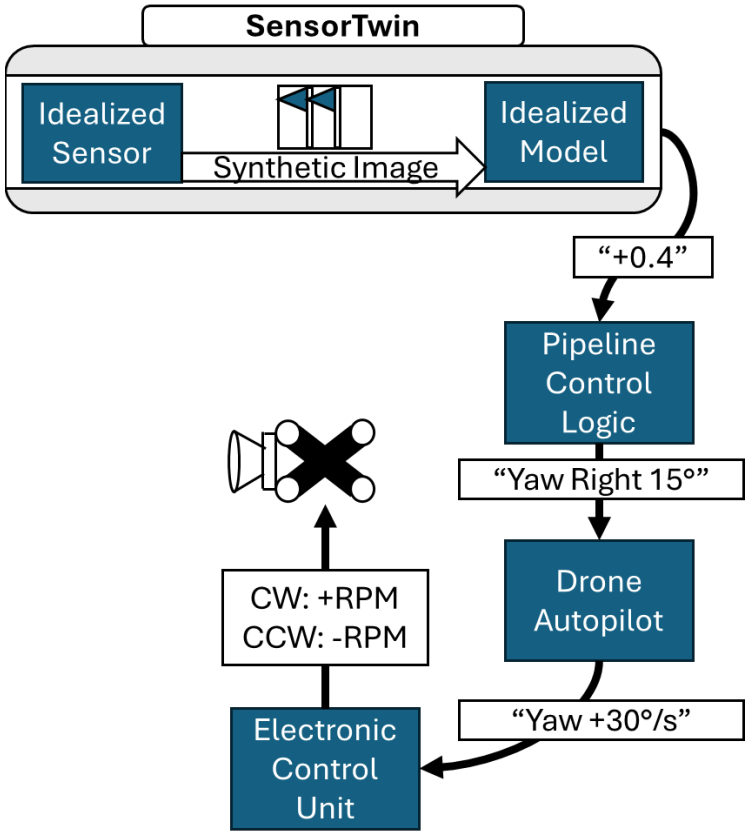


Figure 3.2. Autonomous Actuation Cycle with SensorTwin

The second contribution of SensorTwin is in performance benchmarking, which we use as the proof of concept. Bala et al. make the insightful observation that "only when these factors [target and background specification,] and drone optics are held constant will OODA loop performance come to the fore in determining tracking performance" [2]. SensorTwin directly solves the proposed issues of variable target and background conditions through simulation – for a given trace, the simulated state is deterministic and readily reproducible. Where Bala et al. case specifically on object detection and tracking performance, SensorTwin extrapolates more generally. It can support any semantic function driven by any form of sensor feedback and a vision model, which we subsequently demonstrate by applying SensorTwin to an object avoidance task. Because of the generality of the concept and the determinism that it provides, establishing a theoretic high watermark on a task of interest becomes possible. With a set of benchmark results produced using a deterministic and idealized vision model, we can measure

the remaining headroom for performance improvement of a real engine, which we show in Chapter 4. Similarly, SensorTwin allows digital exploration of modifications to existing platforms. For instance, adding new sensor modalities to an existing platform is a substantial task. The new sensor or camera must be procured, installed, and a supporting model acquired or developed before testing ever begins. SensorTwin allows the simulation of both sensor and model, providing developers the ability to assess whether the addition is worthwhile at a fraction of the engineering effort and cost of a full prototype.

3.4 Aviary – A Simulated World

Aviary [13] is a simulator developed by the SteelEagle team following the work on DigitalDrone but prior to the work on SensorTwin to provide a SITL testing and development capability in support of other project efforts. Aviary neatly solves the previous problem of independent isolated simulated vehicles by introducing a single comprehensive space containing all relevant entities. Prior to the SensorTwin work, Aviary allowed users to generate static objects and operate simple fully simulated drones using the full SteelEagle pipeline. Vision models, simulated or real, may be plugged into Aviary to provide capabilities (e.g. object detection) for simulated drones. One such example is the idealized object detection engine that pre-existed this work and was used extensively in both proof of concept and subsequent benchmarking of SensorTwin. All entities are purely digital without any connection to physical world entities. Figure 3.1 provides a visualization of Aviary's architecture prior to this work.

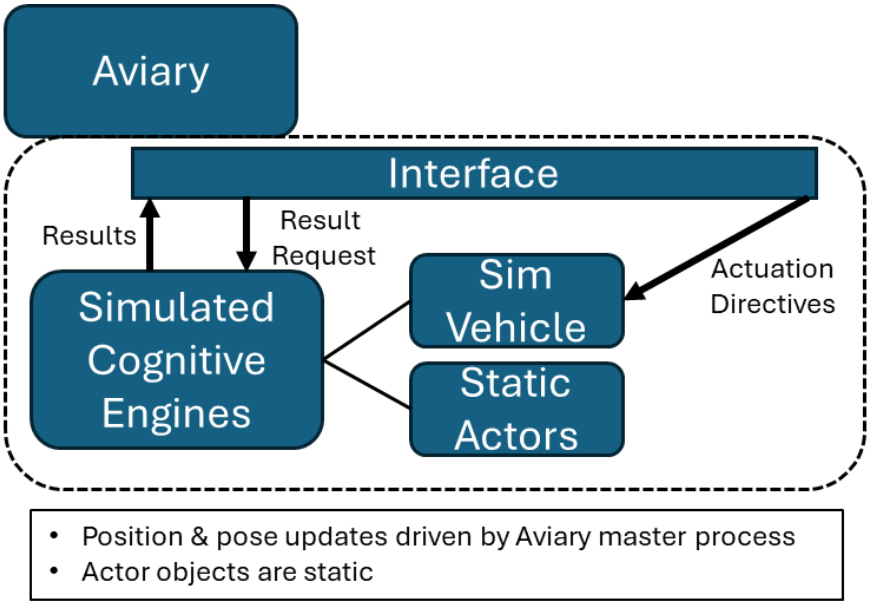


Figure 3.3. Aviary Architecture Without SensorTwin

Within Aviary, entities are characterized as either vehicles or actors. Simulated vehicles are equivalent to a simulated drone and produce a viewport of the simulated space based on the vehicle's current position and pose. A base interface that handles vehicle control wraps the

virtual drones and provides perfect and instantaneous actuation. Actors represent objects of interest for the generation of artificial sensor feedback and are modeled as static, simple 3D geometric figures with an exposed label to facilitate interaction with various engine plugins. Vehicle motion is handled through a main event loop which updates the current position and pose of vehicles based on their respective lateral, vertical, and angular velocities.

3.4.1 Extensions to Existing Aviary Functionality

Implementing SensorTwin required extending Aviary, namely in the form of enabling actor motion via a waypoint system. Ensuring reproducible actor movement patterns was a prerequisite to executing the tracking agility benchmarking tests covered in Chapter 5. Actor way pointing leverages the definition of a singular origin point and the operation of an overarching main event loop by the simulator process to convert geodetic waypoints to the simulated space and ensure a constant rate of motion by the actor towards the next waypoint. Sequences of waypoints are strung together to create actor paths of arbitrary length and complexity. Unlike vehicles, which possess a full control interface, actor pathing between points is exclusively straight-line and constant velocity, with the optional capability for actors to pause at specific waypoints for specified lengths of time.

3.4.2 Idealized Monocular Avoidance Engine

Separate from modifications to existing Aviary components, this work also required the creation of a simulated monocular-vision-based object avoidance engine to conduct the slalom experiment formulated by Bala et al. [2]. To prevent trivialization of the problem and ensure reasonable comparison, the simulated engine functions by producing a depth map based on the drone's current field of view. Unlike the models utilized in the 2024 work, the Aviary engine however benefits from near-perfect information on the actual distance between the supported vehicle and obstacles in its viewport. In contrast to converting to a binary image based on intensity thresholding and actuating based on contours produced by this process, we instead utilize the known distance between vehicle and object to produce a simple quadratic weighting function. This weighting is applied to the normalized lateral angle calculated from the drone's forward axis to the object's position in the frame to produce an actuation direction and magnitude normalized over the range [-1, 1], where a negative sign corresponds to leftwards movement, a positive to rightwards, and larger absolute values to a greater proportion of maximum velocity along the axis.

3.4.3 Virtual Reality Interface – The Goggles

The core idea of SensorTwin hinges on the ability to actuate a physical device based on purely simulated sensor data. This inherently incurs communication and synchronization requirements. Trivially, without the ability to access the state of the simulated world and subsequently make decisions based on the findings, the concept fails. Having constructed a channel to address the first concern, the states of the simulated and physical world must be synchronized. SensorTwin foundationally requires that each feeds the other, requiring a bi-directional link. The physical vehicle's position and pose drive the motion and orientation of the simulated vehicle. Simultaneously, the position and pose of the simulated vehicle provide the reference for what simulated sensor feedback to produce. Accuracy and timeliness in both directions determine the value of SensorTwin, with errors and inaccuracies quickly compounding when the upstream pipeline actuates based off the results returned from Aviary's frame of reference.

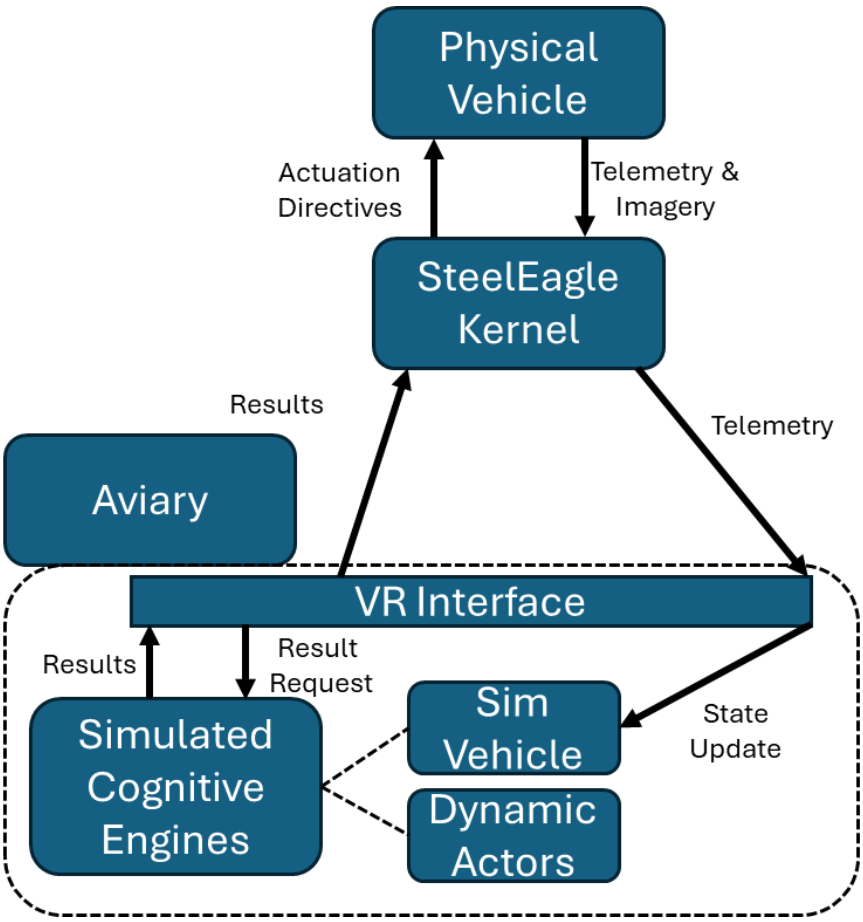


Figure 3.4. Aviary Architecture with SensorTwin

Concretely, this is provided by the Virtual Reality Interface, an adapter bridge that connects physical and simulated vehicles. The VR bridge differs from the baseline control interface that each simulated vehicle is instantiated with in several ways. Firstly, the interface

registers as a subscriber for the corresponding drone's telemetry feed and implements functionality to query telemetry frames and convert them from real-world to simulation space values. Secondly, the VR interface modifies the digital drone movement API, effectively binding the digital vehicle to the physical vehicle through the real telemetry stream. Finally, the interface dispatches requests to Aviary-based computational engines based on the state of the simulated vehicle and subsequently publishes the received results to the kernel through a results socket also maintained within the interface. The bi-directional pipe the VR Interface introduces bridges the digital and physical worlds, allowing end-to-end testing during real flight with the full automation pipeline by injecting simulated stimulus from Aviary's simulated space.

Figure 3.2 below shows Aviary's organization after incorporating the VR Interface. As a brief aside, we refer to this component as a VR interface and not an augmented reality (AR) interface because in this work we interpose on the drone's imagery stream and replace the output entirely with a product from Aviary's simulated space. While there is no technical constraint preventing overlay of simulated space objects on the real image stream, for the purpose of isolating actuation responses entirely to the simulated state we refrain from doing so.

3.4.3.1 Physical-to-Digital Pipeline

Onboard instruments produce position and orientation data that the corresponding driver queries at a fixed rate. The driver packs this information into a telemetry frame defined via Protobuf and publishes it back to the kernel, establishing a telemetry stream which other components of the system may interact with by subscribing to the appropriate socket. At the kernel, telemetry frames are recorded within a datastore implemented over a relational database.

During simulation start-up in Aviary, control interfaces are initialized with a one-to-one mapping of physical to digital vehicles based on the specification in the configuration file provided to Aviary at launch. Each interface subscribes to the corresponding vehicle's telemetry socket and creates a separate background thread responsible for reading the most recent telemetry frame from the datastore. The telemetry receiver operates on a configurable frequency used to determine the polling rate – by default this is set to 30 frames per second. Following receipt of each frame from the datastore, the receiver caches the frame locally and triggers a position update of the simulated vehicle. If the telemetry receiver's frequency exceeds the telemetry update rate from driver to kernel, duplicate frames are expected. Because of this, position and pose updates on the simulated vehicle are designed to be idempotent. The other effect of this is that when loss of connection with the driver occurs, the simulated vehicle becomes bound to the last known position and pose of its physical parent vehicle.

3.4.3.2 Synchronizing the Simulated Space

Each `update_sim_vehicle` call made by the telemetry receiver reads the locally cached frame and verifies that the contents are valid before attempting to modify the state of the wrapped simulated vehicle. We define a valid frame to be simply a telemetry frame whose contents are not empty. In the event that no frame is received (and therefore cached by the receiver) or a frame is received in which required fields are not set, `update_sim_vehicle` falls through as a no-op. Barring these cases, raw telemetry data is extracted for the following: latitude, longitude, relative altitude,

magnetic bearing, gimbal pitch, gimbal roll, and gimbal yaw. Latitude and longitude are received in standard decimal degree format, and altitude is measured relative to the physical vehicle's takeoff position (this is not the same as Above Ground Level / AGL). Magnetic bearing is received as a whole circle bearing with 0° representing magnetic north and wrapping clockwise to 360°. Gimbal yaw is measured similarly but uses the drone's heading as the 0° reference point. Gimbal pitch and roll are measured relative to the horizon with 0° representing the default upright and level position. For gimbal pitch, negative values correspond to orienting the gimbal down towards the ground plane.

Aviary uses a local cartesian coordinate system (with an approximate 1 unit to 1 meter mapping) to manage object locations, requiring conversion of GPS locations, magnetic bearings, and gimbal yaw offsets before attempting to manipulate simulated entities. By convention, the first vehicle instantiated during simulation start becomes the reference origin point for all other objects in the same sim space. Using this origin, each telemetry receiver converts GPS position to a relative position in sim space coordinates and directly sets the wrapped sim vehicle (represented by its attached camera) to the corresponding position. During conversion, relative altitude is mapped directly to the z-value in the local coordinate system. Similarly, magnetic bearing and gimbal yaw are converted to sim angles before the pose of the sim vehicle's camera is directly set to match the reported gimbal pitch, gimbal roll, and converted bearing adjusted for gimbal yaw. Because each update occurs as a direct position or pose set rather than a kinematic model involving position, pose, or velocity targets, repeat `update_sim_vehicle` calls using the same telemetry data produce no change in the sim vehicle's state. Of concern, Aviary runs an event loop to handle sim world and entity updates, triggering a move call on every object and vehicle once per simulator tick. To prevent small per frame errors resulting from position or pose drift, update calls from the telemetry receiver set position and pose targets matched to the values derived from the telemetry frame and a full-stop velocity target.

3.4.3.3 Digital-to-Physical Pipeline

VR interfaces spawn a second background thread that handles out-bound sensor result production and publication. The result producer registers and binds to the results socket specified for the vehicle, running with a configurable frequency like the telemetry receiver. At instantiation the result producer receives a handle to its parent interface's engine holder, a container for various Aviary-based computation engines that produce artificial results based on the sim space state. Result producers query each engine in their holder and package the results into a protocol-compliant result frame before publishing it to the results socket. Once published, the physical drone's driver reads the results from the kernel datastore and makes task transition decisions and actuation calculations accordingly. By design, Aviary engines produce a valid result entry for the protocol-compliant result frame. This allows the packing of multiple results into a single response frame but also implies that the result producer must handle result packing. Due to this, result producers are not totally engine-agnostic and must be updated when new classes of engines (e.g. object detection, obstacle avoidance) are released. However, so long as new engines within an already-defined class produce protocol-compliant results matching existing engines within the same class, no extensions to the VR interface or the result producer thread are required.

3.4.3.4 System Usage

Aviary launches with a TOML configuration file that specifies what vehicles and actors will exist in the sim world. Dynamic actor definitions are straightforward – the file must specify an origin and set of waypoints in GPS coordinate form (along with any pause durations in seconds), as well as the desired movement velocity in meters per second. If no waypoints are specified, objects default to static behavior. In both cases a label must be provided (e.g. “Person”) for idealized engines to recognize the object. SensorTwin augmented vehicle definitions require socket endpoints for the vehicle driver, telemetry, imagery, and engine results. The driver and telemetry sockets are read from to supply the physical-to-digital stream. Images of the simulated space from the digital vehicle’s viewport are published to the imagery socket, while results from the various simulated vision models are published to the results socket. Desired engines are specified on a per-vehicle basis within the config file, allowing for multiple vehicles to operate different sets of engines simultaneously.

Chapter 4

Comparing Performance as Proof of Concept

4.1 Defining Agility Via Flight Characteristics

Agility as a concept is nebulous and subjective. While relative comparison across a limited count of contenders is manageable, rank ordering quickly becomes intractable without defining a set of quantitative, objectively measurable characteristics. Under "agile," Webster's Dictionary provides the simple, intuitive definition of being "marked by ready ability to move with quick easy grace" [1]. While capturing the same essence, earlier research performed on behalf of the U.S. government for manned rotary aircraft defines the term with the following:

> *"Agility can be thought of as the rate of change of maneuverability. It is the quickness with which different maneuver states can be entered or exited…The primary measure of agility is the elapsed time to make the desired change in maneuver state"* [23].

From this stepping-off point, we establish a framework that is logically divided based upon movement dimensions to provide a unified reference for the concept of agility within which to categorize measurements and assessments. This work focuses specifically on multi-rotor style vehicles, and so we partition the framework's categories according to the most relevant flight characteristics in that setting – namely rotational motion, lateral motion, and vertical motion. Under the rotational motion header fall yaw-based maneuvers, angular velocity requirements, and angular acceleration assessments. Lateral motion covers all movement in the horizontal plane and is further sub-divided along two axes: front-back, generally affected by the drone's pitch, and left-right, generally affected by the drone's roll. Vertical motion encompasses changes in the drone's elevation, including vertical acceleration, climb rate, and descent rate. While each component is first-class in influencing a vehicle's overall agility, the scope of this work is primarily concerned with the first two aspects, rotation and lateral motion, with vertical-based concepts proposed and briefly discussed but not explored experimentally.

Briefly discussed in Section 2.1, automation pipelines influence the agility of the supported vehicle. Unlike skilled human pilots who develop an innate sense of how much throttle to apply based on the nuance of the situation, automation pipelines must either learn fine control via model training or be manually tuned by their developers. As a result, the quality, and range, of actuation demands the automation pipeline makes in response to sensor feedback directly influence the display of agility by the physical vehicle.

4.2 Initial SensorTwin Validation by Result Recreation

Foundationally, we expect a functioning SensorTwin to enable the replacement of non-deterministic vision components with deterministic artificial sensor data. In doing so, we remove the inherent variance introduced by such models with the intent of answering the question "how well can I possibly perform on a task if my computer vision were perfect?" Before discussing the larger benchmark suite developed as part of this work, we present as proof of concept a recreation of the earlier experiments performed in the work by Bala et al. but with the respective computational engines replaced by an Aviary-based SensorTwin. To map the OODA Loop to autonomous drone operations Bala et al. measured the capabilities of an automated vehicle supported by a computational engine running a computer vision model [2]. Two task benchmarks were developed: one for object detection and tracking, and one for object avoidance.

4.2.1 Visual Object Tracking – RoboMaster Random Walk

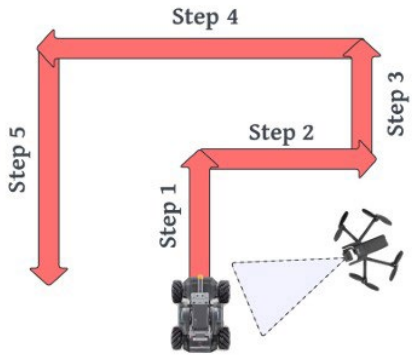


Figure 4.1. RoboMaster Random Walk Example [2]

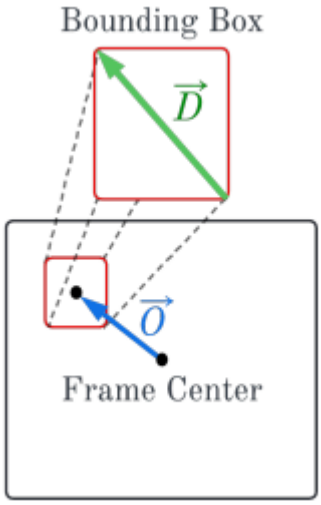
4.2.1.1 Task Description

The object tracking benchmark, discussed in detail in the Visual Object Tracking section of Bala et al.'s 2024 work, involved a small object (a DJI RoboMaster S1 – Figure 4.2.b below) executing a multi-step random walk (Figure 4.1) on a level, near-uniform background. While the robot performed this random walk, a single drone (a Parrot Anafi Gov, the same model used for all measurements in this work – Figure 4.2.a below) attempted to keep the device centered in its field of view. The automation pipeline provided actuation control by performing a detection task on the current frame and providing offset corrections based on the location of the bounding box relative to frame center. Task assessment began once the detection engine registered a valid detection and concluded when either the random walk completed or detection track remained broken for a specified period.

(a) Parrot Anafi Gov (Test Vehicle) [24] (b) DJI RoboMaster S1 (Test Target) [25]

Figure 4.2. RoboMaster Random Walk Test Vehicle & Target [2]

4.2.1.2 Task Evaluation



(a) Visual Depiction of Centering Ratio

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

$$s_i = 1.1^{-c_i}$$

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

(b) Calculating Score

Figure 4.3. Scoring for Visual Object Detection [2]

For this experiment, Bala et al. devised a scoring system that penalizes the distance of the track object from frame center [2], visualized in Figure 4.3 above. Each frame in which a detection occurs contains a bounding box around the detected object. A pixel offset vector, specified as $||\vec{O}_i||$, measures L2 distance between the bounding box center and the center of the frame (the frame origin). This value is scaled by the vector dimension of the bounding box, specified as $||\vec{D}_i||$, yielding a centering ratio, c_i. The centering ratio is subsequently applied to an inverse exponential function, creating a super-linear penalty based on distance from frame origin and producing a score for each frame in the range [0, 1]. Frames in which no detection occurs receive

a score of 0. Frame scores are aggregated to produce an average per frame score for each run, with higher values indicating better performance.

4.2.1.3 Recreation Methodology

We generate in Aviary a target object of approximately the same size as the RoboMaster used in the original experiment. Identically, the trace consists of 35 steps with a mean step length of 5m and direction of travel for the track object constrained to 90° offsets from the drone's starting heading to mirror the cardinal direction constraint. For each velocity setting in the original paper, 1.5m/s, 2.5m/s, and 3.5m/s, we execute five iterations of the experiment. The drone, a Parrot Anafi Gov (as used in the original work), begins at an altitude of 10m with the target object approximately center frame. The drone must track the object until completion of the last step or until loss of track extends beyond the timeout setting – set here as 10 seconds to match Bala et al. Because SensorTwin ensures positive detection when the object is in frame regardless of background and because the simulated object may ignore local topography, the level, near-homogenous ground setting utilized in the original experiment is unnecessary.

4.2.1.4 Experiment Results

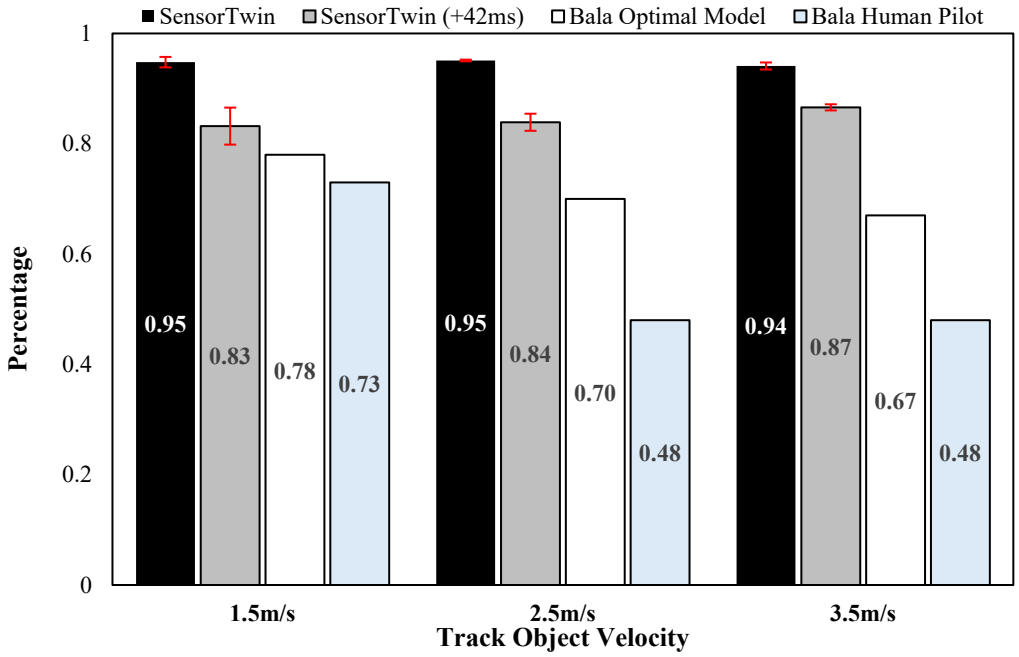


Figure 4.4. Comparison of Average Scores for Visual Object Detection [2]

Figure 4.4 above shows a comparison of results between the work of Bala et al. and our SensorTwin recreation of the experiment. As expected, the system performs better when running the idealized sensing and vision models under SensorTwin than when running a live model. This

supports the concept of using SensorTwin as a tool to gauge how a system might perform with perfect sensing and computer vision. In all 15 test flights executed under SensorTwin, the raw *detection rate*, here the proportion of total frames in which a detection occurs, is 100%.

Notably, computation cost per frame is negligible in Aviary (less than 1ms). To more faithfully match the conditions of the original experiment and produce a better estimate of theoretic best performance, we artificially induce 42ms of additional latency into the simulated detection engine to match the reported cost of the most expensive model previously assessed. The averaged results for the matched-latency iterations are shown in Table 4.1 below.

	SensorTwin (+42ms)	SensorTwin (No Added Latency)	Bala 2024 Optimal Model [2]	Bala 2024 Human Pilot [2]
1.5 m/s	.832	0.948	0.78	0.73
2.5 m/s	.839	0.951	0.70	0.48
3.5 m/s	.866	0.941	0.67	0.48

Table 4.1. Matched Latency Comparison of SensorTwin on Visual Object Detection [2]

The degradation in performance relative to operation with no added latency seems supportive of the observation that greater latency reduces vehicle responsiveness, showing between 7.5% and 11.5% reduction in overall average score. To break out the sources of this degradation in greater detail, we examine *per detection frame average score*, referred to interchangeably as *detection quality* in later sections, defined as the average score over only frames in which detections occur, in addition to raw detection rate, both shown in Table 4.2.

	SensorTwin (No Added Latency)		SensorTwin (+42ms Latency)	
	Detection Rate	Detection Frame Average Score	Detection Rate	Detection Frame Average Score
1.5 m/s	1.00	0.948	0.940	0.832
2.5 m/s	1.00	0.951	0.886	0.840
3.5 m/s	1.00	0.941	0.997	0.867

Table 4.2. SensorTwin Effects of Inference Latency on Detection Rate & Detection Quality

The introduction of additional latency into the SensorTwin pipeline produces two distinct impacts on the automated vehicle during assessment: a reduced overall detection rate and a lower per detection frame score. Recalling that frame score is calculated as a function of the magnitude of the detection object's offset from frame center, a lower average score in frames registering detections suggests that in the aggregate the detection object is further from frame center than the non-latent case. This makes intuitive sense considering the mechanics of the automation pipeline. By introducing a time delay, the drone is in effect actuating towards the position of the track object at some time delta in the past rather than the current position of the object.

The introduction of additional latency also results in a lower detection rate, indicating that at various points in the experiment the track object fully escaped the field of view of the drone but was subsequently reacquired in future frames. During the conduct of this experiment, we observed that loss of track occurred exclusively following 180° reversals in the track object's path. We posit that this results from a combination of the natural delay coupled with an effect we subsequently refer to as pipeline inertia. We define pipeline inertia as the resistance an automated vehicle's pipeline displays in response to changes in the direction and magnitude of actuation requirements during task execution. This is tied to configurable gain rate limits and sampling rate for results from vision models. This phenomenon is explored in more detail during the analysis of the benchmark suite experiments in section 5.5.

4.2.2 Visual Object Avoidance – Slalom Course

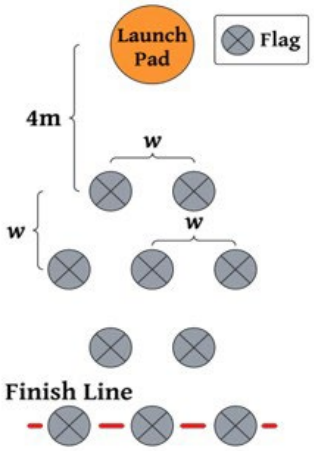


Figure 4.5. Slalom Course Example [2]

4.2.2.1 Task Description

The object avoidance benchmark, covered in Section V of the work by Bala et al., requires a drone to navigate a series of thin racing flags placed in a slalom-style course at a set distance (Figure 4.5). A key component of this experiment is that the drone is monocular, therefore the automation pipeline must perform object detection and depth estimation to successfully navigate the course. The drone begins at a short offset from the first row of flags and successfully completes the course after moving past the last row of flags. At the time of the original experiment execution the automation pipeline implemented avoidance in a way that prevented yawing, requiring the drone to strafe left or right after detecting an obstruction. Separation distance between flags, using the same distance between flags in the same row and between rows, provided the difficulty lever for this evaluation.

4.2.2.2 Task Evaluation

$$(1) \qquad S_w = \frac{t_{min}}{t_{avg}}$$

Figure 4.6. Scoring for Visual Obstacle Avoidance [2]

To avoid over-penalizing drones that suffered from low top-speed, Bala et al. normalized scoring by first performing a control run without obstacles, measuring the average time to completion for the drone, defined as T_{min} [2]. Scoring for each model was subsequently calculated, as shown in Figure 4.6, by taking the average time to completion for a set of runs at a given separation distance, defined as T_{avg}, and dividing the control average by the run average to bound the score between the range of [0, 1], with higher scores indicating better performance. If the drone is unable to complete the run at a given separation distance, or if the drone strikes a flag, it receives a score of 0 for the run and is marked as Did Not Finish (DNF). To be scored at a given separation distance, a drone must complete 80% of the runs executed at that distance.

4.2.2.3 Recreation Methodology

We begin by generating a field of obstacles within Aviary matching the specifications in the original experiment. The obstacles are modeled to be similar height and width to the racing flags described in the original experiment. This process is performed for each value of the separation parameter specified in the original work – 2m, 2.5m, and 3m – to produce the individual SensorTwin trace files. All traces use the 4m separation from drone origin to first row of flags previously used. Because no physical objects exist to serve as visual references for human observers, we make two modifications to the assessment procedure. First, we perform collision detection within the Aviary avoidance engine between the simulated vehicle and the simulated obstacles. If at any point the physical drone enters an actor volume, the trace is terminated and scored DNF similarly to a physical flag collision in the original version. To account for the vehicle being represented as a single point in simulated space, we adjust the collision distance threshold to account for the radius of the vehicle. Second, we condition trace conclusion and clock stop on the drone passing a particular longitude. By orienting the drone and constructing the obstacle field due-east (90°) or due-West (270°), we are able to produce a north-south running finish line immediately beyond the last row of obstacles. The finish longitude is acquired by calculating the total course distance from the origin and is confirmed prior to first test run by placing the physical drone at the corresponding distance and logging the GPS position reported by the drone in its telemetry stream. Once prepared for an iteration, the drone is elevated to a starting height of 1m at the origin point before the VR interface and simulated field trace are activated within Aviary. Activation of the VR interface logs the start time of the trace, and the telemetry receiver logs completion once the drone crosses the longitudinal delineation marking the finish line. Control runs are captured by operating the drone under test conditions with SensorTwin active but in a simulated world containing no obstacles to impede motion.

4.2.2.4 Experiment Results

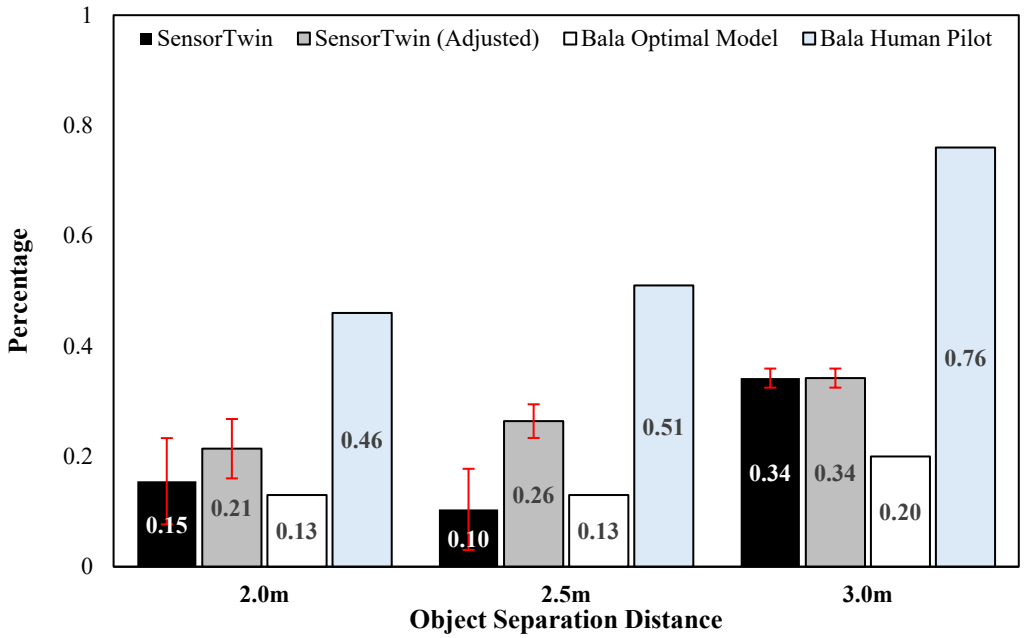


Figure 4.7. Comparison of Average Scores for Visual Object Avoidance [2]

Figure 4.7 shows the comparison of results for SensorTwin against the optimal model and human pilot results reported by Bala et al [2]. During testing for separation values of 2.0m and 2.5m, sustained wind gusts produced performance by the physical drone an order of magnitude worse than all other iterations at that separation. For experiment integrity we include score calculations for the full dataset collected and with the outlier values dropped. Table 4.3 below displays base and adjusted values. The following analysis is conditioned upon the adjusted scores.

Similarly to the Aviary detection engine, inference costs are trivial in the Aviary avoidance engine. Even with this advantage and the benefit of producing a perfect absolute distance depth map (via the idealized engine), an automated drone augmented with SensorTwin still performs significantly worse than a human remote pilot under the same test conditions, which we attribute to the inherent difficulty of avoidance implemented over monocular depth estimation. Score differences ranging from 8% to 14% between SensorTwin and the optimal model from the 2024 work suggest significant headroom for model-based performance on the avoidance task. We attribute this to a mixture of increased responsiveness from the lack of inference cost and more accurate actuation vector calculation from a perfect depth-map.

	Mean Score	Standard Dev	Median	Adjusted Mean	Adjusted Standard Dev
2.0 m	0.155	0.078	0.216	0.214	0.054
2.5 m	0.104	0.074	0.161	0.264	0.030
3.0 m	0.342	0.017	0.335	-	-

Table 4.3. Aggregate Score Metrics for SensorTwin on Visual Object Avoidance

Chapter 5

Agility Benchmarking

5.1 Autonomous Tracking Agility Benchmark

This work contributes a suite of benchmarking tests intended to provide a theoretical performance bound for a given automation pipeline. As shown in the results produced while recreating the RoboMaster and Slalom test results, isolating out the non-determinism of computer vision models and the computational cost associated with state-of-the-art model inference leaves only environmental effects, physical hardware constraints, and the autonomy pipeline as limiting factors when trying to quantify the capability of a drone. Each trace included in the benchmark seeks to either assess some quantifiable component of the drone's agility or provide an intuitive sense of the border of a vehicle's capabilities. The traces within the Autonomous Tracking Agility Benchmark exclusively leverage object detection as the task mechanism. Compared to object avoidance style tasks, object detection and tracking enable much finer-grained manipulation of an assessed platform.

Objectively, we form an assessment of agility by assessing a vehicle in three categories of motion: lateral, rotational, and vertical. The scope of this work focuses on the first two categories (and combinations therein); estimating vertical separation between a vehicle and a target object is not a capability currently implemented within the SteelEagle ecosystem. Though we leave this topic to future work, we also outline (and provide trace generators for) five traces that incorporate a vertical movement component. Subjectively, we seek to refine the intuition around what a platform can reasonably be expected to handle. Within an automated flight environment, certain movement patterns might reasonably be expected to be more challenging to handle than others. Several of the benchmark traces are designed intending to identify such edge cases and determine whether a vehicle-pipeline combination handles them effectively – and if not, to identify where the limits of the system are reached.

Towards these combined ends, the Autonomous Tracking Agility Benchmark consists of 12 traces, designs for which are displayed in Figures 5.1 and 5.2. The difficulty of each trace is modifiable via the lever of simulated object speed. Intuitively the faster the object moves, the greater the actuation demands on the assessed vehicle to maintain a positive track. In the subsequent sections we cover design and intent of each individual trace, explain the experiment procedures and conditions used to collect the first dataset on a physical platform, and then conclude the chapter with a presentation of results for the seven non-vertical traces across three different simulated object speeds.

5.2 Benchmark Trace Overview

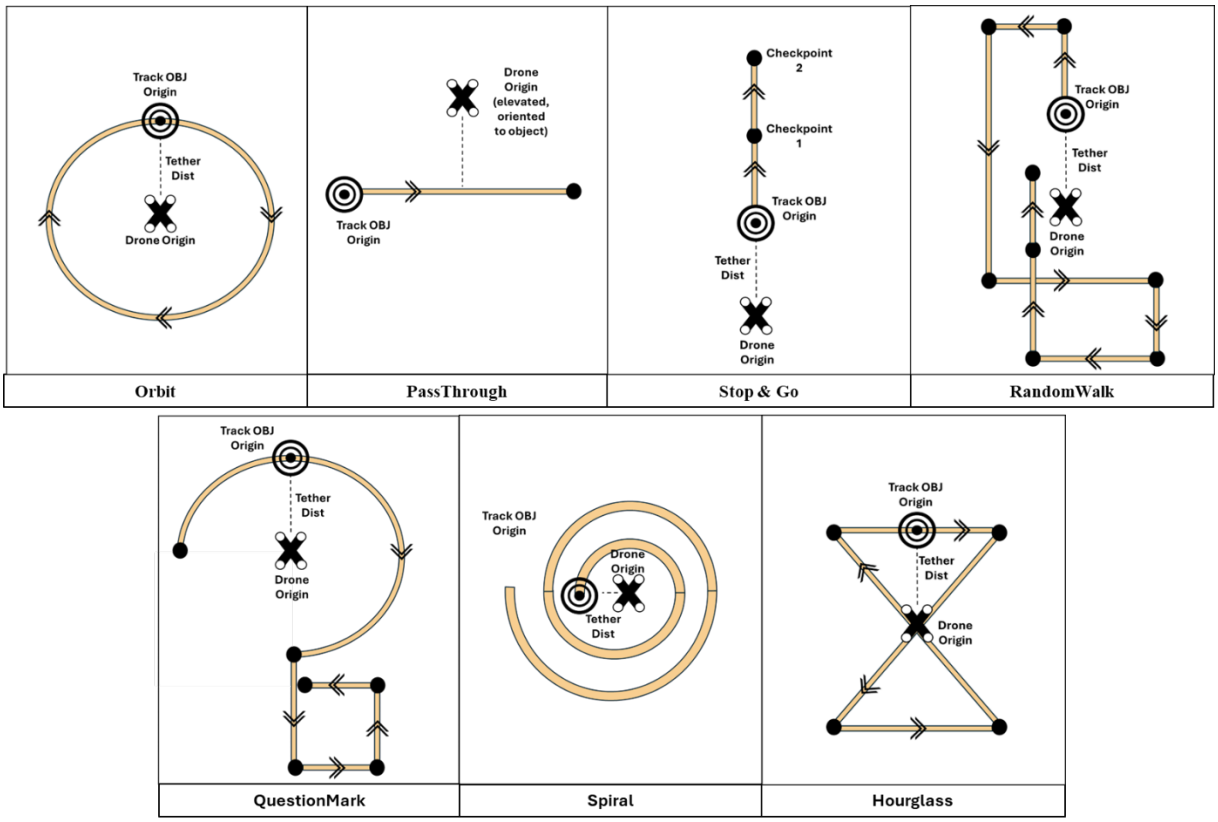


Figure 5.1. Autonomous Tracking Agility Benchmark (ATAB) Trace Diagrams

5.2.1 Orbit (OR)

Orbit is a rotational assessment that fixes the position of the vehicle. At trace start, the simulated object begins moving at constant velocity through a set of waypoints surrounding the drone at tether distance. Completion occurs once the object completes a single revolution of the drone's origin position. Only a single revolution is used to avoid score inflation from reacquisition on additional orbits. To maintain track the drone must generate a yaw rate matching the angular velocity of the simulated object relative to the origin point. On a successful iteration, the drone yaws through a complete revolution without departing the origin point.

During Orbit a natural lag occurs at start due to the object immediately moving while the vehicle is stationary. This requires the drone to rapidly accelerate to a rotational rate greater than the angular velocity of the object to recenter it. Overshoot can occur unless the drone and its supporting automation pipeline match the object's angular velocity, indicating lack of responsiveness, signaled by track loss, or lack of precision, signaled by oscillation.

5.2.2 PassThrough (PT)

PassThrough is a lateral assessment isolating performance along the forward-backward axis. We define passthrough as an object moving directly through a vehicle's position to the opposite side. We classify the maneuver as adversarial due to the likelihood of it breaking track. The track object begins in front of the vehicle beyond the tether distance. On start it moves at a constant velocity on a straight line through the vehicle's origin to a point equidistant behind. In a successful iteration, the drone moves forwards until reaching tether distance with the object. Once at tether distance, the drone then moves directly backwards with minimal gimbal movement until the trace completes.

Starting outside the tether distance forces movement towards the object that the system must counteract. This quick switch in the necessary direction of travel for the vehicle reveals pipeline inertia if present. Track loss tests system resiliency to intermittent detection failures, with the object's back leg providing a predictable path for reacquisition.

5.2.3 Stop & Go (S&G)

Stop & Go is a lateral motion assessment isolating along a singular axis. The simulated object begins in front of the platform at tether distance and moves directly away at a fixed velocity through multiple identical path segments. Between segments the object pauses. In this work, we measure only along the front-back axis but rotating the path segments 90° in either direction while maintaining the same object start point and enforcing a strafe constraint on the assessed platform converts the assessment to the left-right axis. For a successful iteration the drone moves forward maintaining tether distance before stopping as quickly as possible each time the object pauses.

Because the simulated object ignores inertia in a way that a physical vehicle cannot, some degree of oscillation about the tether distance is guaranteed. The magnitude and duration of oscillation informs intuition, with shorter violations of tether distance and fewer velocity direction changes signs of more agile systems.

5.2.4 RandomWalk (RW)

RandomWalk is a direct recreation of the RoboMaster Random Walk experiment introduced by Bala et al [2]. To allow direct comparison this trace allows both rotational and lateral motion but may be converted to a purely lateral assessment with a strafe-only restriction. The simulated object moves through a defined number of steps. Each step is composed of a random distance sampled uniformly from a step-length range and an arbitrarily selected cardinal direction relative to the drone's origin heading. During a successful iteration the drone matches the front-back motions of the object while yawing or strafing in response to perpendicular motion by the object.

RandomWalk requires the vehicle to station-keep at the tether distance in response to arbitrary and immediate changes in direction by the simulated object. Failing to maintain center or tether indicates possible sluggishness in actuation correction by the pipeline or limitations at

the hardware level. Much like with Orbit, oscillation around the center in one or both planes points to pipeline responsiveness and precision limitations. Though not intentionally induced in this trace, passthrough may occur (especially if the simulated object's velocity is higher than the vehicle's maximum or the vehicle is slow to accelerate) and is an obvious sign of the problems described.

5.2.5 QuestionMark (QM)

QuestionMark assesses rotational and lateral capability in sequential ordering. The object first moves through a 270-degree arc around the vehicle's origin. It then immediately transitions to straight line movements consisting of an outward-bound leg followed by a simple box pattern. Object velocity is constant and no pauses occur. In a successful iteration the drone rotates at the origin point until the object begins the outbound leg. After the object transitions to straight line segments, the drone initially moves forward to maintain tether distance, then yaws or strafes once the object turns. After the second turn, the drone moves backwards to maintain tether, before yawing or strafing to maintain track once the object makes its final turn perpendicular to the drone.

While this test primarily evaluates the transition from rotational to lateral motion, drift from the object's partial revolution may induce combined motion earlier than expected. High detection rates with lower per-frame averages are indicative of a platform struggling to balance actuation in multiple dimensions simultaneously.

5.2.6 Spiral (SP)

Spiral forces combined lateral and rotational motion for the duration of the trace. The track object begins at tether distance and follows a revolving path around the vehicle's origin that gradually grows in radius. Throughout the trace, the track object maintains a constant velocity and performs no stops. On a successful iteration the drone matches the object's velocity and maintains a gradual follow-and-bank flight pattern behind the object in-line with the object's already covered path.

The follow-and-bank flight pattern this trace attempts to induce ultimately examines how stable and precise a system is when moving laterally with some degree of yaw. Overshoot or repeated shift of the vehicle from inside the object's path to outside the curve suggest issues with multi-dimension precision and pipeline inertia.

5.2.7 Hourglass (HG)

Hourglass assesses rotational and lateral capability in an adversarial setting. The track object begins outside of tether distance and begins moving perpendicularly to vehicle before turning towards the vehicle's origin point and attempting a passthrough. On reaching the opposite point, the object turns again and moves parallel to its initial segment before turning back towards the origin for a second passthrough. Object velocity is constant and no stops occur. During a

successful iteration the drone first moves forward to reach tether distance, then moves directly backwards to maintain it as the object turns onto the passthrough leg. The drone then executes a follow-and-bank or strafe maneuver keep track of the object on the parallel leg before repeating both steps for the second half of the object's movement.

This trace attempts to produce an understanding of how a vehicle and its automation pipeline perform when placed into a non-cooperative scenario. The additional length of this trace relative to PassThrough is intended to allow a better gauge of how well recovery is performed after track loss.

5.2.8 Omitted Traces

The following traces were not assessed as part of this work due to the previously described challenge of estimating vertical separation in the current SteelEagle system. Generators for each trace were, however, developed during this work and a brief explanation of each trace follows. These traces are presented as possible future work in Chapter 6.

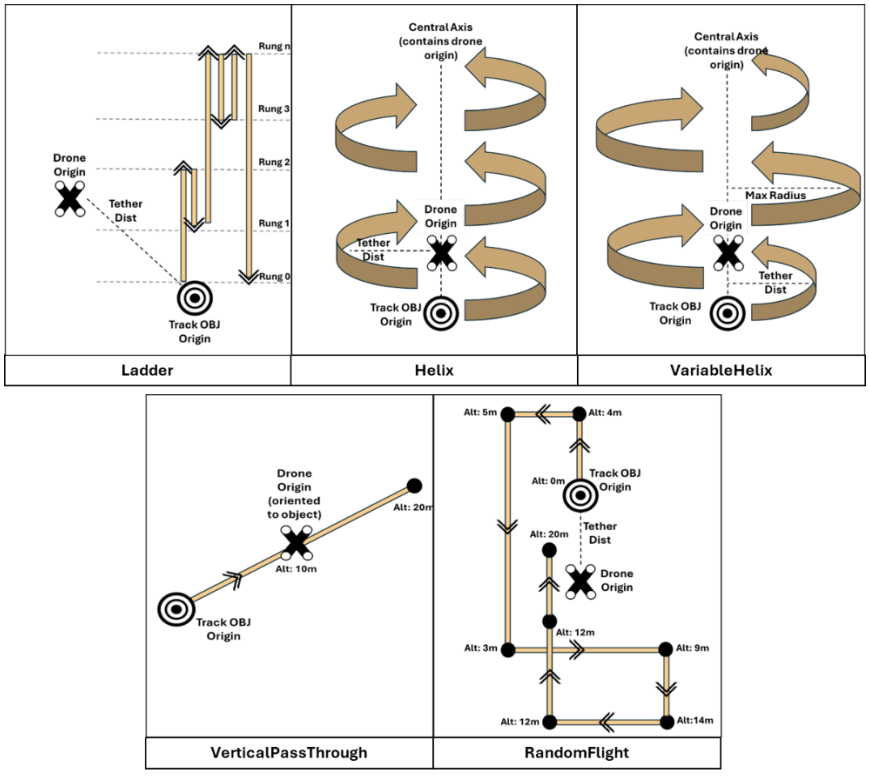


Figure 5.2. Omitted Trace Diagrams, Autonomous Tracking Agility Benchmark

Ladder (LD)
Ladder assesses vertical motion. The track object begins centered at ground level directly in front of the assessed vehicle. On trace start, the track object alternates elevating and descending a series of "rungs." Distance between rungs is constant, but the number of rungs traversed during an elevation or descent is not. The track object's velocity is constant and no pauses occur.

Helix (HX)
Helix assesses a combination of vertical and rotational motion by adding a gradual elevation and additional revolutions to Orbit. The track object begins at ground level centered at the tether distance, then climbs as it revolves around the assessed vehicle. The track object's path uses the tether distance as the radius and velocity remains constant.

Variable Helix (VH)
VariableHelix extends the Spiral trace with a constant-rate elevation component to assess all three dimensions. The track object begins at ground level with a base radius equivalent to the tether distance. The radius of the track object from the drone's origin position both increases and decreases throughout the test, though never to less than the tether distance. Object velocity is constant.

VerticalPassThrough (VPT)
VerticalPassThrough modifies the base PassThrough trace to include constant-rate vertical elevation. The simulated object crosses directly through the drone's position at the drone's starting elevation. After completing the attempted intercept, the simulated object proceeds to an equidistant position above and behind the origin.

RandomFlight (RF)
RandomFlight extends the RandomWalk trace to include variable elevation changes in each step. Start conditions for the trace are identical to RandomWalk – the simulated object begins at ground level forward of the assessed vehicle at tether distance. Velocity for the simulated object is constant and the object moves in straight lines from waypoint to waypoint.

5.3 Experiment Methodology

For each test within the benchmark, the assessed platform is bound to a SensorTwin within an Aviary simulated world via the VR interface described in Chapter 4. Traces exclusively utilize the simulated object-detection engine provided via Aviary and require the assessed platform to maintain a track on the simulated object. At the start of a trace, the simulated track object begins centered in the platform's field of view with the object at a fixed lateral distance. Depending on the trace being executed and the underlying characteristics being evaluated, this distance is either the tether distance or a larger distance (between 150% and 250% of the tether distance) intended to force the platform to actuate forwards.

5.3.1 Condition Specifications

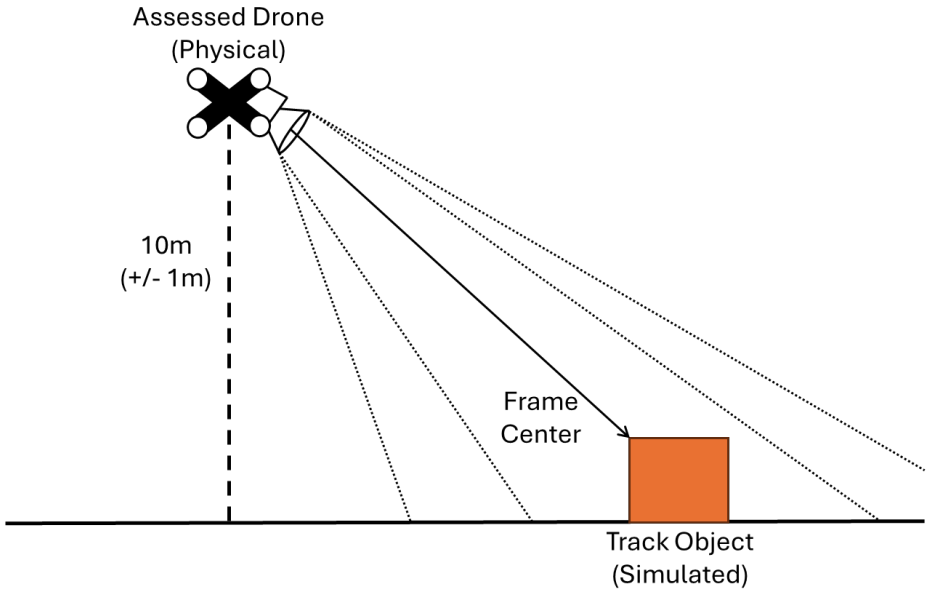


Figure 5.3. Benchmark Start Conditions

Concretely, each experiment was performed using a single Parrot Anafi GOV as the assessed platform. A Dell Latitude 5420 Rugged hosted the vehicle driver, SteelEagle kernel, Aviary instance, and a React-based ground control station (GCS) interface connects to the assessed vehicle through the kernel over a WiFi link. For all experiments, a tether distance of 10m was used and the drone was allowed to yaw with an unbound angular velocity. Because the Parrot Anafi GOV cannot yaw the camera gimbal and the tracking task implementation avoids ever rolling the gimbal, the gimbal remained unlocked, allowing for up-down corrections via gimbal pitch changes, which are mechanically limited to a range of values from 90° (upwards, perpendicular to the drone body) to -90° (downwards, perpendicular to the drone body) [26]. Gimbal pitch in each iteration was adjusted based on the start position of the track object to ensure a centered first frame and consistent track start. For safety concerns the drone was limited to a maximum single-axis lateral velocity of 5m/s, equivalent to the maximum straight-line velocity of the track object during the hardest variant of each trace. Note that because this limitation is applied separately to both left-right and front-back actuation of the drone, the maximum total lateral velocity of the drone is 7.071m/s. We fixed the elevation of the drone to 10m for all traces and bar elevation changes via actuation commands. In practice, local wind conditions produced minor variations in the elevation of the drone throughout trace runs, but these variations were never observed to be greater than 1m above or below the drone's starting elevation.

5.3.2 Trace Execution Procedure

Due to the volume of flight data required for a useful sample size, data collection occurred over multiple flight sessions across several days. Where possible, all iterations at all velocities for a given trace were executed during the same flight session.[1] Prior to execution of an experiment set, the drone origin point was physically marked with a landing and takeoff pad. The drone was powered on while on the landing pad to report its GPS position and magnetic heading. Trace files were regenerated programmatically using the reported telemetry data – this step was required because the actor way-pointing system implemented in Aviary requires waypoints to be specified as decimal degree formatted coordinates rather than local coordinate offsets. Once regenerated, new trace files were manually verified for first-frame object centering at the trace execution altitude before scored runs began. Each trace began with the drone returned to the origin position at an elevation of approximately 10m (within 1m tolerance) and heading matched to the starting heading reported in the initial telemetry read. After resetting the drone's aerial position, the track task was launched via the GCS, which included as a first step a gimbal pitch set command to the appropriate starting pitch for each trace. Only once completion of the gimbal pitch set was reported was the scored trace launched within Aviary, marking actual iteration start.

5.3.3 Trace Scoring

Scoring of the benchmark traces is performed identically to the methodology defined by Bala et al. for the RoboMaster Random Walk [2] (depicted previously in Figure 4.3) to provide a sense of relative comparison to results garnered on a physical detection object with a real computer vision engine operating. Similarly to the original RoboMaster experiment, Aviary's simulated detection engine produces a bounding box around the track object for each frame in which the drone's position and pose result in a capture of the object. Unlike the original experiment, the output of the simulated detection engine is deterministic. If the object is within the drone's field of view and close enough to be visible at sufficient resolution, a bounding box around the simulated object is produced with a confidence value of 100%.

A score is awarded to each frame, s_i, captured during the trace based on the distance of the simulated track object from the center. An inverse exponential function super-linearly penalizes distance from frame center, though unlike Bala et al. we calculate scores only for a penalty of 10%. For frames that do not contain the detection object or in which the detection object is at a distance exceeding the cutoff threshold, a score of 0 is awarded. Per frame scores are aggregated over a trace run to produce average scores for the duration of the run. To evaluate each trace, we examine the raw detection rate, r_{det}, as a proportion of the total frames captured, the average per frame score over all frames captured, s_{avg}, and the average per frame score over only frames captured containing a detection, s_{det}, interchangeably referred to as detection quality.

[1] Mechanical issues encountered during data collection for Spiral coupled with the trace's length required splitting experiment execution over two days. The protocol described in execution procedure was employed to minimize variance in relative starting positions across flight sessions.

5.4 Experiment Results

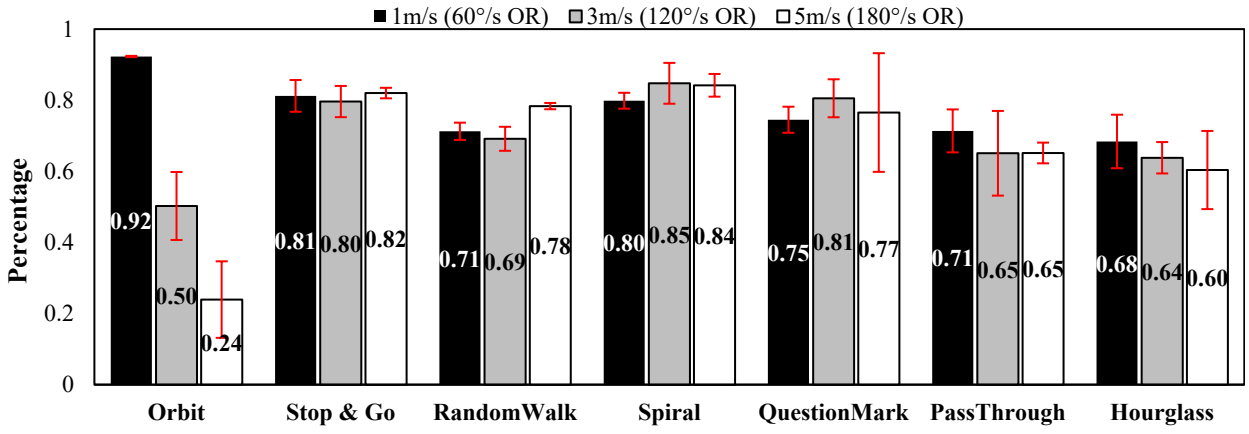


Figure 5.4. Average Scores of Parrot Anafi Gov on ATAB

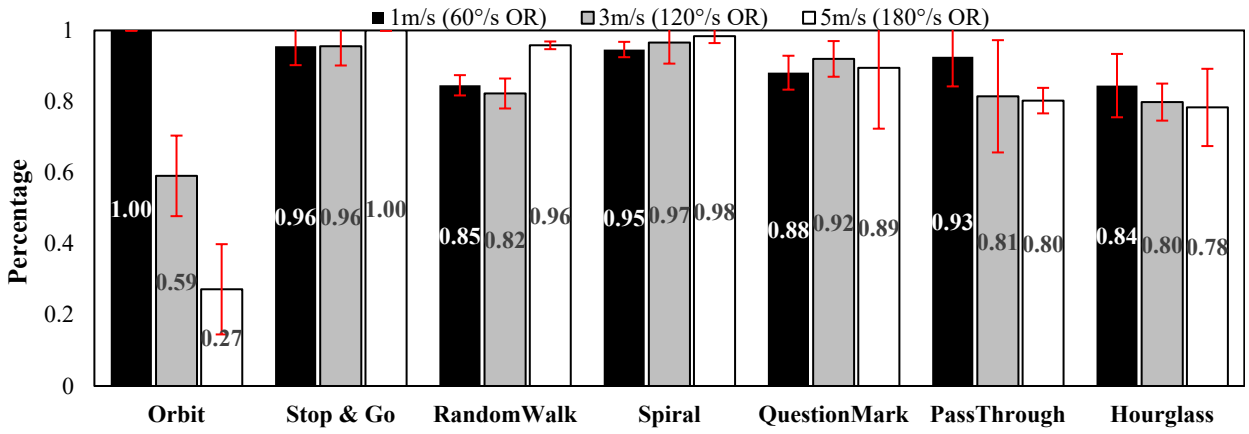


Figure 5.5. Average Detection Rates of Parrot Anafi Gov on ATAB

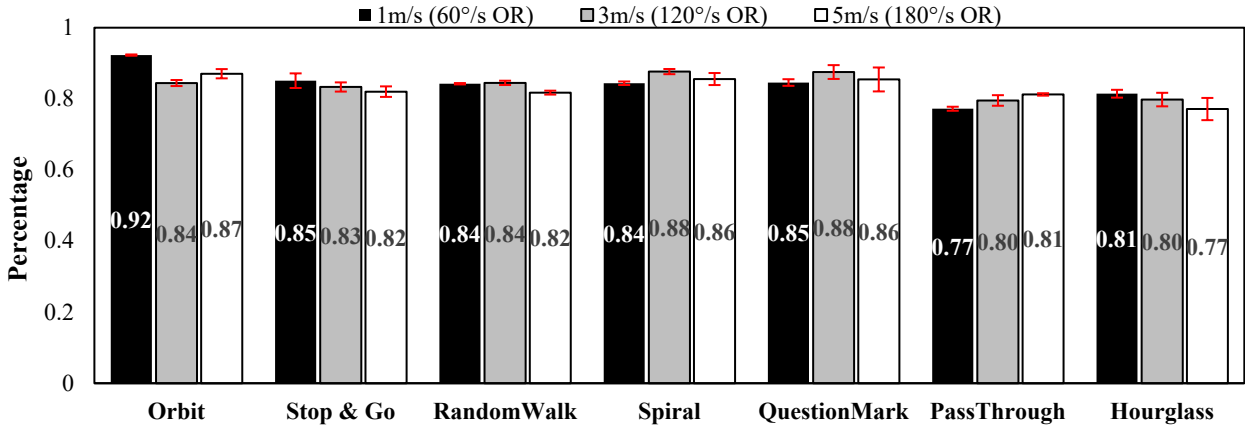


Figure 5.6. Average Detection Quality Scores of Parrot Anafi Gov on ATAB

Figures 5.4, 5.5, and 5.6 above summarize the performance metrics for an Anafi Parrot Gov on the Autonomous Tracking Agility Benchmark across all evaluated tasks. Error bars representing standard deviation are displayed in red for each metric at each velocity. While the lack of additional platforms prevents relative comparison using the produced benchmark results, the data collected allows for a platform specific assessment of tracking agility. Subsequent tables and figures present metrics by trace accompanied by observations made during experimentation and data analysis.

5.4.1 Orbit

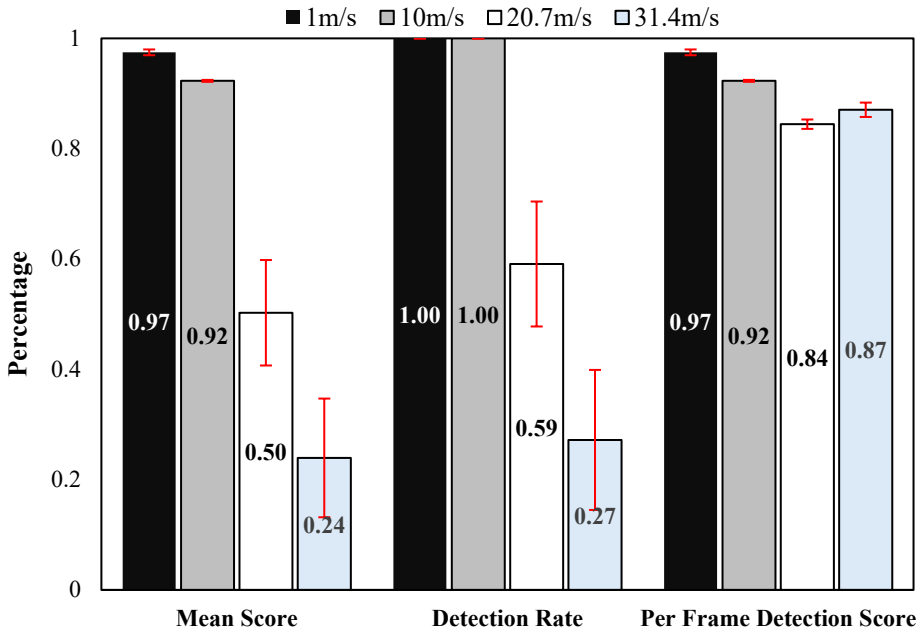


Figure 5.7. Orbit Aggregate Metric Scores

As the only trace to assess purely rotational performance, a different set of track object velocities were utilized during data collection. Data was collected for object movement rates of 1 m/s, 10 m/s (57.3 deg/s), 20.7 m/s (120 deg/s), and 31.4 m/s (180 deg/s). Increased object velocity produces a decrease in vehicle performance as expected, though the relation of per frame detection scores between the 20.7 m/s and 31.4 m/s cases is misleading. The raw data appears to suggest that despite the platform scoring a lower overall detection rate, detection quality is higher when one occurs. In actuality, the 31.4 m/s case benefits from having a smaller sample size of frames, resulting in a disproportionate weighting of the earliest frames in which the track object begins centered without any requirement for the drone to actuate. During experiment execution, we observed that the vehicle was unable to match the 180 deg/s rotational velocity demand and quickly lost track of the object, resulting in all 0-score frames after the initial detection window.

At the lower speeds of 1 m/s and 10 m/s, the vehicle meets both the initial acceleration demand and the steady state velocity demand to keep the track object in frame center, seen by both perfect detection rates and high per frame scores. Upon moving to 20.7 m/s, the vehicle becomes limited by its rotational acceleration capability despite being able to meet and exceed the 120 deg/s rotational velocity demand. As a result, detection rate suffers due to initial loss of track while the drone approaches steady state velocity but per frame scores remain relatively high because the drone briefly exceeds the velocity of the track object to achieve centering once track is reestablished.

5.4.2 PassThrough

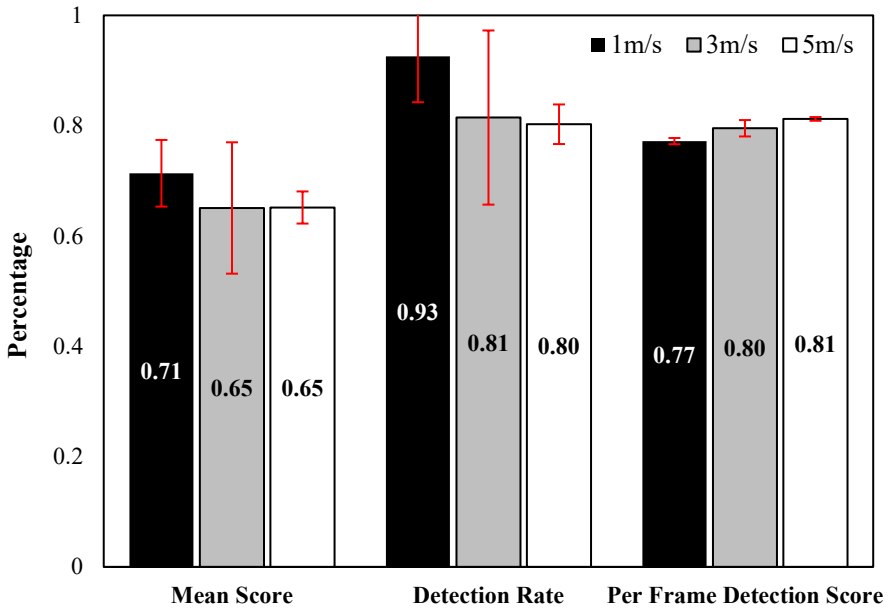


Figure 5.8. PassThrough Aggregate Metric Scores

Given the adversarial nature of the PassThrough trace, we intuitively expect performance to decrease as track object velocity increases. Data collected is supportive of this intuition, with best score and detection rate achieved at 1 m/s. While a minor drop in detection rate occurs between 3 m/s and 5 m/s, the drop is nowhere near as significant as the 11% difference between 1 m/s and 3 m/s. We suspect this to be a side effect of the trace's design, whereby the period between passthrough and trace termination is too short at higher velocities (because the rear segment remains a constant length) to sufficiently penalize the rapid loss of track and failure to reacquire observed at 5 m/s. Given the magnitude of the detection rate change between 1 m/s and the faster iterations and the observed loss of track at both 3 m/s and 5 m/s, we suspect that the minor increase in per frame detection quality is produced by a similar effect to the one observed in the Orbit experiments.

5.4.3 Stop & Go

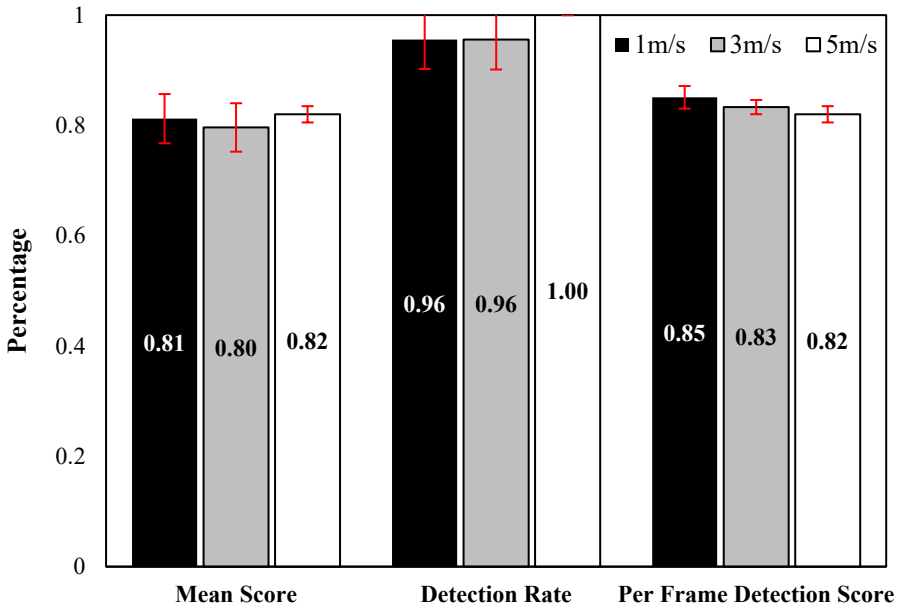


Figure 5.9. Stop & Go Aggregate Metric Scores

Stop & Go is the first trace from the benchmark to produce results suggesting that automation pipeline factors like gain rates can be as impactful as hardware limitations during agility assessments. A purely intuitive approach suggests that the slower the track object moves, the easier the test trace is and the better the performance of the assessed platform should be. While this holds at 1 m/s and 3 m/s, we see the opposite with average score improving on the 5 m/s iterations. In contrast to the slower track object speeds, the drone maintained a perfect detection rate on every iteration conducted at 5 m/s. Interestingly, we still see that per frame scores decrease as track object speed increases, suggesting a possible trade-off between the ability of a vehicle to keep an object in frame through larger dynamic movements and the ability of the vehicle to make finer-grained adjustments to keep an object at a particular position in frame.

5.4.4 RandomWalk

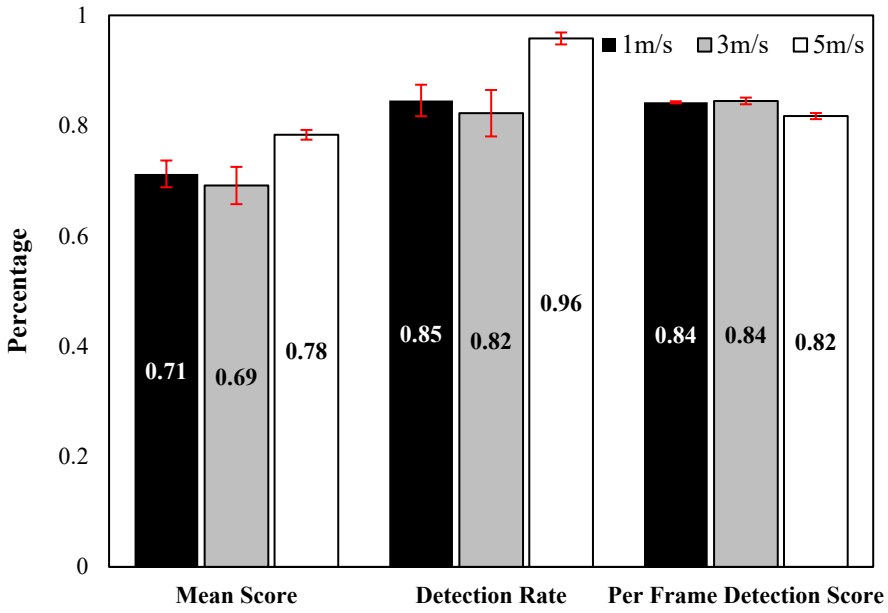


Figure 5.10. RandomWalk Aggregate Metric Scores

Results gathered during the RandomWalk trace provide additional support to the micro-macro motion tradeoff speculation made when discussing Stop & Go. Similarly, we see a slight performance decrease moving from 1 m/s to 3 m/s followed by a performance spike at 5 m/s. Detection rates and per frame detection score remain close for 1 m/s and 3 m/s iterations, with a noticeable improvement in detection rate and degradation in per frame detection quality at 5 m/s.

5.4.5 QuestionMark

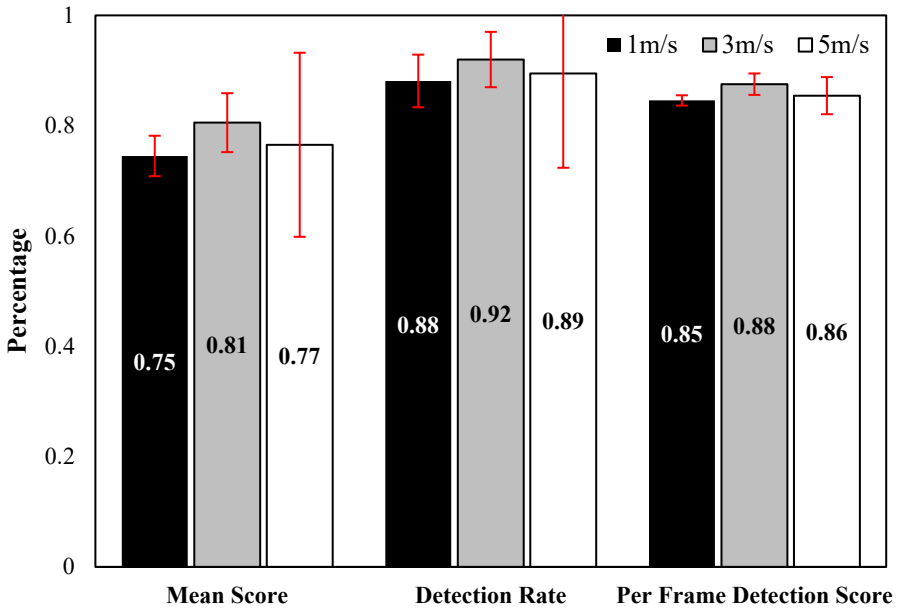


Figure 5.11. QuestionMark Aggregate Metric Scores

Because QuestionMark is primarily a lateral evaluation with a rotational component introduced to assess the ability of the vehicle to transition, the trace uses the 1, 3, 5 m/s track object speeds standard for lateral tests instead of the 10, 20.7, 31.4 m/s speeds used for Orbit. Additionally, the maximum velocity of the assessed platform (15 m/s [26]) is significantly less than the 20.7 and 31.4 m/s speeds, making those iterations poor indicators of the system's capabilities given that it mechanically cannot maintain the tether distance. Optimal performance for the assessed platform during QuestionMark occurred during 3 m/s iterations. Unlike previous traces the data collected during this trace does not initially seem to follow a particular pattern, with performance across average score, detection rate, and per frame detection quality following a rank ordering of 3 m/s, 5 m/s, 1 m/s. Worth highlighting, a major outlier exists in the dataset collected for 5 m/s in which the run score and detection rate registered were roughly halved compared to the other datapoints. If we account for this deviation and exclude the point, the adjusted score for 5 m/s exceeds 3 m/s with detection rate and detection quality demonstrating the above-mentioned trade-off. Similarly to the Spiral experiments, significant lateral oscillation by the assessed vehicle was observed during the rotational portion of the trace at 1 m/s but not at 3 m/s or 5 m/s.

5.4.6 Spiral

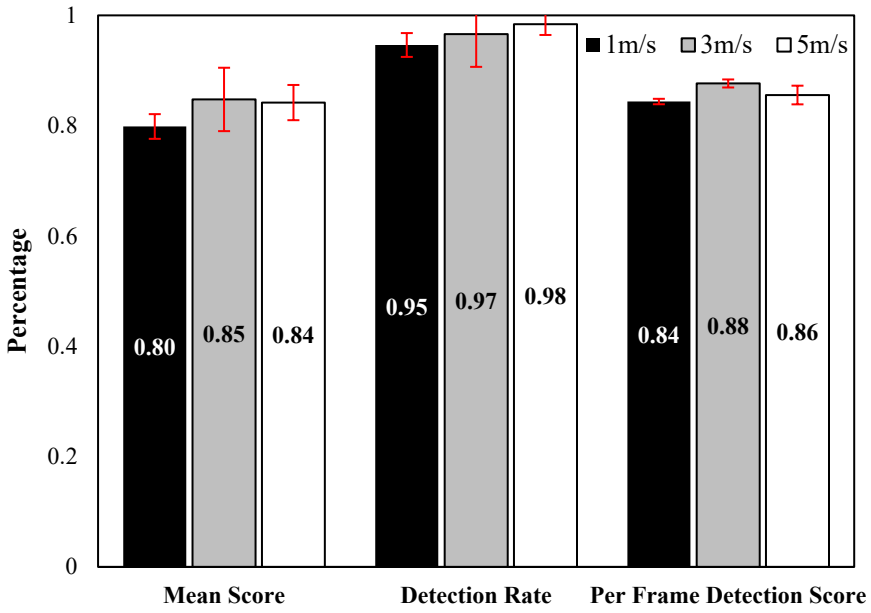


Figure 5.12. Spiral Aggregate Metric Scores

Similarly to QuestionMark, Spiral is primarily a lateral test with the rotational component of the object introduced to assess the vehicle's stability when moving in multiple dimensions simultaneously. As a result, Spiral also uses the 1, 3, 5 m/s track object velocities. The same issues outlined in Section 5.4.5 with vehicle maximum velocity and tether distance preclude using the 10, 20.7, 31.4 m/s velocities used for Orbit. The data collected for Spiral shows a slight departure from the patterns observed for Stop & Go and RandomWalk, which we speculate is related to the addition of rotational motion demands not present in the aforementioned traces. The 1 m/s case produces the worst average score, detection rate, and per frame detection quality, with noticeable improvements in the first two metrics occurring at 3 m/s and 5 m/s. This suggests that the automation pipeline is potentially over-tuned to faster cases, supported by the observation of repeated lateral oscillation by the assessed vehicle over the track object during 1 m/s iterations that were not present during either 3 m/s or 5 m/s. The faster velocity iterations produced the same detection rate/detection quality trade-off observed in Stop & Go and RandomWalk.

5.4.7 Hourglass

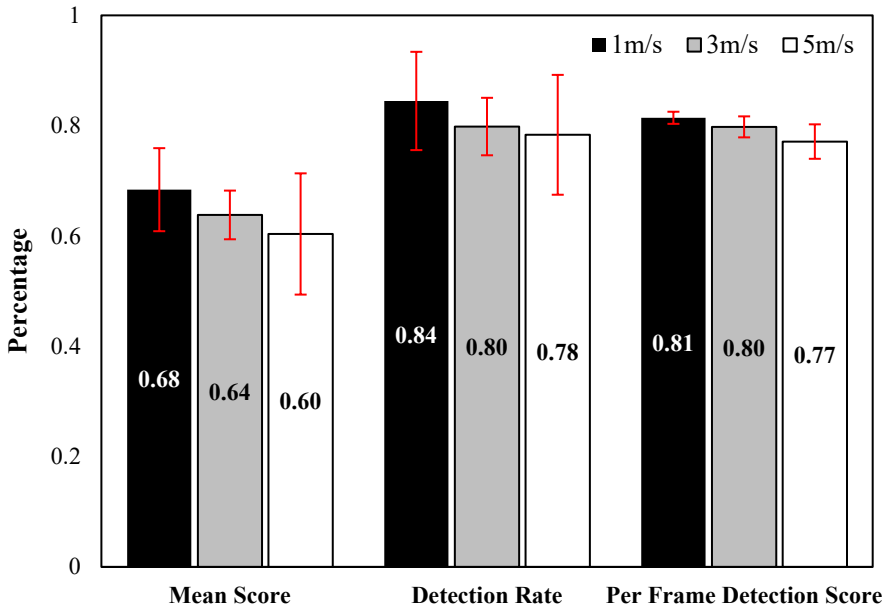


Figure 5.13. Hourglass Aggregate Metric Scores

The Hourglass dataset shows similar patterns to the PassThrough dataset, matching expectations given that Hourglass is effectively composed of multiple passthrough maneuvers followed by reacquisition periods. As velocity of the track object increases, we observe a steady decrease in average score, average detection rate, and per frame detection quality. A major positive outlier exists in the 5 m/s datapoints, with a single iteration producing a perfect detection rate and an iteration score on the order of 50% better than the other points in the set. If we adjust the 5 m/s aggregates to exclude this point, we see an even greater decrease in average score and detection rate moving from 3 m/s to 5 m/s that matches the empirically observed increase in track loss time prevalent as track object velocity increased.

	Mean Score	Median Score	Std Deviation	Avg Detection Rate	Per Frame Avg Score
1 m/s	0.684	0.631	0.075	0.845	0.814
3 m/s	0.638	0.620	0.044	0.798	0.798
5 m/s	0.604 (0.549)	0.553 (0.549)	0.111 (0.008)	0.783 (0.730)	0.771 (0.753)

Table 5.1. Hourglass Aggregate Metric Scores with Outlier Adjusted Values

5.5 Insights from Benchmark Execution

While lack of additional platforms prevents relative comparison in this work, we note two interesting trends that emerge in the dataset collected during benchmarking. The first of these is the counterintuitive tendency for overall average score to increase as the velocity of the track object goes up. This trend is observed in the results gathered for RandomWalk, Spiral, the adjusted results in QuestionMark, and the recreation of the original RoboMaster RandomWalk when additional latency is included, while Stop & Go shows roughly equivalent results for average score across all three measured velocities. We conjecture that this results from the configurable parameters in the automation pipeline being better tuned, or perhaps over-tuned, to settings in which a track object is moving more quickly, such as the 3-5 m/s range. If tuned to higher rate objects, the magnitude of changes generated by the pipeline could conceivably result in overshooting the desired position and pose. The lateral oscillation observed only during 1 m/s iterations in four of the five mentioned traces provides initial supporting evidence of pipeline tuning as a contributing factor. Accordingly, we include further investigation of this effect as a promising concept for future work.

The second possible trend, which we refer to as the micro-macro tradeoff, is less counterintuitive but also less prevalent in the data and will require further experimentation for conclusive results. The underlying concept suggested is that as velocities increase, detection rate improves to a point at the cost of detection quality. Stated another way, while the drone manages to keep the track object in frame a greater percentage of the time when moving quicker itself, its ability to make the fine-grained corrections important for maintaining center decreases. In Stop & Go and RandomWalk as track object speed increases, detection rates increase while per frame detection score decreases. This effect is also partially visible in the adjusted results for QuestionMark and Spiral, with the 1 m/s iterations failing to adhere to the trend. Notably, both QuestionMark and Spiral experienced significant lateral oscillation during the 1 m/s iterations, which could conceivably distort the resulting detection rate and per frame scores. We avoid any conclusive determinations on this effect but recommend it as another topic for future work.

Chapter 6

Future Work & Conclusions

6.1 Conclusion

This thesis presents the concept of SensorTwin, an adaptation of the pre-existing digital twin concept that utilizes simulation to produce a virtual reality effect for automated vehicles. This virtual reality effect enables physical vehicle manipulation through controlled, deterministic, but artificial sensor feedback. We demonstrate a concrete implementation of the concept using the SteelEagle automated drone framework and discuss candidate use cases in automated drone development and performance benchmarking. We perform relative comparisons to the results produced by fully functional vision models in the earlier efforts of Bala et al. [2], incorporating SensorTwin to garner a sense of where a theoretical performance bound might lie as a proof of concept. We conclude by introducing the Autonomous Tracking Agility Benchmark suite as a particular example of performance benchmarking, evaluating an existing pipeline-vehicle pair, and providing baseline results for future relative comparison.

6.2 Future Work

The initial work on SensorTwin and the derived conclusions lend to multiple natural paths for future work:

- **Benchmarking Expansion to Vertical Characteristics.** As mentioned in Section 5.1, the current lack of a vertical separation estimation mechanism in SteelEagle's autonomy pipeline precluded completion of the full benchmark suite developed as part of this work. Five additional traces aimed at evaluating drone motion in the vertical plane, both in isolation and in conjunction with rotational and lateral motion, are available via the trace generator tool. Potential areas of exploration are the impact of climb and descent maneuvers on simultaneous lateral or rotational maneuvers and an assessment of what track object maneuvers including a vertical component produce similar challenges for automated tracking as purely lateral passthrough.
- **Additional Platform Benchmarking.** This work constrained its scope to a single COTS multicopter, the Parrot Anafi Gov as a proof of concept of SensorTwin. Benchmarking additional vehicles across a variety of manufacturers would enable relative comparison of popular platforms and potentially the ability to analyze trends based on various classifications of vehicles, such as by price-point, weight, or manufacturer-specified flight performance.
- **Cognitive Function Expansion.** We examined two common cognitive functions, object detection-enabled tracking and, briefly, object avoidance. Specifically, we examined

performance as a function of specific implementations of either task. Comparison to other approaches within the same type of task, such as to LiDAR-based avoidance or optical flow-based tracking, could produce results enabling a relative comparison across multiple algorithms in the same family of task. Alternatively, SensorTwin could be used to support exploration into segmentation and localization style tasks.

- **Pipeline Metric Tuning Analysis.** To ensure a coherent dataset, we held the configurable characteristics of the autonomy pipeline, namely velocity maximums and gain rates, constant. Evidenced by the results in Section 5.5, our selected settings are certainly not optimal in all cases and may not be optimal in any. Experimentation with varied configuration settings across a set of expected track object speeds would be useful in determining more performant configurations in both general and object movement trait specific settings.
- **Multi-Objective Optimization.** The scoring function utilized during this work penalized performance only based on distance of the track object from center frame despite higher-level task code implementing a tether distance. The unified simulation space in SensorTwin makes calculating absolute distance between the physical vehicle and a simulated object trivial. Experimentation with a multi-objective scoring function that penalizes both center frame distance and tether distance differences could provide useful insights into how pipeline tuning and task logic alter pipeline preferences to actuate the gimbal versus actuating the drone
- **Telemetry Conversion Smoothing.** Telemetry data sourced from real sensors is inherently noisy. The current implementation of SensorTwin does not address the moment-to-moment variance, naively binding the simulated vehicle's pose to the most recent telemetry frame received. Smoothing telemetry data prior to applying it to the simulated vehicle would further increase the fidelity of results and potentially cause a shift in the theoretic performance bound for a task produced by SensorTwin evaluation.
- **Swarm Control Logic Validation.** While this work focused heavily on performance benchmarking, the control logic validation component of SensorTwin is arguably more important when actively developing a system. As swarms become increasingly prevalent, so too will task logic involving multiple vehicles. SensorTwin provides a mechanism for integrating both real and simulated vehicles into testing environments, allowing a progressive graduation in the number of physical vehicles tested as confidence in the underlying code grows.

Bibliography

[1] "Definition of AGILE," *Merriam-webster.com*, 2018. https://www.merriam-webster.com/dictionary/agile. Last accessed on 26 Apr. 2026.

[2] M. Bala et al., "The OODA Loop of Cloudlet-Based Autonomous Drones," in 2024 IEEE/ACM Symposium on Edge Computing (SEC), Rome, Italy, 2024, pp. 178-190, doi: 10.1109/SEC62691.2024.00022.

[3] T. B. Sheridan and W. L. Verplank, "Human and Computer Control of Undersea Teleoperators," MIT Man-Machine Systems Laboratory, Cambridge, MA, Tech. Rep., 1978.

[4] R. Parasuraman, T. B. Sheridan, and C. D. Wickens, "A model for types and levels of human interaction with automation," in IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 30, no. 3, pp. 286-297, May 2000, doi: 10.1109/34.845719.

[5] H. -M. Huang, "Autonomy Levels For Unmanned Systems (ALFUS) framework, volume I: terminology version 2.1," National Institute of Standards and Technology, Tech. Rep., Jan. 2008.

[6] Small Unmanned Aircraft Systems, *Code of Federal Regulations*, Title 14, Part 107, 2026.

[7] Federal Aviation Administration, *14 CFR Part 108 Unmanned Aircraft Systems (UAS) Transporting or Dropping Hazardous Materials Safety Risk Assessment (SRA),* Advisory Circular No. 108-4 (Draft), Mar. 06, 2026.

[8] PX4 Development Team, "Multicopter PID Tuning Guide (Manual/Advanced)," PX4 Guide (main), Dronecode. [Online]. Available: https://docs.px4.io/main/en/config_mc/pid_tuning_guide_multicopter, Last accessed on 28 Apr. 2026.

[9] Jiang, Yupeng & Deng, Yao & Schroder, Sebastian & Liang, Linfeng & Gambhir, Suhaas & James, Alice & Seth, Avishkar & Pirrie, James & Zhang, Yihao & Zheng, Xi. (2025). "A Step-by-Step Guide to Creating a Robust Autonomous Drone Testing Pipeline." 10.48550/arXiv.2506.11400.

[10] General Electric, "GE Launches the Next Evolution of Wind Energy Making Renewables More Efficient, Economic: the Digital Wind Farm," GE News, Schenectady, NY, USA, Press Release, May 19, 2015. [Online]. Available: https://www.ge.com/news/press-releases/ge-launches-next-evolution-wind-energy-making-renewables-more-efficient-economic. Last accessed 26 Apr. 2026.

[11] Siemens AG, "News for the Digital Enterprise," *News for the Digital Enterprise*, no. 2, Apr. 2023. [Online]. Available: https://assets.new.siemens.com/siemens/assets/api/uuid:488375d6-7f15-46e1-bd6c-9caa290e6bee/siemens-news-for-the-digita1-enterprise-en-2-2021-product-news.pdf. Last accessed 26 Apr. 2026.

[12] Bala, Mihir (2025). Towards Fully-Autonomous Ultralight Drones. Carnegie Mellon University. Thesis. https://doi.org/10.1184/R1/29304797.v1

[13] Living Edge Lab, "SteelEagle Architecture," *SteelEagle*, GitHub Pages. [Online]. Available: https://cmusatyalab.github.io/steeleagle/reference/intro, Last accessed 28 Apr. 2026.

[14] M. S. Whalley, "Development and Evaluation of an Inverse Solution Technique for Studying Helicopter Maneuverability and Agility," Aeroflightdynamics Directorate, U.S.

Army Aviation Systems Command, Ames Research Center, Moffett Field, CA, USA, NASA Tech. Memo. NASA TM-102889 / USAAVSCOM Tech. Rep. 90-A-008, Jul. 1991.
[15] J. Verbeke and J. De Schutter, "Experimental maneuverability and agility quantification for rotary unmanned aerial vehicles," *Int. J. Micro Air Veh.*, vol. 9, no. 4, pp. 294–307, Dec. 2017, doi: 10.1177/1756829317736204.
[16] A. Fotouhi, M. Ding and M. Hassan, "Understanding autonomous drone maneuverability for Internet of Things applications," *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Macau, China, 2017, pp. 1-6, doi: 10.1109/WoWMoM.2017.7974336.
[17] Mihir Bala, Thomas Eiszler, Xiangliang Chen, Jan Harkes, James Blakley, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2024. Democratizing Drone Autonomy via Edge Computing. In Proceedings of the Eighth ACM/IEEE Symposium on Edge Computing (SEC '23). Association for Computing Machinery, New York, NY, USA, 40–52. https://doi.org/10.1145/3583740.3626614
[18] D. Iqbal and B. Buhnova, “Digital Twin Design for Autonomous Drones,” in 2024 19th Conference on Computer Science and Intelligence Systems (FedCSIS), Belgrade, Serbia: IEEE, 2024, pp. 119-130. doi: 10.15439/2024F6765.
[19] I. Al-Darraji, F. Q. Khan, T. T. A. M. Hussain, G. K. Georgiou, H. A. A. A. A. Al-Sadi, and A. G. Ismaeel, "Integrating Drones With Digital Twins for Aerial Remote Sensing," *Radio Science*, vol. 60, no. 8, Aug. 2025, Art. no. e2023RS007700, doi: 10.1029/2023RS007700.
[20] N. Grigoropoulos and S. Lalis, "Simulation and Digital Twin Support for Managed Drone Applications," in *Proc. 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Sep. 2020, pp. 1–8, doi: 10.1109/DS-RT49257.2020.9213676.
[21] A. McClellan, J. Lorenzetti, M. Pavone, and C. Farhat, “A physics-based digital twin for model predictive control of autonomous unmanned aerial vehicle landing,” in *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2022, pp. 1-8.
[22] A. K. Ghosh, A. S. Ullah, R. Teti, and A. Kubo, "Developing sensor signal-based digital twins for intelligent machine tools," *Journal of Industrial Information Integration*, vol. 24, p. 100242, Dec. 2021, doi: 10.1016/j.jii.2021.100242.
[23] J. R. Olson and M. W. Scott, "Helicopter Design Optimization for Maneuverability and Agility," in *Proc. 45th Annual Forum of the American Helicopter Society*, Boston, MA, USA, May 1989, pp. 22–24.
[24] GenPac Drones, “Parrot Anafi USA (Gov),” https://www.genpacdrones.com/product/parrot-anafi-usa-gov/, Last accessed April 28, 2026.
[25] DJI, “Robomaster S1,” https://www.dji.com/robomaster-s1, Last Accessed April 28, 2026.
[26] Parrot Drones SAS, *ANAFI USA GOV User Manual*, ver. 6.10.6.0. Paris, France: Parrot Drones SAS, Jan. 2026. [Online]. Available: https://www.parrot.com/assets/s3fs-public/2023-01/ANAFI-USA-GOV-user-manual.pdf, Last accessed on 28 Apr. 2026.