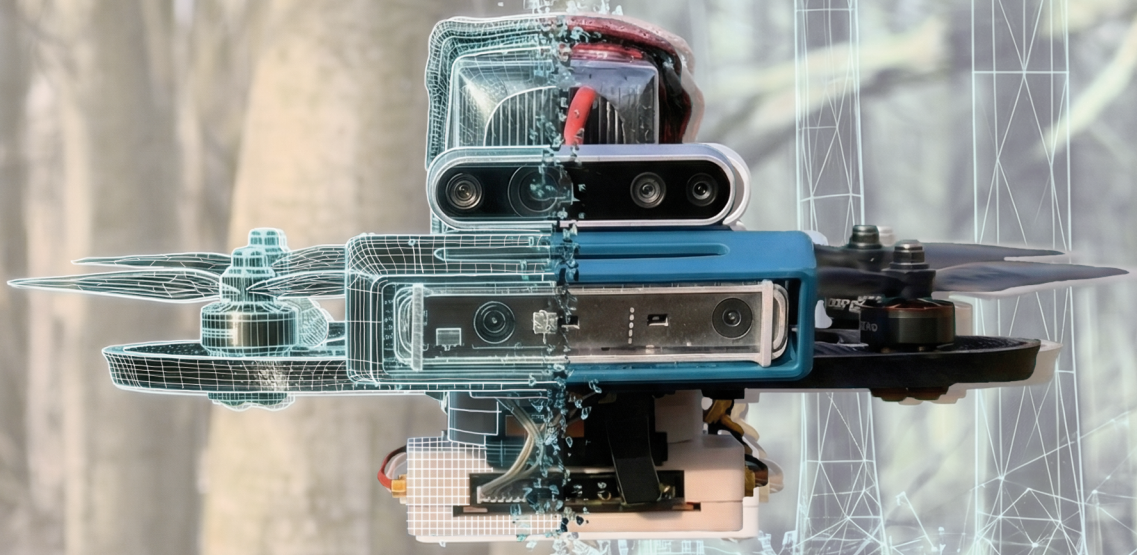
Learning Visual Navigation for Drones in Cluttered Environments

Hang Yu



LEARNING VISUAL NAVIGATION FOR DRONES IN CLUTTERED ENVIRONMENTS

LEARNING VISUAL NAVIGATION FOR DRONES IN CLUTTERED ENVIRONMENTS

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, prof. dr. ir. H. Bijl,
chair of the Board for Doctorates
to be defended publicly on
Wednesday, 13 May 2026, 17:30

by

Hang YU

This dissertation has been approved by the (co)promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. G.C.H.E. de Croon,	Delft University of Technology, *promotor*
Dr. ir. C. de Wagter,	Delft University of Technology, *copromotor*
Independent members:	
Prof. dr. F. A. Oliehoek,	Delft University of Technology
Prof. dr. K. Masania,	Delft University of Technology
Dr. ir. E. van Kampen,	Delft University of Technology
Prof. dr. K. Alexis,	Norwegian University of Science and Technology, Norway
Dr. G. Spigler	Tilburg University, Netherlands

Keywords: Autonomous drones, vision-based obstacle avoidance, reinforcement learning, sim-to-real transfer, latent representation

Printed by: Ipskamp Printing

Cover by: Hang Yu

ISBN 978-94-6518-312-1

An electronic copy of this dissertation is available at
https://repository.tudelft.nl/.

CONTENTS

SUMMARY

Autonomous drones are increasingly deployed in cluttered, GPS-denied environments such as forests, caves, industrial facilities, and urban canyons. In such settings, visual obstacle avoidance is the key enabler for safe and agile flight. Yet, existing methods still struggle with three fundamental challenges: the lack of a unified evaluation framework, the difficulty of balancing safety and agility, and the significant gap between policies trained in simulation and their deployment on real drones. This dissertation addresses these challenges under a unifying research objective: to develop efficient learning-based algorithms for safe and agile onboard navigation in cluttered environments, while providing systematic tools to evaluate and improve their performance.

The first contribution of this thesis is the design of a high-fidelity benchmarking suite for vision-based obstacle avoidance, named *AvoidBench*. Built on Unity for photorealistic rendering and a quadrotor dynamics model for realistic flight behaviour, the suite provides a collection of simulation environments with varying levels of clutter and structure. To enable systematic comparison of algorithms, the benchmark introduces two complementary classes of metrics. Environment metrics, such as traversability, quantify the difficulty of a given scene. Flight performance metrics, including success rate, relative end distance, average goal velocity, and path excess factor, characterize safety and agility from the perspective of the navigating drone. A standardized experimental protocol and software interface allow both model-based and learning-based methods to be evaluated under identical conditions. By benchmarking several state-of-the-art approaches, *AvoidBench* exposes their strengths and weaknesses and establishes a common reference for future research on drone obstacle avoidance.

Building on this platform, the second contribution introduces *MAVRL*, a reinforcement learning-based obstacle avoidance algorithm that explicitly accounts for environment complexity. Instead of prescribing a fixed flight speed, the policy is trained to adaptively adjust its velocity: it accelerates in simple environments and decelerates in dense, cluttered ones. This behaviour is encouraged by performing domain randomization over environment complexity during training and by defining rewards that reflect the safety–agility trade-off. To overcome partial observability and the lack of an explicit mapping module, the method uses a memory-augmented latent representation of depth images. A recurrent variational autoencoder encodes a sequence of depth maps into a compact latent state that is trained to reconstruct both past and current depth, thereby preserving information about the recent scene structure. This latent state, combined with the drone state and navigation target, forms the input to a Proximal Policy Optimization (PPO) agent that outputs acceleration commands.

Experiments in *AvoidBench* show that *MAVRL* achieves a superior Pareto frontier between safety and agility across a wide range of environment complexities. Indoor real-world experiments demonstrate that the learned policy can be deployed on a physical drone with minimal adaptation.

The third contribution, *Depth Transfer*, addresses the simulation-to-real (sim-to-real) gap that arises when deploying vision-based obstacle avoidance policies in the real world. Two main sources of discrepancy are considered: the difference in dynamics and control between simulator and real platform, and the difference between ideal depth in simulation and noisy stereo depth in reality. To reduce the dynamics gap, a reference generator produces high-level commands that can be consistently tracked by both simulated and real drones, thereby decoupling the learned policy from platform-specific low-level controllers. To reduce the perception gap, a domain adaptation framework is proposed in which a depth encoder is first trained in simulation on ground-truth depth and then refined so that real stereo depth images are mapped into an aligned latent space. This enables direct policy transfer without further tuning of the control network. Ablation studies show that *Depth Transfer* substantially improves the performance of an Isaac Gym-trained policy when evaluated in *AvoidBench*, and dense forest flight tests confirm robust obstacle avoidance in challenging outdoor environments. Compared with direct policy fine-tuning, adapting only the encoder is both more sample-efficient and safer, since the control behaviour learned in simulation is preserved.

Finally, the fourth contribution explores how ideas from biological sensorimotor control can improve the efficiency and flexibility of training obstacle avoidance policies. Inspired by the hierarchical perception–action organization found in animals, a novel architecture is proposed that separates high-level waypoint generation from low-level control. The high-level module receives depth maps, target information, and drone state, and outputs short-horizon waypoints. The low-level module tracks these waypoints and generates collective thrust–body rate commands (CTBR). To further stabilize perception, a gimbal-based visual input system is introduced that keeps the camera approximately level, reducing motion blur and viewpoint disturbance during agile manoeuvres. Unlike previous end-to-end RL methods, the low-level controller is trained using differentiable dynamics and gradient-based optimization instead of trial-and-error reinforcement learning. Experiments show that the gimbal-stabilized perception significantly enhances robustness in aggressive flight, and that the hierarchical architecture achieves performance comparable to state-of-the-art end-to-end methods with substantially reduced training time. Moreover, because the low-level controller is task agnostic, it can be reused for other navigation tasks, such as exploration or tracking, by retraining only the high-level policy.

In summary, this dissertation contributes a unified evaluation framework, an adaptive-speed reinforcement learning algorithm, a sim-to-real transfer method for depth-based navigation, and a bio-inspired hierarchical perception–action architecture for efficient policy learning. Together, these elements advance the state of the art in learning-based visual navigation for drones in cluttered environments. They demonstrate that it is possible to balance safety and agility, to transfer complex policies from

simulation to reality, and to accelerate training by exploiting structure in both the task and the underlying dynamics. These results provide a solid foundation for future work on lightweight, robust, and versatile autonomous drones that can operate reliably in the complex environments of the real world.

1

INTRODUCTION

Well begun is half done.

Aristotle

1

Deep within the irradiated ruins of the Chernobyl Nuclear Power Plant's Unit 5, a drone navigates a labyrinth of twisted steel and darkness [1]. Deprived of GPS signals and human access, the aircraft relies entirely on onboard LiDAR and collision tolerant algorithms to map the hazardous interior. Simultaneously, amidst the dense, unstructured canopies of a North American forest, a Skydio drone weaves through a complex web of branches at high speed [2]. Assisting police in a search and rescue operation, it executes split second trajectory adjustments to avoid obstacles that would be invisible to a remote pilot's video feed.

These two distinct scenarios, the Flyability Elios inspecting critical infrastructure in a GPS denied confinement and the Skydio autonomously traversing a dynamic natural environment, illustrate a fundamental paradigm shift in Unmanned Aerial Vehicle (UAV) technology. We are witnessing the evolution of drones from passive, remotely piloted "flying cameras" into active, intelligent robotic agents capable of independent spatial awareness and decision making.

Historically, the operational envelope of UAVs was strictly limited by the availability of Global Navigation Satellite Systems (GNSS) and the skill of the human operator. In "cluttered" environments, such as collapsed buildings, dense forests, or urban canyons, signal interference and loss often resulted in mission failure or catastrophic crashes. The inability to autonomously perceive and react to the physical world has long been the primary bottleneck preventing the widespread deployment of UAVs in complex disaster relief and industrial applications.

Autonomous obstacle avoidance addresses this critical limitation. It is not merely a safety feature but a complex synthesis of multi-modal sensing (including stereo vision, millimeter-wave radar, and LiDAR) [3, 4], Simultaneous Localization and Mapping (SLAM) [5–7], and real-time path planning algorithms [8–10]. As the capabilities of drones increase, more scientific questions arise: how to ensure safe flight in complex environments, how to reduce the size of drones to navigate narrow passages, and how to make drones fly fast enough to save time, etc. The following sections will discuss the challenges, previous work, and research questions related to drone navigation in cluttered environments.

1.1. CHALLENGES AND PREVIOUS WORK

Autonomous navigation in cluttered, GPS-denied environments requires a delicate balance between perception accuracy, computational efficiency, and flight agility. The core challenge lies in enabling a Micro Air Vehicle (MAV) to perceive its surroundings via onboard sensors, plan collision-free trajectories, and execute control commands with millisecond-level latency. Existing literature approaches this problem primarily through two distinct paradigms: Model-based approaches, which rely on explicit geometric reconstruction and kinematic planning, and Learning-based approaches, which leverage data-driven policies to map sensory inputs directly to control actions.

(a)

(b)

Figure 1.1: (a) Flyability Elios drone investigates the condition of Chernobyl's Unit 5 reactor core [1]. (b) Skydio drone autonomously navigates through a dense forest during a search and rescue mission [2].

1.1.1. MODEL-BASED APPROACHES

Classical visual navigation for multirotors has traditionally followed a *model-based pipeline*: perception builds a local or global map of the environment, a planner searches for a collision-free and dynamically feasible trajectory, and a feedback controller tracks this trajectory using an explicit dynamics model of the vehicle. In field robotics, this paradigm has been extensively exploited in subterranean and GPS-

denied settings, where aerial and legged robots jointly explore large-scale underground networks. Graph-based subterranean exploration methods maintain a topological graph of frontiers and unexplored regions and use heuristic graph search, often variants of A*, to select safe, informative exploration paths for the robot team [11–13]. Related work combines these global planners with motion-primitive based local exploration that repeatedly expands a tree or lattice in the state space using dynamically feasible primitives, enabling agile flight through unknown 3D environments [14].

For local navigation in cluttered environments, optimization-based trajectory planners have become particularly prominent. Instead of searching over discrete paths, these methods formulate trajectory generation as a constrained optimization problem, in which collision costs, smoothness, and dynamic feasibility are encoded directly in a continuous cost functional. Ego-Planner is a representative example: it maintains a local map and optimizes a spline-based trajectory under collision constraints, achieving high responsiveness without relying on a globally consistent Euclidean signed distance field (ESDF) [15]. Similar ideas are embedded in high-performance simulation and control frameworks, where GPU-accelerated physics engines such as Isaac Gym, Isaac Lab, Aerial Gym, and Flightmare provide realistic quadrotor dynamics and allow real-time evaluation of candidate trajectories or controllers under diverse environmental conditions [16–20].

Another important branch of model-based navigation uses motion-primitive planning as its core abstraction. In this setting, the planner operates over a library of short, dynamically feasible motion segments that are precomputed to respect the quadrotor's dynamics and actuation limits. Search-based planners then select and sequence these primitives to explore unknown environments or reach local goals while avoiding obstacles [14]. This discretization transforms kinodynamic planning into graph search in the state space and allows planners to explicitly trade off exploration gain, goal-directed behavior, and safety. Building on this framework, recent systems integrate motion primitives with learned perception modules: deep collision predictors are used to assess the safety of each primitive, while the underlying search and dynamics constraints remain fully model-based [21, 22].

Model-based planning is typically coupled with well-engineered control stacks that track the planned trajectories at high frequency. Quadrotor simulators and middleware such as ROS–Gazebo-based platforms, RotorS, and AirSim provide modular models of UAV dynamics, actuators, and sensor noise, and are widely used to design and evaluate low-level controllers, planners, and perception pipelines in realistic scenarios [23–27]. Mapping frameworks such as OctoMap and Voxblox further enable incremental 3D occupancy and signed-distance representations that are well-suited for model-based planning and collision checking [28, 29].

However, precisely because these pipelines are carefully engineered, they tend to become systemically complex and heavily constrained. A typical stack may include state estimation, dense or semi-dense mapping, global and local planning, trajectory optimization, and low-level control, each with its own assumptions, parameters, and failure modes [10]. Constraints introduced at each layer, for example on map resolu-

tion, trajectory smoothness, or safety margins, can accumulate and make the overall system overly conservative and difficult to tune or adapt to new platforms and environments. At the same time, explicit mapping is almost always a core component: maintaining probabilistic 3D maps or ESDF on board (for instance with OctoMap, Voxblox, or similar structures) consumes substantial memory and computational resources, which is particularly problematic for small UAVs with limited onboard compute and power budgets [26, 28, 29]. In dense clutter, high map resolution is required to resolve narrow gaps and thin obstacles, further increasing the load, even when the task may only require short-horizon, reactive avoidance rather than full global mapping. These limitations motivate the exploration of more compact learning-based or hybrid architectures that reduce system complexity, relax some of the hand-crafted constraints, and potentially bypass heavy explicit mapping while still ensuring safe, agile navigation in cluttered environments.

1.1.2. LEARNING-BASED APPROACHES

In parallel to classical model-based pipelines, learning-based approaches have rapidly advanced visual navigation for drones in cluttered environments. A first line of work focuses on *imitation learning* from expert policies. In agile flight, expert planners or optimal-control solvers are used in simulation to compute near-optimal trajectories with access to full state and map information; then, convolutional or transformer-based networks are trained to imitate these expert actions using only visual observations, enabling high-speed, vision-based flight in natural environments such as forests and urban canyons [30, 31]. Recent work further combines imitation with deep reinforcement learning (RL), using expert demonstrations to bootstrap RL-based training for vision-based agile flight and thereby reducing exploration burden while retaining the performance benefits of RL fine-tuning [32, 33]. In this way, the literature gradually transitions from purely supervised learning to reinforcement-learning-based navigation.

Going beyond pure imitation, deep RL has shown that model-free visuomotor policies can match or even surpass traditional optimal control baselines in agility-oriented tasks. In drone racing, simulation-based policies have reached and exceeded champion-level human performance, outperforming carefully engineered pipelines that combine trajectory optimization with model predictive control (MPC) [34, 35]. Related comparisons between optimal control and RL in autonomous racing show that RL can exploit subtle interactions between perception and dynamics to push the limits of performance when sufficient simulated experience is available [35]. In cluttered environments, RL has been extended to perception-aware policies that explicitly account for the reliability of the perception pipeline when choosing aggressive maneuvers, improving safety at high speed [36]. Complementary work learns speed adaptation policies that modulate the forward velocity of a quadrotor based on obstacle density, allowing the vehicle to decelerate in dense clutter and accelerate in open space [37]. These works illustrate a progression from general learning-based controllers toward increasingly sophisticated RL-based navigation policies.

A substantial body of research addresses the challenge of high-dimensional visual input through *representation learning*. Instead of feeding raw depth or RGB images directly into the control policy, these methods first learn compact latent encodings that preserve collision-relevant information. Task-driven compression approaches, for example, learn depth-image encoders optimized specifically for collision prediction and navigation, showing clear advantages over generic autoencoders with the same latent dimensionality [38]. On legged robots, similar ideas are used to learn state representations and navigation policies jointly, allowing robots to move through cluttered and dynamic environments using a compact visual latent state rather than full sensor streams [39]. Beyond local controllers, learned latent spaces have been employed as state spaces for latent-space motion planning, where sampling-based planners operate directly in a low-dimensional learned manifold instead of raw sensor or configuration spaces [40]. In many of these works, representation learning is tightly coupled with RL: the latent space is optimized to be predictive and approximately Markovian, which simplifies the downstream RL problem [41, 42].

Handling partial observability and temporal context is another key theme in learning-based visual navigation. Recurrent architectures such as long short-term memory (LSTM) networks [43] and other memory modules are widely used to model temporal dependencies, often by compressing an observation history into an internal belief state that supports prediction and control. Memory-based deep RL for UAV navigation shows that recurrent policies can outperform purely feedforward ones in environments with limited or intermittent observations [44]. Similar principles appear in navigation systems that combine recurrent latent encoders with RL, where a history of depth or image features is compressed into a predictive latent state that is more informative for downstream control than single-frame observations [38, 39, 45]. Beyond recurrent memory, world models explicitly learn a latent dynamics model that predicts future latent states (and sometimes rewards) from past observations and actions, enabling planning or policy learning via imagined rollouts in the learned latent space [46–50]. Thus, recurrent networks and world-model-based approaches are increasingly integrated into RL formulations for navigation.

Learning-based navigation is not limited to fully supervised or RL setups; self-supervised approaches also play an important role and often act as a precursor or complement to reinforcement learning. BADGR learns a predictive model of collision probabilities from self-supervised driving data and uses it for navigation, avoiding the need for manually labeled trajectories [51]. On small aerial platforms, self-supervised monocular distance learning and learning to fly by crashing demonstrate that useful depth and collision cues can be extracted directly from monocular video and from crash events, enabling low-cost obstacle avoidance without dense labels [52, 53]. Related work also validates self-supervised monocular distance learning in space robotics through experiments conducted on the International Space Station (ISS), where a free-flying SPHERES platform used stereo vision as a trusted signal while learning monocular distance estimation during operation [54]. Complementary work in neuromorphic vision uses event cameras and biologically inspired motion-vision processing to detect gaps and navigate through dense clutter, offering an alternative sensing and

processing paradigm for agile flight [55, 56]. Many recent systems combine these self-supervised perception models with RL-based decision making, underscoring the trend from general learning-based components toward fully RL-driven navigation stacks [57–59].

Overall, learning-based approaches, ranging from imitation and self-supervision to deep reinforcement learning, have demonstrated that data-driven visuomotor policies can complement and, in some regimes, rival classical model-based pipelines. At the same time, they introduce new challenges related to training stability, sample complexity, and deployment robustness. These challenges are particularly pronounced for RL-based methods and motivate a more focused discussion on the specific obstacles faced by reinforcement learning in this domain.

1.1.3. Challenges in Reinforcement Learning

Despite the impressive progress of learning-based visual navigation, RL for drones still faces two fundamental challenges: bridging the sim-to-real gap and achieving sufficient training efficiency for complex vision-based tasks.

Sim-to-real gap RL for aerial robots is almost always trained in simulation, where large-scale interaction can be collected safely and cheaply. However, transferring such policies to physical drones is hampered by mismatches between simulated and real dynamics, sensing, and visual appearance. Differences in mass, motor response, aerodynamics, sensor noise, or external disturbances such as wind can severely degrade the performance of a policy that was optimized under idealized simulation assumptions [60, 61]. On the perception side, rendered images often fail to capture fine geometric details, motion blur, and illumination changes present in real environments, leading to substantial visual domain shift [62–64]. From an information-theoretic viewpoint, these shifts can be interpreted as changes in the input bottleneck, which are known to harm generalization if not explicitly addressed [65, 66].

A wide range of techniques has been proposed to mitigate this sim-to-real gap. Domain randomization exposes the policy to a distribution of simulated dynamics and appearances in the hope that the real world becomes just another sample from this distribution [60, 62]. Domain adaptation and domain generalization further seek to align feature distributions or learn domain-invariant representations using adversarial or discrepancy-based objectives [67–74]. These approaches have enabled zero-shot sim-to-real transfer for simple vision-based flight from purely synthetic training images [63, 64], and robust policies that combine simulated and real data [61]. Yet, they still struggle when the real deployment conditions deviate significantly from the assumed parameter ranges or from the visual variability seen during training.

Recent work therefore explores more structured ways of closing the gap by tightening the coupling between simulation and real dynamics. Differentiable or hybrid simulation frameworks refine dynamics models using real flight data and then backpropagate through these models to rapidly adapt the policy in an online fashion [75].

By learning residual dynamics on top of a simple physics model and embedding them in a differentiable simulator, such approaches can adapt to unmodeled effects such as payload changes, wind, or delays within a few seconds of real-world interaction, going beyond what standard model-free RL such as Proximal Policy Optimization (PPO) [76] can achieve in the same time budget. Differentiable quadrotor simulators that are fully GPU-native and integrate physics and rendering, such as DiffAero, further increase throughput and provide a unified platform for exploring hybrid differentiable and RL algorithms and sim-to-real transfer for flight control [77]. In parallel, large-scale GPU-accelerated simulation frameworks like Isaac Gym, Isaac Lab, and Aerial Gym offer multi-modal, batched environments for training vision-based policies and studying transfer at scale [16–19, 78].

Another promising direction is to increase the fidelity of the simulated visual environment. Instead of relying on hand-modeled primitives or standard game engines, several works use 3D reconstruction and Gaussian radiance fields to render high-quality, photorealistic scenes for policy training [79, 80]. By combining differentiable dynamics with high-fidelity 3D Gaussian Splatting (3DGS), GRaD-Nav and GRaD-Nav++ train visual navigation policies (and even vision-language-action models) that transfer from simulation to real drones without additional fine-tuning, while retaining the ability to adapt online to new instances within the same task class [79, 80]. Nevertheless, even with such high-fidelity simulators, the coverage of real-world variability, such as weather, lighting conditions, and sensor noises, remains incomplete. Bridging the sim-to-real gap for aggressively flying drones in cluttered, visually complex environments therefore remains an open and central challenge.

Training efficiency and scalability A second major obstacle for RL-based navigation is training efficiency. End-to-end policies that map raw images directly to low-level motor or CTBR commands operate in very high-dimensional observation and action spaces and must solve long-horizon tasks with sparse and delayed rewards. Even in carefully designed simulators, such policies often require millions of interaction steps to reach reasonable performance [16, 17, 20, 77, 81]. This makes naïve end-to-end RL from pixels prohibitive in practice, especially when one ultimately aims to deploy on real hardware where each rollout is costly and potentially risky.

To reduce the burden on RL, many recent works deliberately structure the decision space instead of learning a monolithic mapping from images to actuators. One prominent line of research uses action abstraction and hierarchy: the RL policy operates at a higher-level interface (for example desired velocities, accelerations, or waypoints), while a separate low-level controller is responsible for tracking these commands. This idea is closely related to earlier arguments in evolutionary robotics that abstraction can help cross the reality gap by restricting learning to robust, transferable decision variables while leaving execution details to lower-level mechanisms [82]. Champion-level drone racing systems, for instance, convert visual observations into geometric gate information and then train RL policies that output control inputs in a structured space on top of a carefully designed control stack [34, 83]. For more complex navigation tasks, high-level inspection or avoidance policies output motion setpoints, such as de-

sired linear velocities and yaw rates, or acceleration references, that are then converted into CTBR commands by classical feedback controllers or MPC [45, 59, 84–87]. In parallel, teacher–student bootstrapping further improves sample efficiency: privileged-state teachers are first trained in a low-dimensional state space and then distilled into vision-based students, which are subsequently fine-tuned with asymmetric actor–critic RL [32, 33]. All these strategies aim to confine RL to a better-conditioned subproblem, making the exploration and credit-assignment problem more tractable.

Beyond architectural and algorithmic choices, differentiable simulation has emerged as a powerful tool to further improve data efficiency. Instead of estimating policy gradients from stochastic rollouts alone, analytic policy gradient (APG) methods exploit differentiable dynamics to backpropagate tracking-error gradients directly through time, leading to one to four orders of magnitude fewer samples than standard model-free RL in several control benchmarks [88]. Building on this idea, recent work applies differentiable physics to flight control and vision-based navigation. Vision-based agile flight can be trained end-to-end by backpropagating through differentiable dynamics models and depth-rendering pipelines, achieving high success rates and significantly reduced wall-clock training time compared to purely model-free RL [89, 90]. Differentiable quadrotor simulators such as DiffAero integrate physics and depth or LiDAR rendering in a GPU-native fashion, providing both high-throughput rollouts and gradient information for hybrid differentiable and RL algorithms [77]. Other systems combine differentiable simulation with task-specific perception or scene representations, for example by leveraging 3DGS-based distance fields for gradient-based navigation and rapid online adaptation [79, 80].

While these advances substantially improve training efficiency, they also introduce their own constraints: differentiable formulations often require smooth surrogate costs and simplified geometry and can suffer from vanishing or exploding gradients over long horizons [88]. Moreover, many efficient formulations still assume relatively structured tasks or observation spaces. Achieving sample-efficient RL for general, long-horizon, vision-based navigation in cluttered 3D environments, while maintaining safety and robustness, remains an open problem and a key motivation for the hierarchical and hybrid approaches developed in this dissertation.

1.2. PROBLEM STATEMENT AND RESEARCH QUESTIONS

In this dissertation, we aim to advance the state of the art in autonomous drone navigation in cluttered environments by addressing the challenges outlined above. And our methods can be evaluated in a benchmarking platform and compared with prior works. The main research objective is as follows:

Research objective

To develop efficient learning-based algorithms for safe and agile onboard navigation in cluttered environments, and to optimize the trade-off between safety and agility.

The complexity of visual obstacle avoidance, particularly the intricate interaction with dynamic environments, has led to the lack of a unified platform for algorithm evaluation.

The first research question can be summarized as:

Research Question 1

How can we systematically evaluate vision-based obstacle avoidance algorithms for autonomous drones?

To address the difficulty of reproducing obstacle avoidance experiments in real-world environments, we will construct a high-fidelity simulation environment that includes both a realistic drone dynamics model and photo-realistic visual rendering. The evaluation metrics will be divided into environmental metrics and flight performance metrics, which will be used to quantify environmental complexity and to assess flight safety and agility, respectively.

We will benchmark state-of-the-art algorithms to validate the rationality of the evaluation system. Moreover, the evaluation results will serve as the foundation for designing our own obstacle avoidance algorithm. A key focus is to leverage the environmental complexity metrics to shape the algorithm's adaptability to different environments. Therefore, the second research question can be summarized as:

Research Question 2

How can we train an obstacle avoidance policy that adaptively adjusts its flight speed to the complexity of the environment?

Reinforcement learning typically relies on domain randomization to bridge the sim-to-real gap. By randomizing environmental complexity during training, it becomes possible to learn a policy that adaptively adjusts its flight speed, decelerating in complex scenarios and accelerating in simpler ones. This strategy aims to achieve an effective balance between safety and agility. In addition, for reactive algorithms that use drone control commands as policy actions, it is important to incorporate temporal information and memory into the observations. For example, recurrent neural networks can be used to encode a history of recent sensor inputs. Enriching the policy input with such temporal and environmental information can further enhance the safety and reliability of obstacle avoidance flight.

Next, we will deploy the policy trained in simulation on real drones in real obstacle

environments like forests and use these experiments to further investigate the practical challenges of simulation-to-real transfer:

Research Question 3

How can we effectively bridge the sim-to-real gap for vision-based obstacle avoidance on autonomous drones?

The gaps in vision-based obstacle avoidance tasks can be broadly divided into two categories: drone dynamics gaps and visual perception gaps. Dynamics gaps are often mitigated using domain randomization. In addition, when high-level control commands are used as actions, it is crucial to ensure that the tracking performance of the low-level controller on the real drone matches that in simulation. For visual gaps, since we primarily use depth maps as input, the main source of discrepancy comes from the depth sensing error of the real camera relative to the ideal depth in simulation. Moreover, significant differences between the simulated and real environments can cause substantial shifts in the input domain.

After mitigating the sim-to-real gap, we need to reconsider the obstacle avoidance solution from a higher-level perspective. In particular, we ask whether the incorporation of real animal behavior can further improve the training efficiency of RL methods on-policy. Therefore, the last research question is:

Research Question 4

How can animal-inspired hierarchical perception-action architecture be used to train drone obstacle avoidance more efficiently?

Animals in nature typically do not act purely end-to-end. Signals from the brain are first abstracted into high-level actions, such as turning left or right, and then translated into low-level control commands for the body. Adopting a similar hierarchical navigation strategy may lead to a more efficient training process. In addition, low-level components may be reused across different high-level tasks; for example, a low-level obstacle avoidance controller can be directly employed in exploration or target-tracking tasks. This hierarchical design also creates new opportunities for online learning and continual adaptation.

1.3. THESIS OUTLINE

The remainder of this thesis is organized as follows: Chapter 2 presents a high-fidelity Unity-based benchmarking suite for drone visual obstacle avoidance. It defines evaluation metrics tailored to Research Question 1 and benchmarks several state-of-the-art algorithms to validate the proposed evaluation system. Building on this simulator, Chapter 3 addresses Research Question 2 by introducing a varying-speed obstacle avoidance policy trained in simulation with memory-augmented latent representa-

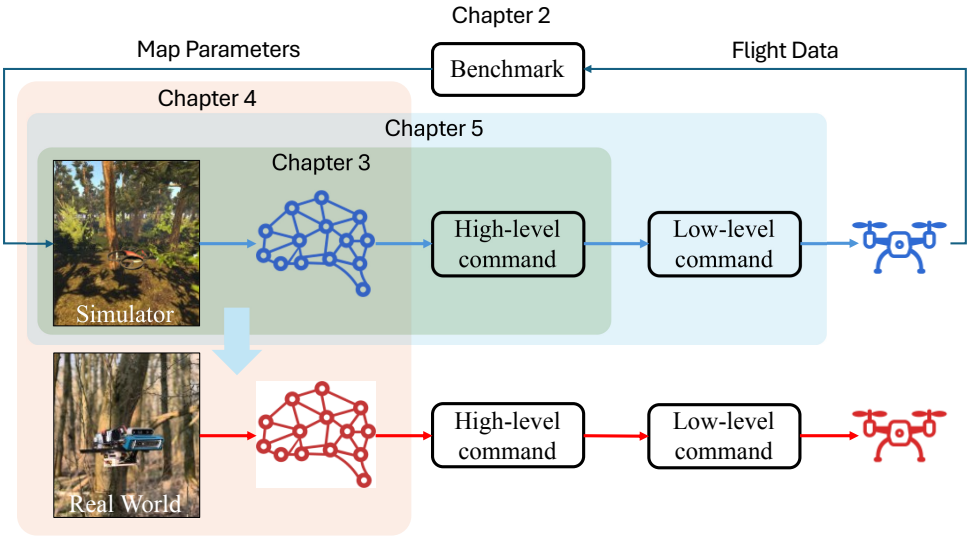


Figure 1.2: The dissertation is structured around the four research questions: Chapter 2 addresses Research Question 1, Chapter 3 Research Question 2, Chapter 4 Research Question 3, and Chapter 5 Research Question 4.

tions. Chapter 4 focuses on Research Question 3 and studies how to transfer these policies from simulation to real drones, tackling both dynamics and perception gaps. Finally, Chapter 5 addresses Research Question 4 and investigates how bio-inspired hierarchical perception–action architectures can improve the training efficiency and reusability of drone obstacle avoidance policies.

2

HIGH-FIDELITY VISION-BASED OBSTACLE AVOIDANCE BENCHMARKING SUITE FOR MULTI-ROTORS

Building on the problem formulation in Chapter 1, the first step toward safe and agile autonomous flight in clutter is to make progress measurable and comparable. Vision-based obstacle avoidance algorithms are often evaluated under ad hoc settings, which obscures how performance changes across environments and makes it difficult to understand the safety–agility trade-off in a principled way. To address Research Question 1, this chapter therefore introduces a high-fidelity benchmarking suite that couples photo-realistic rendering with realistic quadrotor dynamics, and provides standardized scenarios, protocols, and metrics for assessing both environmental complexity and flight performance. This benchmarking foundation is essential not only for fair comparison with prior work, but also for guiding the design choices of the learning-based methods developed in the subsequent chapters.

This chapter is based on the following article:

H. Yu, G. C. H. E. de Croon and C. De Wagter, *AvoidBench: A high-fidelity vision-based obstacle avoidance benchmarking suite for multi-rotors*, IEEE International Conference on Robotics and Automation (2023)

2.1. INTRODUCTION

Autonomous drones are increasingly deployed in scenarios where human access is limited or dangerous, such as forest rescue and inspection, cave exploration [11, 12, 14], and greenhouse or warehouse monitoring [91]. Across these applications, obstacle avoidance is a core capability that directly determines both safety and mission effectiveness. Vision-based obstacle avoidance is particularly attractive because cameras are compact, lightweight, and low-cost, while providing rich information about nearby structure. At the same time, there is currently no widely adopted protocol for evaluating vision-based obstacle avoidance algorithms under comparable conditions. As a result, it is difficult to quantify progress at the field level or to make fair, apples-to-apples comparisons between different methods. This motivates the need for a reliable and reproducible benchmarking suite.

Standardized benchmarks have played a similar role in other research communities. For example, computer vision and natural language processing (NLP) benefit from well-established benchmarks such as KITTi [92], ImageNet [93], and GLUE [94]. These benchmarks provide shared data, clear metrics, and common evaluation protocols, which collectively accelerate iteration and make results comparable across papers and over time. Moreover, a strong benchmark lowers the barrier for adoption by providing reference implementations and by making it straightforward to reproduce baselines.

However, benchmarking in robotics is inherently more challenging than in dataset-driven fields. First, robots operate in closed loop: the sensory observations depend on the actions selected by the autonomy algorithm, so performance cannot be evaluated on a fixed dataset alone. Second, success depends on an integrated pipeline that couples perception, planning, and control, and is further influenced by external factors such as illumination, motion blur, actuation limits, and disturbances (e.g., wind gusts). Consequently, robotics benchmarks must balance realism, repeatability, and computational efficiency, while still providing interpretable measurements of both task outcome and flight quality.

Simulation is a natural tool to address these challenges, because it enables controlled experiments and repeated trials under identical conditions. Several mature simulators exist for multirotors. For example, Gazebo-based simulators such as Hector [23] and RotorS [24] provide well-tested dynamics and interfaces. Nevertheless, Gazebo does not provide photo-realistic visuals, which is a major limitation when benchmarking vision-based autonomy. Low visual fidelity can increase the sim-to-real gap and can reduce how well simulation-based benchmarking results generalize to real robotic platforms.

Recent advances in game engines offer an opportunity to improve realism without sacrificing controllability. Rendering engines such as Unity3D and Unreal Engine can generate photo-realistic scenes with rich textures, materials, and lighting. Airsim is a robot simulation platform developed by Microsoft that uses Unreal Engine 4 [25] and provides both C++ and Python APIs. It also supports hardware-in-the-loop (HITL) and software-in-the-loop (SITL) with flight controllers such as PX4. However, Airsim

is relatively heavy-weight and tightly couples rendering with the drone dynamics and control, which can limit simulation speed and complicate large-scale benchmarking. Flightmare [20] is a more flexible simulator for quadrotors that combines Unity3D rendering with a quadrotor dynamics model (based on RotorS [24]) and supports fast, parallelized simulation for reinforcement learning.

Several benchmarking efforts for autonomous drones already exist. MAVBench [26] evaluates multiple mission types (e.g., search-and-rescue, mapping, and planning) using Airsim and Unreal Engine 4 as a backend. While MAVBench provides end-to-end photo-realistic simulation, its evaluation focuses mainly on time and energy, and it does not explicitly quantify environmental complexity. BOARR [27] is tailored to obstacle avoidance benchmarking, but relies on Gazebo and therefore provides limited visual realism. In vision-based obstacle avoidance, visual realism can substantially affect perception quality and, in turn, the resulting control behavior. This observation motivates a benchmark that combines photo-realistic rendering, realistic dynamics, and environment-aware evaluation metrics.

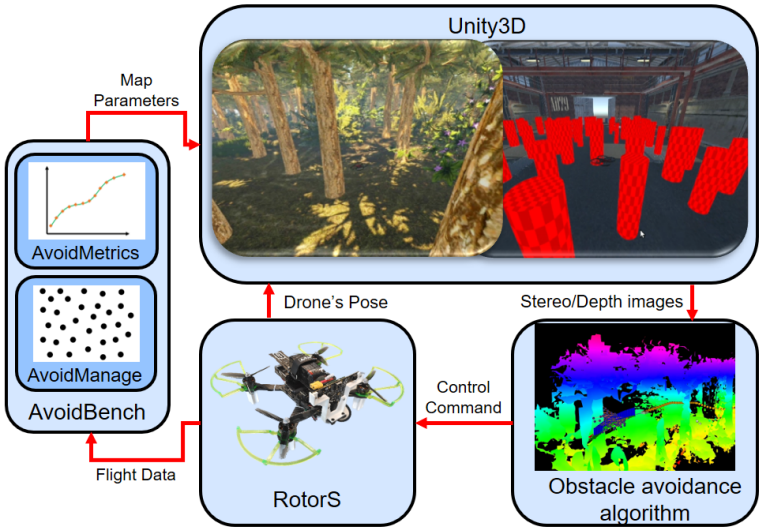


Figure 2.1: The basic framework of AvoidBench. It uses the dynamics model from RotorS and photo-realistic scenes from Unity3D. Users can implement their obstacle avoidance algorithms by Python or C++ interfaces.

In this chapter, we propose *AvoidBench*, a benchmarking suite for evaluating the performance of vision-based obstacle avoidance algorithms. We build AvoidBench on Flightmare because it is comparatively lightweight and can achieve higher simulation speed than Airsim, while still supporting photo-realistic rendering through Unity3D. At the time of writing, AvoidBench includes two representative environments for benchmarking: a forest scene and an indoor warehouse scene. Within each scene, the obstacle distribution and map complexity can be systematically varied, allowing re-

searchers to probe failure modes and to quantify how algorithm performance changes with environment difficulty. In addition, we propose a complete set of evaluation metrics that jointly measure flight performance and environment complexity, enabling a more informative analysis than success/failure alone.

AvoidBench makes four main contributions: (1) we develop a benchmark with photo-realistic visual scenes and realistic multirotor dynamics that enables objective comparison across algorithms and operating conditions, and we validate the benchmark with real-world experiments; (2) expanding upon Nous et al.'s work [95], we propose a complete set of evaluation metrics that covers both flight performance and environment complexity; (3) we benchmark representative obstacle avoidance approaches—learning-based, optimization-based, and motion-primitive-based—and analyze their behavior in both simulation and real-world tests; and (4) we release an open-source software framework that allows users to integrate their own algorithms in either Python or C++, and to relate performance to interpretable environment metrics.

2.2. BENCHMARKING PIPELINE

Building on Flightmare, AvoidBench provides an end-to-end pipeline for benchmarking vision-based obstacle avoidance for multirotors in simulation. The benchmark repeatedly executes start-to-goal navigation trials in procedurally generated maps, logs the sensory observations and flight states, and computes a collection of metrics that summarize both task outcome and flight quality. By relying on Unity3D for photo-realistic rendering, the visual observations are closer to those encountered in real deployments, which helps to reduce the visual sim-to-real gap. As shown in Figure 2.2, AvoidBench consists of four main modules: (1) virtual scenes implemented in Unity3D; (2) a communication interface (AvoidLib) between the Unity side and the C++ side, implemented with ZeroMQ; (3) AvoidManage, which integrates rendering and dynamics and manages the execution of benchmark trials; and (4) AvoidMetrics, which computes both performance and environment metrics. In the remainder of this section, we describe each module in more detail.

2.2.1. VIRTUAL SCENES

In AvoidBench, a *scene* denotes an editable environment in which obstacle avoidance algorithms are evaluated under controlled yet realistic visual conditions. While Flightmare provides a solid simulation backbone, its default interaction with the environment is limited for benchmarking requirements such as adding terrain variation, vegetation (e.g., bushes in a forest), or changing obstacle textures in an indoor scene. To support systematic evaluation, AvoidBench currently provides two environments: an outdoor forest and an indoor warehouse. As shown in Figure 2.3, the outdoor scene is a 160 m×160 m forest with trees, bushes, and undulating terrain, whereas the indoor scene contains geometrical obstacles with different textures and opacity. Below we define two concepts that are used throughout the benchmark: the *map*, which

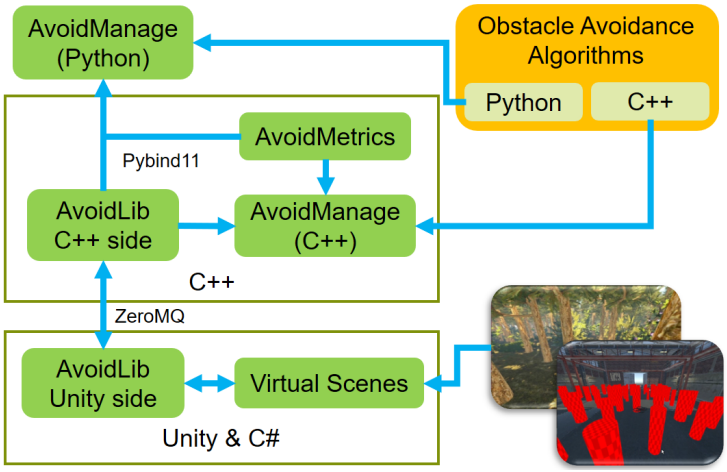


Figure 2.2: Different modules of AvoidBench. AvoidLib is used to communicated between the Unity side and C++ side. AvoidManage contains the setup of virtual scenes and the whole process of flight and benchmarking. AvoidMetrics is the package of performance and environment metrics.

specifies an obstacle field, and the *trial*, which specifies a start-goal task within a map.

Map The benefit of a simulator is that it is easy to create and auto-generate different maps for the flight tests. Users can easily change the number or position of outdoor trees as well as the size and opacity of texture on indoor obstacles. To evaluate the obstacle avoidance ability of drones, it is necessary to follow certain rules to generate obstacles. To generate the actual obstacle field, AvoidBench utilizes a rudimentary and well-known method for procedural generation named Poisson Disc Sampling [96]. This method has a considerable advantage over random sampling as it creates a uniform distribution of obstacles. Furthermore, it allows us to control the minimum distance between obstacles.

As shown in Figure 2.3, we model indoor obstacles as textured cylinders. Users can vary obstacle size and texture opacity, as well as the Poisson-disc sampling radius that determines the minimum separation between obstacles. Generating a new indoor map therefore only requires choosing a new random seed. For the outdoor scene, our goal is to evaluate algorithms in the presence of non-geometrical obstacles that are common in natural environments. We therefore place both red trees and bushes on uneven terrain, and users can control the ratio between red trees and bushes to adjust the mixture of obstacle types.

Trial Given a general map, we perform different *trials*. Each trial has a different start point and goal point generated by a random seed. The drone can traverse the

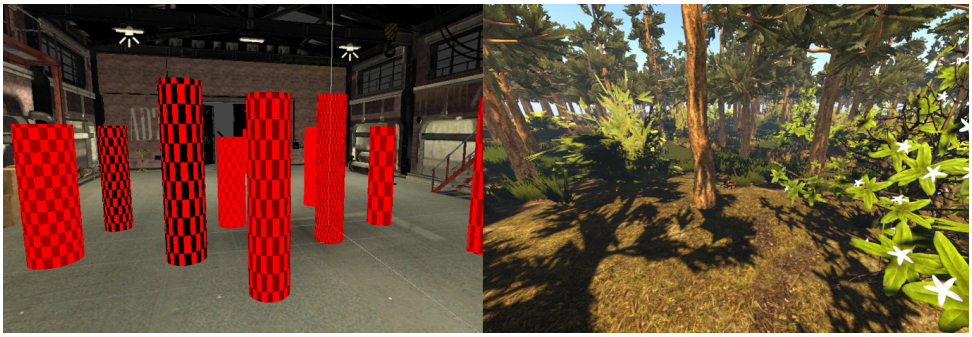


Figure 2.3: Indoor and outdoor scenes. There are many geometrical obstacles but with different opacity of texture in indoor scene (left). Outdoor scene contains red trees and non-geometrically-shaped obstacles such as the bushes (right).

whole map as much as possible. We use the mesh collider from Unity3D to check if the drone collided with obstacles. If the algorithm can command the drone to the goal within one meter without collision, this trial is marked as "*finished*". Apart from collisions, there can be a variety of situations that lead to trial failure. An algorithm e.g., might get stuck in a dead end, unable to find or move to the goal. By setting a maximum time parameter, AvoidBench automatically stops the trial if the time limit has been exceeded.

2.2.2. AVOIDLIB

Similar to Flightmare, AvoidBench uses ZeroMQ to implement the communication layer between Unity (rendering and collision checking) and the C++ side (dynamics, control, and benchmarking logic). ZeroMQ is an asynchronous messaging library designed for distributed and concurrent applications, which makes it suitable for high-rate simulation loops. Through AvoidLib, the benchmark can (i) send map-generation parameters to Unity, (ii) update the drone pose used for rendering, (iii) receive stereo images for perception, and (iv) obtain collision flags from Unity3D's physics and collider system. Because the environment metrics introduced below require an accurate representation of the 3D environment, we also retain Flightmare's capability to export point clouds from the simulator.

2.2.3. AVOIDMANAGE

AvoidManage integrates the rendering engine with the simulated multirotor dynamics and serves as the central *manager* of the benchmark. Conceptually, it operates as a state machine that executes trials, records data, and decides when a trial should terminate. At each control step, AvoidManage queries the obstacle avoidance algorithm

for a control command, advances the multirotor dynamics model, and retrieves the corresponding sensory observations from Unity. The resulting flight data are streamed to AvoidMetrics to evaluate trial performance online (e.g., to detect collisions) and to compute summary metrics at the end of each run. To facilitate rapid prototyping, we also provide Python bindings via Pybind11, allowing researchers to implement and evaluate their obstacle avoidance algorithms in either C++ or Python.

2.2.4. AVOIDMETRICS

AvoidBench is designed not only to report whether a drone reaches its goal, but also to explain *why* performance changes across environments. To this end, AvoidMetrics quantifies both the environment (i.e., how challenging a map is to traverse) and the resulting flight behavior (i.e., safety, efficiency, agility, and computational cost). Measuring these aspects jointly enables a more informative analysis than single-number scores: it becomes possible to compare algorithms under matched difficulty, to identify the regimes in which an approach fails, and to anticipate how a method may behave when transferred to real-world environments.

ENVIRONMENT METRICS

AvoidBench currently reports two complementary environment metrics: *Traversability* and *Relative Gap Size*. Traversability captures free-space availability in a geometry-agnostic manner, whereas relative gap size provides an interpretable proxy for the minimum obstacle separation in procedurally generated maps.

Traversability *Traversability* was introduced in [97] as a metric that quantifies how difficult it is for a drone of a given size to traverse an environment. It was proposed as an alternative to obstacle density, because density alone ignores the drone's physical dimensions and often correlates poorly with actual avoidance difficulty. A key advantage of traversability is that it does not assume a particular obstacle geometry. Instead, it estimates the expected free-flight distance by sampling N positions p in the environment; for each position, a heading h is selected and a collision checker is used to compute the collision-free distance s. Averaging these distances yields an uncorrected measure of free space, which is made dimensionless by dividing by the drone diameter d_{drone}:

$$TRAV = \frac{1}{d_{drone} \cdot N} \sum_{i=0}^{N} s(p(i), h(i)) \tag{2.1}$$

In AvoidBench, we apply two practical modifications to improve repeatability while preserving the intuition of the original metric. Intuitively, traversability represents how far the drone could travel in the environment *without* relying on an active obstacle avoidance strategy. First, instead of uniformly random sampling, we use a grid-based

sampling scheme across multiple heights of the 3D map, which reduces variance and makes repeated evaluations more comparable. Second, at each sampled location, we evaluate a set of direction vectors distributed evenly on a 2D circle rather than a single random heading. For each direction, we compute the corresponding free-path length and then average across directions and locations. Because the metric is based on sampling rather than on a specific obstacle model, it can be applied consistently to complex scenes that include non-cylindrical obstacles.

Relative Gap Size The second environment metric, *relative gap size*, is less generic than traversability because it is directly tied to the Poisson-disc sampling radius used during procedural map generation. It aims to capture the minimum clearance between obstacles in an interpretable way, by expressing that clearance in multiples of the drone diameter. Specifically, it adjusts the Poisson radius $r_{poisson}$ by subtracting the average obstacle width $\hat{w}_{obstacle}$, and then normalizes the result by the drone diameter d_{drone}:

$$RGS = \frac{r_{poisson} - \hat{w}_{obstacle}}{d_{drone}} \tag{2.2}$$

The relative gap size should not fall below 1. Otherwise, even an ideal controller would be unable to traverse the obstacle field because obstacles are placed too close together relative to the drone size.

PERFORMANCE METRICS

AvoidBench currently reports seven complementary performance metrics. Together, they capture task completion (safety), path efficiency, smoothness/energy usage, goal-directed speed, progress in failed trials, and computational cost.

Success Rate The *success rate* is the most direct indicator of safety in AvoidBench. Each map is evaluated through multiple trials, where a trial is considered *finished* if the drone reaches the goal (within 1 m) without colliding with any obstacle. The success rate is then computed as the ratio between the number of finished trials $T_{finished}$ and the total number of executed trials T_{whole}. Although simple, this metric is essential because it summarizes whether an algorithm can reliably complete the task under a given environment complexity and start-to-goal distance.

$$SR = \frac{T_{finished}}{T_{whole}} \tag{2.3}$$

Path Excess Factor Beyond success, it is important to assess how efficiently an algorithm reaches the goal. The *path excess factor* measures how much longer the ex-

ecuted trajectory is compared with the shortest feasible path through the free space. It is computed as:

$$PEF = \frac{d_{trav} - d_{min}}{d_{min}} \times 100\,\% \tag{2.4}$$

Here, d_{trav} denotes the total distance travelled by the drone, and d_{min} is the minimum free-path distance between start and goal computed by a path-search algorithm (A* [13] in our case). The path excess factor should be interpreted as the excess distance covered by the drone relative to the shortest path. So, 0 % means no excess distance, whereas a 100 % means that the drone covered twice the distance it could have.

Energy Cost We use the integral of jerk (the time derivative of acceleration) as a proxy for *energy cost* and motion smoothness. In practice, aggressive and jerky maneuvers tend to increase power consumption, induce vibrations, and stress the platform. Hence, a lower energy cost generally corresponds to smoother and more energy-efficient flight:

$$EC = \int_0^{t_{trial}} \|jerk\|_2 \, \mathrm{d}t \tag{2.5}$$

Average Goal Velocity The *average goal velocity* measures how quickly an algorithm progresses toward the goal. While the trial duration alone is a common indicator of speed, it is not directly comparable across maps with different start-to-goal distances. We therefore normalize by the free-path minimum distance d_{min} and convert the time into a velocity-like quantity. With t_{trial} denoting the trial duration, we define:

$$AGV = \frac{d_{min}}{t_{trial}} \tag{2.6}$$

Relative End Distance When a trial fails (e.g., due to collision or timeout), it is still informative to quantify how much progress was made toward the goal. The *relative end distance* measures the remaining fraction of the start-to-goal displacement at the final drone position. In AvoidBench, it is defined using the start-goal vector $\mathbf{a}$ and the start-to-final-position vector $\mathbf{b}$ (Figure 2.4) as:

$$RED = \frac{|\mathbf{a} - \mathbf{b}|}{|\mathbf{a}|} \tag{2.7}$$

A value of $RED = 0$ indicates that the drone reached the goal, whereas $RED = 1$ corresponds to no progress from the start location. Smaller values therefore indicate better goal-directed progress.

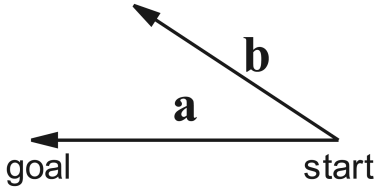


Figure 2.4: Calculation method of relative end distance. **a** is the vector of start point to goal point. **b** is the vector of start point to drone's final location.

Processing Time The *processing time* measures the computational cost of an obstacle avoidance algorithm by recording the runtime of each control iteration. During a trial, all processing periods are stored so that AvoidBench can report statistics such as the mean and standard deviation. Because processing time depends on hardware, algorithms should be evaluated on the same machine for fair comparison. This metric is also practically relevant: long runtimes introduce control delays, reduce the effective control frequency, and may leave fewer resources for other autonomy tasks.

Contrast Factor The *contrast factor* is the only metric that explicitly compares two sets of experiments. It is designed to quantify relative performance while reducing the confounding influence of different start-to-goal distances. In addition to comparing two different algorithms, the same formulation can compare simulator versus real-world runs of the *same* algorithm, which provides a compact way to quantify sim-to-real discrepancy.

From the perspective of probability, the success rate is related to the tested algorithms, the minimum free-path distance from start point to goal point, and the complexity of maps which can be represented by traversability. We use Ξ to represent the obstacle avoidance algorithm and τ as a certain map. The normalized traversability P_τ can be calculated as:

$$P_\tau = \frac{TRAV}{TRAV_{max}} \tag{2.8}$$

Where $TRAV_{max}$ is the value of traversaility when there is no obstacle in the map (only boundaries of the map have an impact on this value). Assuming the obstacles in the map are uniform, the probability of no collision for a certain algorithm within a certain distance is p, and the probability of no collision at twice distance should be p^2. Obviously, the success rate is exponentially related to the flying distance in theory. Then the success rate $\hat{SR}$ can be estimated as:

$$\hat{SR}_\Xi = f(\Xi, P_\tau)^{\frac{{}^\Xi d_{min}}{\lambda}} \tag{2.9}$$

Where λ is a scale factor of distance, and ${}^\Xi d_{min}$ is the minimum free-path distance from start point to goal point randomly generated for algorithm Ξ. $f(\Xi, P_\tau)$ indicates

the non-collision probability for a certain distance determined by λ when algorithm Ξ is excuted in the map with normalized traversability P_τ. From AvoidBench, ${}^{\Xi}d_{min}$ and P_τ are known, and $S\hat{R}_\Xi$ can be replaced by statistical results from (2.3). However, we still cannot get $f(\Xi, P_\tau)$ because the λ is unknown. By logarithmizing both sides of (2.9), we can obtain the ratio of different algorithms' performance in the map with same traversability:

$$CF = \frac{\ln f(\Xi_1, P_\tau)}{\ln f(\Xi_2, P_\tau)} = \frac{{}^{\Xi_2}d_{min} \ln SR_{\Xi_1}}{{}^{\Xi_1}d_{min} \ln SR_{\Xi_2}} \tag{2.10}$$

If Ξ_1 and Ξ_2 represent the experiments of the same algorithm in the simulator and real world, then contrast factor can be used to measure the sim2real gap even though they have different flying distance.

2.3. SIMULATION AND EXPERIMENT

2.3.1. OBSTACLE AVOIDANCE ALGORITHMS

In order to demonstrate that AvoidBench can evaluate a diverse set of obstacle avoidance strategies, we benchmark three representative state-of-the-art approaches: Agile-Autonomy [30] (learning-based), Ego-planner [15] (optimization-based), and MBPlanner [14] (motion-primitive-based). For Agile-Autonomy, we use the original implementation and retrain the network using our benchmark scenes so that the policy experiences the same visual domain as the other baselines. MBPlanner is originally designed for exploration; to adapt it to our start-to-goal navigation tasks, we replace the exploration gain with a goal-directed term based on the distance to the goal, ensuring that the drone consistently moves toward a fixed target. All three algorithms are evaluated using a virtual stereo camera, and depth maps are computed via Semi-Global Matching [98] to provide a consistent depth input across methods.

2.3.2. SIMULATION RESULTS

For the outdoor scene, we generate 30 maps and evaluate 30 trials per map. The ratio of bush trees to red trees is set to 2:3, and the Poisson-disc sampling radius is varied in the interval [2.3, 5.8] to systematically change map complexity. Start and goal locations are generated from fixed random seeds so that all three algorithms face identical tasks and environments, enabling a fair comparison. All simulation experiments are executed on a standard laptop with 16 GB memory, an i7 11800H CPU, and an RTX3050 GPU. The image frame rates in the outdoor and indoor environments are 20 Hz and 40 Hz, respectively.

To highlight how performance changes with environment difficulty, we report results as a function of traversability. For visualization, we group trials into five traversability intervals and compute statistics within each bin.

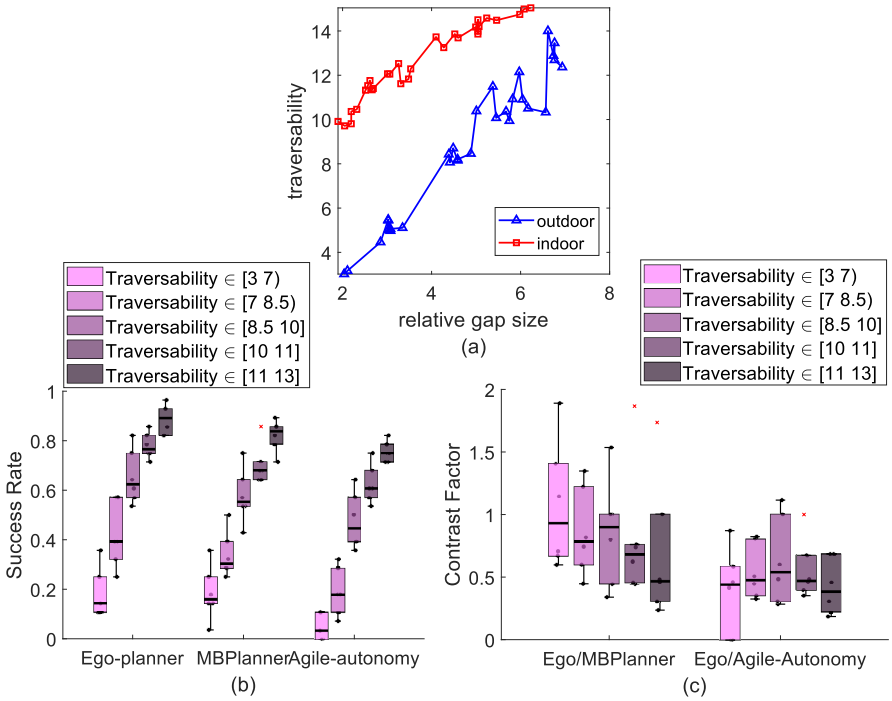


Figure 2.5: (a) is the relationship between traversability and relative gap size of both indoor scene and outdoor scene. (b) is the success rate of different algorithms. (c) is the contrast factors of Ego-planner to MBPlanner and Ego-planner to Agile-Autonomy.

Figure 2.5(b) shows how the success rate of the three algorithms varies with traversability. The results are grouped into five traversability intervals to make the overall trend easier to interpret. Obviously, for the three algorithms, success rate increases significantly with the increase of traversability, which shows that the performance of obstacle avoidance algorithms has a great relationship with the environment metrics we proposed. The results indicate that Ego-planner and MBPlanner achieve similar safety levels (success rate), and that both outperform Agile-Autonomy in terms of collision-free task completion in these settings. Figure 2.5(c) is the result of contrast factor calculated by success rate. A contrast factor close to 1 for "Ego / MBPlanner" suggests comparable performance, whereas values below 1 for "Ego / Agile-Autonomy" indicate that Ego-planner achieves a higher success rate than Agile-Autonomy under matched traversability. The relationship between traversability and relative gap size is shown in Figure 2.5(a). Due to the non-geometrical structure of bushes, traversability in the outdoor scene does not increase strictly linearly with relative gap size, unlike the indoor scene where obstacles are more regular. Overall, traversability serves as a more general and geometry-agnostic indicator of environment complexity.

Figure 2.6(a) and 2.6(b) report the path excess factor and energy cost, respectively (lower is better). Missions are sorted by traversability. A lower path excess factor indicates a trajectory closer to the shortest feasible route, while a lower energy cost indicates smoother flight with fewer high-jerk maneuvers. Ego-planner and MBPlanner exhibit clearer improvements in both metrics as traversability increases, whereas Agile-Autonomy generally achieves lower energy cost across missions, with a weaker dependence on traversability.

Another metric with significant differences is average goal velocity (Figure 2.6(c)). Agile-Autonomy has been designed for high-speed flight, which results in higher flight speeds than those of Ego-planner and MBPlanner. Relative end distance provides an additional perspective on failure behavior, because it reflects how far the drone remained from the goal when a trial terminated unsuccessfully. As shown in Figure 2.6(d), in the results of Relative end distance, Ego-planner remains the safest algorithm and we can see relative end distance also has a decreasing trend with traversability.

For processing time (Figure 2.6(e) and 2.6(f)), Ego-planner and Agile-Autonomy are the fastest. Although Ego-planner performs mapping, it avoids explicit ESDF construction, which reduces computational overhead. Agile-Autonomy relies on an end-to-end neural network and therefore does not require map building, leading to short runtimes as well. In contrast, MBPlanner's mapping and planning pipeline incurs the highest processing time among the three algorithms.

2.3.3. REAL WORLD TESTS

EXPERIMENT PLATFORM

To verify that the proposed metrics can objectively evaluate different algorithms in real deployments, we built a quadrotor platform for experimental validation (Figure 2.7(a)). The drone uses a 6-inch Armattan frame equipped with Emax 2306 motors

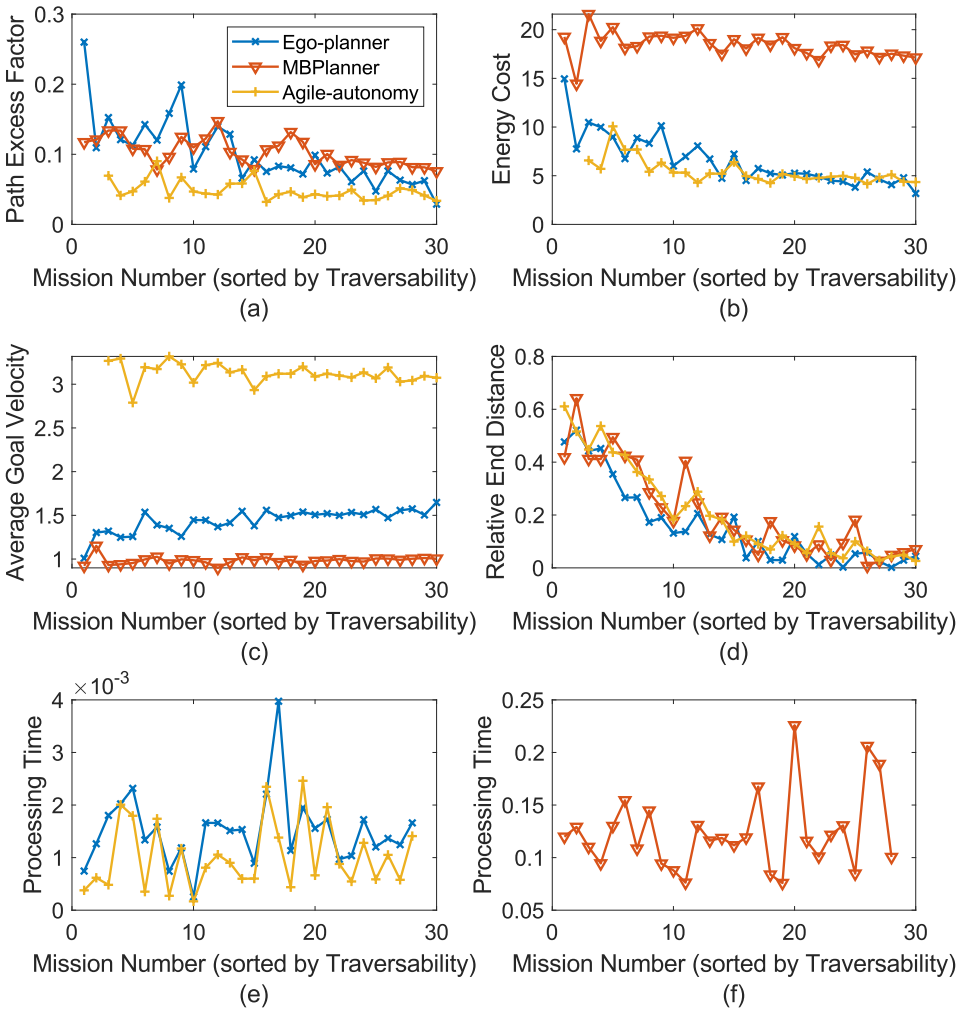
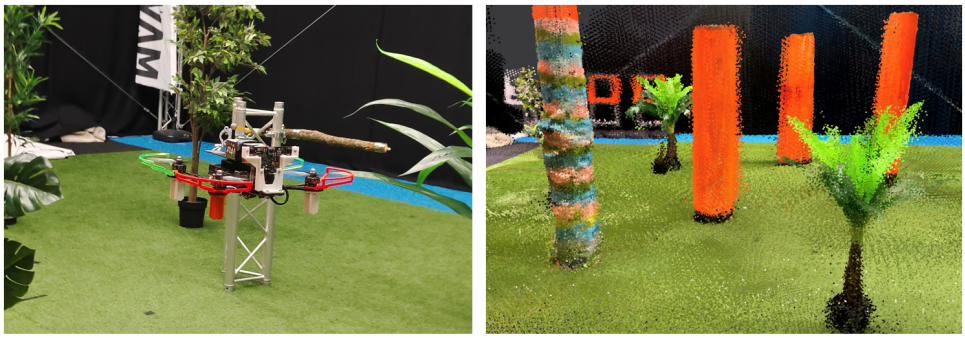


Figure 2.6: Results of different metrics relating to the increasing traversability missions. (a) shows the results of path excess factor. (b) shows the results of path cost. (c) shows the results of average goal velocity. (d) shows the results of relative end distance. (e)-(f) show the results of processing time.

and 5-inch three-bladed propellers, which provides sufficient thrust margin for agile flight while carrying the sensing and computing payload.

Computation is performed on an NVIDIA Jetson Xavier NX module (384-core GPU, 48 Tensor Cores, and a 6-core ARM CPU). The obstacle avoidance framework outputs low-level control commands consisting of collective thrust ratio and body rates. These commands are sent to a Pixhawk Mini 4 flight controller running PX4. For perception, the platform carries an OAK-D stereo camera, which provides dense depth maps at 400p resolution and 30 Hz. This setup mirrors the simulated evaluation pipeline and enables a direct comparison between simulation and real-world benchmarking.



(a) (b)

Figure 2.7: (a): Experiment platform for real world tests. (b): High-precision point cloud map built by a lidar sensor.

EXPERIMENT RESULTS

For real-world experiments, we arrange three maps with different levels of clutter and geometric complexity. To compute traversability in the real world, we build high-precision point cloud maps using a LiDAR sensor (one example is shown in Figure 2.7(b)). For each map, we run 20 trials. Using (2.8), we obtain the normalized traversability of each real-world map and then select simulation results from the corresponding traversability interval for comparison; the aggregated results are shown in Figure 2.8.

From Figure 2.8(a), we see that at comparable normalized traversability, simulation success rates are consistently higher than those obtained in the real world. This gap is expected, as real flights introduce additional uncertainty (e.g., sensing noise, calibration errors, actuator delays, and unmodeled disturbances) that can reduce safety margins. We also compute the contrast factor for "Simulation / Real world" as shown in Figure 2.8(b). Because this metric compensates for differences in flight distance, the contrast factors cluster closer to 1 in most cases, except for the easiest map where the limited number of trials may affect the estimate. Finally, Figure 2.8(c) reports the path excess factor in real-world tests, which shows a decreasing trend with traversability

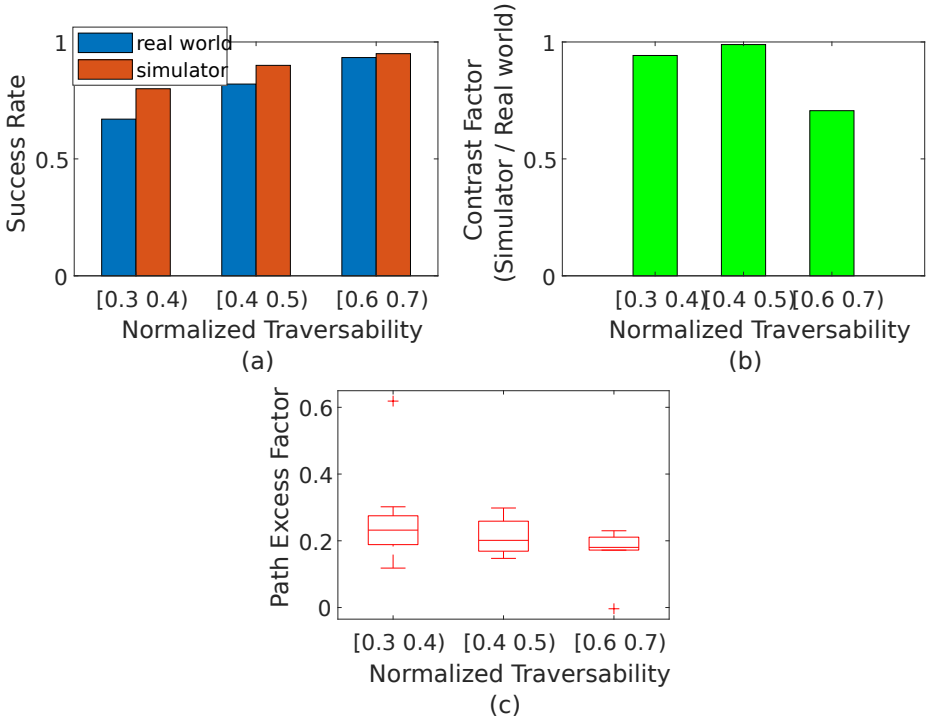


Figure 2.8: Comparison of real world tests and simulation tests. (a) shows the results of success rate when real world maps have similar traversability with simulator. (b) shows the results of contrast factor of "simulator / real world". (c) shows the results of path excess factor in real world tests.

consistent with the simulation results.

2.4. CONCLUSION

In this chapter, we introduced AvoidBench, a high-fidelity benchmarking suite for vision-based obstacle avoidance. AvoidBench combines photo-realistic Unity3D environments with multirotor dynamics and a standardized trial protocol, enabling controlled and repeatable evaluation across a broad range of map complexities. Importantly, the benchmark reports both environment metrics (e.g., traversability and relative gap size) and performance metrics (e.g., success rate, efficiency, smoothness/energy usage, speed, progress in failed trials, and runtime), which supports analysis beyond success/failure alone.

Through extensive simulations and real-world experiments, we demonstrated that the proposed metrics provide consistent and interpretable trends. In particular, success rate and efficiency metrics correlate strongly with traversability, and the contrast factor offers a practical way to compare performance across different distances and between simulation and real deployments. Overall, AvoidBench provides a principled foundation for the learning-based methods developed in the subsequent chapters, where environment-aware evaluation is essential to reason about the safety–agility trade-off.

3

LEARN TO FLY IN CLUTTERED ENVIRONMENTS WITH VARYING SPEED

With the benchmarking pipeline and complexity-aware metrics established in Chapter 2, we can move beyond comparison and use the same framework to shape policy design. The results and analysis enabled by AvoidBench highlight a recurring limitation of fixed-speed navigation: conservative speeds waste agility in simple scenes, while aggressive speeds reduce safety in dense clutter. Motivated by Research Question 2, this chapter leverages the simulator, protocols, and safety–agility evaluation criteria introduced earlier to develop an obstacle avoidance policy that adapts its flight speed to perceived environmental difficulty. In particular, it introduces a reinforcement learning approach trained under randomized environment complexity and equipped with a memory-augmented latent representation, so that the learned behavior can remain reactive while still accounting for recently observed obstacles and local structure.

This chapter is based on the following article:
H. Yu, C. De Wagter and G. C. H. E. de Croon, *MAVRL: Learn to Fly in Cluttered Environments With Varying Speed*, IEEE Robotics and Automation Letters (2025)

3.1. INTRODUCTION

Obstacle avoidance is a fundamental capability for autonomous micro aerial vehicles (MAVs) that must fly in close proximity to obstacles while relying on lightweight onboard sensing and computation. Over the past decade, a wide range of obstacle avoidance algorithms has been proposed [21, 22, 30, 36, 51], and learning-based perception and control have become particularly influential. However, integrating these methods into reinforcement learning (RL) pipelines [34, 44] remains challenging: the policy must learn to interpret high-dimensional observations, remain safe under partial observability, and still make steady progress toward a goal.

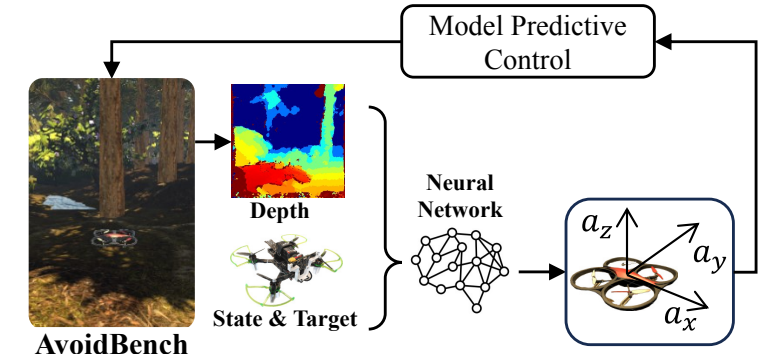
A key, yet often under-emphasized, design choice in drone obstacle avoidance is how speed is handled. Many approaches assume a fixed (or narrowly tuned) cruising speed, which can be overly conservative in simple scenes [15] and insufficiently safe in dense clutter [30]. The idea of adaptive speed has also been explored [37]; however, that work relies primarily on RL for speed selection while still depending on mapping and traditional path optimization. More broadly, autonomous drone RL typically faces a trade-off between end-to-end learning from raw sensory inputs [22, 63] and using compact, fixed-dimensional latent representations [38, 39]. End-to-end learning can discover rich visuomotor behaviors, but it is data- and compute-intensive. Latent representations improve training efficiency, yet purely reactive policies that observe only the current latent state (without an explicit mapping module) can lose track of previously observed obstacles and become stuck in front of complex structures.

In this chapter, we introduce a new obstacle avoidance pipeline termed Memory-Augmented Varying-speed Reinforcement Learning (MAVRL). As shown in Figure 3.1(a), MAVRL takes a depth map together with the drone and target states as input and outputs acceleration commands. These acceleration commands are then tracked by a model predictive controller (MPC) from Agilicious [99], which produces the body rates and thrust commands required by the flight controller. The full pipeline is trained in simulation on randomly generated scenes spanning a wide range of obstacle configurations and clutter levels.

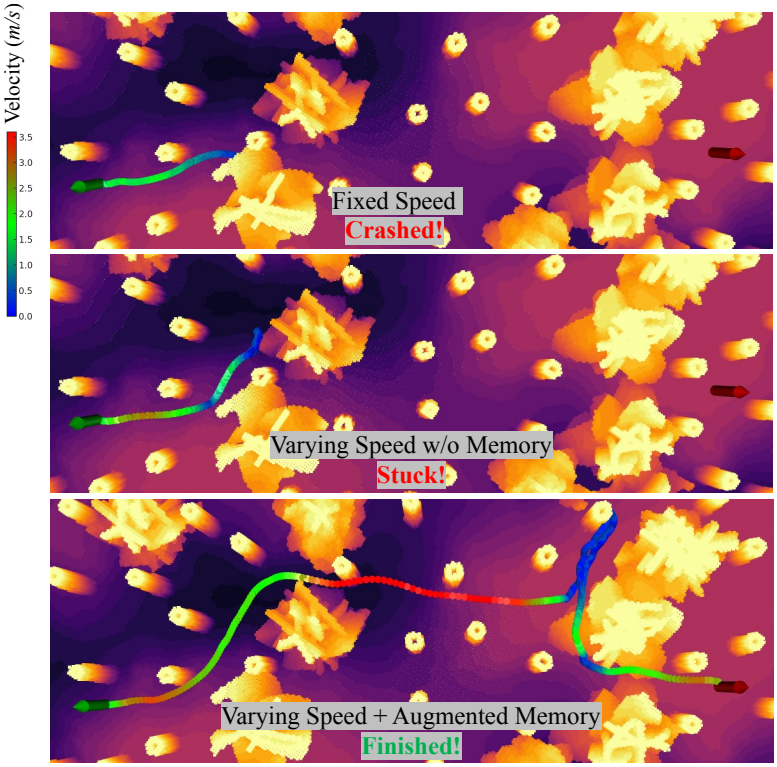
A central component of MAVRL is a latent representation that explicitly incorporates memory. Instead of encoding only the current depth observation, the representation encourages the policy to retain information about obstacles observed within a recent time window, even after they leave the camera field of view. As illustrated in Figure 3.1(b), combining this memory-augmented representation with a varying-speed policy allows the drone to avoid oscillations and dead-ends in clutter, achieving both higher safety and better flight efficiency.

Contributions. The main contributions of this chapter are:

- **Memory-augmented latent representation.** We design a latent state that retains recent depth information by reconstructing both past and current depth observations (and optionally future ones). This provides a more explicit and structured memory signal than standard latent encoders. Extensive ablation studies show that this representation improves RL-based obstacle avoidance, especially



(a)



(b)

Figure 3.1: (a) is the basic framework of MAVRL. (b) illustrates drone's trajectories in a Cluttered Environment. Fixed-speed flight often results in collisions with large obstacles. Absence of augmented memory leads to frequent entrapment in such obstacles. In contrast, MAVRL-equipped flights demonstrate safe and efficient navigation through complex terrains.

when large obstacles temporarily occlude the target direction.

- **Varying-speed RL policy and systematic evaluation.** We train a policy that adapts its speed to the perceived navigational difficulty of the local environment. In simulation, we compare MAVRL against strong baselines, including Agile-Autonomy [30] and Ego-planner [15], and demonstrate superior performance in cluttered scenarios with a favorable safety–agility balance.
- **Real-world deployment.** We deploy the trained network on a real quadrotor with minimal post-simulation fine-tuning, demonstrating that the proposed perception and control pipeline is practical on onboard hardware.

3.2. Related Work

This chapter is closely related to two research directions: (i) learning-based obstacle avoidance policies for MAVs, and (ii) compact latent representations that make RL with visual inputs more sample-efficient. We briefly summarize representative work in both areas and highlight the gap addressed by MAVRL.

3.2.1. Learning-based Obstacle Avoidance

Recent studies [21, 22, 30] have shown substantial progress in learning-based obstacle avoidance. In supervised learning, Agile-Autonomy [30] uses a trajectory generation procedure based on Metropolis–Hastings sampling and trains a neural network to imitate the resulting collision-free behavior. Reinforcement learning has also delivered impressive performance when combined with high-fidelity simulation [34, 35]; for example, [34] introduced *Swift*, which can outperform expert human pilots in racing tasks. At the same time, many RL-based systems rely on additional modules (e.g., mapping, state estimation, or obstacle detection) or on priors about the environment, which can reduce robustness when the scene structure changes.

Several works attempt to reduce these dependencies. [36] trained a vision-based policy through reinforcement learning using a teacher policy that had access to full state information. [59] proposed a 2D navigation planner trained with Soft Actor-Critic (SAC) on LiDAR-based costmaps, although the sensor payload can limit practical deployment on small MAVs. Other approaches [44, 57, 63] use RGB inputs and learn latent features suitable for RL training. Beyond supervised and RL methods, self-supervised approaches have also been explored for obstacle avoidance [51–53, 55]. Notably, [55] adapts flying speed to obstacle density with bio-inspired design. In contrast, our goal is to train a single RL policy that can operate across environments of varying complexity and learn an adaptive speed profile within the same training framework.

3.2.2. LATENT REPRESENTATIONS

Given the high dimensionality of visual observations, an effective representation is crucial for stable and efficient policy learning. End-to-end visuomotor learning has been demonstrated with depth or RGB inputs [22, 30, 63], but performance can degrade in dense clutter and success rates may remain limited [38]. Latent representations are therefore widely used across tasks such as image understanding [100] and navigation [39, 40]. [40] introduced a latent representation for sampling-based motion planning, combining an autoencoder for compressing images, a dynamics model for predicting the next latent state, and a learned collision checker.

For depth-based navigation, Mihir *et al.* [38] proposed a collision-oriented encoding that preserves thin structures more reliably than a standard VAE [101] at the same latent dimensionality. [39] presented a learning pipeline for quadrupedal navigation in clutter using a pre-trained representation that couples a VAE encoder with an LSTM [43] for predicting future latent states.

Our work is inspired by [39], but differs in how memory is introduced. Rather than emphasizing future prediction, we structure the latent space to retain *explicit* information about recent depth history, which proves particularly beneficial when navigating cluttered environments dominated by large obstacles.

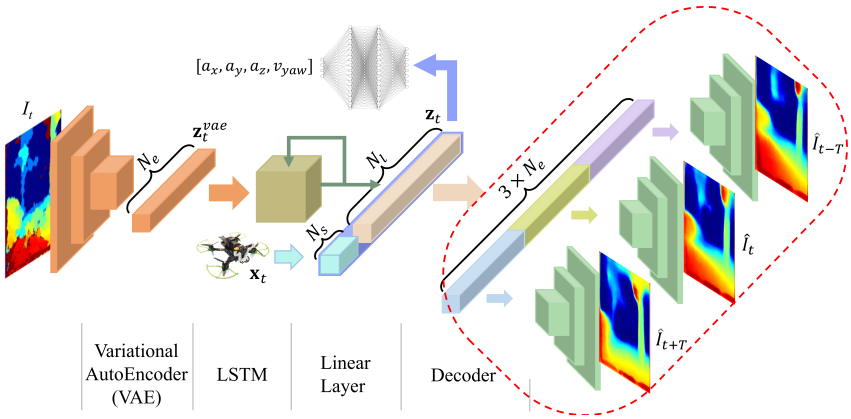
3.3. MEMORY-AUGMENTED REPRESENTATION

In this section, we describe how MAVRL constructs a compact state representation for depth-based navigation and how this representation is augmented with short-term memory. The key idea is to encode each depth image into a low-dimensional latent vector and then use a recurrent model to aggregate a sequence of observations. In doing so, the policy can retain information about obstacles that were visible moments ago but are no longer in the current field of view.

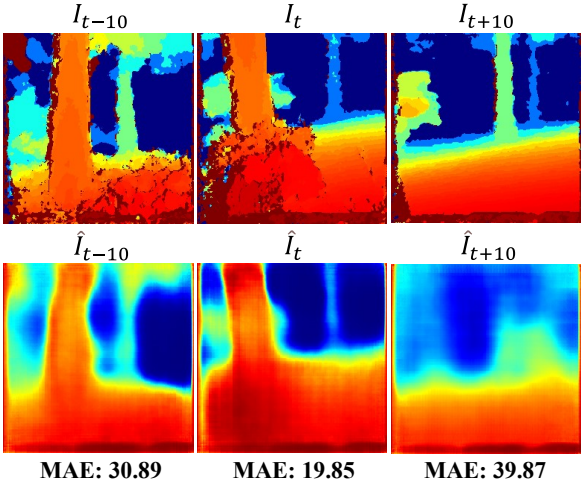
As illustrated in Figure 3.2(a), we first encode the current depth image I_t with a Variational Autoencoder (VAE), producing a latent vector $\mathbf{z}_t^{vae}$. A single-layer Long Short-Term Memory (LSTM) network then processes the latent sequence and outputs the final latent state $\mathbf{z}_t$, which is intended to summarize the recent depth history. We concatenate $\mathbf{z}_t$ with a low-dimensional vector $\mathbf{x}_t$ that contains the drone state and target-related variables, and we train a Proximal Policy Optimization (PPO) policy [76] to generate acceleration commands from this combined input.

Because jointly training a VAE, an LSTM, and an RL policy from scratch is unstable, we adopt a staged training procedure:

- **Warm-start policy training.** We first train an initial PPO policy while using fixed (randomly initialized) VAE and LSTM components. This policy learns basic goal-reaching behavior and produces trajectories that cover diverse viewing angles and distances.
- **Dataset collection and VAE training.** Using rollouts from the warm-start policy,



(a)



(b)

Figure 3.2: (a) depicts MAVRL's network architecture. The depth image, encoded into a latent space by VAE, is processed by LSTM to create a memory-augmented representation. This, combined with the drone's state and target data, informs the acceleration command via PPO. The network within the red dotted box, used for LSTM training, is not for reference. (b) compares original and reconstructed depth images from latent space $\mathbf{z}_t$, showing better quality for past and current images than for future ones(highest MAE loss).

we collect sequences of depth images and train the VAE to reconstruct depth observations. In this stage, we do not train the LSTM yet.

- **LSTM training with a frozen encoder.** We then freeze the VAE encoder and train the LSTM on the collected sequences so that its output latent state can reconstruct selected past/current (and optionally future) depth images, thereby injecting an explicit memory signal.
- **Final PPO training with adaptive speed.** Finally, we retrain PPO end-to-end with the learned VAE and LSTM components to obtain an obstacle avoidance policy that adapts its speed to environments of varying complexity.

3.3.1. ENCODING DEPTH IMAGES

Our pipeline is based on AvoidBench [81], a high-fidelity simulator with photo-realistic scenes. Instead of directly acquiring depth images, AvoidBench uses a semi-global matching algorithm (SGM) [98] from a virtual stereo camera to replicate realistic depth errors, reducing the gap between simulation and reality. We use AvoidBench to generate depth images for training the VAE.

Consider a depth image at time t, denoted by $I_t \in \mathbb{D}$, where $\mathbb{D}$ is the set of all depth images. We use a VAE to encode each image into a latent space, $\mathbf{z}_t^{vae} \in \mathbb{Z}^{N_e}$, where $\mathbb{Z}^{N_e}$ represents all possible latent spaces and $N_e = 64$ is the dimension of VAE latent space. The VAE training employs an encoder-decoder framework without recurrent structures, using convolutional neural networks for both the encoder and decoder.

The encoder includes six convolutional layers, each followed by a ReLU activation function. The output from the final convolutional layer is flattened and then split into two components by two fully connected layers, representing the mean μ and variance σ^2. The latent space $\mathbf{z}_t^{vae}$ is sampled from a Gaussian distribution characterized by μ and σ^2. The decoder, mirroring the encoder, comprises six deconvolutional layers, each also followed by a ReLU activation function. The output of the last deconvolutional layer passes through a sigmoid activation function to yield the reconstructed depth image I_t^{recon}. The loss function for the VAE is detailed in Equation 3.1.

$$
\begin{aligned}
\mathscr{L}_{VAE} &= \mathscr{L}_{recon} + \beta_{norm}\mathscr{L}_{KL} \\
\mathscr{L}_{recon} &= \text{MSE}(I_t, I_t^{recon}) \\
\mathscr{L}_{KL} &= \frac{1}{2}\sum_{i=1}^{N_e}(1 - \mu_i^2 - \sigma_i^2 + \log(\sigma_i^2))
\end{aligned}
\tag{3.1}
$$

where β_{norm} is the weight of Kullback-Leibler (KL) loss, I_t^{recon} is the reconstructed depth image from latent space $\mathbf{z}_t^{vae}$. The MSE loss is used to calculate the reconstruction loss $\mathscr{L}_{recon}$, while KL loss is used to calculate the KL divergence between the latent space and the Gaussian distribution.

3.3.2. Memory-augmented Latent Representation

As shown in Figure 3.2(a), the VAE output $\mathbf{z}_t^{vae}$ is fed into a single-layer LSTM that integrates information over time. The LSTM produces a latent state $\mathbf{z}_t \in \mathbb{Z}^{N_l}$ with $N_l = 256$, which is intentionally larger than the VAE latent dimension ($N_e = 64$) to provide additional capacity for storing temporal context. During policy learning, $\mathbf{z}_t$ is concatenated with the state-and-goal vector $\mathbf{x}_t$ and passed to PPO.

To explicitly encourage memory, we attach a lightweight decoder head during LSTM training. Specifically, the concatenated vector $[\mathbf{z}_t, \mathbf{x}_t]$ is mapped through a fully connected layer to a vector of size $3 \times N_e$, which is split into three latent codes that are decoded (with the shared VAE decoder) into a past, current, and future depth image $\hat{I}_{t-T}$, $\hat{I}_t$, and $\hat{I}_{t+T}$. The offset T specifies the temporal distance for reconstruction and thus controls the memory horizon. By reconstructing past and current observations, the LSTM is forced to preserve obstacle information that may no longer be visible at time t, while the optional future reconstruction provides an additional regularization signal. The corresponding training objective is defined in Equation 3.2.

$$\mathscr{L}_{\text{LSTM}} = \sum_{i=-1,0,1} \lambda_i \cdot \text{MSE}(I_{t+iT}, \hat{I}_{t+iT}) \quad (\lambda_i \in \{0,1\}) \tag{3.2}$$

where $\hat{I}_{t+iT}$ denotes the depth image reconstructed from the latent state $\mathbf{z}_t$. The binary coefficient λ_i selects which of the three reconstruction targets (past, current, future) are included in the loss. By changing the set of active λ_i values, we can directly control whether the LSTM is encouraged to store only the current observation, to retain a short history, or to also regularize toward near-future prediction. The influence of different λ_i configurations is evaluated in the experiments.

Figure 3.2(b) visualizes depth images reconstructed from $\mathbf{z}_t$. Reconstruction quality is consistently higher for past and current observations than for the future frame, suggesting that the LSTM learns to encode recent history more reliably than it can predict unseen geometry. This behavior is expected: future depth depends on control decisions and on parts of the scene that may not yet be observed, whereas past observations correspond to already measured geometry. In practice, this reinforces our design choice to use past/current reconstruction as the primary mechanism for explicit memory.

3.4. Reinforcement Learning for Obstacle Avoidance

We detail the reinforcement learning algorithm used to train our obstacle avoidance policy. We utilize PPO, a policy gradient method, to optimize the policy network by maximizing expected rewards. We treat obstacle avoidance as a Markov Decision Process (MDP), which structures decision-making in stochastic environments. Our approach, distinct from other RL-based strategies [36, 59], is tailored to handle environments of varying complexity, allowing the policy to adaptively respond to environmental challenges.

3.4.1. Problem Formulation

We use the AvoidBench simulator [81] for RL environment setup. Our drone, equipped with a stereo camera, uses PPO for training. For enhanced efficiency, we replace RotorS [24] dynamics model with a simpler kinematics model to allow parallel data generation with multiple drones. The control command includes 3D acceleration and a 1D yaw rate. The kinematics model is:

$$\dot{p} = R_b^w v, \quad \dot{v} = a, \tag{3.3}$$

where p is the drone's position in the world frame, v is its velocity in the body frame, R_b^w is the rotation matrix from the body to the world frame, and a is the body-frame acceleration. This simplified kinematics model is used solely for policy training. For benchmarking against other methods, the RotorS dynamics model is employed.

The Markov Decision Process (MDP) for our model is defined as a tuple $(\mathscr{S}, \mathscr{A}, \mathscr{P}, \mathscr{R}, \gamma)$, where $\mathscr{S}$ represents the state space, $\mathscr{A}$ denotes the action space, $\mathscr{P}$ defines the transition probability, $\mathscr{R}$ is the reward function, and γ signifies the discount factor. The state space $\mathscr{S}$ comprises the current latent representation $\mathbf{z}_t$, along with the drone's state and target information $\mathbf{x}_t$. The action space $\mathscr{A}$ includes the acceleration in the body frame and the yaw rate. The transition function is deterministic, governed by Equation 3.3. Thus, for any state s and action a, the transition probability $\mathscr{P}(s'|s,a)$ is 1 for the unique next state s' and 0 for others.

As illustrated in Figure 3.2(a), the drone's state and target information at time t, denoted as $\mathbf{x}_t$, is represented by a vector of length N_s. In our specific case, N_s is equal to 7. The coordinate system is shown in Figure 3.3(a): the drone's position and velocity in world frame at time t are denoted as $\mathbf{p}(x_t, y_t, z_t)$ and $\mathbf{v}(v_{xt}, v_{yt}, v_{zt})$, respectively, with the target position represented as $\mathbf{p_g}(x_g t, y_g t, z_g t)$. The vector from the drone to the target is $\mathbf{d} = \mathbf{p_g} - \mathbf{p}$, which represented as (d_{xt}, d_{yt}, d_{zt}), and the drone's heading angle is ψ. The bearing angle β is defined as the angle between the body frame's x_b axis and the target vector, while the track angle χ represents the direction of horizontal velocity in the world frame. The drone's state and target information $\mathbf{x}_t$ is defined as:

$$\begin{aligned} \mathbf{x}_t &= [d_{\text{hor}}, v_{\text{hor}}, \beta', d_{zt}, v_{zt}, \chi', \psi], \\ d_{\text{hor}} &= \ln(\sqrt{d_{xt}^2 + d_{yt}^2} + 1), \quad v_{\text{hor}} = \sqrt{v_{xt}^2 + v_{yt}^2}, \\ \beta' &= \beta + \psi = \arctan\left(d_{yt}/d_{xt}\right), \\ \chi' &= \chi - \psi = \arctan\left(v_{yt}/v_{xt}\right), \end{aligned} \tag{3.4}$$

Here, d_{hor} is the log horizontal distance to the target, v_{hor} the drone's horizontal velocity, v_{zt} the vertical velocity, β' the direction to the target in the world frame, χ' the velocity direction in the body frame, and d_{zt} the vertical distance to the target.

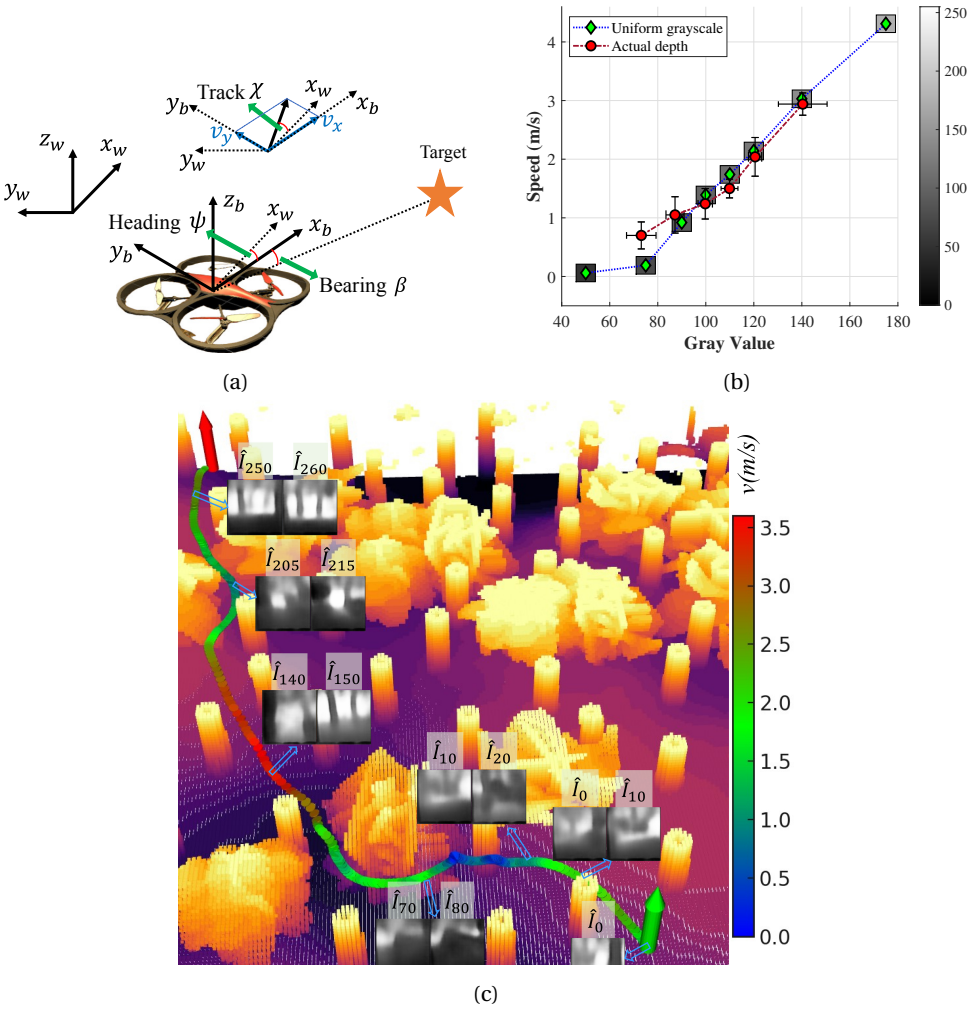


Figure 3.3: (a) illustrates the drone's coordinate system, with the bearing angle β between body axis x_b and the target vector, and the track angle χ as the horizontal velocity's direction in the world frame. (b) displays the drone's average speed in response to uniformly bright gray images (green diamond), and the mean and standard deviation of the speed when the input are actual depth images from (c) (red circles). (c) presents an adaptive drone trajectory in a cluttered environment. The blue hollow arrows point to the reconstructed depth maps at the corresponding positions in the trajectory. The drone decelerates when navigating complex obstacles and accelerates in simpler scenarios, demonstrating dynamic speed adjustment based on obstacle density.

3.4.2. Reward Functions

The reward function is designed to ensure that the drone flies safely and efficiently. As reaching the target and avoiding collisions are sparse rewards, we introduce a progressive reward to efficiently guide the drone. The progressive reward is defined as follows:

$$\begin{aligned} r_{\text{progress}} = & \lambda_d \cdot d_{\text{hor}} + \lambda_b \cdot \left|\chi' + \psi - \beta'\right| + \\ & \text{sign}(v_{\text{hor}} - v_{\text{max}}) \cdot \lambda_v \cdot v_{\text{hor}} + \lambda_z \cdot d_z + \\ & \lambda_f \cdot |\chi'| + \lambda_a \cdot \|\mathbf{a}_{t-1} - \mathbf{a}_t\|, \end{aligned} \tag{3.5}$$

where λ_d, λ_b, λ_v, λ_z, λ_f, and λ_a are weights for each term. $\mathbf{a}_t$ is the acceleration from the policy at time t. The first two terms guide the drone towards the target by penalizing horizontal distance and promoting correct directionality. The third term penalizes high horizontal velocity (activated when $v_{\text{hor}} > v_{\text{max}}$ and $\lambda_v = 0$ for $v_{\text{hor}} < v_{\text{max}}$, where v_{max} represents the threshold for penalizing horizontal velocity). The fourth term addresses vertical distance, the fifth encourages forward flight, and the last penalizes jerk for smoother flight.

Then the whole reward function is defined as:

$$r = \begin{cases} r_{\text{exceed}} & \text{if } (p_t < p_{min} \text{ or } p_t > p_{max}) \\ & (p \in \{x, y, z\}) \\ \dfrac{r_{\text{arrive}}}{TRAV} & \text{if } \|\mathbf{d}\| < d_{min} \\ r_{\text{collision}} & \text{if collision} \\ r_{\text{progress}} & \text{otherwise} \end{cases} \tag{3.6}$$

where p_{min} and p_{max} are the minimum and maximum values of the coordinates at the boundary, respectively. We define r_{exceed} as the boundary-exceed penalty, r_{arrive} as the target arrival reward which can be obtained when the distance from the drone to the target point is less than d_{min}, and $r_{\text{collision}}$ as the collision penalty. $TRAV$, introduced by Nous et al. [97], measures environmental clutter, accounting for drone's size and complex obstacle shapes. Higher values indicate easier navigation. PPO, trained within a fixed time window, incentivizes faster flight for higher arrival rewards. To balance safety and agility, the arrival reward inversely correlates with $TRAV$, while the horizontal velocity penalty in Equation 3.5, moderates speed.

The episode terminates once any of the previously mentioned conditions are met, following which the drone is reset to a new random starting point. In our setup, the values are configured as follows: $r_{\text{exceed}} = -2.0$ for exceeding boundaries, $r_{\text{arrive}} = 10.0$ for reaching the target, and $r_{\text{collision}} = -2.0$ for collisions. The traversability range, $TRAV$, is set between 3 and 13. The progressive reward, r_{progress}, ranges from -0.2 to 0 as defined in Equation 3.5. Notably, this progressive reward is considerably smaller than the other rewards.

3.4.3. TRAINING IN VARYING COMPLEXITY ENVIRONMENTS

We use AvoidBench [81] to set up the RL environment, which is designed to test vision-based obstacle avoidance algorithms. AvoidBench builds on Flightmare [20] but includes larger bushes as obstacles, offering more environmental complexity than Flightmare's thin red trees. This complexity is adjustable via the radius of the Poisson distribution.

To enhance training efficiency, we start with a warm-up in a simpler environment (12-meter Poisson radius) to facilitate learning of basic navigation and achieving high rewards. We then increase the complexity (Poisson radius between 3.0 and 5.4 meters) to train the drone on speed adjustment relative to environmental density—speeding up in simpler settings and slowing in denser ones. This adaptive speed feature is crucial for balancing agility and safety in cluttered environments.

As illustrated in Figure 3.3(b), the task involves the drone flying from the green arrow (start point) to the red arrow (target). The trajectory is color-coded to represent the drone's velocity. Additionally, we display some predicted depth images generated by our memory-augmented latent representation. These images are presented as pairs of $(\hat{I}_{t-10}, \hat{I}_t)$. For instance, in the pairs $(\hat{I}_0, \hat{I}_{10})$ and $(\hat{I}_{10}, \hat{I}_{20})$, it is evident that the drone retains memory of the depth image $\hat{I}_{10}$ seen 10 timestamps earlier. Observations from pairs $(\hat{I}_{10}, \hat{I}_{20})$, $(\hat{I}_{70}, \hat{I}_{80})$, and $(\hat{I}_{205}, \hat{I}_{215})$ demonstrate the drone's tendency to decelerate when encountering complex obstacles and to accelerate when observed distances in the flight direction are larger, as seen in $(\hat{I}_{140}, \hat{I}_{150})$ and $(\hat{I}_{250}, \hat{I}_{260})$.

We collected depth images and their corresponding speeds along the trajectory in Figure 3.3(b) at 0.1s intervals. The images were divided into six groups based on their average grayscale value, with each group containing an equal number of images. The average grayscale and speed, along with their standard deviations, were then calculated to plot the curve of red circles shown in Figure 3.3(c). Speed responses to uniformly bright gray images, represented by green diamonds, are measured during stable zero acceleration periods, closely matching real navigation speeds. The plots show an inverse relationship between drone speed and obstacle proximity, with faster speeds when obstacles are distant and slower speeds as they get closer.

3.5. EXPERIMENTS

To assess MAVRL's effectiveness, we conducted a series of experiments. We trained various latent representations to predict past, current, and future depths, then compared their performance using a policy network with consistent parameters. We also evaluated MAVRL's varying speed feature by comparing a fixed-speed training regime to Agile-Autonomy [30] and Ego-planner [15], focusing on success rate and the balance between safety and agility. Finally, we implemented our network on an actual drone with minimal fine-tuning. It should be noted that all training and testing are performed in static obstacle environments.

All simulation experiments are run on a server with an Intel Core i7-13700K CPU

and an NVIDIA GeForce RTX 4090 GPU. To get enough training results for statistical analysis, we create 5 docker containers in the server and run per configuration 5 parallel, independent training processes.

3.5.1. LATENT REPRESENTATION

To assess the effectiveness of our latent representation, we conducted several ablation studies. Initially, we trained the network end-to-end, followed by training the policy network with encoder in the loop (no decoder and reconstruction loss). Then, we trained the policy with LSTM in the loop while the encoder was pre-trained as a VAE model.

We trained five policies with the same parameters but different random seeds for each ablation study, conducting 600 iterations with checkpoints every 20. Each policy was tested on four maps, performing 25 trials per map, resulting in $5 \times 4 \times 25$ trials per study. As depicted in Figure 3.4, our memory-augmented latent representation (purple line) surpassed both the encoder in-the-loop (pink line) and LSTM in-the-loop (yellow line) policies in obstacle avoidance, with shaded areas showing standard deviations.

Further, to assess the necessity of LSTM, we experimented with inputs of embedded I_{t-20} and I_t from the data buffer directly without an LSTM policy. The cyan line in Figure 3.4 indicates that the success rate is significantly higher when using the memory-augmented latent representation with LSTM compared to using embedded inputs alone. This confirms that LSTM's continuous memory capabilities substantially enhance performance in obstacle avoidance tasks.

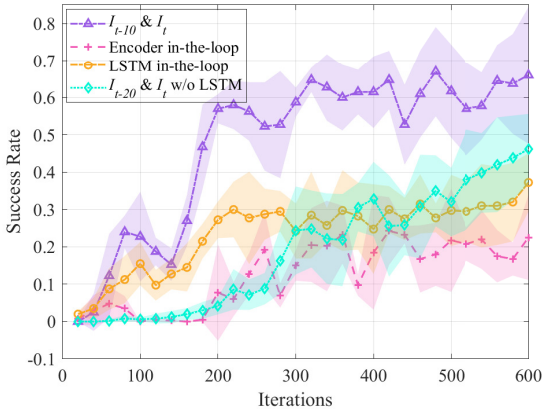


Figure 3.4: Average success rates of I_{t-10} & I_t supervised LSTM latent space policy (purple line), encoder in-the-loop without LSTM policy (pink line), LSTM in-the-loop policy (yellow line), and embedded I_{t-20} & I_t without LSTM policy (cyan line). The shaded area represents the standard deviation.

To evaluate the effectiveness of the memory-augmented latent representation, we conducted an experiment where the LSTM was trained using various reconstruction configurations. These configurations were then employed to train the policy network. We investigated eight distinct types of latent representations for this study:

- Current depth image prediction I_t,
- Future depth prediction I_{t+10} with only embedded depth as LSTM's inputs,
- Future depth prediction I_{t+10} with embedded depth, current actions $\mathbf{a}_t$ and states $\mathbf{x}_t$ as LSTM's inputs,
- Current depth I_t, with short-term future I_{t+10},
- Current depth I_t, with long-term future I_{t+20},
- Current depth I_t, and past depth I_{t-10},
- Current depth I_t, with more distant past I_{t-20},
- Current I_t, past I_{t-20}, and future I_{t+10} depth maps, also with current actions $\mathbf{a}_t$ and states $\mathbf{x}_t$ as LSTM's inputs,

where predicting current depth [22] and predicting future depth [39] separately are the most common in the literature. Considering the onboard computing constraints, we set the high-level control frequency to 10 Hz. To assess the impact of different memory lengths, we tested various T values to reconstruct current and past depths (I_t&I_{t-T}). Success rates of obstacle avoidance in the same evaluation environments are shown in Table 3.1. For the ablation studies, we selected $T = 20$, the best value from the fine-tuned hyperparameter results, and used $T = 10$ as a comparison reference.

Table 3.1: Influence of different T.

T	0	1	5	10	15	20	30
success rate	0.50	0.58	0.61	0.64	0.67	**0.74**	0.61

To evaluate how various latent representations affect policy network performance and the benefits of augmented memory, we conducted an extensive testing regimen. The policy network, using consistent latent representations, was trained ten times, each with a unique random seed, saving checkpoints every 20 iterations for a total of 600 iterations. The evaluation was conducted the same as the ablation study, and the results are detailed in Figure 3.5 and Table 3.2.

In Table 3.2, we compared the highest success rate checkpoints for each latent representation, detailing average success rates and standard deviations. Figure 3.5 displays the success rates of I_t, I_{t+10} without actions, I_{t+10} with actions, the superior combinations I_t&I_{t-20} (the better one compared with I_t&I_{t-10}), and I_t&I_{t+10} (the better one compared with I_t&I_{t+20}). The P-values from permutation tests [102] for each latent

representation compared to I_t and I_{t+10} are provided in the last two columns of Table 3.2. A P-value below 0.05 signifies significant differences. For example, the P-value for I_t&I_{t-20} versus I_{t+10} is 0.0016, suggesting only a 0.16% chance that the results are from the same distribution. Comparatively, the P-value for I_t&I_{t-20} versus I_t is 0.0, indicating a significant difference.

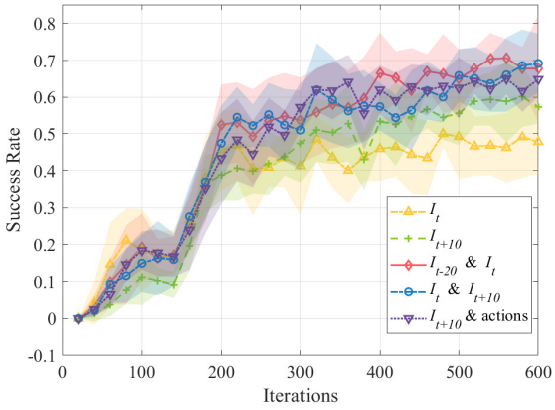


Figure 3.5: Success rates of I_t, I_{t+10} without actions, I_{t+10} with actions, and the superior combinations I_t&I_{t-20} and I_t&I_{t+10}. The shadow area represents the standard deviation.

Table 3.2: Comparison of different types of latent space.

Latent space	Mean ± std	P-Value (for I_t)	P-Value (for I_{t+10})
I_t	0.500 ± 0.037	-	-
I_{t+10}	0.601 ± 0.083	0.0015	-
I_{t+10} with actions	0.650 ± 0.042	0.0	0.0686
I_t&I_{t-10}	0.636 ± 0.089	0.0001	0.1913
I_t&I_{t-20}	**0.707 ± 0.051**	0.0	0.0016
I_t&I_{t+10}	0.692 ± 0.080	0.0	0.0110
I_t&I_{t+20}	0.664 ± 0.100	0.0001	0.0680
I_{t-20}&I_t&I_{t+10} with actions	0.700 ± 0.101	0.0	0.0120

Thus, we deduce that augmenting current depth with past or future information consistently outperforms predictions based solely on current depth. The combination of I_t and I_{t-20} emerged as the most effective, closely followed by the combination of current and future depth I_t&I_{t+10} which are both much better than only predicting future depth. Adding actions as input when predicting the future also improves the performance. Predicting I_{t-20}, I_t, and I_{t+10}, along with auxiliary predictions of the

drone's state and action, performs similarly to the latent representation obtained by predicting only I_t and I_{t-20}. This leads us to conclude that predicting past depth can be more beneficial in our task.

Since the LSTM training dataset was gathered using an initial policy, we compiled Table 3.3 to show the variations in predicting past, current, and future depths across different latent spaces, both before and after retraining with PPO. The mean and standard deviation, calculated from the grayscale values of the depth images, reveal that future prediction errors are the highest, though they slightly improve when actions are incorporated for I_{t+10}. Lower errors indicate that the latent space retains more features (3.2(b)). Reconstruction accuracy decreases slightly after retraining PPO.

Table 3.3: Errors of different prediction items.

Predicted Item	Mean ± std	
	(Before PPO retraining)	(After PPO retraining)
I_{t-20}	26.96 ± 12.01	34.07 ± 12.48
I_{t-10}	22.35 ± 10.09	30.19 ± 12.24
I_t	19.31 ± 8.96	22.66 ± 8.69
I_{t+10}	41.44 ± 22.04	49.46 ± 20.05
I_{t+20}	49.67 ± 25.51	52.68 ± 21.20
I_{t+10} with actions	36.34 ± 19.04	47.55 ± 18.18

Memory-augmented latent spaces significantly enhance drone navigation in environments with large obstacles, as shown in Figure 3.1(b). Drones with memory navigate around large obstacles more effectively, opting for longer, safer paths, while those without memory often become entrapped. This highlights the crucial role of memory in improving obstacle avoidance, particularly with larger obstructions.

3.5.2. BENCHMARKING FOR VARYING SPEED POLICY

Since predicting past and future depth images significantly improves policy performance, we use the combination of I_t and I_{t-20} as the latent representation for the following experiments. To evaluate the impact of a varying speed policy, we compared a policy network trained with variable speeds against one with a fixed speed. For the fixed speed setup, we modified the velocity penalty in the reward formula (Equation 3.5) to $\lambda_v \cdot |v_{\text{hor}} - v_{\text{desire}}|$, where v_{desire} is the desired velocity. While λ_v was set high, it remained lower than the collision penalty, allowing consistent speed learning. Both of the fixed speed model and varying speed model were then benchmarked against the learning-based Agile-Autonomy method [30] and the optimization-based Ego-planner [15] using the AvoidBench framework [81].

For MAVRL, it utilized the MPC controller from Agilicious [99] when outputting acceleration commands, which then generated body rate and thrust commands for

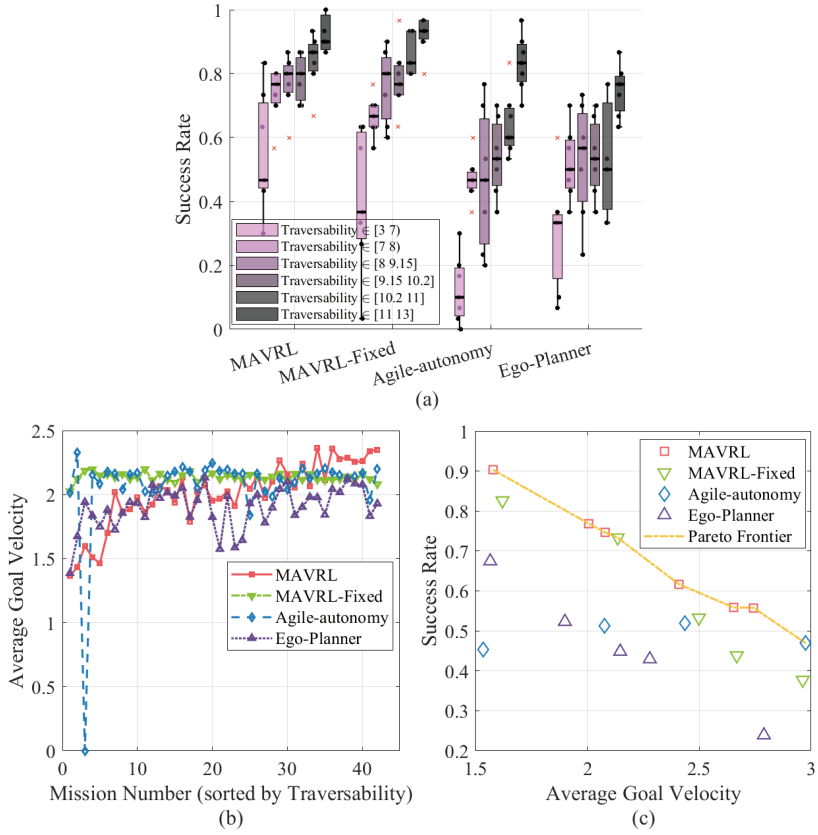


Figure 3.6: (a) is the success rate of 2 different MAVRL versions, Agile-Autonomy and Ego-Planner. (b) is the average goal velocity (AGV) of 2 different MAVRL versions, Agile-Autonomy and Ego-Planner. (c) is the Pareto frontier of success rate versus average flight speed.

the drone. Figure 3.6(a) shows the success rates over six groups, each with seven maps and 30 trials per map (1260 runs per method), indicating MAVRL with varying speed often performs best. Figure 3.6(b) explores the link between average goal velocity (AGV) [81] and map complexity, revealing MAVRL with varying speed tends to have higher AGV in less complex environments, while all algorithms maintain a similar AGV (around 2.0 m/s).

To validate MAVRL's superior performance across various agility levels when employing varying speeds, we fine-tuned the reward function parameters of both MAVRL variants to achieve different average flight speeds. This led to the construction of a Pareto frontier of success rate versus average flight speed, as shown in Figure 3.6(c). The results confirm that MAVRL with varying speed forms the Pareto frontier, dominating the results of the other methods. However, its average speed could not exceed 3.0 m/s due to flight distance limitations, although the maximum speed reached 5.5 m/s.

3.5.3. REAL WORLD TESTS

Figure 3.7: Real world test of MAVRL.

To validate MAVRL's real-world efficacy, we implemented our network on a real drone, maintaining the same architecture and hyperparameters as in our simulation experiments. Our test setup included a quadrotor equipped with 5-inch propellers and a Realsense D435i camera, powered by a Jetson Xavier NX featuring a 384-core GPU, 48 Tensor Cores, and a 6-core ARM CPU. On the RTX 4090 server, the network inference speed reaches 275Hz, while on the Xavier NX it reaches 15Hz. To ensure sufficient onboard computing power for MPC and data recording, we set the acceleration control frequency to 10Hz on both the simulator and the real drone, with the MPC generating a high-frequency low-level control command at 100Hz.

Depth images were captured with a Realsense D435i stereo camera facing forward on the drone, which has the same resolution and field of view setups as simulator. The simulation improved depth map accuracy by using the SGM algorithm without dis-

tortion, while the Realsense D435i achieved centimeter-level accuracy up to 3 meters in real world. Indoor positioning information is provided by Optitrack, while outdoor positioning information is provided by a RealSense T265 tracking camera.

Due to the real scene's environmental background shown in Figure 3.7 being too close to the obstacles, the VAE and LSTM trained in simulation struggled to differentiate between obstacles and background effectively. We collected approximately 1,200 real depth maps and fine-tuned the VAE and LSTM using a smaller learning rate, consistent with the training method described in Section III for these components. When using the PPO network trained in the original simulation environment, the latent space generated by the fine-tuned VAE and LSTM was still able to effectively perform obstacle avoidance navigation tasks. As shown in Figure 3.7, the drone successfully navigated a cluttered environment, utilizing a latent representation augmented with past memory. In the supplementary video, we show that a network fine-tuned indoors can operate outdoors but is more prone to collisions in forests due to small branches and leaves being underrepresented in the latent space.

3.6. CONCLUSION

Our approach leverages memory-augmented latent representations to endow the drone with a recollection of past scenarios. Experimental results demonstrated that reconstructing a more extensive history of past and current depth information significantly enhances the drone's performance in reinforcement learning-based obstacle avoidance tasks. Additionally, we established that adopting a varying speed strategy not only improves success rates but also strikes an optimal balance between safety and agility. The successful deployment of our network on a real drone, requiring minimal fine-tuning, marks a significant achievement. Looking forward, we will focus on improving the prediction and avoidance of dynamic obstacles while ensuring the retention of information about small obstacles, such as branches and leaves.

4

LEARNING TO SEE LIKE A SIMULATOR FOR REAL-WORLD DRONE NAVIGATION

Chapter 3 shows that high-fidelity simulation and complexity-aware training can produce strong varying-speed policies, but practical deployment requires these policies to remain reliable when both perception and dynamics deviate from the simulator. This motivates Research Question 3: bridging the sim-to-real gap for vision-based obstacle avoidance. Building directly on the learned representations and policies from the previous chapter, this chapter focuses on transferring depth-based navigation from simulation to real drones by explicitly addressing the two dominant sources of mismatch. It introduces a depth transfer approach that aligns simulated and real depth at the feature level via domain adaptation, while also reducing control sensitivity through a reference generation strategy that can be tracked consistently across platforms. Together, these components aim to preserve the behavior learned in simulation while enabling robust zero-shot deployment in visually complex real environments such as forests.

This chapter is based on the following article:
H. Yu, C. De Wagter and G. C. H. E. de Croon, *Depth Transfer: Learning to See Like a Simulator for Real-World Drone Navigation*, IEEE Robotics and Automation Letters (2025)

4.1. INTRODUCTION

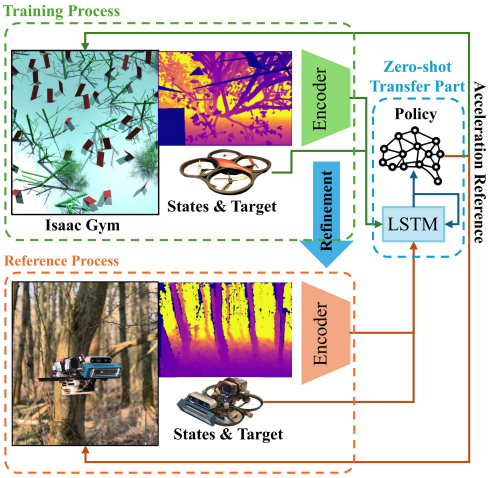
Vision-based navigation allows drones to fly autonomously in cluttered environments using only onboard sensing. Compared with map-based planning pipelines that rely on accurate global localization, reactive vision-based policies can be more tolerant to drift and intermittent perception failures, which makes them attractive for GPS-denied outdoor flight. At the same time, learning such policies directly in the real world is expensive and risky: collecting sufficient coverage of diverse obstacles and lighting conditions often requires extensive trial-and-error and can easily lead to collisions. Consequently, reinforcement learning (RL) is typically performed in simulation, where millions of interactions can be generated safely and efficiently using modern GPU-accelerated platforms [16, 19, 20]. However, this convenience comes with a well-known challenge: policies trained on simulated perception frequently degrade when deployed on real sensors due to domain shift.

Among visual modalities, depth is often considered a strong candidate for sim-to-real transfer because it provides geometry while discarding most appearance-specific factors such as texture and color. Nevertheless, "depth" in simulation and in the real world can be qualitatively different. Simulators typically provide clean and complete ground-truth depth, whereas real depth estimated from stereo matching is affected by textureless regions, reflective surfaces, motion blur, and lighting changes, producing missing pixels and spurious discontinuities. These artifacts can significantly alter the obstacle cues available to the policy and therefore become a major bottleneck for zero-shot deployment.

To address this gap, we propose a depth transfer method that explicitly aligns stereo depth images with simulated ground truth through representation learning and domain adaptation. Our key idea is to separate perception from control during training: we first learn a compact latent representation of depth using a variational autoencoder (VAE), and then train an RL policy on the latent state. Afterward, we adapt the encoder to real (or alternative simulator) stereo depth by enforcing feature-level domain invariance, so that the downstream policy can operate on a latent space that is consistent with what it encountered during RL training. This design allows us to reuse the same policy while updating only the perception module when the observation distribution changes.

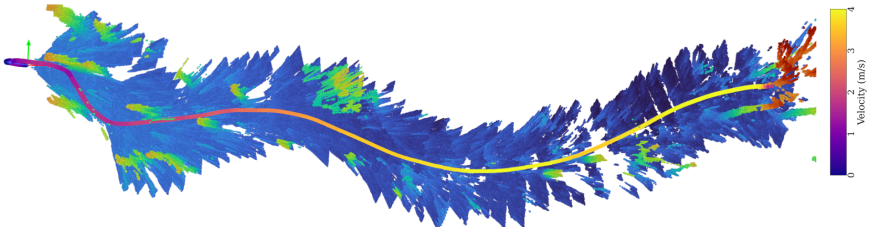
We evaluate the proposed approach in both simulation and real-world environments. In particular, we benchmark zero-shot transfer in AvoidBench [81] and validate real-world flight in forested scenes, where depth estimation noise and viewpoint changes are pronounced. Figure 4.1(b,c) illustrates the evaluation environments and representative trajectories. Together, these experiments demonstrate that aligning depth representations substantially improves robustness compared with directly deploying a simulator-trained encoder on real stereo depth.

The remainder of this chapter is organized as follows. We first review related work on RL-based obstacle avoidance and sim-to-real transfer. We then introduce the proposed depth transfer framework, including latent representation learning and feature-



(a)

(b)



(c)

Figure 4.1: (a) is the basic framework of the obstacle-free navigation system. When training, the depth images are generated from IsaacGym and the policy is trained in the simulator. During real-world deployment, the depth images are collected from real environments and the policy is evaluated using the refined encoder. (b) illustrates the forest environment for policy evaluation. (c) shows the trajectory from (b), the point clouds here are only for visualization.

level domain adaptation. Next, we describe the RL formulation and training details for obstacle avoidance. Finally, we present comprehensive evaluations in simulation (IsaacGym and AvoidBench) and real-world environments.

Our main contributions are as follows.

- We propose a depth transfer method that aligns stereo depth images with simulator ground truth, enabling RL policies trained entirely in the IsaacGym simulator to generalize effectively to both the photo-realistic simulator AvoidBench and real-world environments.
- We improve latent representation quality by optimizing the VAE architecture and applying min-pooling dilation to better capture obstacle structures in depth maps.

4.2. Related Work

This chapter is closely related to two research threads. The first concerns reinforcement learning for vision-based navigation and obstacle avoidance, where policies are trained end-to-end from raw sensory inputs to control commands. The second focuses on sim-to-real transfer, including domain randomization and domain adaptation techniques that reduce the discrepancy between simulated observations and real sensor measurements. In the following, we briefly review representative approaches and highlight how our method differs by adapting depth at the representation level while keeping the downstream policy fixed.

4.2.1. RL-based Obstacle Avoidance

RL-based obstacle avoidance methods typically formulate navigation as a Markov decision process and learn a policy through trial-and-error interactions. For MAVs, this learning paradigm is appealing because it can capture coupled perception–control effects and does not require explicit mapping or obstacle detection, which are often brittle at high speed. At the same time, the learned policy is only as reliable as the observation distribution it is trained on, which makes the choice of visual representation and training environment particularly important. RL has been widely applied to navigation tasks on various robotic platforms, including drones [34, 35]. Although effective in state-based settings, training RL agents directly from raw pixel data remains challenging due to the high dimensionality of visual inputs. To overcome the difficulty of training vision-based RL policies directly from high-dimensional input, previous work has adopted a two-stage strategy. First, an expert policy is trained using state-based observations, which are typically low-dimensional and easier to learn. Then, a vision-based student policy is trained via imitation learning to mimic the expert [33, 36].

Despite these successes, directly training end-to-end visuomotor policies can be sample-inefficient and may overfit to simulator-specific cues. An alternative solution

is to use representation learning with staged pretraining, which first extracts compact and informative features from high-dimensional inputs before training the policy, offering improved sample efficiency and generalization compared to fully end-to-end learning. For instance, Mihir *et al.*[38] introduced the Depth Image-based Collision Encoder (DCE), which addresses the challenge of high-dimensional depth input by compressing it into a compact latent space, while simultaneously enhancing the preservation of fine obstacle details through geometric expansion[22]. A well-structured latent space improves both navigation and obstacle avoidance performance. To incorporate temporal context, David *et al.* [39] used the Long Short-Term Memory (LSTM) to predict future depth from historical latent features. Building on this idea, a subsequent approach [85] enhances memory representation by predicting both the previous and current depth images, yielding a more structured and explicit latent space that outperforms memory-free baselines and methods relying solely on future predictions. We verify that optimizing the VAE architecture to improve depth reconstruction, combined with simple pixel-level dilation, can also preserve more obstacle details and enhance navigation success.

4.2.2. SIM-TO-REAL TRANSFER AND DOMAIN ADAPTATION

Transferring learned navigation policies from simulation to the real world has been studied extensively, and solutions range from increasing robustness during training to explicitly aligning source and target domains. Because our approach relies on stereo depth, we focus on methods that address observation mismatch rather than only dynamics mismatch.

Multiple techniques have been proposed to improve sim-to-real transfer. To improve generalization, domain randomization has been widely used by introducing physical perturbations in simulators [34, 35, 60, 62]. While effective for robotic dynamics and obstacle layouts, it struggles with complex visual discrepancies. For instance, [61] proposed using visually diverse simulation data, but this approach suffers from limited coverage and reduced training efficiency.

By optimizing contrastive loss between source and target embeddings, a drone racing policy can achieve zero-shot transfer [103]. However, this requires anchor–target image pairs with consistent scene elements, such as gate positions. In obstacle avoidance, real-world obstacles often differ in shape and size from simulation, making such correspondence difficult and reducing the effectiveness of contrastive loss.

Stereo vision has a number of challenges that lead to imperfect depth maps, including texture-poor areas, reflections, areas that are occluded in one of the images, etc. This leads to artifacts and invalid regions in stereo depth images, making them substantially different from ground-truth depth images available in simulations. One partial solution to this is to perform stereo vision with images generated in the simulator [30, 85]. However, doing this during reinforcement learning is computationally expensive and still differs from real-world data. Reducing the resolution of the depth map, as suggested in [104], can help mitigate the domain gap introduced by stereo

vision. However, this process often results in the loss of fine details, making it less effective for detecting thin or small obstacles.

Domain adaptation provides an alternative by aligning models trained on one domain with data from another. It has been widely explored in computer vision [67, 69, 70], and more recently in reinforcement learning [66]. Many methods adopt a two-stage pipeline: (1) learning domain-invariant representations, followed by (2) RL training [73, 74]. However, these methods often require data from multiple domains during training and are typically still validated in visual environments like CarRacing or CARLA [78].

In this work, we propose a feature-level domain adaptation method to bridge the gap between simulated and real-world depth. A VAE trained on IsaacGym ground truth depth is aligned with stereo depth via domain adaptation, enabling an RL policy trained in simple simulation to transfer effectively to AvoidBench [81] and real-world environments.

4.3. DEPTH TRANSFER BASED ON DOMAIN ADAPTATION

Following the training pipeline proposed in [85], we adopt a similar approach to train an RL agent for obstacle-free navigation. The VAE is used to embed high-dimensional depth images into a 64-dimensional latent space, which is then processed by a LSTM module. The LSTM is trained for the reconstruction of both the previous and current depth images, producing a 256-dimensional latent representation that encodes temporal information. This output is concatenated with the drone's state and target information and is used as input to the RL policy.

Concretely, the pipeline decouples perception learning and policy optimization. We first train a policy with ground-truth depth in a lightweight simulator to obtain stable control behavior, and then refine only the perception front-end so that real stereo depth is mapped into the same latent space. This staged strategy reduces the burden on RL to simultaneously solve representation learning and control, and it allows adaptation using unlabelled target-domain depth images without requiring additional RL rollouts in the real world.

To bridge the visual gap between simulation and the real world, we propose a depth transfer method based on domain adaptation. This approach aligns the latent spaces of depth images across domains, allowing the policy trained in simulation to be directly deployed in real-world environments. The full training process consists of the following steps:

1. Train an initial PPO policy in IsaacGym with a few obstacles using ground-truth depth images. The encoder and LSTM are integrated but randomly initialized and kept frozen during this step.
2. Collect depth images with the initial policy and train the VAE to map into a 64-dimensional latent space.

3. Train the LSTM to reconstruct both past and current depth images from the latent embeddings.
4. Retrain the PPO policy using the outputs from the trained LSTM, concatenated with the drone's state and target position, as policy input.
5. Perform domain adaptation by retraining the encoder on stereo depth images from IsaacGym, AvoidBench, or the real world to align their latent space with that of the simulated ground truth.
6. Evaluate the policy in AvoidBench and real-world environments using the refined encoder.

The first stage establishes a strong control policy under idealized depth, while the second stage focuses on making the depth encoder robust to the artifacts present in stereo reconstruction. We next describe the latent representation learning module and then introduce the domain adaptation objective used for sim-to-real depth transfer.

4.3.1. LATENT REPRESENTATION LEARNING

We employ a VAE to learn a latent representation of depth images. This latent space serves two roles: it compresses high-dimensional depth observations into a compact state for the RL policy, and it provides a feature space where domain alignment can be performed more reliably than at the pixel level. In practice, this design also reduces the sensitivity of the policy to sparse missing values in stereo depth, because the encoder can learn to "denoise" and complete local structure through the reconstruction objective. The VAE consists of an encoder E and a decoder D. The encoder maps depth images X to a latent space $\mathbf{z}_{\text{vae}}$, while the decoder reconstructs the depth images from this latent representation. The VAE is trained by minimizing both the reconstruction loss and the Kullback-Leibler (KL) divergence between the learned latent distribution and a prior distribution.

Unlike previous work [85], which relied solely on convolutional layers for feature extraction, we incorporate a residual neural network (ResNet) architecture to enhance the quality of the learned latent space. The ResNet architecture consists of a series of residual blocks, each containing two convolutional layers with skip connections. These skip connections facilitate gradient flow, allowing the network to learn more complex features and improving the reconstruction quality of depth images. In addition, to solve the problem of thin obstacles in depth maps, we apply a simple pixel-level dilation to expand the obstacle sizes in the depth images, which we call the min-pooling dilation. The min-pooling dilation can be expressed as follows:

$$Y_{i,j} = \min_{(m,n)\in\mathscr{R}_{i,j}} X_{m,n} \tag{4.1}$$

where X is the input depth map, $\mathscr{R}_{i,j}$ represents the receptive field (i.e., the pooling window) centered at position (i, j), and Y is the resulting output after the min-pooling operation.

As shown in the top part of Figure 4.2, the VAE is trained on depth images from IsaacGym to extract compact latent representations. The encoder consists of a 16-channel convolutional layer followed by six residual blocks, and outputs the mean and variance of the latent distribution. The decoder mirrors the encoder to reconstruct the input depth image. The VAE is optimized with a combination of reconstruction and KL divergence losses.

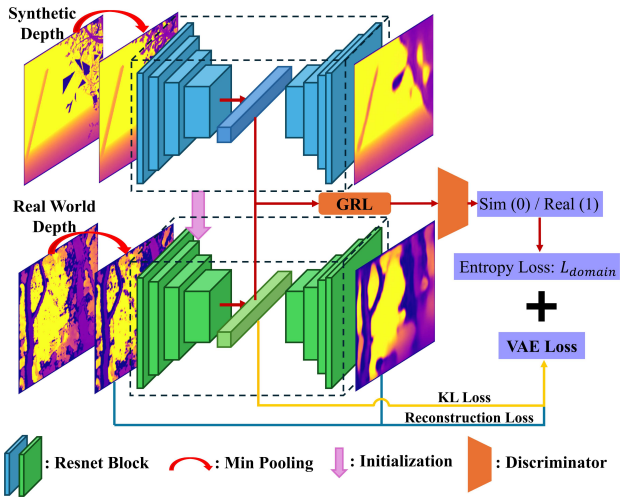


Figure 4.2: Domain adaptation for depth transfer. The VAE is trained using depth images from IsaacGym. The encoder maps depth images to a latent space, while the decoder reconstructs the depth images. The GRL and discriminator are used to align the latent spaces of simulated and real-world depth images.

As illustrated in Figure 4.1(a), the latent vector from the encoder is used as input to the LSTM. Following the architecture in [85], the LSTM is trained to reconstruct both the previous and current depth images to capture temporal context. Specifically, the LSTM output at time t is passed through a fully connected (FC) layer and then split into two latent vectors, $\hat{\mathbf{z}}_{t-T}$ and $\hat{\mathbf{z}}_t$, which correspond to the predicted latent space for time steps $t-T$ and t, respectively. These two latent vectors are then passed through the same VAE decoder used during pretraining to reconstruct the corresponding depth images, $\hat{Y}_{t-T}$ and $\hat{Y}_t$. Its output is concatenated with the drone's state and target position to train the PPO policy. The LSTM is optimized using the following loss:

$$\mathscr{L}_{\text{LSTM}} = \sum_{i=-1,0} \text{MSE}(Y_{t+iT}, \hat{Y}_{t+iT}), \tag{4.2}$$

where $\hat{Y}_{t+iT}$ denotes the reconstructed depth image at time $t+iT$ from the latent input $\mathbf{z}_t$, and T is the temporal interval between prediction steps. Following the setting validated in [85], we set $T = 20$.

4.3.2. SIM-TO-REAL DEPTH TRANSFER

After training the policy in IsaacGym, we collect stereo depth images from simulators or real-world environments and refine the VAE to align the resulting latent space with that learned from ground-truth depth in simulation, as shown in the bottom part of Figure 4.2. Before encoding, stereo depth images are preprocessed with min-pooling dilation to enlarge obstacle regions. The VAE is initialized with weights from simulation and fine-tuned using the collected stereo data.

To bridge the domain gap, we adopt a Gradient Reversal Layer (GRL) and a discriminator. The use of GRL and domain discriminator for adversarial domain adaptation was first introduced by Ganin et al. [68]. This approach enables feature alignment through backpropagation without requiring explicit domain labels during training. The GRL is placed between the latent space and discriminator to reverse gradients during backpropagation, encouraging the encoder to generate domain-invariant representations. The discriminator is trained to distinguish between simulated and real-world latent features, thus guiding the encoder toward alignment. During the forward pass, the GRL behaves as an identity function, i.e., $\mathbf{z}_{\text{grl}} = \mathbf{z}_{\text{vae}}$. where $\mathbf{z}_{\text{grl}}$ is the output of the GRL. During backpropagation, the gradient of the loss function L with respect to the input is reversed and scaled by λ_{grl}:

$$\frac{\partial L}{\partial \mathbf{z}_{\text{vae}}} = -\lambda_{\text{grl}} \frac{\partial L}{\partial \mathbf{z}_{\text{grl}}} \tag{4.3}$$

where λ_{grl} is a hyperparameter that controls the strength of the gradient reversal.

The discriminator is a binary classifier consisting of two fully connected layers followed by a log-softmax layer. It outputs a probability p indicating whether the input latent vector originates from the ground depth or the stereo depth. The entropy loss of the discriminator is combined with the VAE loss to update the VAE for stereo depth images. The loss function for the VAE is then updated as follows:

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{recon}} + \beta \mathcal{L}_{\text{KL}} + \gamma \mathcal{L}_{\text{domain}} \tag{4.4}$$

where $\mathcal{L}_{\text{domain}}$ is the domain loss, and γ is a hyperparameter that balances the domain loss with the reconstruction loss and KL divergence.

The domain adaptation loss $\mathcal{L}_{\text{domain}}$ is defined as a binary cross-entropy loss applied to the discriminator, which distinguishes whether a latent vector $\mathbf{z}_{\text{vae}}$ comes from ground-truth or stereo depth images:

$$\mathcal{L}_{\text{domain}} = -\mathbb{E}_{\mathbf{z}_{\text{vae}}^{\text{gt}}} \log D_{\text{dis}}(\mathbf{z}_{\text{vae}}^{\text{gt}}) - \mathbb{E}_{\mathbf{z}_{\text{vae}}^{\text{stereo}}} \log\left(1 - D_{\text{dis}}(\mathbf{z}_{\text{vae}}^{\text{stereo}})\right), \tag{4.5}$$

where $\mathbf{z}_{\text{vae}}^{\text{gt}} = E(X^{\text{gt}})$ and $\mathbf{z}_{\text{vae}}^{\text{stereo}} = E(X^{\text{stereo}})$ are latent vectors extracted from ground-truth and stereo depth images, respectively. The discriminator $D_{\text{dis}}(\cdot)$ outputs the probability that the input belongs to the ground-truth domain, and $\mathbb{E}[\cdot]$ denotes the batch-wise expectation.

Since the GRL inverts the gradient during backpropagation, the encoder is trained adversarially to fool the discriminator, encouraging the latent representations of ground truth and stereo depth images to become indistinguishable.

4.4. Reinforcement Learning for Obstacle Avoidance

In this section, we describe the reinforcement learning procedure used to train the obstacle avoidance policy. The policy takes as input the latent representation produced by the depth encoder (and the recurrent state) and outputs high-level navigation commands, which are tracked by a low-level controller. By training the policy on the latent space rather than raw depth, we obtain a compact and structured observation that is easier to optimize and can later be preserved during sim-to-real adaptation. We emphasize that the RL stage is performed entirely in simulation: real-world data are only used to adapt the encoder, enabling zero-shot deployment of the learned control policy.

4.4.1. Task Formulation

The obstacle avoidance task is formulated as a Markov Decision Process (MDP), defined by the tuple $\langle \mathscr{S}, \mathscr{A}, \mathscr{P}, \mathscr{R}, \gamma \rangle$. Here, $\mathscr{S}$ denotes the state space, which includes the drone's state, target information, and the latent representation extracted from depth images. The action space $\mathscr{A}$ is defined as $\boldsymbol{a} = [\boldsymbol{a}_{\text{cmd}}, \dot{\psi}]$, where $\boldsymbol{a}_{\text{cmd}}$ is the commanded acceleration and $\dot{\psi}$ is the yaw rate.

In our case, the transition function $\mathscr{P}$ is deterministic. We use the Aerial Gym simulator [19] for training the RL agent. Aerial Gym is built on top of the IsaacGym physics engine, which provides a drone model with realistic dynamics.

Given an action command $\boldsymbol{a}$, we employ the reference trajectory generator from Paparazzi[1] to produce smooth motion trajectories. By tuning the trajectory smoothing parameters, we can closely match the drone's response in simulation to that of the real platform, effectively reducing the sim-to-real gap caused by differences in dynamics and control.

A low-level controller maps the reference trajectory to the corresponding thrust and torque commands, which are applied to the drone model. The drone's state is subsequently updated through rigid-body dynamics simulated by the physics engine.

4.4.2. Observations and Reward Function

Observations

At each time step t, the observation $\boldsymbol{o}_t$ consists of a 14-dimensional physical state vector and a 256-dimensional perception vector obtained from the VAE and LSTM. The full observation is defined as:

$$\boldsymbol{o}_t = \left[\log d_{\text{hor}},\ d_z,\ \boldsymbol{d}_{\text{norm}},\ \boldsymbol{v}_{\mathscr{W}},\ \boldsymbol{e},\ \boldsymbol{\omega}_{\mathscr{B}},\ \boldsymbol{z}_t\right], \tag{4.6}$$

where d_{hor} and d_z denote the horizontal and vertical distances to the target; $\boldsymbol{d}_{\text{norm}}$ is the normalized direction vector to the target; $\boldsymbol{v}_{\mathscr{W}}$ is the linear velocity in the world

[1] wiki.paparazziuav.org

frame; $\boldsymbol{e} = [\phi,\ \theta,\ \psi]^\top$ and $\boldsymbol{\omega}_{\mathcal{B}}$ are the Euler angle and the angular velocity in the body frame; and $\boldsymbol{z}_t$ is the 256-dimensional latent representation of the depth image.

REWARD FUNCTION

The reward function consists of a **progress reward** r_{prog} and several **terminal rewards**. The progress reward is defined as:

$$\begin{aligned} r_{\text{prog}} = \ & \lambda_d \cdot \log d_{\text{hor}} + \lambda_z \cdot d_z + \max(0,\ v_{\text{hor}} - v_{\max}) \cdot \lambda_v \cdot v_{\text{hor}} \\ & + \lambda_{\text{dir}} \cdot \|\boldsymbol{d}_{\text{norm}} - \boldsymbol{v}_{\text{norm}}\|_1 + \lambda_{\text{input}} \cdot \|\boldsymbol{\omega}_{\mathcal{B}}\| \\ & + \lambda_{\text{perc}} \cdot \left(|v_{\mathcal{B},y}| + \max(0,\ -v_{\mathcal{B},x})\right), \end{aligned} \tag{4.7}$$

where λ_d, λ_z, λ_v, λ_{dir}, λ_{input}, and λ_{perc} are negative hyperparameters that shape the agent's behavior.

The first two terms penalize the horizontal and vertical distances to the target, encouraging the drone to approach it. The third term penalizes excessive horizontal velocity v_{hor} when it exceeds the predefined limit $v_{\max}$, ensuring safe and controlled motion. The fourth term penalizes the angular deviation between the normalized distance vector $\boldsymbol{d}_{\text{norm}}$ and velocity vector $\boldsymbol{v}_{\text{norm}}$, guiding the drone to move directly toward the target. The fifth term penalizes high angular velocity $\boldsymbol{\omega}_{\mathcal{B}}$, promoting stable attitude control. Finally, the last term introduces a perception-aware penalty by discouraging lateral and backward motion in the body-frame velocity $\boldsymbol{v}_{\mathcal{B}} = [v_{\mathcal{B},x},\ v_{\mathcal{B},y},\ v_{\mathcal{B},z}]$. This encourages the drone to maintain a forward-facing flight. This behavior does not emerge reliably without explicit penalization and was found to improve obstacle avoidance performance in our experiments.

This progress reward is designed to balance goal-reaching efficiency, flight stability, and safety, guiding the drone toward the target while maintaining a favorable attitude.

The overall reward function, composed of the progress reward and several terminal rewards, is defined as:

$$r = \begin{cases} r_{\text{exceed}}, & \text{if } (p_t < p_{\min} \text{ or } p_t > p_{\max}), \quad p \in \{x, y, z\} \\ r_{\text{arrive}}, & \text{if } \|\boldsymbol{d}\| < d_{\min} \\ r_{\text{collision}}, & \text{if collision occurs} \\ r_{\text{prog}}, & \text{otherwise} \end{cases} \tag{4.8}$$

where $p_{\min}$ and $p_{\max}$ define the allowable position boundaries, and r_{exceed} is the penalty for flying out of bounds. The term $d_{\min}$ is the arrival threshold, and r_{arrive} denotes the reward for successfully reaching the target. The term $r_{\text{collision}}$ penalizes collisions with obstacles, while r_{prog} refers to the progress reward defined in Equation 4.7.

4.5. EXPERIMENTS

We conduct a set of experiments to evaluate both components of the proposed method: (i) the quality of the learned depth representation and its ability to preserve obstacle geometry, and (ii) the resulting navigation performance under zero-shot transfer across simulators and in real environments. Unless otherwise specified, we report success rate and collision statistics and compare against encoder/policy variants that isolate the impact of each design choice.

4.5.1. DEPTH RECONSTRUCTION WITH MIN-POOLING DILATION

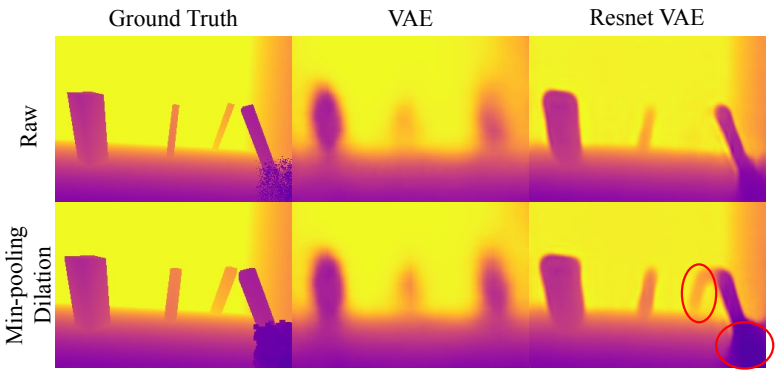


Figure 4.3: Depth reconstruction with min-pooling dilation. The first column shows the original depth and after dilation. The second and third columns compare the reconstructed depth images from a standard VAE and a VAE with a ResNet architecture.

Different from prior work [85], we apply min-pooling dilation to enlarge obstacle regions in depth images. Intuitively, this operation acts as a conservative "safety margin" in the perception space: thin structures and partial depth returns are expanded so that the encoder and policy are encouraged to treat them as solid obstacles. This is particularly beneficial for stereo depth, where small gaps and missing pixels can otherwise lead to underestimating the occupied space. As shown in the first column of Figure 4.3, this makes obstacles more prominent and easier to detect. The second and third columns compare reconstructions from a standard VAE [85] and a ResNet-based VAE, with the latter producing outputs much closer to the ground truth. When combined with min-pooling dilation, reconstruction quality improves further—especially for nearby regions—by preserving fine details and better separating obstacles from the background.

To assess the impact of the ResNet-based VAE and min-pooling dilation on obstacle avoidance, we trained four policies with different latent representations: standard VAE, VAE with dilation, ResNet VAE, and ResNet VAE with dilation. Evaluation was performed on 24 maps of varying complexity, each with 10 trials. As shown in Table 4.1,

both the ResNet architecture and min-pooling yield substantial performance gains, with their combination achieving the highest success rate. While the absolute success rates may appear modest, the task setting is highly challenging, featuring dense and randomized obstacle layouts. This controlled comparison allows us to attribute performance changes to representation quality versus observation preprocessing, rather than to differences in RL hyperparameters or training environments.

4.5.2. VISUALIZATION AND ANALYSIS OF LATENT REPRESENTATIONS

The RL policy will be trained in IsaacGym with ground truth depth images, since using the stereo depth means the simulator always need to render two images and run the stereo matching algorithm, which is time-consuming and not efficient. The goal of the domain adaptation is to align the latent spaces of the depth images from different sources, so that the policy trained in IsaacGym can be directly transferred to the visually more realistic AvoidBench and real-world environments.

To support the VAE encoder domain adaptation, we collected about 80,000 ground-truth depth images and 80,000 stereo depth images from IsaacGym, using the former as source data and the latter as target data. We further collected roughly 40,000 stereo depth images from AvoidBench and 6,000 real-world depth images with an Intel RealSense camera, both as target data. All simulation data were acquired automatically using a pre-trained policy, without manual intervention.

Table 4.2 summarizes an ablation study on the GRL-related hyperparameters λ_{grl} and γ, which control the gradient reversal strength and domain loss weight, respectively. The results show that our method remains effective across a range of values. The highest success rate (77.5%) is achieved when $\lambda_{\text{grl}} = 1.0$ and $\gamma = 50$, corresponding to a balanced domain loss. Performance degrades only when γ is too small ($\gamma = 10$), indicating excessive discriminator strength. These findings suggest that our framework is robust to hyperparameter choices.

To evaluate the effectiveness of feature-level domain adaptation, we use t-SNE [105] to visualize the latent space (Figure 4.4). The visualization includes four categories: ground-truth depth from IsaacGym (orange), stereo depth from IsaacGym (yellow), AvoidBench (blue), and real-world data (purple).

In the top-left plot of Figure 4.4, a single encoder trained on ground-truth depth produces clear domain separation, especially between ground truth and the three stereo-style depth categories. After applying domain adaptation (top-right), the distributions become closely aligned, indicating effective feature-level adaptation.

The bottom plots examine whether an encoder refined with IsaacGym or AvoidBench stereo depth can generalize to real-world inputs. Compared to the top-right configuration (with dedicated encoders), both single-encoder variants show poorer alignment with real-world depth images.

We quantify the alignment using the Geometric Separability Index (GSI) [106], where lower values indicate better inter-class mixing. We computed the GSI values for the

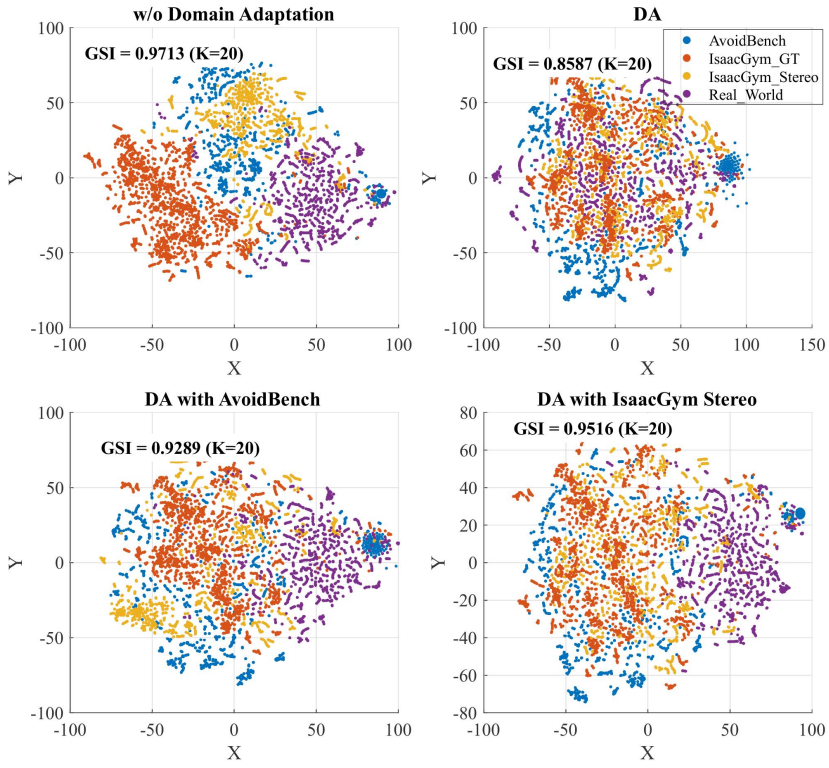


Figure 4.4: t-SNE visualization of latent space alignment. The plot includes ground truth depth images from IsaacGym (orange), stereo depth images from IsaacGym (yellow), stereo depth images from AvoidBench (blue), and real-world depth images (purple).

Table 4.1: Ablation studies for Resnet VAE and min-pooling dilation

	VAE	VAE & Dilation	Resnet VAE	Resnet VAE & Dilation
Success Rate (%)	50.4	64.2	59.6	**71.7**

Table 4.2: Ablation study of GRL hyperparameters

λ_{grl} / γ	0.5 / 50	1 / 1	1 / 10	1 / 20	1 / 50
Domain loss	0.542	0.231	0.309	0.453	**0.610**
Success Rate (%)	75	61.3	68.8	**77.5**	**77.5**

ground truth depth images from IsaacGym and real-world depth images under four different encoder configurations, as reported in each plot of Figure 4.4. The configuration where all categories use different encoders achieves the lowest GSI value (0.8587), indicating the best latent space alignment. In contrast, the configuration where the encoder was refined using stereo depth images from AvoidBench results in a higher GSI value (0.9289), though still lower than the baseline where all categories use the same encoder (0.9713). Refining the encoder with stereo depth images from IsaacGym also improves alignment (0.9516) for real-world depth images, though it is less effective than using AvoidBench for adaptation. These results confirm that domain adaptation effectively aligns the latent spaces of different depth inputs, with the best alignment achieved using separate encoders or AvoidBench-refined features for real-world deployment.

4.5.3. ZERO-SHOT POLICY TRANSFER IN ISAACGYM

We first test transfer in a controlled simulator setting to isolate perception shift while keeping dynamics and reward definitions consistent. Specifically, we evaluate policies with different encoders on stereo depth generated under varying rendering and noise configurations. These experiments quantify how much of the performance drop is attributable to the depth representation alone.

We assess the effectiveness of our depth transfer method by deploying a policy trained on ground truth depth images to stereo depth images in IsaacGym. Evaluation is conducted across eight maps with different obstacle layouts, each comprising ten trials with randomized start and target positions. The success rate is defined as the percentage of trials in which the drone reaches the target without collisions.

As shown in Figure 4.5, the policy trained on ground truth depth fails to generalize to stereo depth images without domain adaptation (left), resulting in frequent collisions and missed targets. After applying domain adaptation (right), the latent space is better

aligned, enabling successful navigation.

We also trained a stereo baseline policy using stereo depth images from IsaacGym. Owing to the higher computational cost, it required about 50 hours to converge, compared to 18 hours for the policy trained on ground-truth depth. All experiments were run on an RTX 4090 GPU. The evaluation results are summarized in Table 4.3. The *GT* column represents the upper bound performance, where evaluation is conducted using ground truth depth images. The *Stereo Baseline* column shows the performance of the stereo baseline policy, which is lower than the *GT* upper bound due to the challenges of training with stereo depth images. The *Stereo* column presents the evaluation results for a policy trained on ground truth depth images but evaluated directly with stereo depth images without any refinement, resulting in a significant performance drop. The *Refinement w/o DA* column shows results where the encoder is refined solely using the VAE loss, without domain adaptation. In contrast, the *Domain Adaptation* column represents a policy trained on ground truth depth images but evaluated with stereo depth images after applying domain adaptation, demonstrating improved generalization. All of the above policies use the VAE latent space as input. The highest success rate is achieved by the *LSTM & DA* configuration, which incorporates temporal memory via an LSTM and refines the encoder but not the policy using both reconstruction and domain adaptation losses.

4.5.4. ZERO-SHOT POLICY TRANSFER IN AVOIDBENCH

We next benchmark the same policies in AvoidBench, which provides more diverse scenes and photo-realistic rendering. This setting is closer to real-world deployment and stresses generalization across obstacle layouts and visual conditions. We follow the standardized evaluation protocol and report results across multiple environment difficulties.

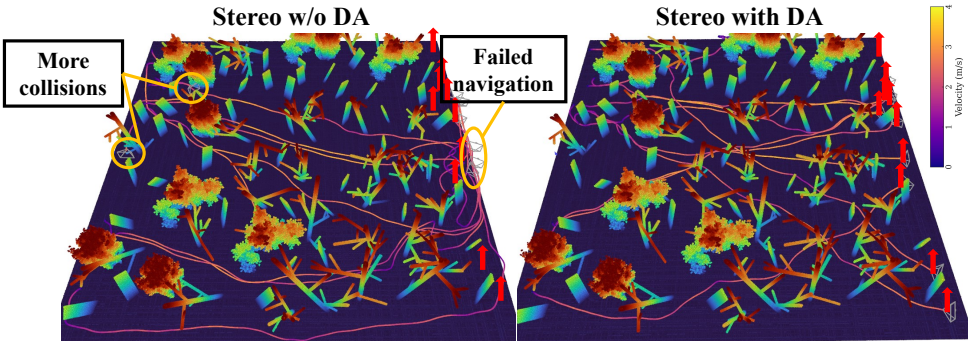


Figure 4.5: Policy evaluation in IsaacGym with different encoders. The left picture shows the policy evaluated with stereo depth images without domain adaptation, and the right picture shows the policy evaluated with stereo depth images after domain adaptation.

Table 4.3: Evaluations across different latent space configurations.

	GT	Stereo Baseline	Stereo	Refinement w/o DA	Domain Adaptation	LSTM & DA
Success Rate (%)	92.5	78.8	40.0	63.8	77.5	**81.2**

To evaluate the overall performance of our obstacle avoidance approach, we benchmarked the proposed method using AvoidBench. Two policies with different maximum speed limits were trained by fine-tuning the progress reward hyperparameters in Equation 4.7. These policies were then evaluated using stereo depth images as input. As shown in Figure 4.6, the x-axis denotes the average goal velocity [81], while the y-axis indicates the corresponding success rate. Each marker in the plot represents aggregated results over 1260 trials (30 trials per map across 42 maps).

Each policy is evaluated using three encoder configurations: an upward-pointing triangle denotes the encoder trained solely on ground truth depth images from IsaacGym without domain adaptation; a diamond-shaped marker represents the encoder refined with stereo depth images from IsaacGym; and a downward-pointing triangle indicates the encoder refined with stereo depth images from AvoidBench. Although the policies are trained in IsaacGym using ground truth depth images, the domain adaptation is conducted separately using stereo depth data from different sources. The points connected by blue dotted lines represent the results of the same policy using latent spaces generated by different encoders, while the red dotted line denotes the Pareto frontier.

Square markers represent MAVRL [85], which trains both the encoder and policy

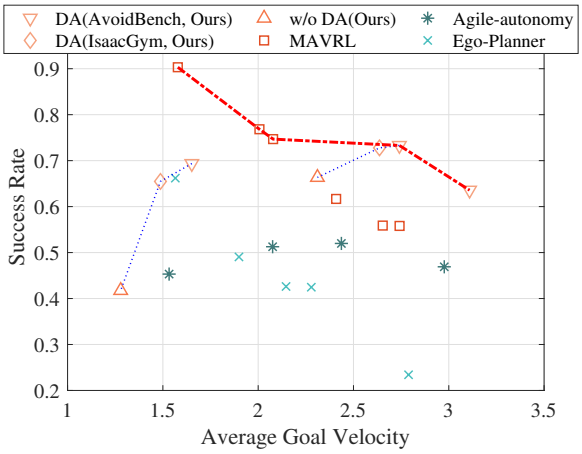


Figure 4.6: Pareto front of the proposed method compared to existing approaches.

directly in AvoidBench using stereo depth. Star and cross markers indicate Agile-Autonomy [30] and Ego-Planner [15], respectively. MAVRL performs best in low-speed scenarios (average velocity $< 2.1\ m/s$), likely due to full training in AvoidBench. In contrast, our method dominates the high-speed regime, benefiting from a reference generator that ensures consistent trajectory tracking and stable flight. Despite being trained entirely in IsaacGym, our policy generalizes effectively via domain adaptation—even when refined with stereo depth from different sources.

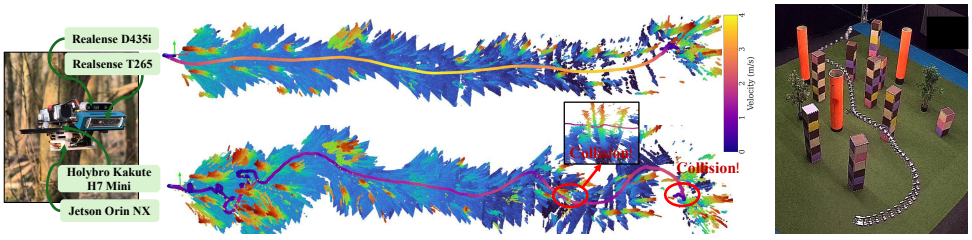
For the same policy, using different target data for domain adaptation leads to varying performance. The upward trend of the blue dashed line shows that employing stereo depth maps from IsaacGym and AvoidBench for domain adaptation substantially improves obstacle avoidance, with AvoidBench target data yielding the highest success rate and speed. These results highlight the effectiveness of our domain adaptation method in aligning latent spaces across domains, enabling robust zero-shot policy transfer across simulation and real-world environments.

4.5.5. ZERO-SHOT POLICY TRANSFER IN REAL-WORLD ENVIRONMENTS

Finally, we validate zero-shot transfer in real outdoor environments. The goal is to assess whether the adapted encoder can compensate for the compound effects of stereo noise, motion blur, and unmodeled environmental factors. We report qualitative trajectories and quantitative success statistics over multiple runs.

To validate our depth transfer method in real-world scenarios, we deploy the trained policy on a drone with a 150 mm wheelbase, equipped with an RealSense D435i for depth and a RealSense T265 for state estimation. The low-level controller runs on a Holybro Kakute H7 Mini with Betaflight, while the high-level policy runs on a Jetson Orin NX.

The RealSense D435i runs at 30Hz, while the policy network outputs actions at 10Hz to match onboard processing limits. These actions are sent to a low-level Model Predictive Control (MPC) operating at 100Hz [99]. With a maximum speed of 5.0m/s,



(a) (b)

Figure 4.7: Real-world experiments conducted in both outdoor (a) and indoor (b) environments demonstrate the effectiveness and generalization of our domain-adapted depth transfer method.

Table 4.4: Statistical results of real forest flight experiments.

	Success rate (%)	Average speed (m/s)	Number of collisions	Average jerk (m/s^3)
w/ DA	**70.0**	**2.15**	**3**	0.0437
w/o DA	20.0	0.64	5	**0.0385**

the drone travels 0.5m in 100ms, which is sufficient for timely reactions. An adaptive speed strategy [85] further improves robustness in complex environments.

Figure 4.7a illustrates the real-world flight trajectories of the drone using two different encoders: one trained on IsaacGym ground truth depth images (bottom trajectory) and another refined with domain adaptation (top trajectory). The results demonstrate that domain adaptation enables direct transfer of the policy trained in IsaacGym to real-world environments, allowing the drone to successfully avoid obstacles and reach the target. In contrast, when using the unrefined encoder, the drone exhibited erratic lateral movements in dense obstacle regions and collided twice, while also maintaining a lower overall speed compared to the domain-adapted encoder.

Table 4.4 presents the quantitative results from the cluttered forest flight experiments. With domain adaptation, both the success rate and average speed improved significantly, while the number of collisions was reduced. Most failures during domain-adapted flights occurred when the state estimator experienced substantial drift or when the drone encountered extremely small branches.

We also test the same policy in a challenging indoor environment (Figure 4.7b) using the encoder refined with outdoor stereo depth. The drone navigates narrow corridors and avoids obstacles, demonstrating strong generalization of our method to diverse real-world settings.

4.6. CONCLUSION

We presented a depth transfer approach that improves the sim-to-real robustness of vision-based obstacle avoidance for MAVs. The method combines three complementary ideas: (i) learning a compact latent representation of depth via a VAE, (ii) improving reconstruction fidelity around obstacles using min-pooling dilation, and (iii) aligning source and target depth distributions with feature-level domain adaptation. By decoupling perception adaptation from policy learning, we are able to keep the RL policy fixed and adapt only the encoder using unlabelled target-domain depth.

Extensive experiments in IsaacGym and AvoidBench demonstrate that the proposed encoder design and adaptation strategy lead to higher success rates under zero-shot transfer compared with baselines that deploy a simulator-trained encoder directly. Real-world flight results further confirm that representation-level depth alignment can mitigate common stereo artifacts and preserve safe navigation behavior in cluttered

outdoor scenes.

While the current study focuses on stereo depth, the overall framework is applicable to other geometric representations (e.g., monocular depth estimation or event-based depth proxies) and suggests a practical direction for deploying learning-based navigation systems: train control in simulation where interaction is cheap, and update perception using lightweight adaptation when the observation domain changes.

APPENDIX 4.A REFERENCE GENERATOR

The drone model is described as a rigid body with mass m, and a set of propellers that generate thrust. The motion of the drone is governed by the following rigid body dynamics:

$$\begin{bmatrix} \dot{\boldsymbol{p}}_{\mathcal{W}} \\ \dot{\boldsymbol{q}}_{\mathcal{W}\mathcal{B}} \\ \dot{\boldsymbol{v}}_{\mathcal{W}} \\ \dot{\boldsymbol{\omega}}_{\mathcal{B}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{\mathcal{W}} \\ \frac{1}{2}\boldsymbol{q}_{\mathcal{W}\mathcal{B}} \otimes \begin{bmatrix} 0 & \boldsymbol{\omega}_{\mathcal{B}}^{\mathsf{T}} \end{bmatrix}^{\mathsf{T}} \\ \boldsymbol{g}_{\mathcal{W}} + \frac{1}{m}\boldsymbol{R}_{\mathcal{W}\mathcal{B}}(\boldsymbol{f}_a + \boldsymbol{f}_d) \\ \boldsymbol{J}^{-1}(\boldsymbol{\tau}_a + \boldsymbol{\tau}_d - \boldsymbol{\omega}_{\mathcal{B}} \times \boldsymbol{J}\boldsymbol{\omega}_{\mathcal{B}}) \end{bmatrix}, \tag{4.9}$$

where $\boldsymbol{p}_{\mathcal{W}}$, $\boldsymbol{q}_{\mathcal{W}\mathcal{B}}$, $\boldsymbol{v}_{\mathcal{W}}$, and $\boldsymbol{\omega}_{\mathcal{B}}$ represent the drone's position, orientation (quaternion), linear velocity, and angular velocity, respectively. The thrust and torque produced by the propellers are denoted by $(\boldsymbol{f}_a, \boldsymbol{\tau}_a)$, while $(\boldsymbol{f}_d, \boldsymbol{\tau}_d)$ represent the aerodynamic drag. $\boldsymbol{g}_{\mathcal{W}}$ is the gravitational acceleration in the world frame, while $\boldsymbol{J}$ is the drone's inertia matrix. The rotation matrix $\boldsymbol{R}_{\mathcal{W}\mathcal{B}}$ transforms vectors from the body frame to the world frame.

We use the acceleration $\boldsymbol{a}_{\text{cmd}}$ and yaw rate $\dot{\psi}$ as the action space for the RL agent. A reference generator maps these commands into smooth attitude and thrust references to ensure feasible tracking. The desired roll (ϕ_{sp}), pitch (θ_{sp}), and yaw (ψ_{sp}) angles are computed as:

$$\begin{aligned} \theta_{\text{sp}} &= \tan^{-1}\left(\frac{a_{x,\text{cmd}}}{a_{z,\text{cmd}}}\right), \quad \phi_{\text{sp}} = \tan^{-1}\left(\frac{-a_{y,\text{cmd}}}{\sqrt{a_{z,\text{cmd}}^2 + a_{x,\text{cmd}}^2}}\right), \\ \psi_{\text{sp}} &= \psi + \dot{\psi}\Delta t, \end{aligned} \tag{4.10}$$

where $a_{x,\text{cmd}}, a_{y,\text{cmd}}, a_{z,\text{cmd}}$ are body-frame accelerations, and Δt (0.01 s) is the simulation time step.

To generate a smooth angular velocity reference, we define the attitude error $\boldsymbol{e}_\theta$ based on the desired quaternion $\boldsymbol{q}_{\mathcal{W}\mathcal{B},\text{sp}}$, and update the reference angular velocity using a second-order underdamped system with saturation constraints:

$$\begin{aligned} \boldsymbol{e}_\theta &= \boldsymbol{q}_{\mathcal{W}\mathcal{B}} - \boldsymbol{q}_{\mathcal{W}\mathcal{B},\text{sp}}, \\ \dot{\boldsymbol{\omega}}_{\mathcal{B},\text{sp}} &= -2\boldsymbol{\omega}_{\text{n}}\boldsymbol{\zeta}\boldsymbol{\omega}_{\mathcal{B},\text{sp}} - \boldsymbol{\omega}_{\text{n}}^2\boldsymbol{e}_\theta, \\ \dot{\boldsymbol{\omega}}_{\mathcal{B},\text{sp}} &= \text{clip}(\dot{\boldsymbol{\omega}}_{\mathcal{B},\text{sp}}, -\dot{\boldsymbol{\omega}}_{\max}, \dot{\boldsymbol{\omega}}_{\max}), \\ \boldsymbol{\omega}_{\mathcal{B},\text{sp}} &= \boldsymbol{\omega}_{\mathcal{B},\text{sp}} + \dot{\boldsymbol{\omega}}_{\mathcal{B},\text{sp}}\Delta t, \\ \boldsymbol{\omega}_{\mathcal{B},\text{sp}} &= \text{clip}(\boldsymbol{\omega}_{\mathcal{B},\text{sp}}, -\boldsymbol{\omega}_{\max}, \boldsymbol{\omega}_{\max}). \end{aligned} \tag{4.11}$$

Here, $\boldsymbol{\omega}_{\text{n}}$ is the natural frequency controlling convergence speed, and $\boldsymbol{\zeta}$ is the damping ratio that balances responsiveness and overshoot. $\dot{\boldsymbol{\omega}}_{\max}$ and $\boldsymbol{\omega}_{\max}$ limit angular acceleration and velocity to ensure physical feasibility and stability.

Similarly, the thrust is updated using a rate-limited approach:

$$\dot{T}_{\text{sp}} = \text{clip}\left(\frac{T_{\text{sp}} - T}{T_{\text{sim}}}, -\dot{T}_{\max}, \dot{T}_{\max}\right), \quad T_{\text{sp}} \leftarrow T + \dot{T}_{\text{sp}}\Delta t. \tag{4.12}$$

where T_{sim} is the RL action duration (0.1s in our case), and $\dot{T}_{\text{max}}$ is the maximum allowable thrust rate.

The reference linear acceleration, linear velocity, position, orientation, and angular velocity are then updated through Equation 4.9. This ensures that the reference states evolve smoothly, aligning with the drone's rigid body dynamics while maintaining stability and responsiveness.

5

HIERARCHICAL REINFORCEMENT LEARNING WITH DIFFERENTIABLE DYNAMICS

After Chapter 4 mitigates the key perception and dynamics gaps that hinder deployment, an important remaining question is how to further improve training efficiency and modularity without sacrificing performance. This directly leads to Research Question 4, which asks whether bio-inspired structure can accelerate learning and improve reusability. Drawing on the insight that end-to-end visuomotor policies remain expensive to train for long-horizon navigation, this chapter introduces a hierarchical perception–action architecture that separates high-level waypoint generation from low-level dynamic control. Inspired by biological navigation and stabilization mechanisms, it also adopts gimbal-stabilized perception to provide more consistent inputs during agile flight. Crucially, the low-level controller is trained via differentiable optimization through a dynamics model, reducing reliance on trial-and-error reinforcement learning and enabling faster convergence while retaining strong obstacle avoidance behavior.

5.1. INTRODUCTION

Biological navigation rarely operates through direct, muscle-level actuation. Instead, natural animals typically rely on hierarchical behavioral control, in which higher-level decisions (e.g., target selection or obstacle avoidance) are translated into intermediate motion primitives and ultimately into reflexive motor patterns. Biohybrid and cyborg navigation studies provide a particularly clear demonstration of this organization, because they can inject structured commands into the nervous system and observe how the animal's innate control loops transform them into coordinated locomotion. For example, cyborg insects navigate and avoid obstacles using simple sensory-level commands—such as left/right turning cues or forward activation—that trigger innate reflexive steering responses [107]. Robo-pigeons exhibit graded motor responses, where brain microstimulation parametrically modulates turning angle, turning curvature, and turning radius during outdoor flight [108]. Remote-controlled rats respond to virtual left/right cues delivered through somatosensory stimulation and sustain movement through reward-based forward drive, revealing another form of hierarchical cue–motivation control [109]. Across these systems, navigation emerges from multi-level directives—high-level cues, mid-level motion parameters, and low-level motor patterns—rather than direct manipulation of muscle-level actuation.

In addition, birds maintain remarkably stable visual perception during rapid flight by actively stabilizing their heads relative to the environment. Species such as pigeons exhibit distinct head-locking and head-bobbing behaviors that generate motion-invariant visual frames through vestibular feedback and rapid neuromuscular compensation [110, 111]. These biological gaze-stabilization mechanisms ensure that downstream perceptual and navigational processes operate on clean, stable sensory input. Inspired by these principles, modern drone systems increasingly adopt gimbal-mounted cameras to mechanically decouple visual perception from airframe motion [112, 113].

In contrast to end-to-end visual navigation approaches that map raw images directly to low-level motor commands or collective thrust and bodyrates commands (CTBR), our framework adopts a hierarchical perception–action architecture aligned with biological systems. A gimbal-stabilized camera first provides reliable visual input; a high-level network generates waypoint-based behavioral intentions; and a low-level controller converts these intentions into dynamically consistent CTBR commands. This hierarchical design mirrors the multi-level organization observed in birds and other animals, enabling more interpretable, robust, and biologically grounded drone navigation.

As illustrated in Figure 5.1, depth images from a gimbal-stabilized camera are embedded using a convolutional neural network (CNN) pretrained via Variational Autoencoder (VAE). The embedded latent features are concatenated with the drone's state information (position, velocity, orientation, angular velocity) and target point coordinates, forming the input to a high-level policy network. This network outputs waypoint commands that specify desired future positions. The front-end high-level policy is trained using Proximal Policy Optimization (PPO) within a simulated environ-

ment featuring randomized obstacle layouts to promote generalization. Compared to end-to-end training from depth latend space and drone's states to CTBR commands, this hierarchical approach simplifies the learning problem, enhances interpretability, and improves sample efficiency. The waypoint commands serve as intermediate goals that guide the low-level controller, which translates them into dynamically feasible CTBR commands while respecting the drone's physical constraints. Besides, instead of using PPO, the low-level controller is trained via differentiable optimization which means the dynamics model is differentiable and can be unrolled through time. This allows for gradient-based optimization of the low-level controller, leading to more precise and efficient control policies.

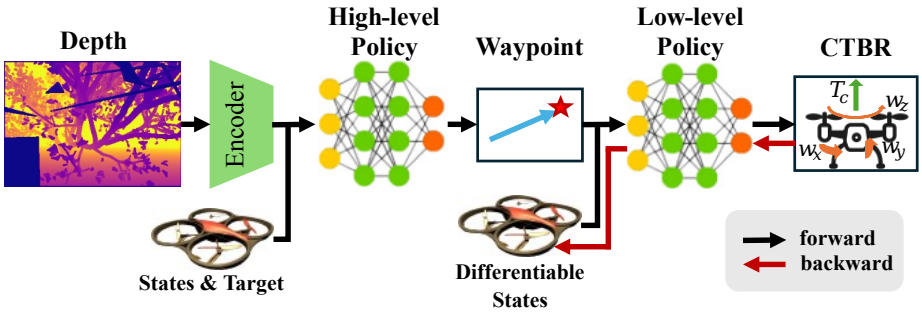


Figure 5.1: Hierarchical perception–action architecture for drone navigation.

Our main contributions are summarized as follows:

- Inspired by biological hierarchical control systems, we propose a novel hierarchical perception–action architecture for drone navigation that separates high-level waypoint generation from low-level dynamic control. Ablation studies demonstrate that this hierarchical design significantly outperforms end-to-end approaches in terms of navigation success rate and sample efficiency.
- We introduce a differentiable dynamics model for the low-level controller, enabling gradient-based optimization of control policies. This approach leads to more accurate and dynamically consistent CTBR commands, improving flight stability and responsiveness.
- Inspired by avian gaze stabilization mechanisms, we employ a gimbal-stabilized camera to provide stable visual input for navigation. Ablation experiments show that this setup enhances perception quality and navigation performance compared to fixed-camera configurations.

5.2. RELATED WORK

5.2.1. RL-BASED OBSTACLE AVOIDANCE AND NAVIGATION

End-to-end training from raw high-dimensional sensory input to low-level control commands is quite challenging. Most works choose to transform the raw sensory input

into a low-dimensional state space as the input for RL. By converting onboard images into the pose of gates and combining with the drone's state, [34] achieves vision-based autonomous drone racing. Equipped with an RL policy that outputs CTBR commands trained end-to-end, the AI drone outperforms champion-level human pilots for the first time. SkyDreamer [83] achieves end-to-end vision-to-control drone racing by feeding a gate-segmentation mask (with IMU gyros and motor RPM) into a Dreamer-style world-model RL policy that outputs direct motor PWM commands, eliminating explicit state estimation and cascaded control at deployment. However, for more complex navigation tasks such as obstacle avoidance in forest environments, end-to-end training from simplified visual information to low-level control commands such as CTBR or motor PWM remains quite difficult and sample-inefficient.

[32] attributes the inefficieny of end-to-end vision RL mainly to hard exploration under high-dimensional observations, and improves sample efficiency by bootstrapping: train a privileged-state RL teacher, distill it to a vision-based student, then fine-tune with RL using an asymmetric actor-critic setup. To improve sample efficiency, some works choose to use a higher-level action space, such as velocity [84] or acceleration [85, 86]. In our previous work [85], VAE and LSTM are used to encode a sequence of depth images into a low-dimensional latent space, which is then combined with the drone's state as the input to a PPO policy that outputs acceleration commands. A MPC controller is followed to convert the acceleration commands into CTBR commands. Malczyk et al. [84] leverage semantics-augmented perception (masked depth plus egocentric maps) to learn an inspection navigation policy that outputs high-level motion setpoints—desired linear velocity and yaw rate—which are then tracked by a separate low-level controller rather than directly commanding actuators.

5.2.2. DIFFERENTIABLE SIMULATION FOR DRONE NAVIGATION

Differentiable simulation has been increasingly adopted for robot learning and control due to its ability to provide gradient information through complex dynamics, enabling more efficient optimization of control policies [75, 88, 90]. Wiedemann et al. [88] exploit differentiable simulation to train trajectory-tracking controllers offline with Analytic Policy Gradients (APG), i.e., backpropagating tracking-error gradients through a differentiable dynamics model to directly optimize the policy (instead of sampling-based RL or online MPC). Building on offline controller optimization via analytic gradients in differentiable simulators, Pan et al. [75] extends this idea to real-world online adaptation: it continuously learns a residual dynamics model from flight data to refine a hybrid differentiable simulator, and then backpropagates through it to update the policy within seconds under unseen disturbances. Differentiable simulation is highly effective for training drone control policies because the underlying dynamics are inherently differentiable, enabling direct gradient-based optimization. However, applying it to vision-based avoidance is more challenging since perception and collision/event interactions are often non-differentiable, making end-to-end backpropagation difficult.

To address this, [90] enables gradient-based training for vision-based obstacle avoid-

ance by using a fully differentiable physics + depth-rendering simulator and replacing discrete collisions with a smooth distance-based barrier cost (computed from analytic distances to simple geometric obstacles and the closest-distance operator), so policy gradients can be backpropagated through time. DiffAero [77] is a GPU-native differentiable quadrotor simulator (physics + depth/LiDAR rendering in PyTorch) that enables large-scale training and benchmarking of gradient-based and hybrid policy learning, with an ONNX→ROS/PX4 deployment pipeline. Same as [90], its sensing/geometry queries rely on primitive obstacles (e.g., spheres/cuboids) and analytic ray–primitive intersections to compute closest-range/shortest-distance measurements efficiently. These works enable efficient differentiable avoidance by representing obstacles as simple primitives and using analytic closest-distance (or ray–primitive) computations to define smooth barrier costs. However, such primitive-only geometry struggles to model complex irregular obstacles, leading to a sim-to-real gap that is difficult to eliminate.

Compared with prior differentiable-avoidance methods that rely on primitive obstacle geometries, GRaD-Nav [79, 80] computes the nearest-obstacle distance from a 3D Gaussian Splatting (3DGS) reconstruction (via a 3DGS-derived point cloud within the camera FOV), which can better capture real scene geometry and thus helps reduce the sim-to-real gap while retaining a distance-based, gradient-friendly avoidance signal. However, it relies on pre-building a 3DGS map, which typically requires substantial offline data collection and compute.

Our work builds on these differentiable-avoidance foundations but introduces a hierarchical architecture that separates high-level waypoint generation from low-level dynamic control. For the high-level policy, we use PPO to handle the non-differentiable perception and obstacle interactions. For the low-level controller, we leverage differentiable dynamics to enable gradient-based optimization of CTBR commands, improving flight stability and responsiveness. This hierarchical design combines the strengths of both RL and differentiable optimization, leading to more effective and robust drone navigation. Futhermore, when migrating to different high-level tasks, such as tracking and exploration, only the high-level policy needs to be retrained while the low-level controller can be reused directly, greatly improving the reusability of the learned controller.

5.3. METHODOLOGY

In this section, we detail our hierarchical perception–action architecture for drone navigation, comprising a high-level waypoint policy trained via PPO and a low-level controller optimized through differentiable dynamics. The simulator for training and evaluation is built on IsaacLab, which provides accurate quadrotor dynamics and realistic rendering.

5.3.1. PROBLEM FORMULATION

We formulate vision-based obstacle avoidance as a Markov decision process (MDP) $\langle \mathcal{X}, \mathcal{A}, P, r, \gamma \rangle$. At time step t, the system is in state $x_t \in \mathcal{X}$, executes an action $\boldsymbol{a}_t \in \mathcal{A}$, transitions according to

$$x_{t+1} \sim P(\cdot \mid x_t, \boldsymbol{a}_t), \tag{5.1}$$

and receives reward $r_t = r(x_t, \boldsymbol{a}_t)$ with discount factor $\gamma \in (0,1)$. The objective is to learn a stochastic policy $\pi_\psi(\boldsymbol{a}_t \mid x_t)$ that maximizes the expected discounted return

$$J(\psi) = \mathbb{E}_{\pi_\psi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]. \tag{5.2}$$

The state is defined as

$$x_t = \left(\boldsymbol{p}_t, \boldsymbol{q}_t, \boldsymbol{v}_t, \boldsymbol{\omega}_t, \boldsymbol{g}_t, \boldsymbol{I}_t\right), \tag{5.3}$$

where $\boldsymbol{p}_t$ is the drone position, $\boldsymbol{q}_t$ is the attitude, $\boldsymbol{v}_t$ is the linear velocity, $\boldsymbol{\omega}_t$ is the angular velocity, $\boldsymbol{g}_t$ specifies the goal, and $\boldsymbol{I}_t$ is the onboard depth image. The action corresponds to a CTBR command for quadrotor control,

$$\boldsymbol{a}_t = \boldsymbol{u}_t = \left(c_t, \boldsymbol{\omega}_t^{\text{cmd}}\right) \in \mathcal{U}, \qquad \boldsymbol{\omega}_t^{\text{cmd}} = \left(\omega_{x,t}^{\text{cmd}}, \omega_{y,t}^{\text{cmd}}, \omega_{z,t}^{\text{cmd}}\right), \tag{5.4}$$

where c_t is collective thrust and $\boldsymbol{\omega}_t^{\text{cmd}}$ is the commanded body-rate vector.

Although the task can be described by a single Markov policy $\pi_\psi(\boldsymbol{u}_t \mid x_t)$, we implement it as the composition of two neural modules evaluated at the same control frequency: a waypoint generator and a low-level CTBR controller. Let $\boldsymbol{z}_t$ denote the latent representation extracted from the depth image $\boldsymbol{I}_t$. The waypoint generator takes as input

$$o_t^{\text{wp}} = \left(\boldsymbol{z}_t, \Delta\boldsymbol{g}_t, \boldsymbol{q}_t, \boldsymbol{v}_t\right), \qquad \Delta\boldsymbol{g}_t = \mathcal{T}\left(\boldsymbol{p}_t, \boldsymbol{g}_t\right), \tag{5.5}$$

where $\mathcal{T}(\cdot)$ expresses the goal in the world frame. It outputs an instantaneous waypoint command

$$\boldsymbol{w}_t = \pi_\theta^{\text{wp}}\left(o_t^{\text{wp}}\right), \qquad \boldsymbol{w}_t \in \mathcal{W}. \tag{5.6}$$

We parameterize $\boldsymbol{w}_t$ as

$$\boldsymbol{w}_t = \left(\Delta\psi_t, \Delta h_t, v_t^{\text{ref}}\right), \tag{5.7}$$

where $\Delta\psi_t$ is a heading offset, Δh_t is a vertical offset, and v_t^{ref} is a desired forward speed. This waypoint provides a compact, interpretable navigation command that is updated at every time step based on the current perception features and goal information.

The low-level controller conditions on the tracking state

$$\hat{x}_t = \left(\boldsymbol{p}_t, \boldsymbol{q}_t, \boldsymbol{v}_t, \boldsymbol{\omega}_t\right) \tag{5.8}$$

and the generated waypoint command $\boldsymbol{w}_t$, and outputs the CTBR command

$$\boldsymbol{u}_t = \pi_\phi^{\text{ctl}}(\hat{x}_t, \boldsymbol{w}_t). \tag{5.9}$$

Composing both modules yields the overall Markov policy

$$\boldsymbol{u}_t = \pi_\psi(x_t) = \pi_\phi^{\text{ctl}}\Big(\hat{x}_t, \pi_\theta^{\text{wp}}(o_t^{\text{wp}})\Big), \qquad \psi \doteq (\theta, \phi), \tag{5.10}$$

which maps the current state to a CTBR command and therefore defines a standard MDP.

5.3.2. LATENT SPACE REPRESENTATION OF DEPTH IMAGES

To extract compact and informative features from high-dimensional depth images, we still employ the same latent space representation method introduced in Chapter 4. Following MAVRL, we adopt a memory-augmented latent depth representation to compress high-dimensional perception while retaining short-term obstacle memory. A CNN-based VAE encodes each depth observation into a compact latent z_t^{vae} (64-D), and a single-layer LSTM aggregates these latents into a higher-capacity state z_t (256-D). During representation learning, z_t is decoded (sharing the VAE decoder) to reconstruct the current and past depth maps using an MSE-based auxiliary loss, explicitly encouraging the latent to store recent observations beyond the current field of view.

This latent representation is beneficial because it compresses high-dimensional depth observations into a compact state that is easier to learn from, while the LSTM and auxiliary reconstruction losses encourage the latent to retain short-term geometric memory (e.g., recently seen obstacles outside the current FOV), improving sample efficiency and robustness compared with direct pixel-based policies.

5.3.3. HIGH-LEVEL WAYPOINT GENERATION POLICY VIA PPO

PPO is a widely used policy-gradient method for training RL policies. It maximizes a clipped surrogate objective that constrains the policy update and avoids overly large parameter changes. The clipped objective is

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t\right)\right], \tag{5.11}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies, $\hat{A}_t$ is an advantage estimate, and ϵ controls the clipping range. In our framework, PPO is used to train the high-level waypoint generation policy π_θ^{wp}.

The input to π_θ^{wp} is the concatenation of the depth latent $\boldsymbol{z}_t$, the relative goal information $\Delta\boldsymbol{g}_t$, the attitude $\boldsymbol{q}_t$, and the linear velocity $\boldsymbol{v}_t$. The relative goal information encodes both the vertical displacement to the target and the direction toward it:

$$\Delta\boldsymbol{g}_t \doteq \begin{bmatrix} \Delta h_t^g \\ \hat{\boldsymbol{d}}_t \end{bmatrix} \in \mathbb{R}^4, \qquad \Delta h_t^g = g_{z,t} - p_{z,t}, \qquad \hat{\boldsymbol{d}}_t = \frac{\boldsymbol{g}_t - \boldsymbol{p}_t}{\|\boldsymbol{g}_t - \boldsymbol{p}_t\|_2}. \tag{5.12}$$

Here, Δh_t^g is the height difference between the goal and the drone, and $\hat{\boldsymbol{d}}_t$ is the 3D unit vector pointing from the drone toward the goal.

The output of the waypoint policy is the waypoint command $\boldsymbol{w}_t$. As illustrated in Figure 5.2, we parameterize

$$\boldsymbol{w}_t = \left(\Delta\psi_t, \Delta h_t, v_t^{\text{ref}}\right), \tag{5.13}$$

where $\Delta\psi_t$ is a heading offset, Δh_t is a vertical offset, and v_t^{ref} is a reference forward speed.

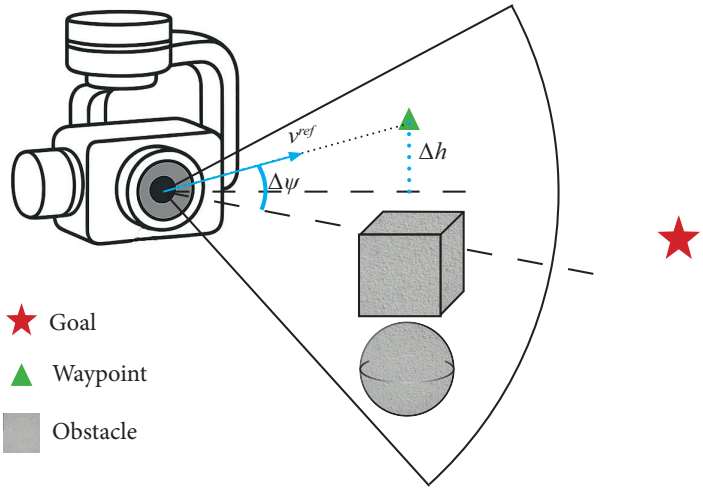


Figure 5.2: Illustration of waypoint generation.

The reward encourages goal reaching while penalizing unsafe behaviors:

$$r = \begin{cases} r_{\text{exceed}}, & \text{if } (p_t < p_{\min} \text{ or } p_t > p_{\max}), \quad p \in \{x, y, z\}, \\ r_{\text{arrive}}, & \text{if } \|\boldsymbol{d}_t\| < d_{\min}, \\ r_{\text{collision}}, & \text{if collision occurs}, \\ r_{\text{prog}}, & \text{otherwise}, \end{cases} \tag{5.14}$$

where $\boldsymbol{d}_t = \boldsymbol{g}_t - \boldsymbol{p}_t$ is the goal displacement vector and $d_{\min}$ is the arrival threshold. The terms r_{exceed} and $r_{\text{collision}}$ penalize boundary violations and collisions, respectively, while r_{arrive} provides a terminal reward upon reaching the goal.

The progress reward is defined as

$$r_{\text{prog}} = \lambda_z^{\text{hl}} |\Delta h_t^g| + \lambda_{\text{dir}}^{\text{hl}} \left(1 - \begin{bmatrix} \cos(\psi_t + \Delta\psi_t) \\ \sin(\psi_t + \Delta\psi_t) \end{bmatrix}^{\top} \hat{\boldsymbol{d}}_{t,1:2}\right) - \lambda_v^{\text{hl}} v_t^{\text{ref}}, \tag{5.15}$$

where $\hat{\boldsymbol{d}}_{t,1:2}$ denotes the first two components of $\hat{\boldsymbol{d}}_t$, ψ_t is the current yaw angle, and λ_z^{hl}, $\lambda_{\text{dir}}^{\text{hl}}$, and λ_v^{hl} weight the height-alignment, direction-alignment, and speed-regularization terms, respectively. In our implementation these coefficients are negative, such that larger misalignment incurs a stronger penalty and higher forward

speed is encouraged. This shaping directly constrains the waypoint output to align the drone heading and altitude with the goal direction, which empirically stabilizes training and accelerates convergence. The waypoint generation policy is encouraged to fly fast, but the terminal penalties for collisions and boundary violations lead it to reduce the reference speed in cluttered environments to maintain safety.

5.3.4. LOW-LEVEL CONTROL POLICY VIA DIFFERENTIABLE DYNAMICS

QUADROTOR DYNAMICS MODEL

The low-level controller π_ϕ^{ctl} maps the current tracking state $\hat{x}_t$ and the waypoint command $\boldsymbol{w}_t$ to a CTBR command $\boldsymbol{u}_t$. Unlike the high-level waypoint policy trained via PPO, we optimize the low-level controller using differentiable dynamics, enabling gradient-based optimization of control policies. Since IsaacLab does not support differentiable simulation, we implement a separate differentiable dynamics module in PyTorch for low-level controller training. This model can fit the dynamics of the IsaacLab [18] quadrotor simulator with high accuracy, which allows us to directly use depth images rendered based on IsaacLab.

We employ a rigid-body quadrotor model with a rate inner-loop that maps the CTBR command $\boldsymbol{u}_t = (c_t, \boldsymbol{\omega}_t^{\text{cmd}})$ to forces and torques. Using the previously defined state variables $\boldsymbol{s}_t = (\boldsymbol{p}_t, \boldsymbol{q}_t, \boldsymbol{v}_t, \boldsymbol{\omega}_t)$, the continuous-time dynamics are

$$\begin{cases} \dot{\boldsymbol{p}}_t = \boldsymbol{v}_t, \\ \dot{\boldsymbol{v}}_t = \dfrac{1}{m}\big(\boldsymbol{F}_t + \boldsymbol{F}_{\text{drag},t}\big) + \boldsymbol{g}, \\ \dot{\boldsymbol{\omega}}_{b,t} = \mathbf{I}^{-1}\big(\boldsymbol{\tau}_t - \boldsymbol{\omega}_{b,t} \times (\mathbf{I}\boldsymbol{\omega}_{b,t})\big), \\ \dot{\boldsymbol{q}}_t = \frac{1}{2}\, \boldsymbol{q}_t \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{b,t} \end{bmatrix}. \end{cases} \tag{5.16}$$

where $\mathbf{R}(\boldsymbol{q}_t) \in SO(3)$ is the rotation matrix corresponding to $\boldsymbol{q}_t$, $\boldsymbol{\omega}_{b,t} = \mathbf{R}(\boldsymbol{q}_t)^\top \boldsymbol{\omega}_t$ is the body-frame angular velocity, $\mathbf{I}$ is the body-frame inertia matrix, and $\otimes$ denotes quaternion multiplication. The thrust command is converted to physical thrust by $T_t = c_t T_{\max}$ and applied along the body z axis, yielding the world-frame thrust force

$$\boldsymbol{F}_t = \mathbf{R}(\boldsymbol{q}_t) \begin{bmatrix} 0 \\ 0 \\ T_t \end{bmatrix}. \tag{5.17}$$

We include quadratic aerodynamic drag

$$\boldsymbol{F}_{\text{drag},t} = -\tfrac{1}{2}\rho C_d A\, \boldsymbol{v}_t \|\boldsymbol{v}_t\|_2. \tag{5.18}$$

To model the body-rate inner-loop, the commanded rate $\boldsymbol{\omega}_t^{\text{cmd}}$ is tracked by a PD controller that generates a body torque

$$\boldsymbol{\tau}_t = \mathbf{K}_p\big(\boldsymbol{\omega}_t^{\text{cmd}} - \boldsymbol{\omega}_{b,t}\big) - \mathbf{K}_d\, \boldsymbol{\omega}_{b,t}, \qquad \boldsymbol{\tau}_t \leftarrow \text{clip}(\boldsymbol{\tau}_t, -\boldsymbol{\tau}_{\max}, \boldsymbol{\tau}_{\max}), \tag{5.19}$$

where $\mathbf{K}_p$ and $\mathbf{K}_d$ are diagonal gain matrices and $\boldsymbol{\tau}_{\max}$ denotes per-axis torque limits.

We discretize the model with a semi-implicit Euler integrator: linear velocity and body rates are updated first using $\dot{\boldsymbol{v}}_t$ and $\dot{\boldsymbol{\omega}}_{b,t}$, then position is updated using the new velocity, and the quaternion is integrated and renormalized to unit length. The simulator is implemented in PyTorch and supports batched execution across parallel environments, enabling differentiable optimization through the dynamics.

DIFFERENTIABLE OPTIMIZATION

The discretized quadrotor dynamics in (5.16) define a differentiable transition

$$\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{u}_t), \qquad \boldsymbol{u}_t = \pi_\theta(\boldsymbol{s}_t), \tag{5.20}$$

which enables gradient propagation through the unrolled simulator. For a horizon objective $\mathscr{L} = \sum_{t=0}^{H-1} \ell(\boldsymbol{s}_{t+1}, \boldsymbol{u}_t)$, the analytic policy gradient can be obtained by back-propagation through time:

$$\frac{\partial \mathscr{L}}{\partial \theta} = \sum_{t=0}^{H-1} \left(\frac{\partial \ell_t}{\partial \boldsymbol{s}_{t+1}} \frac{\partial \boldsymbol{s}_{t+1}}{\partial \theta} + \frac{\partial \ell_t}{\partial \boldsymbol{u}_t} \frac{\partial \boldsymbol{u}_t}{\partial \theta} \right), \qquad \ell_t \doteq \ell(\boldsymbol{s}_{t+1}, \boldsymbol{u}_t). \tag{5.21}$$

The state sensitivity satisfies the recursion

$$\frac{\partial \boldsymbol{s}_{t+1}}{\partial \theta} = \mathbf{A}_t \frac{\partial \boldsymbol{s}_t}{\partial \theta} + \mathbf{B}_t \frac{\partial \boldsymbol{u}_t}{\partial \theta}, \qquad \mathbf{A}_t \doteq \frac{\partial \boldsymbol{s}_{t+1}}{\partial \boldsymbol{s}_t}, \ \mathbf{B}_t \doteq \frac{\partial \boldsymbol{s}_{t+1}}{\partial \boldsymbol{u}_t}. \tag{5.22}$$

To improve stability for long-horizon rollouts, we adopt a one-step truncated sensitivity by stopping gradient propagation through $\boldsymbol{s}_t$ beyond one transition. This yields the approximation

$$\frac{\partial \boldsymbol{s}_{t+1}}{\partial \theta} \approx \mathbf{A}_t \mathbf{B}_{t-1} \frac{\partial \boldsymbol{u}_{t-1}}{\partial \theta} + \mathbf{B}_t \frac{\partial \boldsymbol{u}_t}{\partial \theta}, \tag{5.23}$$

This truncation preserves the local dependence of $\boldsymbol{s}_{t+1}$ on both $\boldsymbol{s}_t$ and $\boldsymbol{u}_t$ via $\mathbf{A}_t$ and $\mathbf{B}_t$, while avoiding long products of Jacobians across time, thereby mitigating gradient explosion and enabling stable gradient-based optimization of the low-level control policy in differentiable simulation.

The loss function for low-level controller training encourages waypoint tracking and penalizes aggressive maneuvers. The per-step loss is defined as

$$\begin{aligned} \ell(\boldsymbol{s}_{t+1}, \boldsymbol{u}_t) = {} & \lambda_\psi^{\mathrm{ll}} \left(\psi_{t+1} - \psi_{t+1}^{\mathrm{wp}}\right)^2 + \lambda_h^{\mathrm{ll}} \left(h_{t+1} - h_{t+1}^{\mathrm{wp}}\right)^2 + \lambda_v^{\mathrm{ll}} \left(v_{x,t+1}^b - v_{t+1}^{\mathrm{ref}}\right)^2 \\ & + \lambda_{\mathrm{dir}}^{\mathrm{ll}} \left(1 - \hat{\boldsymbol{v}}_{xy,t+1}^{w\,\top} \boldsymbol{e}(\psi_{t+1}^{\mathrm{wp}})\right) + \lambda_{\mathrm{flat}}^{\mathrm{ll}} \left\| \left(\boldsymbol{g}_{t+1}^b\right)_{xy} \right\|_2^2 \\ & + \lambda_u^{\mathrm{ll}} \|\boldsymbol{u}_t\|_2^2 + \lambda_{\Delta u}^{\mathrm{ll}} \|\boldsymbol{u}_t - \boldsymbol{u}_{t-1}\|_2^2, \end{aligned} \tag{5.24}$$

where ψ_{t+1} is the yaw angle, h_{t+1} is the altitude, and $v_{x,t+1}^b$ is the forward velocity in the body frame at time $t+1$, while ψ_{t+1}^{wp}, h_{t+1}^{wp}, and v_{t+1}^{ref} are the waypoint-provided

heading, height, and speed references. The heading-alignment term uses the normalized horizontal velocity direction in the world frame,

$$\boldsymbol{e}(\psi_{t+1}^{\mathrm{wp}}) = \begin{bmatrix} \cos(\psi_{t+1}^{\mathrm{wp}}) \\ \sin(\psi_{t+1}^{\mathrm{wp}}) \end{bmatrix}, \qquad \hat{\boldsymbol{v}}_{xy,t+1}^{w} = \frac{\left(\mathbf{R}(\boldsymbol{q}_{t+1})\, \boldsymbol{v}_{t+1}^{b}\right)_{xy}}{\left\| \left(\mathbf{R}(\boldsymbol{q}_{t+1})\, \boldsymbol{v}_{t+1}^{b}\right)_{xy} \right\|_2}, \tag{5.25}$$

To encourage near-level flight, we penalize the horizontal components of gravity expressed in the body frame,

$$\left(\boldsymbol{g}_{t+1}^{b}\right)_{xy} = \left(\mathbf{R}(\boldsymbol{q}_{t+1})^{\top} \boldsymbol{g}^{w}\right)_{xy}. \tag{5.26}$$

All coefficients λ^{ll} are set to negative values, such that larger tracking errors and larger control effort yield higher penalties in $\ell(\cdot)$. Finally, $\|\boldsymbol{u}_t\|_2^2$ penalizes control effort and $\|\boldsymbol{u}_t - \boldsymbol{u}_{t-1}\|_2^2$ penalizes rapid changes in the control command.

5.4. EXPERIMENTS

To evaluate the effectiveness of our hierarchical perception–action architecture with differentiable dynamics, and to quantify the benefits of gimbal-stabilized perception, we conduct extensive experiments in simulation. We compare our method against end-to-end baselines and perform ablation studies to isolate the contribution of each component.

Using IsaacLab [18], we construct a suite of obstacle-avoidance tasks with environments of varying complexity. Obstacles include panels and procedurally generated fake trees with randomized trunks and branches (Figure 5.3), whose irregular geometries are difficult to approximate with simple primitives. Obstacle locations are randomized according to a Poisson process to generate diverse layouts. In each episode, the drone starts from a random initial position and must reach a specified goal while avoiding collisions. All experiments are run on a workstation with an Intel Core i7-13700K CPU and an NVIDIA RTX 4090 GPU. To accelerate training, we collect experience using 128 parallel simulation environments.

5.4.1. HIERARCHICAL PERCEPTION–ACTION ARCHITECTURE

We compare our hierarchical perception–action architecture against an end-to-end baseline trained with PPO that directly maps depth latents and drone states to CTBR commands. The baseline uses the same depth-latent representation as our method to ensure a fair comparison. Both methods are trained under identical conditions, including the same number of training steps and the same environment settings. Because our hierarchical architecture explicitly decomposes the task into high-level waypoint generation and low-level control, it is not straightforward to design reward functions that are strictly identical across the two approaches. Instead, we design the reward for the end-to-end baseline to encourage behavior similar to the hierarchical policy, while still enabling stable learning of goal reaching and collision avoidance.

For evaluation, we generate 10 test maps with higher obstacle density than the training environment to reduce the likelihood of overfitting. For each map, we sample 10 start–goal pairs, resulting in 100 trials in total to compute the success rate. One example test map and its 10 trajectories are shown in Figure 5.3. Figure 5.4 reports results from checkpoints saved at different training steps. The blue (with gimbal) and purple (without gimbal) curves correspond to our hierarchical architecture, while the orange (with gimbal) and green (without gimbal) curves correspond to the end-to-end baseline. The hierarchical approach converges substantially faster and achieves higher final performance in both success rate and AGV. Specifically, our method reaches a success rate of 78% (with gimbal) and 72% (without gimbal), compared to approximately 40% (with gimbal) and 28% (without gimbal) for the end-to-end baseline. Likewise, it achieves an AGV of about 4.0 m/s (with gimbal) and 3.6 m/s (without gimbal), versus roughly 2.6 m/s (with gimbal) and 1.6 m/s (without gimbal) for the baseline. This gap highlights the benefit of decomposing navigation into waypoint planning and low-level control: the decomposition simplifies the learning problem and allows each module to specialize. Moreover, the hierarchical method requires only about one quarter of the training time to reach a comparable success rate to the end-to-end baseline, indicating improved sample efficiency. Furthermore, the hierarchical method requires only one-third of the training time to achieve performance comparable to the end-to-end baseline in terms of both success rate and AGV.

5.4.2. GIMBAL-STABILIZED PERCEPTION

We further investigate the impact of gimbal-stabilized perception on navigation performance. As shown in Figure 5.4, for both the hierarchical and end-to-end methods, the gimbal-stabilized camera (blue and orange curves) consistently outperforms the fixed-camera configuration (purple and green curves). Although the gap is smaller than that between hierarchical and end-to-end architectures, both success rate and AGV improve with stabilization. This improvement is attributable to the more stable and consistent visual input provided by the gimbal.

To better understand how stabilization affects the learned behavior, we analyze the distributions of the drone's roll and pitch angles during testing. Figure 5.5a shows the probability density of roll and pitch angles for both configurations. Angle samples are collected during steady flight, excluding time intervals near the start and goal to avoid the influence of takeoff and landing maneuvers. The horizontal axis denotes angle (degrees), and the vertical axis denotes probability density. For roll, the gimbal-stabilized configuration (blue) exhibits a broader distribution than the fixed-camera configuration (orange), while both remain centered near 0°. This indicates that, with stabilization, the drone can execute more aggressive roll maneuvers without degrading the effective camera viewpoint. In contrast, the fixed-camera configuration tends to constrain roll angles to avoid excessive image tilt that can compromise perception. For pitch, the difference is less pronounced: both configurations show similarly narrow, near-zero-centered distributions. This suggests that pitch is less affected by camera stabilization, likely because forward flight intrinsically requires pitch adjustments re-

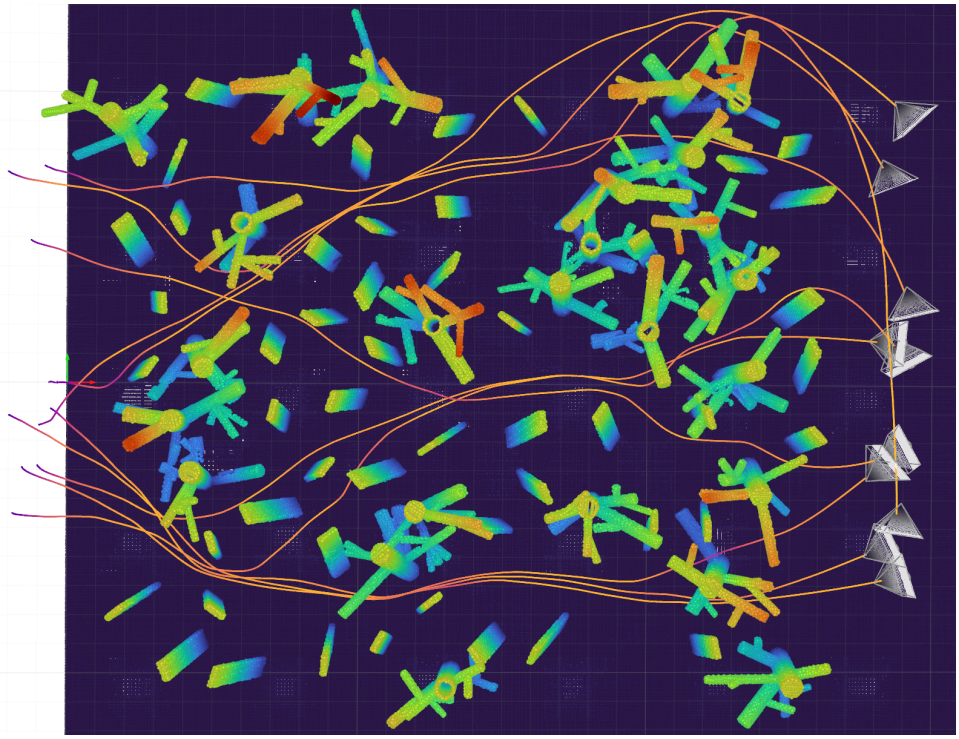


Figure 5.3: Illustration of one test map with 10 trajectories from different start–goal pairs.

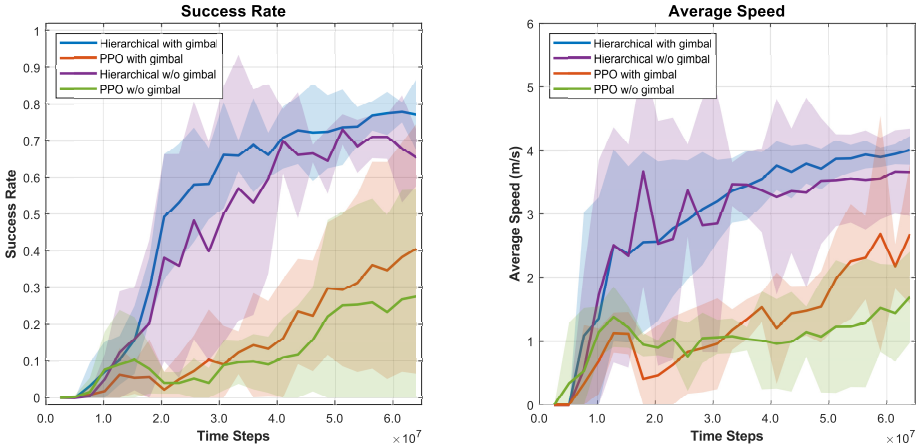


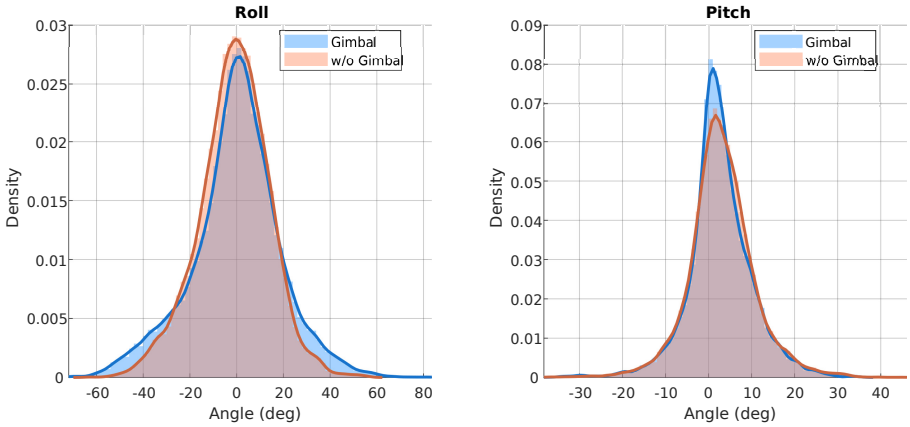
Figure 5.4: Training curves on the test suite (success rate and AGV) for the hierarchical method and the end-to-end baseline, with and without gimbal-stabilized perception.

gardless of obstacle avoidance. Overall, these results indicate that gimbal-stabilized perception enables more dynamic maneuvering, especially in roll which contributes to the observed performance gains.

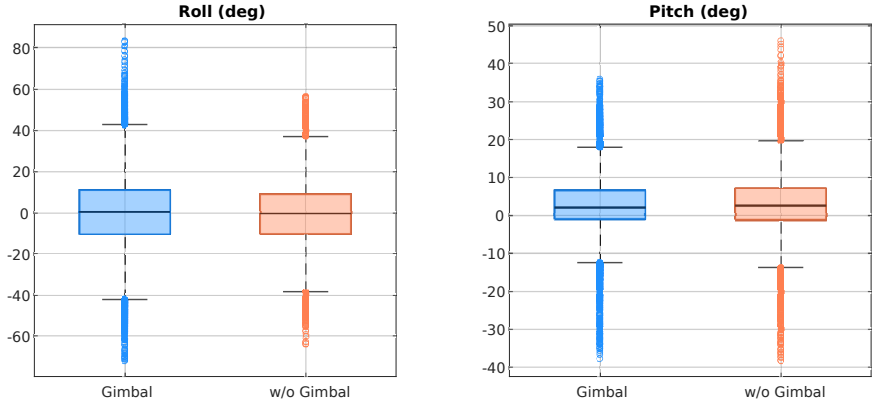
The boxplots in Figure 5.5b further summarize these statistical differences. The gimbal-stabilized setup shows a larger interquartile range for roll, indicating higher variability and more aggressive maneuvers, while maintaining a median close to 0°. The longer whiskers also suggest occasional larger roll angles that remain feasible due to the stabilized view. For pitch, both configurations exhibit smaller interquartile ranges and similar medians, reinforcing that pitch behavior is comparatively insensitive to stabilization. These statistics are consistent with the probability density results and support the conclusion that gimbal stabilization primarily improves roll maneuverability, which likely contributes to higher success rates and faster flight (higher AGV).

5.5. CONCLUSION

This chapter addressed Research Question 4 by showing that bio-inspired hierarchy and stabilization can improve both training efficiency and modularity for vision-based drone navigation. We proposed a hierarchical perception–action architecture that uses gimbal-stabilized depth input, trains a high-level waypoint policy with PPO, and optimizes a low-level CTBR controller via differentiable quadrotor dynamics. Experiments in IsaacLab demonstrate that this decomposition substantially accelerates convergence and improves final success rate and AGV compared with an end-to-end PPO baseline, indicating that waypoint-level commands simplify exploration and credit assignment while allowing the low-level controller to specialize in dynamically consistent tracking. Ablations further show that gimbal stabilization consistently improves performance and enables more aggressive yet controllable roll maneuvers by providing more stable visual input. Overall, combining hierarchical decision making with differentiable low-level optimization yields a more sample-efficient, interpretable, and reusable navigation framework than end-to-end visuomotor learning.



(a) Probability distributions of roll (left) and pitch (right) angles



(b) Boxplots of roll (left) and pitch (right) angles

Figure 5.5: Analysis of roll and pitch angle distributions with and without gimbal-stabilized perception.

6

CONCLUSIONS AND FUTURE WORK

Chapters 2–5 established a coherent pipeline for learning-based drone navigation in clutter: a standardized benchmark for evaluating safety and agility, a varying-speed RL policy that adapts to environment complexity, a depth-transfer strategy that enables reliable sim-to-real deployment, and a hierarchical architecture that improves training efficiency via differentiable optimization. With these components in place, Chapter 6 returns to the research objective defined in Chapter 1 and consolidates the main findings of the dissertation by explicitly answering the four research questions, distilling the key empirical lessons from both simulation and real-world experiments, and clarifying the remaining limitations. It then outlines several promising directions for future work—most notably robust GPS-denied localization in challenging outdoor flight, reduced-sensor navigation using minimal cues (e.g., IMU and optical flow), and extending the hierarchical perception–action design to broader tasks such as exploration, tracking, and, ultimately, manipulation-enabled autonomy.

6.1. ANSWERS TO THE RESEARCH QUESTIONS

Research Question 1

How can we systematically evaluate vision-based obstacle avoidance algorithms for autonomous drones?

In Chapter 2, we proposed a benchmarking suite named **AvoidBench** for vision-based obstacle avoidance algorithms for autonomous drones. The suite comprises a set of simulation environments with varying levels of complexity, a collection of evaluation metrics, and a standardized testing protocol. Environment metrics such as *Ttraversability* are used to quantify environmental complexity. Safety is assessed using metrics such as *success rate* and *relative end distance*, while agility is evaluated using metrics such as *average goal velocity* and *path excess factor*, which characterize the motion of the drone under the guidance of the obstacle avoidance algorithm. Several state-of-the-art obstacle avoidance algorithms were evaluated using the proposed benchmarking suite, and the results provided insight into their respective strengths and weaknesses.

Research Question 2

How can we train an obstacle avoidance policy that adaptively adjusts its flight speed to the complexity of the environment?

In Chapter 3, we proposed a novel reinforcement learning based obstacle avoidance algorithm **MAVRL**. During training, we apply domain randomization to the environment complexity, which enables the learned policy to adapt its flight speed to the perceived difficulty of the surroundings, slowing down in cluttered regions and speeding up in simpler ones. A memory-augmented latent representation that encodes a history of observations allows the policy to better capture the structure of the environment and make more informed decisions. Ablation studies show that using an RNN to reconstruct both past and current depth frames yields a latent space representation that leads to substantially better policies than variants that reconstruct only the current depth or predict only future depth.

Benchmarking results, represented by Pareto frontiers, show that the proposed algorithm outperforms baseline methods in terms of the trade-off between safety and agility across different levels of environmental complexity. Indoor real-world experiments further validate the effectiveness and robustness of our approach in practical deployment scenarios.

Research Question 3

How can we effectively bridge the sim-to-real gap for vision-based obstacle avoidance on autonomous drones?

In Chapter 4, we proposed **depth transfer**, a novel sim-to-real transfer learning

approach for vision-based obstacle avoidance on autonomous drones. To address dynamics and control gaps, we employ a reference generator that produces high-level reference commands which can be reliably tracked by both simulated and real drones, thereby reducing the impact of dynamics mismatch. To tackle visual gaps, we use domain adaptation techniques to align the feature distributions between simulated and real-world depth images. This feature-level alignment makes the latent space induced by real-world depth maps more closely resemble that of simulated inputs, and thus improves the performance of policies trained in simulation when they are deployed on real platforms.

Extensive ablation studies show that **depth transfer** enables policies trained in Isaac Gym to achieve a significant performance boost when transferred to other simulators such as AvoidBench. Moreover, dense forest flight tests on real drones further validate the effectiveness of our approach in challenging real-world deployment scenarios.

Research Question 4

How can animal-inspired hierarchical perception-action architecture be used to train drone obstacle avoidance more efficiently?

In Chapter 5, we proposed a novel **hierarchical perception–action architecture** and a gimbal-based stabilized visual input system for vision-based obstacle avoidance on autonomous drones, inspired by how animals process visual information. The hierarchical architecture consists of a high-level waypoint generator, which uses target information and depth maps to propose short-horizon waypoints, and a low-level waypoint tracking policy, which follows these waypoints and outputs CTBR commands. To further improve training efficiency, we train the low-level policy using differentiable optimization rather than reinforcement learning.

Ablation studies show that the gimbal-based stabilized visual input system significantly improves obstacle avoidance performance by providing more stable and consistent visual inputs, especially during aggressive maneuvers. In addition, the hierarchical perception–action architecture greatly enhances training efficiency, achieving performance comparable to end-to-end reinforcement learning methods while requiring substantially less training time.

6.2. CONCLUSIONS

Research Objective

To develop efficient learning-based algorithms for safe and agile onboard navigation in cluttered environments, and to optimize the trade-off between safety and agility.

Now let us return to the research objective formulated in Chapter 1 and ask to

what extent this dissertation has achieved that goal. I would like to begin with the two words **safety** and **agility**, which appear throughout this thesis. Our solution is built on a simple and intuitive idea: a drone should fly faster in simple environments and slower in complex ones. This principle balances safety with efficient use of the drone's power and dynamic capabilities. The **MAVRL** approach presented in Chapter 3 demonstrates that reinforcement learning can effectively learn a policy that adapts its speed to environmental complexity. Notably, our results outperform state-of-the-art model-based (Ego-Planner [15]) and learning-based (Agile-Autonomy [30]) algorithms on the Pareto frontiers of *success rate* versus *average goal velocity*. To support a fair and systematic comparison, Chapter 2 introduces **AvoidBench**, a benchmarking suite for vision-based obstacle avoidance algorithms. The proposed metrics quantify environmental complexity, safety, agility, and computational efficiency, providing a unified framework for evaluating different methods.

To enable onboard navigation, we applied the algorithms developed in simulation to real-world settings. The MAVRL algorithm can be directly deployed for drone obstacle avoidance in laboratory environments for several reasons. First, training on depth reduces part of the sim-to-real gap because depth primarily reflects scene geometry and is far less sensitive than RGB to texture, illumination, and appearance changes, which dominate the distribution shift. Second, laboratory scenes are relatively simple, with obstacles that are mostly regular geometric shapes, so a policy trained in simulation can be transferred by retraining only the VAE. In contrast, outdoor environments such as dense forests exhibit far more complex obstacle geometries. Notably, such forests include thin and irregularly shaped small branches that are not well encoded by standard VAEs. Moreover, rendering stereo images and then running stereo matching algorithm introduces considerable computational and latency costs. For this reason, it is preferable to train with ground-truth depth in simulation, although this choice would normally enlarge the sim-to-real gap.

The **depth transfer** approach proposed in Chapter 4 effectively addresses this problem. By performing feature-level domain adaptation, it allows policies trained in Isaac Gym with ground-truth depth to be deployed directly in AvoidBench and in real forest environments. Compared with fine-tuning the entire policy, fine-tuning only the encoder module is both more efficient and safer. Benchmark results further show that depth transfer improves the success rate of MAVRL during high-speed flight.

Using ground-truth depth rendered directly from simulation does improve the efficiency of training. This naturally leads to the question of whether the training efficiency of obstacle avoidance policies can be improved even further. The **hierarchical perception–action architecture** proposed in Chapter 5 provides an affirmative answer. By decomposing the obstacle avoidance task into high-level waypoint generation and low-level waypoint tracking, and by employing differentiable optimization to train the low-level policy, we significantly reduce training time while maintaining performance that is comparable to end-to-end reinforcement learning methods.

Overall, this dissertation has made substantial progress toward the research objective of developing efficient learning-based algorithms for safe and agile onboard

navigation in cluttered environments. The proposed methods not only improve the balance between safety and agility but also enhance training efficiency and facilitate sim-to-real transfer. Together, these contributions provide a solid foundation for future research on autonomous drone navigation in complex real-world settings.

6.3. FUTURE WORK

Our extensive outdoor experiments indicate that robust autonomous flight in GPS-free forests remains challenging, even after the improvements introduced in this thesis. In practice, the limiting factor is not only policy learning, but also the reliability of the state estimate used to close the control loop. Although our commercial visual inertial odometry system (RealSense T265) is reasonably robust, it still suffers from substantial drift and occasional tracking failures during high-speed flight. These failures are partly stochastic because they depend strongly on illumination and scene texture, leading to run-to-run variability that is difficult to anticipate. As a consequence, obstacle-avoidance performance in real forests remains unsatisfactory: despite domain randomization, reliable success still requires the odometry to remain reasonably accurate, since state-estimation drift directly corrupts the control loop and can overwhelm the learned avoidance behavior.

This observation raises a broader question about biological flight. When human pilots fly through forests, they do not rely on an explicit, globally consistent odometry estimate in the way our current autonomy stack implicitly does. This suggests two complementary directions for improving real-world performance. First, more robust state estimation will certainly help, including VIO designs that better handle rapid motion, motion blur, and lighting changes. However, improved VIO alone is unlikely to fully resolve the issue, because any vision-based estimator can still degrade in challenging conditions. Second, the navigation policy itself should be made less dependent on accurate global odometry. A promising direction is to investigate whether policies that receive only IMU signals and optical flow can still achieve reliable obstacle avoidance and progress through forests. While such minimal sensing makes learning harder due to reduced observability, combining it with the hierarchical perception–action architecture proposed in Chapter 5 may make the problem tractable by delegating short-horizon reactive control to a robust low-level module and reserving longer-horizon decisions for a higher-level policy.

If the odometry camera can be eliminated, replacing the stereo depth camera with a monocular camera would be highly desirable, as it would significantly improve the drone's overall payload and energy efficiency and thus its endurance. Recent foundation models for monocular depth estimation [114, 115] can already run in real time on NVIDIA Jetson Orin NX, and the improved onboard CPU and GPU capabilities open up new possibilities for more capable perception and navigation on resource-constrained drones.

The hierarchical perception–action architecture proposed in Chapter 5 could be further exploited beyond its current role in improving training efficiency. A natural next

step is to apply this architecture to other navigation tasks such as exploration and tracking, where the low-level policy would be reused without retraining and only the high-level policy would be adapted. For example, the high-level module could be trained to propose the next unexplored location in exploration, or to generate waypoints that follow a moving target in tracking. Since the underlying control required by these tasks is closely related, sharing a common low-level controller is both natural and potentially very efficient. Moreover, the same architecture opens up interesting directions for online learning. In particular, online fine-tuning of the high-level waypoint generator, while keeping the low-level controller fixed, should be considerably simpler than adapting a fully end-to-end network, and is therefore a promising avenue for future research.

Looking further ahead, I believe autonomous navigation for drones is likely to evolve along two main directions. The first is toward lighter and safer autonomous platforms, making drones easier to integrate into everyday life and more capable of serving human needs. Considerable work is already being done in this area, spanning algorithms that couple event-based sensing with low-latency neuromorphic control for agile flight [116, 117], hardware that provides energy-efficient neuromorphic processors that allow spike-based computation and learning on board [118], and airframe design that targets insect-scale, lightweight flapping-wing platforms with favorable lift generation and maneuverability under strict payload constraints [119, 120]. The second direction concerns improving the **operability** of drones. At present, most drone algorithms primarily target navigation, which means that the range of feasible tasks is largely constrained by onboard sensing, such as aerial photography [121], exploration and mapping [122], or gas-leak detection [123]. Endowing drones with manipulation capabilities analogous to grasping [124, 125] and combining this with recent advances in reinforcement learning and sim-to-real transfer could substantially expand the set of tasks they can perform and make autonomous drones far more useful in real-world applications.

REFERENCES

[1] Flyability. *Elios 2 Tested at the Chernobyl Nuclear Power Plant*. https://www.flyability.com/news/chernobyl-mission. Accessed: 2025-12-11. Nov. 2020.

[2] Skydio. *Skydio 2 In Action: First Responder Deployments Show the Power of Autonomous Drones*. https://www.skydio.com/blog/skydio-2-first-responder-using-drone-autonomy. Accessed: 2025-12-11. June 2020.

[3] T. Shan, B. Englot, C. Ratti, and D. Rus. "LVI-SAM: Tightly-coupled Lidar-Visual-Inertial Odometry via Smoothing and Mapping". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 5692–5698. DOI: 10.1109/ICRA48506.2021.9561996.

[4] S. Wang, S. Li, Y. Zhang, S. Yu, S. Yuan, R. She, Q. Guo, J. Zheng, O. K. Howe, L. Chandra, S. Srijeyan, A. Sivadas, T. Aggarwal, H. Liu, H. Zhang, C. Chen, J. Jiang, L. Xie, and W. P. Tay. "UAVScenes: A Multi-Modal Dataset for UAVs". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2025, pp. 28946–28958.

[5] T. Qin, P. Li, and S. Shen. "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator". In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020. DOI: 10.1109/TRO.2018.2853729.

[6] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang. "FAST-LIO2: Fast Direct LiDAR-Inertial Odometry". In: *IEEE Transactions on Robotics* 38.4 (2022), pp. 2053–2073. DOI: 10.1109/TRO.2022.3141876.

[7] G. Balamurugan, J. Valarmathi, and V. P. S. Naidu. "Survey on UAV navigation in GPS denied environments". In: *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. 2016, pp. 198–204. DOI: 10.1109/SCOPES.2016.7955787.

[8] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen. "Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3529–3536. DOI: 10.1109/LRA.2019.2927938.

[9] W. Zhang, J. Li, W. Yu, P. Ding, J. Wang, and X. Zhang. "Algorithm for UAV path planning in high obstacle density environments: RFA-star". In: *Frontiers in Plant Science* 15 (2024), p. 1391628.

[10] A. Merei, H. Mcheick, A. Ghaddar, and D. Rebaine. "A Survey on Obstacle Detection and Avoidance Methods for UAVs". In: *Drones* 9.3 (2025). ISSN: 2504-446X. DOI: 10.3390/drones9030203. URL: https://www.mdpi.com/2504-446X/9/3/203.

[11] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter. “Graph-based subterranean exploration path planning using aerial and legged robots”. In: *Journal of Field Robotics* 37.8 (2020), pp. 1363–1388.

[12] M. Kulkarni, M. Dharmadhikari, M. Tranzatto, S. Zimmermann, V. Reijgwart, P. De Petris, H. Nguyen, N. Khedekar, C. Papachristos, L. Ott, *et al.* “Autonomous teamed exploration of subterranean environments using legged and aerial robots”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 3306–3313.

[13] P. Hart, N. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/tssc.1968.300136. URL: https://doi.org/10.1109/tssc.1968.300136.

[14] M. Dharmadhikari, T. Dang, L. Solanka, J. Loje, H. Nguyen, N. Khedekar, and K. Alexis. “Motion primitives-based agile exploration path planning for aerial robotics”. In: *IEEE International Conference on Robotics and Automation (ICRA), Paris, France*. 2020.

[15] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao. “Ego-planner: An esdf-free gradient-based local planner for quadrotors”. In: *IEEE Robotics and Automation Letters* 6.2 (2020), pp. 478–485.

[16] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, *et al.* “Isaac gym: High performance gpu-based physics simulation for robot learning”. In: *arXiv preprint arXiv:2108.10470* (2021).

[17] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. “Isaac Gym: High Performance GPU Based Physics Simulation For Robot Learning”. In: *NeurIPS Datasets and Benchmarks*. 2021. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/28dd2c7955ce926456240b2ff0100bde-Abstract-round2.html.

[18] M. Mittal *et al.* “Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning”. In: *arXiv preprint arXiv:2511.04831* (2025). URL: https://arxiv.org/abs/2511.04831.

[19] M. Kulkarni, T. J. L. Forgaard, and K. Alexis. *Aerial Gym – Isaac Gym Simulator for Aerial Robots*. 2023. arXiv: 2305.16510 [cs.RO]. URL: https://arxiv.org/abs/2305.16510.

[20] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza. “Flightmare: A flexible quadrotor simulator”. In: *Conference on Robot Learning*. PMLR. 2021, pp. 1147–1157.

[21] H. Nguyen, S. H. Fyhn, P. De Petris, and K. Alexis. “Motion Primitives-based Navigation Planning using Deep Collision Prediction”. In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 9660–9667. DOI: 10.1109/ICRA46639.2022.9812231.

[22] M. Kulkarni, H. Nguyen, and K. Alexis. “Semantically-enhanced deep collision prediction for autonomous navigation using aerial robots”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2023, pp. 3056–3063.

[23] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. v. Stryk. “Comprehensive simulation of quadrotor uavs using ros and gazebo”. In: *International conference on simulation, modeling, and programming for autonomous robots*. Springer. 2012, pp. 400–411.

[24] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. “Rotors—a modular gazebo mav simulator framework”. In: *Robot operating system (ROS)*. Springer, 2016, pp. 595–625.

[25] S. Shah, D. Dey, C. Lovett, and A. Kapoor. “Airsim: High-fidelity visual and physical simulation for autonomous vehicles”. In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[26] B. Boroujerdian, H. Genc, S. Krishnan, W. Cui, A. Faust, and V. Reddi. “Mavbench: Micro aerial vehicle benchmarking”. In: *2018 51st annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE. 2018, pp. 894–907.

[27] T. T. Du Montcel, A. Nègre, J.-E. Gomez-Balderas, and N. Marchand. “BOARR: A benchmark for quadrotor obstacle avoidance based on ROS and RotorS”. In: (2019).

[28] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. “OctoMap: An efficient probabilistic 3D mapping framework based on octrees”. In: *Autonomous robots* 34.3 (2013), pp. 189–206.

[29] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto. “Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 1366–1373.

[30] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza. “Learning High-Speed Flight in the Wild”. In: *Science Robotics*. Oct. 2021.

[31] A. Bhattacharya, N. Rao, D. Parikh, P. Kunapuli, Y. Wu, Y. Tao, N. Matni, and V. Kumar. “Vision transformers for end-to-end vision-based quadrotor obstacle avoidance”. In: *arXiv preprint arXiv:2405.10391* (2024).

[32] J. Xing, A. Romero, L. Bauersfeld, and D. Scaramuzza. “Bootstrapping Reinforcement Learning with Imitation for Vision-Based Agile Flight”. In: *CoRR* abs/2403.12203 (2024). URL: https://doi.org/10.48550/arXiv.2403.12203.

[33] J. Xing, A. Romero, L. Bauersfeld, and D. Scaramuzza. “Bootstrapping Reinforcement Learning with Imitation for Vision-Based Agile Flight”. In: *Proceedings of The 8th Conference on Robot Learning*. Vol. 270. PMLR. 2025, pp. 2542–2556.

[34] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. “Champion-level drone racing using deep reinforcement learning”. In: *Nature* 620.7976 (2023), pp. 982–987.

[35] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza. “Reaching the limit in autonomous racing: Optimal control versus reinforcement learning”. In: *Science Robotics* 8.82 (2023).

[36] Y. Song, K. Shi, R. Penicka, and D. Scaramuzza. “Learning Perception-Aware Agile Flight in Cluttered Environments”. In: *IEEE International Conference on Robotics and Automation*. 2023, pp. 1989–1995. DOI: 10.1109/ICRA48891.2023.10160563.

[37] G. Zhao, T. Wu, Y. Chen, and F. Gao. “Learning Speed Adaptation for Flight in Clutter”. In: *IEEE Robotics and Automation Letters* (2024).

[38] M. Kulkarni and K. Alexis. “Task-Driven Compression for Collision Encoding Based on Depth Images”. In: *Advances in Visual Computing*. Ed. by G. Bebis, G. Ghiasi, Y. Fang, A. Sharf, Y. Dong, C. Weaver, Z. Leo, J. J. LaViola Jr., and L. Kohli. Cham: Springer Nature Switzerland, 2023, pp. 259–273. ISBN: 978-3-031-47966-3.

[39] D. Hoeller, L. Wellhausen, F. Farshidian, and M. Hutter. “Learning a State Representation and Navigation in Cluttered and Dynamic Environments”. In: *IEEE Robot. Autom. Lett.* (2021), pp. 5081–5088. DOI: 10.1109/LRA.2021.3068639.

[40] B. Ichter and M. Pavone. “Robot Motion Planning in Learned Latent Spaces”. In: *IEEE Robot. Autom. Lett.* 4.3 (2019), pp. 2407–2414. DOI: 10.1109/LRA.2019.2901898.

[41] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. “Learning latent dynamics for planning from pixels”. In: *International conference on machine learning*. 2019, pp. 2555–2565.

[42] M. Laskin, A. Srinivas, and P. Abbeel. “Curl: Contrastive unsupervised representations for reinforcement learning”. In: *International conference on machine learning*. PMLR. 2020, pp. 5639–5650.

[43] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[44] A. Singla, S. Padakandla, and S. Bhatnagar. “Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge”. In: *IEEE transactions on intelligent transportation systems* 22.1 (2019), pp. 107–118.

[45] A. Devo, J. Mao, G. Costante, and G. Loianno. “Autonomous single-image drone exploration with deep reinforcement learning and mixed reality”. In: *IEEE Robot. Autom. Lett.* 7.2 (2022), pp. 5031–5038.

[46] D. Ha and J. Schmidhuber. “World Models”. In: *arXiv preprint arXiv:1803.10122* (2018). DOI: 10.48550/arXiv.1803.10122.

[47] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. “Learning Latent Dynamics for Planning from Pixels”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 2555–2565. URL: https://proceedings.mlr.press/v97/hafner19a.html.

[48] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. “Dream to Control: Learning Behaviors by Latent Imagination”. In: *arXiv preprint arXiv:1912.01603* (2019). DOI: 10.48550/arXiv.1912.01603.

[49] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. “Mastering Atari with Discrete World Models”. In: *International Conference on Learning Representations (ICLR)*. arXiv:2010.02193. 2021. DOI: 10.48550/arXiv.2010.02193.

[50] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. “Mastering Diverse Domains through World Models”. In: *arXiv preprint arXiv:2301.04104* (2024). DOI: 10.48550/arXiv.2301.04104.

[51] G. Kahn, P. Abbeel, and S. Levine. “BADGR: An Autonomous Self-Supervised Learning-Based Navigation System”. In: *IEEE Robot. Autom. Lett.* 6.2 (2021), pp. 1312–1319. DOI: 10.1109/LRA.2021.3057023.

[52] K. Lamers, S. Tijmons, C. De Wagter, and G. de Croon. “Self-supervised monocular distance learning on a lightweight micro air vehicle”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 1779–1784.

[53] D. Gandhi, L. Pinto, and A. Gupta. “Learning to fly by crashing”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3948–3955.

[54] K. Van Hecke, G. C. H. E. de Croon, D. Hennes, T. P. Setterfield, A. Saenz-Otero, and D. Izzo. “Self-supervised learning as an enabling technology for future space exploration robots: ISS experiments on monocular distance learning”. In: *Acta Astronautica* 140 (2017), pp. 1–9. DOI: 10.1016/j.actaastro.2017.07.026.

[55] T. Schoepe, E. Janotte, M. B. Milde, O. J. Bertrand, M. Egelhaaf, and E. Chicca. “Finding the gap: neuromorphic motion-vision in dense environments”. In: *Nature Communications* 15.1 (2024), p. 817.

[56] A. Bhattacharya, M. Cannici, N. Rao, Y. Tao, V. Kumar, N. Matni, and D. Scaramuzza. “Monocular Event-Based Vision for Obstacle Avoidance with a Quadrotor”. In: *Proceedings of The 8th Conference on Robot Learning*. Ed. by P. Agrawal, O. Kroemer, and W. Burgard. Vol. 270. Proceedings of Machine Learning Research. PMLR, June 2025, pp. 4826–4843. URL: https://proceedings.mlr.press/v270/bhattacharya25a.html.

[57] M. Kim, J. Kim, M. Jung, and H. Oh. “Towards monocular vision-based autonomous flight through deep reinforcement learning”. In: *Expert Systems with Applications* 198 (2022), p. 116742. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.116742.

[58] C. Niu, C. Newlands, K.-P. Zauner, and D. Tarapore. “An embarrassingly simple approach for visual navigation of forest environments”. In: *Frontiers in Robotics and AI* 10 (2023).

[59] K. Nakhleh, M. Raza, M. Tang, M. Andrews, R. Boney, I. Hadžić, J. Lee, A. Mohajeri, and K. Palyutina. “Sacplanner: Real-world collision avoidance with a soft actor critic local planner and polar state representations”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 9464–9470.

[60] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox. “Closing the sim-to-real loop: Adapting simulation randomization with real world experience”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019, pp. 8973–8979.

[61] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine. “Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight”. In: *2019 international conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 6008–6014.

[62] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza. “Deep drone racing: From simulation to reality with domain randomization”. In: *IEEE Transactions on Robotics* 36.1 (2019), pp. 1–14.

[63] F. Sadeghi and S. Levine. “CAD2RL: Real Single-Image Flight Without a Single Real Image”. In: *Robotics: Science and Systems XIII* (2017).

[64] F. Sadeghi and S. Levine. “Cad2rl: Real single-image flight without a single real image”. In: *arXiv:1611.04201* (2016).

[65] H. He, P. Wu, C. Bai, H. Lai, L. Wang, L. Pan, X. Hu, and W. Zhang. “Bridging the sim-to-real gap from the information bottleneck perspective”. In: *arXiv preprint arXiv:2305.18464* (2023).

[66] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou. “Transfer learning in deep reinforcement learning: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.11 (2023), pp. 13344–13362.

[67] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell. “Cycada: Cycle-consistent adversarial domain adaptation”. In: *International conference on machine learning*. Pmlr. 2018, pp. 1989–1998.

[68] Y. Ganin and V. Lempitsky. “Unsupervised Domain Adaptation by Backpropagation”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1180–1189.

[69] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy. “Domain generalization: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* 45.4 (2022), pp. 4396–4415.

[70] J. Peng, Y. Huang, W. Sun, N. Chen, Y. Ning, and Q. Du. “Domain adaptation in remote sensing image classification: A survey”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 15 (2022), pp. 9842–9859.

[71] A. Farahani, S. Voghoei, K. Rasheed, and H. R. Arabnia. “A brief review of domain adaptation”. In: *Advances in data science and information engineering: proceedings from ICDATA 2020 and IKE 2020* (2021), pp. 877–894.

[72] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner. “Darla: Improving zero-shot transfer in reinforcement learning”. In: *International conference on machine learning*. PMLR. 2017, pp. 1480–1490.

[73] J. Xing, T. Nagata, K. Chen, X. Zou, E. Neftci, and J. L. Krichmar. “Domain adaptation in reinforcement learning via latent unified state representation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 10452–10459.

[74] D. Li, L. Meng, J. Li, K. Lu, and Y. Yang. “Domain adaptive state representation alignment for reinforcement learning”. In: *Information Sciences* 609 (2022), pp. 1353–1368.

[75] J. Pan, J. Xing, R. Reiter, Y. Zhai, E. Aljalbout, and D. Scaramuzza. “Learning on the Fly: Rapid Policy Adaptation via Differentiable Simulation”. In: *arXiv preprint arXiv:2508.21065* (2025).

[76] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. “Proximal policy optimization algorithms”. In: *arXiv:1707.06347* (2017).

[77] X. Zhang, R. Wang, Y. Ren, J. Sun, H. Fang, J. Chen, and G. Wang. “DiffAero: A GPU-Accelerated Differentiable Simulation Framework for Efficient Quadrotor Policy Learning”. In: *arXiv preprint arXiv:2509.10247* (2025).

[78] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. “CARLA: An open urban driving simulator”. In: *Conference on robot learning*. PMLR. 2017, pp. 1–16.

[79] Q. Chen, J. Sun, N. Gao, J. Low, T. Chen, and M. Schwager. “GRaD-Nav: Efficiently Learning Visual Drone Navigation with Gaussian Radiance Fields and Differentiable Dynamics”. In: *arXiv preprint arXiv:2503.03984* (2025).

[80] Q. Chen, N. Gao, S. Huang, J. Low, T. Chen, J. Sun, and M. Schwager. “GRaD-Nav++: Vision-Language Model Enabled Visual Drone Navigation with Gaussian Radiance Fields and Differentiable Dynamics”. In: *arXiv preprint arXiv:2506.14009* (2025).

[81] H. Yu, G. C. H. E. de Croon, and C. De Wagter. “AvoidBench: A high-fidelity vision-based obstacle avoidance benchmarking suite for multi-rotors”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 9183–9189. DOI: 10.1109/ICRA48891.2023.10161097.

[82] K. Y. Scheper and G. C. H. E. de Croon. “Abstraction as a Mechanism to Cross the Reality Gap in Evolutionary Robotics”. In: *International Conference on Simulation of Adaptive Behavior*. Cham: Springer International Publishing, 2016, pp. 280–292. DOI: 10.1007/978-3-319-43488-9_25.

[83] A. Verraest, S. Bahnam, R. Ferede, G. de Croon, and C. De Wagter. “SkyDreamer: Interpretable End-to-End Vision-Based Drone Racing with Model-Based Reinforcement Learning”. In: *arXiv preprint arXiv:2510.14783* (2025).

[84] G. Malczyk, M. Kulkarni, and K. Alexis. “Semantically-Driven Deep Reinforcement Learning for Inspection Path Planning”. In: *IEEE Robotics and Automation Letters* 10.7 (2025), pp. 7206–7213. DOI: 10.1109/LRA.2025.3575331.

[85] H. Yu, C. Wagter, and G. C. H. E. de Croon. “MAVRL: Learn to Fly in Cluttered Environments With Varying Speed”. In: *IEEE Robotics and Automation Letters* 10.2 (2025), pp. 1441–1448. DOI: 10.1109/LRA.2024.3522778.

[86] H. Yu, C. De Wagter, and G. C. H. E. de Croon. “Depth Transfer: Learning to See Like a Simulator for Real-World Drone Navigation”. In: *IEEE Robotics and Automation Letters* 10.11 (2025), pp. 11848–11855. DOI: 10.1109/LRA.2025.3617729.

[87] G. Xu, T. Wu, Z. Wang, Q. Wang, and F. Gao. “Flying on Point Clouds with Reinforcement Learning”. In: *arXiv preprint arXiv:2503.00496* (2025).

[88] N. Wiedemann, V. Wüest, A. Loquercio, M. Müller, D. Floreano, and D. Scaramuzza. “Training Efficient Controllers via Analytic Policy Gradient”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 1349–1356. DOI: 10.1109/ICRA48891.2023.10160581.

[89] Y. Zhang, Y. Hu, Y. Song, D. Zou, and W. Lin. “Back to Newton’s Laws: Learning Vision-based Agile Flight via Differentiable Physics”. In: *arXiv preprint arXiv:2407.10648* (2024).

[90] Y. Zhang, Y. Hu, Y. Song, D. Zou, and W. Lin. “Learning vision-based agile flight via differentiable physics”. In: *Nature Machine Intelligence* 7.6 (June 2025), pp. 954–966. ISSN: 2522-5839. DOI: 10.1038/s42256-025-01048-0. URL: https://doi.org/10.1038/s42256-025-01048-0.

[91] R. Mlambo, I. H. Woodhouse, F. Gerard, and K. Anderson. “Structure from motion (SfM) photogrammetry with drone data: A low cost method for monitoring greenhouse gas emissions from forests in developing countries”. In: *Forests* 8.3 (2017), p. 68.

[92] A. Geiger, P. Lenz, and R. Urtasun. “Are we ready for autonomous driving”. In: *The KITTI Vision Benchmark Suite, 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3354–3361.

[93] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[94] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. “GLUE: A multi-task benchmark and analysis platform for natural language understanding”. In: *arXiv preprint arXiv:1804.07461* (2018).

[95] R. Veder and G. de Croon. *AvoidBench*. http://resolver.tudelft.nl/uuid:302426ff-a4a4-4f8a-967a-331ea71b1ba1.

[96] R. Bridson. “Fast Poisson disk sampling in arbitrary dimensions.” In: *SIGGRAPH sketches* 10.1 (2007), p. 1.

[97] C. Nous, R. Meertens, C. De Wagter, and G. De Croon. “Performance evaluation in obstacle avoidance”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 3614–3619.

[98] H. Hirschmuller. “Stereo processing by semiglobal matching and mutual information”. In: *IEEE Transactions on pattern analysis and machine intelligence* 30.2 (2007), pp. 328–341.

[99] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, *et al.* “Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight”. In: *Science robotics* 7.67 (2022), eabl6259.

[100] A. Sellami and S. Tabbone. “Deep neural networks-based relevant latent representation learning for hyperspectral image classification”. In: *Pattern Recognition* 121 (2022), p. 108224. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2021.108224.

[101] C. Doersch. “Tutorial on variational autoencoders”. In: *arXiv:1606.05908* (2016).

[102] R. J. Boik. “The Fisher-Pitman permutation test: A non-robust alternative to the normal theory F test when variances are heterogeneous”. In: *British Journal of Mathematical and Statistical Psychology* 40.1 (1987), pp. 26–42.

[103] J. Xing, L. Bauersfeld, Y. Song, C. Xing, and D. Scaramuzza. “Contrastive learning for enhancing robust scene transfer in vision-based agile flight”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 5330–5337.

[104] Y. Zhang, Y. Hu, Y. Song, D. Zou, and W. Lin. “Learning vision-based agile flight via differentiable physics”. In: *Nature Machine Intelligence* (2025), pp. 1–13.

[105] L. Van der Maaten and G. Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[106] C. Thornton. “Separability is a learner’s best friend”. In: *4th Neural Computation and Psychology Workshop, London, 9–11 April 1997: Connectionist Representations*. Springer. 1998, pp. 40–46.

[107] M. Ariyanto, X. Zheng, R. Tanaka, C. M. M. Refat, N. Hirota, K. Yamamoto, and K. Morishima. “Biohybrid Behavior-Based Navigation with Obstacle Avoidance for Cyborg Insect in Complex Environment”. In: *Soft Robotics* 12.4 (2025). PMID: 39930959, pp. 498–516. DOI: 10.1089/soro.2024.0082.

[108] K. Fang, H. Mei, Y. Tang, W. Wang, H. Wang, Z. Wang, and Z. Dai. “Grade-control outdoor turning flight of robo-pigeon with quantitative stimulus parameters”. In: *Frontiers in Neurorobotics* 17 (2023), p. 1143601.

[109] S. K. Talwar, S. Xu, E. S. Hawley, S. A. Weiss, K. A. Moxon, and J. K. Chapin. “Rat navigation guided by remote control”. In: *Nature* 417.6884 (2002), pp. 37–38.

[110] M. F. Land. “Motion and vision: why animals move their eyes”. In: *Journal of Comparative Physiology A* 185.4 (1999), pp. 341–352.

[111] L. M. Theunissen and N. F. Troje. “Head stabilization in the pigeon: role of vision to correct for translational and rotational disturbances”. In: *Frontiers in neuroscience* 11 (2017), p. 551.

[112] X. Liu, Y. Yang, C. Ma, J. Li, and S. Zhang. “Real-time visual tracking of moving targets using a low-cost unmanned aerial vehicle with a 3-axis stabilized gimbal system”. In: *Applied Sciences* 10.15 (2020), p. 5064.

[113] A. Manecy, J. Diperi, M. Boyron, N. Marchand, and S. Viollet. “A novel hyperacute gimbal eye to implement precise hovering and target tracking on a quadrotor”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 3212–3218. DOI: 10.1109/ICRA.2016.7487490.

[114] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao. “Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 10371–10381.

[115] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao. “Depth Anything V2”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang. Vol. 37. Curran Associates, Inc., 2024, pp. 21875–21911. DOI: 10.52202/079017-0688.

[116] F. Paredes-Vallés, J. J. Hagenaars, J. Dupeyroux, S. Stroobants, Y. Xu, and G. C. H. E. de Croon. “Fully neuromorphic vision and control for autonomous drone flight”. In: *Science Robotics* 9.90 (2024), eadi0591. DOI: 10.1126/scirobotics.adi0591.

[117] A. Vitale, A. Renner, C. Nauer, D. Scaramuzza, and Y. Sandamirskaya. “Event-driven Vision and Control for UAVs on a Neuromorphic Chip”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 103–109. DOI: 10.1109/ICRA48506.2021.9560881.

[118] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang. “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning”. In: *IEEE Micro* 38.1 (2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359.

[119] M. Karásek, F. T. Muijres, C. D. Wagter, B. D. W. Remes, and G. C. H. E. de Croon. “A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns”. In: *Science* 361.6407 (2018), pp. 1089–1094. DOI: 10.1126/science.aat0350.

[120] G. de Croon, K. de Clercq, R. Ruijsink, B. Remes, and C. de Wagter. “Design, Aerodynamics, and Vision-Based Control of the DelFly”. In: *International Journal of Micro Air Vehicles* 1.2 (2009), pp. 71–97. DOI: 10.1260/175682909789498288.

[121] J. L. Morgan, S. E. Gergel, and N. C. Coops. “Aerial Photography: A Rapidly Evolving Tool for Ecological Management”. In: *BioScience* 60.1 (Jan. 2010), pp. 47–59. ISSN: 0006-3568. DOI: 10.1525/bio.2010.60.1.9.

[122] B. Zhou, Y. Zhang, X. Chen, and S. Shen. “FUEL: Fast UAV Exploration Using Incremental Frontier Structure and Hierarchical Planning”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 779–786. DOI: 10.1109/LRA.2021.3051563.

[123] B. P. Duisterhof, S. Li, J. Burgués, V. J. Reddi, and G. C. H. E. de Croon. “Sniffy Bug: A Fully Autonomous Swarm of Gas-Seeking Nano Quadcopters in Cluttered Environments”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 9099–9106. DOI: 10.1109/IROS51168.2021.9636217.

[124] W. Deng, H. Chen, B. Ye, H. Chen, Z. Li, and X. Lyu. “Whole-Body Integrated Motion Planning for Aerial Manipulators”. In: *IEEE Transactions on Robotics* 41 (2025), pp. 6661–6679. DOI: 10.1109/TRO.2025.3626619.

[125] F. Ruggiero, V. Lippiello, and A. Ollero. “Aerial Manipulation: A Literature Review”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1957–1964. DOI: 10.1109/LRA.2018.2808541.

CURRICULUM VITÆ

Hang YU

04-02-1996 Born in Anhui, China.

EDUCATION

2014–2018 **Northwestern Polytechnical University, Xi'an, China**
Bachelor in Mechanical Design & Manufacturing and Their Automation

2018–2021 **Northwestern Polytechnical University, Xi'an, China**
Master in Aerospace Applied Science and Engineering

2021–2025 **Delft University of Technology, Delft, the Netherlands**
Ph.D. in Control and Simulation, Aerospace Engineering
Promotor: Prof. dr. GCHE (Guido) de Croon & Dr. C. (Christophe) de Wagter

LIST OF PUBLICATIONS

JOURNAL PAPERS

3. M. Shi, R. de Silva, **H. Yu**, R. Polvara, M. Popović, Perception-Aware Autonomous Exploration in Feature-Limited Environments, arXiv preprint arXiv:2603.15605, 2026.
2. **H. Yu**, C. De Wagter, G. C. H. E. de Croon, Depth Transfer: Learning to See Like a Simulator for Real-World Drone Navigation, IEEE Robotics and Automation Letters, 2025.
1. **H. Yu**, C. De Wagter, G. C. H. E. de Croon, MAVRL: Learn to Fly in Cluttered Environments With Varying Speed, IEEE Robotics and Automation Letters, 2025.

CONFERENCE PAPERS

1. **H. Yu**, G. C. H. E. de Croon, C. De Wagter, AvoidBench: A high-fidelity vision-based obstacle avoidance benchmarking suite for multi-rotors, IEEE International Conference on Robotics and Automation (ICRA), 2025.

ACKNOWLEDGEMENTS

It has been a long journey. It began with my master's supervisor suggesting that I pursue a doctorate, or perhaps even earlier, in the scattered dreams of my childhood. Compared with the intense pace of pursuing a PhD in China, life in the Netherlands gave me more room to breathe, to look back, and to ask what this path really means. Along the way, many people helped me, encouraged me, and, in their own ways, answered that question with me.

Guido de Croon, my promoter, thank you for being both rigorous and kind. Whenever my research ran into a wall, you always seemed to find a door. I still remember walking into our meetings with a familiar mix of trepidation and anticipation, worrying whether I had enough progress to show, and quietly hoping to hear the next unexpected idea from you. Your curiosity and joy in scientific exploration reminded me that research is not only about results, but also about how we choose to stay passionate when things are uncertain. Christophe de Wagter, my co-promoter, thank you for your sharp eye and your insistence on understanding the essence of a problem. You do not let details slide, and you do not accept convenient explanations. Watching you work taught me that persistence in seeking the truth is not a slogan, but a habit built one careful step at a time.

My first year in a foreign country was full of small surprises and quiet struggles. I had to figure out transport, get used to different food, learn the unwritten rules of daily life, and slowly find my place. I am deeply grateful to Yingfu for helping me before I arrived in the Netherlands and for the early conversations that helped me shape my research direction when everything still felt vague and unfamiliar.

The Dutch weather is often cold and overcast, but my life here was warmed by people. Wencan and Shihao, we were incredibly lucky to come to the Netherlands together. When adaptation felt like a long uphill walk, we kept each other moving. And all those meals and games we had with Lifei, Guofeng, Jianxin, and Bin—I really hope we can experience them together again in the future. Yiyuan, I am so glad we ended up in the same department and even as neighbors for so long. The ordinary days we shared, small chats, quick help, and simple company, are the memories I will keep closest. I am quite introverted, and in that first year especially, I often needed time before I could open up. That is why I am so thankful to Shushuai, Mier, Ziqing, Bo, Cheng, Yifei, and Sherry. You made life easier without making it feel heavy, and you offered guidance in ways that felt natural and gentle. Liming, it is still hard to believe how lucky it was that we met. From our graduation trip before coming to the Netherlands, to experiments, shared meals, and long conversations during our PhD

years, what stayed with me most was your endless stream of ideas and your optimism. You had a way of making even difficult days feel manageable.

The first year of research was also a test. My work started with Flightmare, and it was my first time stepping into fields that felt completely foreign to me, such as Unity and C#. I was lucky that Yvo was there. Thank you for your patience, your help, and all the discussions that saved me from getting stuck alone for too long. The PhD path can be lonely, but at MAVLab I rarely felt alone. I spent most of my four years surrounded by my office mates. Yilun, Stein, and Jesse, thank you for the many conversations, serious and casual, technical and philosophical. Talking with you made me realize that what keeps a person going is not only discipline, but also a genuine passion for research and for life. It was truly special to spend these four years together in the 3.19 office. Reinier, our new office mate, although we have not known each other for long, I admire your intelligence and your work ethic, especially the way you dig into the mechanisms behind algorithms like SNN and RL until they finally make sense. Shenqi and Moji, thank you for joining the group. It was a pleasure to exchange ideas with you, and I feel honored that my thoughts and suggestions could be useful as you pursue related research. Chaoxiang, although our research topics are quite different, I am truly grateful for the PCB design and fabrication support you provided for my project. I hope we will keep talking about football and perhaps collaborate again in the future.

I would also like to thank my colleagues at MAVLab, including Stavrom, Robin, Till, Elijah, Sven, Sunyi, Sunyou, Dequan, and many others. Even when it was just a quick coffee break, a lab chat, or a shared dinner, those moments made the long journey feel lighter. I would also like to thank my colleagues and friends at C&S, including Jan, Jiayu, Wenyin, Rowena, and others. Thank you for making our department not only a workplace, but a community.

Mengjie, my girlfriend, you may think I have given you more than you have given me, but the truth is that I would not have made it through these four years without you. The warmth of our shared memories, and the hope we hold for what comes next, carried me through the moments when motivation ran thin. You made this journey not only bearable, but meaningful. Mom and Dad, being your son is one of the greatest blessings in my life. Our weekly video calls are my safest corner, the moments when I can truly relax and feel at home. Thank you for your unconditional support and for always believing in me throughout my studies.

Having come this far, I often return to the question that started it all: what is the significance of pursuing a doctorate? The answer I have learned from the people around me is simple. It is a privilege to explore what you love, and it is even more meaningful if that exploration can, in some small way, make the people around you and the quiet corners of this world a little better.

Hang Yu
Delft, January 2026