Efficient Imitation Learning for Robust, Adaptive, Vision-based Agile Flight Under Uncertainty

by

Andrea Tagliabue

B.S. Automation Engineering, Politecnico di Milano (2015)
M.S. Robotics, Systems and Control, Swiss Federal Institute of Technology Zürich (2018)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN AUTONOMOUS SYSTEMS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2024

Authored by:	Andrea Tagliabue Department of Aeronautics and Astronautics May 17, 2024
Certified by:	Jonathan P. How R. C. Maclaurin Professor of Aeronautics and Astronautics, MIT, Thesis Supervisor
Certified by:	Sertac Karaman Professor, Aeronautics and Astronautics, MIT, Thesis Supervisor
Certified by:	Igor Gilitschenski Assistant Professor, Computer Science, UoT, Thesis Supervisor
Accepted by:	Jonathan P. How R. C. Maclaurin Professor of Aeronautics and Astronautics, MIT Chair, Graduate Program Committee

Efficient Imitation Learning for Robust, Adaptive, Vision-based Agile Flight Under Uncertainty

by

Andrea Tagliabue

Submitted to the Department of Aeronautics and Astronautics
on May 17, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN AUTONOMOUS SYSTEMS

ABSTRACT

Existing robust model predictive control (MPC) and vision-based state estimation algorithms for agile flight, while achieving impressive performance, still demand significant onboard computation, preventing deployment on robots with tight Cost, Size, Weight, and Power (CSWaP) constraints. The existing imitation learning strategies that can train computationally efficient deep neural network policies from those algorithms have limited robustness and/or are impractical (large number of demonstrations, training time), limiting rapid policy learning once new mission specifications or flight data become available. This thesis details efficient imitation learning strategies that make policy learning from MPC more practical while preserving robustness to uncertainties. First, this thesis contributes a method for efficiently learning trajectory tracking policies from robust MPC, enabling learning of a policy that achieves real-world robustness from a single real-world or simulated mission. Second, it presents a strategy for learning from MPCs with time-varying operating points, exploiting nonlinear models, and enabling acrobatic flights. The obtained policy has an onboard inference time of only 15 μs and can perform a flip on a UAV subject to uncertainties. Third, it extends the previous approaches to vision-based policies, enabling onboard sensing-to-action with milliseconds-level latency, reducing the computational cost of vision-based state estimation, while using data from a single real-world mission. Fourth, it presents a method to reduce control errors under uncertainties, demonstrating rapid adaptation to unexpected failures and uncertainties while avoiding the challenging reward tuning/design of existing methods. Finally, this thesis evaluates the proposed contributions in simulation and hardware, including flights on an insect-scale (sub-gram), soft-actuated, flapping-wing UAV. The methods developed in this thesis achieve the world's first deployment of policies learned from MPC on sub-gram soft-actuated aerial robots.

Acknowledgments

First and foremost, I would like to thank my advisor, Professor Jonathan P. How, for the immense dedication, support, and countless hours invested in advising and mentoring me. I am grateful you have shown me how to be a researcher, encouraging me to tackle challenging problems and creating numerous fruitful collaboration opportunities, all while ensuring I am on a productive track. Thank you also for creating an environment that enabled me to truly focus on research, as I never once had to worry about funding, and for providing all the necessary tools.

My gratitude extends to my thesis committee members, Professor Sertac Karaman and Professor Igor Gilitschenski. Your work has been a great inspiration to me. I sincerely appreciate our many exciting discussions and your encouragement, motivation, and support of my ideas. Thank you also for always finding time to meet despite your very busy schedules.

I also thank my thesis readers, Professor Michael Everett and Professor Jesus Tordesillas. Michael, thank you for all your help throughout this journey, from the technical discussions to your mentorship. Before joining ACL, I remember reading your work and thinking, "This is what I want to do!". Jesus, thank you for your friendship and your mentorship. You are a role model not only as a researcher but also as a dear friend.

I am genuinely grateful to Professor YuFeng (Kevin) Chen for offering me the once-in-a-lifetime opportunity to work with his incredible aerial platform. Professor Chen, during our collaboration, the SoftFly has evolved at such a fast rate that I will not be surprised to see it autonomously zipping around campus in the next months. I am also thankful to Yi-Hsuan (Nemo) Hsiao for the hard work, the sharp thinking, and the long nights spent together debugging and tuning controllers. I also thank Suhan Kim for carefully manufacturing actuators and sensors throughout our various collaborations.

This work would not have been possible without the generous support of the AFOSR and the members of the Neural-inspired Sparse Sensing and Control for Agile Flight MURI team. I thank Regan Kubicek, Professor Sarah Bergbreiter, Professor Steven L. Brunton, and Professor J. Nathan Kutz for the fruitful discussions and collaboration; Professor Urban Fasel for the talks and for identifying the rotational drag coefficient of the SoftFly; Professor Bing W. Brunton and Professor Thomas Daniel for reminding us of the imperfect nature of our engineered systems.

A big thanks goes to all the members of the Aerospace Controls Lab for the countless discussions and the pleasant working environment. Special thanks to Aleix, Andrew, Kota,

Lakshay, Lucas, Mason, Nick, and Joana (thanks for the snacks!)... for making every day enjoyable. I am particularly grateful to Dr. Parker Lusk for the many enlightening discussions and the incredible software infrastructure at ACL, and to Dr. Yulun Tian for his invaluable mentorship. Of course, a special thank you must also go to my dear friends Xiaoyi (Jeremy) Cai and Dr. Dong-Ki Kim for their constant support in life and academia and for the many adventures we have shared, including excursions with Jesus. Jeremy, your ability to maintain a positive attitude in any situation has been incredibly inspiring. Dong-Ki, your mentorship and our shared travels, from Kyoto to Philadelphia and Bellagio, have been some of the highlights of my journey. I also want to express my gratitude to Tong Zhao for his dedication, the long nights spent testing, and for owning the best mug in the lab.

A big thank you goes to the staff members in the Aero-Astro Department, particularly Bryt, for always efficiently handling the logistics of travel and research.

I am also immensely grateful to Dr. Juan Nieto, Dr. Ali Agha, Dr. Mina Kamel, Professor Roland Siegwart, Professor Mark Mueller, Professor Luca Carlone, and Dr. Kasra Khosoussi for mentoring me and introducing me to the world of robotics. Thank you also to Professor Navid Azizan for the many insightful discussions.

A heartfelt thanks to my friends: Lorenzo, Giovanni, Arianna, Marko, Paco, Angel, Adam, Masayuki, Thomas and Matt..., for the support and the brief, yet restorative, moments and adventures we have shared during trips back home and conferences.

Special thanks go to my parents, Elena and Alessandro, for their unwavering love and support, and for instilling in me the values of critical thinking and perseverance. I also owe a great deal of gratitude to my sister Chiara for her constant encouragement and her creative contributions—drawing the drones that have featured prominently in many diagrams of my papers.

Lastly, my deepest thanks go to my partner Simone, for her sacrifices and constant presence by my side through all the ups and downs. Simone, your love and support have been my anchor throughout this journey.

This work was funded by the Air Force Office of Scientific Research MURI (FA9550-19-1-0386).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Existing control and state estimation algorithms, such as robust/adaptive Model Predictive Control (MPC) and visual odometry, have enabled impressive performance under uncertainties on complex, agile robots. However, their computational cost often limits opportunities for real-time, onboard deployment on platforms with limited cost, size, weight and power (CSWaP), such as UAVs at insect scale. To address the computational challenges for onboard deployment of those algorithms, *Imitation Learning (IL)* is increasingly employed to generate computationally-efficient deep neural network policies that are trained to imitate task-relevant demonstrations collected from the computationally-expensive algorithms. However, existing IL methods suffer from sample-inefficiency, requiring to collect a large number of demonstrations. Critically, this sample-inefficiency introduces significant challenges: (i) it necessitates a substantial number of queries to the resource-intensive MPC expert, requiring expensive training equipment; (ii) it hinders learning from very high-dimensional MPC experts, where each MPC query is computationally expensive; (iii) it results in a considerable volume of queries to the training environment, limiting data collection in computationally intensive simulations or demanding numerous hours of real-time demonstrations on a physical robot,

which is impractical. Moreover, this approach complicates updating the policy when the MPC expert undergoes changes due to (iv) tuning, (v) model updates, or when (vi) performing new tasks is required, such as tracking different sets of trajectories. Additionally, obtaining robust policies remains challenging, as uncertainties at deployment not accounted during training may induce deviations of the policy's input distribution from its training distribution—an issue known as *covariate shift*. Further, when learning policies that produce actions from images, existing works improve robustness and generalization by relying on visual abstractions as input to the policy. However, those abstractions need to be hand-crafted, and may discard critical information that can instead enhance performance. Lastly, existing adaptive strategies that enable rapid performance recovery under uncertainty often rely on model-free Reinforcement Learning (RL). This results in computationally expensive training procedures that do not exploit prior knowledge of the robot/environment available, and demand careful design of reward functions.

This thesis aims to address the aforementioned limitations, which is critical to enable rapid deployments of robust, adaptive, vision-based flight controllers on computationally constrained platforms in uncertain environments. In addition, solutions developed in this thesis contribute to the first MPC-based agile flight demonstrations on a sub-gram, soft-actuated, insect-scale Unmanned Aerial Vehicle (UAV).

1.2 Problem Statement

This thesis studies the challenging problem of *efficient imitation learning from model-based algorithms for onboard sensing and robust, adaptive control for agile flight under uncertainties on CSWaP-constrained robots*. This problem is divided into the following subproblems:

1. **Efficient Imitation Learning of Robust Trajectory Tracking MPC.**

   MPC [1]–[4] enables impressive performance on complex, agile robots [5]–[9]. However, its computational cost often limits the opportunities for onboard, real-time deploy-

ment [10], or takes away critical computational resources. Recent works have mitigated MPC's computational requirements by relying on computationally efficient deep neural networks (DNNs) that are trained to imitate task-relevant demonstrations generated by MPC. Such demonstrations are generally collected via Guided Policy Search (GPS) [11]–[14] and IL [15]–[17]. However, a common issue in existing IL methods (e.g., Behavior Cloning (BC) [18]–[20], Dataset-Aggregation (DAgger) [21]) is that they require collecting a relatively large number of MPC demonstrations, preventing demonstration collection with a real robot, and requiring a simulation environment that accurately represents the deployment domain. One of the causes for such demonstration inefficiency is the need to take into account and correct for the compounding of errors in the learned policy [21], which may otherwise create shifts (*covariate shifts*) from the training distribution, with catastrophic consequences [18]. Approaches to robustify the learning procedure, such as Domain Randomization (DR) [22], [23] and DART [24], introduce further challenges, for example requiring to apply disturbances/model changes during training. Data and computational-efficiency challenges in IL can be mitigated by data augmentation (DA) strategies, based on augmenting the training data with extra input-output samples *efficiently-generated* from the collected demonstrations [11], [18], [20], [25], [26]. However, existing methods for MPC [11], [25], [26] do not explicitly account for uncertainties, not only in the way the demonstration are generated, but more importantly in the way the samples are generated, resulting in policies with limited robustness to uncertainties. An **efficient (data, training-time) IL procedure** that can generate **robust trajectory tracking policies from MPC** relying on few demonstrations, potentially collected from a real robot, remains therefore an open challenge.

2. **Efficient Imitation Learning of Acrobatic Policies from Nonlinear MPC.** Agile and acrobatic flight demands the ability to learn policies from MPCs that leverage significantly different operating points, therefore requiring **nonlinear models**. An

MPC strategy that accounts for uncertainties and nonlinear models is nonlinear robust tube MPC (RTMPC) [27]. This strategy requires solving two optimization problems, an MPC-based trajectory planning step, and an additional MPC-based tracking step. While nonlinear RTMPC [27] achieves robustness and agile control, the computational requirements in solving the large optimization problems in [27] further complicate the inefficiencies highlighted in *Problem Statement 1.* A computationally and data-efficient IL procedure that can generate robust policies from MPC using nonlinear models for acrobatic flights continues to be an open question.

3. **Efficient Imitation Learning of a Robust Visuomotor Policy.** IL [18], [21], [28] has been extensively employed to generate computationally-efficient *sensorimotor* policies for mobile robots [12], [13], [15], [16], [23], [29]. These policies produce **control commands from raw sensory data**, bypassing the computational cost of control *and* state estimation, with benefits in terms of latency and robustness. However, the same inefficiencies of existing IL methods highlighted in *Problem Statement 1* remain. These inefficiencies are made worse by the effects of *sensing uncertainties* and *sim2real* gaps in the sensorial (e.g., visual) inputs. Leveraging high-fidelity simulators on powerful computers in combination with DR avoids demanding data-collection on the real robot, but it introduces *sim-to-real* gaps that are especially challenging when learning *visuomotor* policies, i.e., that directly use raw pixels as input. For this reason, sensorimotor policies trained in simulation often leverage as input easy-to-transfer visual abstractions, such as feature tracks [15], depth maps [12], [30], intermediate layers of a convolutional NN (CNN) [31], or a learned latent space [32]. However, all these abstractions discard information that may instead benefit task performance. DA approaches that augment demonstrations with extra images and stabilizing actions improve robustness and sample efficiency of IL [20], [33]–[36]. However, they (i) rely on handcrafted heuristics for the selection of extra images and the generation of corresponding actions, (ii) do not explicitly account for the effects of uncertainties

when generating the extra data [20], [33]–[36], and (iii) often leverage ad-hoc image acquisition setups for DA [33], [34]. As a consequence, their real-world deployment has been mainly focused on tasks in 2D (steering a Dubins car [20], [34]). A DA strategy for efficient, robust sensorimotor learning that does not rely on heuristics and that does not rely on visual abstractions remains therefore an open challenge.

4. **Efficient Imitation Learning of a Robust and Adaptive Policy from MPC.** The deployment of agile robots in uncertain environments requires not only robustness but also **rapid onboard adaptation** to mitigate performance degradation due to uncertainties. Although approaches that combine robust and adaptive variants of MPC, as detailed in [2], [37]–[40], achieve impressive levels of robustness and adaptability in real-world conditions, they still suffer from high computational costs, and inefficiencies in policy learning, as outlined in *Problem Statement 1*, remain.

   Recent model-free RL approaches, such as Rapid Motor Adaptation (RMA), have demonstrated impressive adaptation and generalization performance on a variety of robots/conditions [41]–[43], including in the context of adaptive attitude control on UAVs [44]. However, obtaining RMA policies requires reward selection and tuning, which may be challenging for the combined position and attitude control on UAVs. A procedure that enables efficient policy learning from MPC and rapid onboard adaptation thus continues to be an unresolved issue.

5. **Efficient, Robust and Agile Flight on a Sub-Gram Aerial Robot.** Flying insects exhibit incredibly agile flight abilities, being capable of performing a flip in only 0.4 ms [45], flying under large wind disturbances [46]–[48], and withstanding collisions [49], [50]. Insect-scale flapping-wing Micro Aerial Vehicles (MAVs) [51]–[54] have the potential to replicate these robustness and agile flight properties, extending their applications to tight and narrow spaces that become difficult for larger scale MAVs [55]–[59]. A key capability needed for the deployment of sub-gram MAVs in

real-world missions is the ability to accurately track desired agile trajectories while being robust to real-world uncertainties, such as collisions and wind disturbances. However, achieving robust, accurate, and agile trajectory tracking in sub-gram MAVs has significant challenges. First, their exceptionally fast dynamics [60] demand high-rate feedback control loops to ensure stability and rapid disturbance rejection, while the small payload limits onboard computation capabilities. Additionally, in order to maximize the lifespan of the robot components, control actions need to be planned in a way that is aware of actuation constraints. For instance, soft dielectric elastomer actuators (DEAs) suffer dielectric breakdown under a high electric field, posing a hard restraint on the maximum operating voltage [61]. Furthermore, the lifetime of passively-rotating wing hinges can be substantially extended under moderate control inputs. Lastly, manufacturing imperfection due to the small scale, and hard-to-model unsteady flapping-wing aerodynamics make it difficult to identify accurate models for simulation and control. A strategy for compute efficient onboard agile flight control for sub-gram, soft-actuated UAVs that account for uncertainties remains therefore an open challenge.

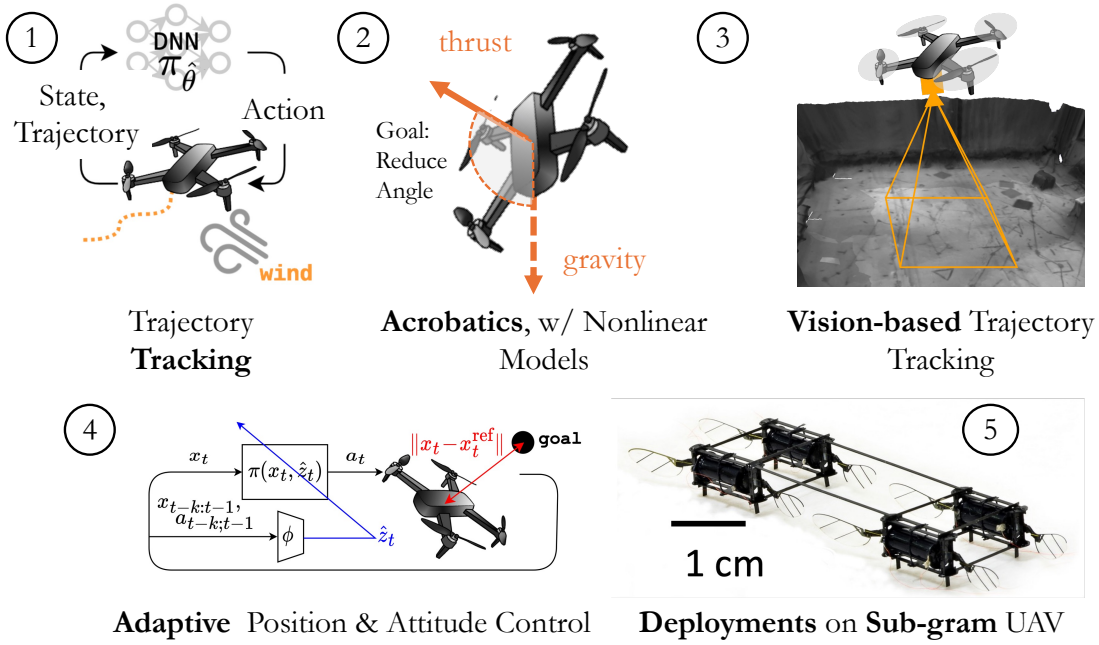


Figure 1.1: Graphic representation of the contributions of this thesis.

1.3 Thesis Contributions and Structure

This thesis aims to address the aforementioned gaps by presenting efficient imitation learning strategies for learning policies robust in the real-world. The contributions are shown in Fig. 1.1 and Table 1.1.

Table 1.1: Contributions of this thesis.

Contributions: Efficient, Robust Imitation Learning for...	**Chapter**	**Ref.**	**Media**
① **Trajectory Tracking**	3	[62], [63]	video
② **Acrobatic Flights**	4	[63]	video
③ **Vision-Based Flight Control**	5	[64], [65]	video, code
④ **Rapid Adaptation**	6	[66]	video
⑤ **Control of Sub-Gram, Soft-Actuated UAVs**	7	[67] [1]	video

[1] awarded finalist for the best paper in Dynamics and Control at the 2023 International Conference in Robotics and Automation.

1.3.1 Efficient, Robust Imitation Learning from MPC for Trajectory Tracking

The first contribution of this thesis is summarized as follows:

- A procedure to *efficiently* learn *robust* trajectory tracking policies from MPC. The procedure is: 1. *demonstration-efficient*, as it requires a small number of queries to the training environment, resulting in a method that enables learning from a single MPC demonstration collected in simulation or on the real robot; 2. *training-efficient*, as it reduces the number of computationally expensive queries to the computationally expensive MPC expert using a computationally efficient DA strategy; 3. *generalizable*, as it produces policies robust to disturbances not experienced during training.

- Extensive simulations and comparisons with state-of-the-art IL methods and robustification strategies.

- We validate the proposed approach by providing the first experimental (hardware) demonstration of zero-shot transfer of a DNN-based trajectory tracking controller for an aerial robot, learned from a single demonstration, in an environment (low-fidelity simulation or controlled lab environment) without disturbances, and transferred to an environment with wind-like disturbances.

1.3.2 Generalization to Agile Flights using Nonlinear Models

The second contribution of this thesis is summarized as follows:

- Generalize the demonstration-efficient policy learning strategy proposed in Contribution 1 with the ability to efficiently learn robust and generalizable policies from variants of MPC that use nonlinear models. This is challenging, as the framework leveraged to perform DA in the linear case, the ancillary controller in RTMPC, requires expensive computations to generate extra actions for DA in the nonlinear case. This results

in long training times and computational inefficiencies when performing DA. This contribution overcomes the computational-efficiency issues in the ancillary controller of nonlinear RTMPC by generating an approximation of the ancillary controller that is used to more efficiently compute the actions corresponding to extra state samples for DA. Additionally, this contribution presents a policy fine-tuning procedure to minimize the impact on performance introduced by our approximation.

- A formulation of nonlinear RTMPC for acrobatic flights on multirotors.
- Extensive simulations and comparisons with state-of-the-art IL methods and robustification strategies.
- Experimental evaluation on the challenging task of acrobatic maneuvers on a multirotor under uncertainties.

1.3.3 Generalization to Vision-based Flight Control

The third contribution of this thesis is summarized as follows:

- A DA strategy enabling efficient (demonstrations, training time) learning of a *sensorimotor* policy from MPC. The policy generates actions using raw images and other measurements, instead of the full-state estimate in our Contribution 1, and is robust in the real world to a variety of uncertainties. Our approach is grounded in the output feedback RTMPC framework theory, unlike previous DA methods that rely on handcrafted heuristics, and uses a Neural Radiance Field (NeRF) to generate images.
- A procedure to apply our methodology for tracking and localization on a multirotor using images, altitude, attitude and velocity data.
- Real-time deployments, demonstrating (in more than 30 flights) successful agile trajectory tracking with policies learned from a **single demonstration** that use onboard fisheye images to infer the horizontal position of a multirotor despite aggressive 3D

motion, and subject to a variety of sensing and dynamics disturbances. Our policy has an average inference time of only 1.5 ms onboard a small GPU (Nvidia Jetson TX2) and is deployed at 200 Hz.

1.3.4 Generalization to Rapid Adaptation

The fourth contribution of this thesis is summarized as follows:

- Extension of efficient and robust IL strategy from MPC for trajectory tracking [62] with the ability to learn robust *and adaptive* policies, therefore reducing tracking errors under uncertainties while maintaining high learning efficiency. Key to our work is to leverage the performance of an RMA-like [41] adaptation scheme, but without relying on RL, therefore avoiding reward selection and tuning.

- Procedure to apply the methodology to the task of adaptive position and attitude control for a multirotor, demonstrating for the first time RMA-like adaptation to uncertainties that cause position and orientation errors, unlike previous work [44] that only focuses on adaptive attitude control.

- Real-time deployment onboard a UAV, demonstrating position and attitude control adaptation to large disturbances not experienced during training (e.g., sudden propeller failure) during trajectory tracking, and robustness and adaptation to the *simultaneous* presence of multiple disturbances.

1.3.5 Hardware Demonstrations on a Sub-gram, Flapping-wing Aerial Robot

The fifth contribution of this thesis is summarized as follows:

- The first computationally-efficient strategy for robust, MPC-like control of sub-gram MAVs. Our approach employs a deep-learned neural network (NN) policy that is

trained to reproduce a trajectory tracking RTMPC, leveraging the strategy developed in Contribution 1 (our previous IL work [62]).

- A cascaded control strategy for Sub-gram, Soft-Actuated, Flapping-wind aerial robots, where the attitude controller in [68] is modified with a model adaptation method to compensate for the effects of uncertainties.

- Experimental evaluation on the MIT SoftFly [60], an agile sub-gram MAV (0.7 g), showing a 60% reduction in maximum trajectory tracking errors over [60], while being real-time implementable (2 kHz) on a small computational platform.

- The first strategy to achieve aggressive and fast flights on a new variant of a sub-gram MAV [69] (0.76 g), leveraging the methodology developed in Contribution 2. The presented approach repeatably achieved in experiments an horizontal velocity above 120 cm/s, being four-times faster than existing work [69], using a policy learned from 4 demonstrations and that runs at 1 kHz on a small computer.

Chapter 2

Related Work

2.1 Efficient and Robust Motor Policies

Explicit MPC. A well-established method to generate a fast approximation of linear MPC is *explict* MPC [1], where a policy is pre-computed offline by partitioning the feasible state space and by solving an optimal control problem for each region; the online optimization problem is reduced to finding the region corresponding to the current state via a look-up table or DNNs [70], [71]. However, the memory requirement and computational complexity of explicit MPC grow exponentially in the number of constraints [1], [70], and these methods have been mainly studied for linear systems, and do not leverage task-specific demonstrations to identify the more relevant parts of the feasible state space, therefore missing opportunities to optimize their performance or training efforts for those more relevant regions. Our work is motivated by the computational reduction opportunities obtained by DNN [72], but we leverage task-relevant demonstrations and IL to focus the learning efforts on the most relevant parts of the policy input space, while learning from MPC that use nonlinear models.

IL from MPC. Imitation-learning policies from MPC is a strategy widely employed in the robotics literature to reduce the onboard computational cost of this type of controller. Close to our work, [15] learns to perform acrobatic maneuvers with a quadrotor from MPC using

Table 2.1: Strategies for policy learning from model-based planners/controllers. We highlight that most of the approaches that have been successfully deployed in the real world required expensive offline training/data collection procedures, while approaches that are efficient at training time did not explicitly account for uncertainties. Our work unifies these aspects, enabling efficient learning of fast policies that account for uncertainties.

Method	Explicitly Accounts for uncertainties	Data-efficient training	Compute-efficient training	Allows both off/on-policy data collection	Demonstrations from both sim. and real robot	State ≥ 10 and under-actuated	Real-world and agile deployment
BC [18]	No	No	No	n.a.	Yes	Yes	No
DAgger [16], [29]	No	No	No	n.a.	Yes	Yes	Yes
DR [22], [23]	Yes	No	No	Yes	Yes	Yes	Yes
GPS [11], [12]	No	Yes	Yes	n.a.	No	Yes	No
MPC-Net [14]	No	Yes	Yes	No	Yes	Yes	No
LAG-ROS [74]	Yes	n.a.	No	No	No	No	No
[25] (DA)	No	No	Yes	No	No	No	No
[26] (DA)	No	Yes	Yes	No	No	No	No
SA (proposed)	Yes	Yes	Yes	Yes	Yes	Yes	Yes

DAgger combined with DR, by collecting 150 demonstrations in simulation. Ref. [29] uses DAgger combined with an MPC based on differential dynamic programming (DDP) [73] for agile off-road autonomous driving using about 24 laps[1] around their racetrack for the first DAgger iteration. All these examples demonstrate the *performance* that can be achieved when imitation-learning policies from MPC, but they also highlight that current methods require a large number of interactions with the MPC expert and the training environment, resulting in longer training times or complex data collection procedures, as summarized in Table 2.1

Robustness in IL. Robustness in IL is needed to compensate for the distribution shifts caused by the *sim2real* or *lab2real* (i.e., when collecting demonstrations on the real robot in a controlled environment and then deploying in the real world) transfers. Robustness to these types of *shifts* is achieved by modifying the training domain so that its dynamics match the ones encountered in the deployment domain [22], [75]. An extremely effective method is DR [22], which applies random model errors/disturbances, sampled from a predefined

[1]Obtained using Table 2 in [29], considering 6000 observation/action pairs sampled at 50 Hz while racing on a 30 m long racetrack with an average speed of 6 m/s.

set of possible perturbations, during data collection in simulation. An alternative avenue relies on modifying the actions of the expert to ensure that the state distribution visited at training time matches the one encountered at deployment time, such as in DART [24] and Ref. [76]. Although effective, these approaches require many demonstrations/interactions with the environment in order to take into account all the possible instantiations of model errors/disturbances that might be encountered in the target domain, limiting the opportunities for *lab2real* transfers, or increasing the data collection effort when training in simulation. Our work will exploit extra information available to the MPC to reduce the number of MPC/environment interactions.

Data Augmentation for Efficient/Robust IL. GPS [11]–[14], [17], introduced first the idea to use trajectories from model-based planners, including MPC, to generate state-action samples (guiding samples) for improved sample efficiency in policy learning. Specifically, Ref. [11] leveraged an iterative linear quadratic regulator (iLQR) [6] expert to generate guiding samples around the optimal trajectory found by the controller. Similarly, the authors in [14] observe that adding extra states and actions sampled from the neighborhood of the optimal solution found by the iLQR expert can reduce the number of demonstrations required to learn a policy when using DAgger. However, while GPS methods are in general more sample-efficient than IL, the nominal plans and the distribution of guiding-samples they generate do not *explicitly* account for model and environment uncertainties, resulting in policies with limited robustness. Ref. [13] for example, demonstrates in simulation robustness to up to 3.3% in weight perturbations of a multirotor, while our approach demonstrates robustness to perturbations up to 30%. Our work leverages a robust variant of MPC called RTMPC [3], [27], to provide robust demonstrations and a DA strategy that accounts for the effects of uncertainties. Specifically, the DA strategy is obtained by using an outer-approximation of the robust control invariant set (*tube*) as a support of the sampling distribution, ensuring that the guiding samples produce robust policies. This idea is related to the recent LAG-ROS framework [74], which provides a learning-based method

to compress a global planner in a DNN by extracting relevant information from the robust tube. LAG-ROS emphasizes the importance of nonlinear contraction-based controllers (e.g., CV-STEM [77]) to obtain robustness and stability guarantees. Our contribution emphasizes instead minimal requirements - namely a tube and an *efficient* DA strategy - to achieve demonstration-efficiency and robustness to real-world conditions. By decoupling these aspects from the need for complex control strategies, our work greatly simplifies the controller design. Additionally, different from LAG-ROS, the DA procedures presented in our work do not require solving a large optimization problem for every extra state-action sample generated (achieving computational efficiency during training) and can additionally leverage interactive experts (e.g., DAgger) to trade off the number of interactions with the environment with the number of extra samples from DA (further improving training efficiency).

Recent work [25], [26] exploited local approximations of the solutions found when solving the nonlinear program (NLP) associated with MPC to efficiently generate extra state-actions samples for DA in policy learning. Similar to our work, [25] uses a parametric sensitivity-based approximation of the solution to efficiently generate extra states and actions. Different from our work, however, their method proposes sampling of the entire feasible state space to learn a policy, while our work focuses instead on task-relevant demonstrations, a more computationally and data-efficient solution. The recently presented extension [26] solves this issue by leveraging interactive experts (e.g., DAgger). However, both [25], [26] do not *explicitly* account for the effects of uncertainties, neither in the design of the expert, nor in the way that extra states are generated, resulting in policies with limited robustness. Thanks to our robust expert, our approach not only accounts for uncertainties during demonstration collection and in the distribution of samples for DA, but it can additionally account for the errors introduced in the DA procedure by further constraint tightening and updating the tube size. Additionally, thanks to the strong prior on the state distribution under uncertainty produced by the tube in RTMPC, our DA strategy can quickly cover the task-relevant parts of the state space, obtaining demonstration-efficiency. Last, unlike prior work, we experimentally

validate our approach, demonstrating it on a system whose models have a large state size (state size 8 and 10), whereas previous work focuses on lower-dimensional systems (state size 2) and only in simulation.

Robustness and Computational Challenges in MPC for Agile Flight. MPC has been widely employed in the aerial robotics community [78], enabling impressive performance in trajectory tracking and minimum-time planning for agile flights, and particularly in drone racing [79]–[81]. However, the authors of [79] highlight that one of the biggest drawbacks of MPC is in its required computational resources, limiting its deployment on platforms with a small computational budget. In addition, they highlight that their MPC tends to fail when subject to a large external force disturbance or model errors. Our work is motivated by these findings and employs robust variants of MPC that explicitly account for uncertainties, such as disturbances and model errors, while reducing the computational complexity of MPC. Impressive agile flight has also been achieved by MPC with models learned offline [82], [83] or online [84], [85], or with MPC combined with non-parametric adaptation laws [86]. Our approach can benefit these fields, as RTMPC can explicitly account for uncertainties in learned models and can account for the dynamics introduced by adaptation laws [2], reducing the constraint violations observed in [86].

2.2 Visuomotor Imitation Learning

Robust/efficient IL of sensorimotor policies. Table 2.2 presents state-of-the-art approaches for sensorimotor policy learning from demonstrations (from MPC or humans), focusing on mobile robots.

The approach presented in our Contribution 2 (Chapter 5), named Tube-NeRF, is the only method that (i) explicitly accounts for uncertainties, (ii) is efficient to train, and (iii) does not require visual abstractions (iv) nor specialized data collection setups. Related to our research, [36] employs a NeRF for DA from human demonstrations for manipulation, but uses

Table 2.2: Approaches that learn visuomotor policies from demonstrations. Highlights: (i) real-world deployments of policies trained entirely in simulation often require visual-abstractions (e.g., [15]), losing information about the environment. Instead, approaches that directly use images (ii) benefit from data collection in the real-world, but require a large number of demonstrations (e.g., [29]), or (iii) leverage DA strategies that however use specialized data collection equipment and their deployment has been focused on 2D domains ([20], [34]) or to generate low-dimensional, discrete actions for aerial tasks (e.g., move left-right [33]). Last, all the considered DA approaches use ad-hoc heuristics to select extra sensorial data and/or to compute the corresponding actions.

Method	Domain of training data	Policy directly uses images	No special data collection equipment	Demo.-efficient	Explicitly accounts for uncertainties	Avoids Hand-Crafted Heuristics for Data Augm.	Real-world deployment (2D/3D, Domain)
[13] (**MPC-GPS**)	Sim.	Yes	Yes	No	No	N.A.	No (3D, Aerial)
[12] (**PLATO**)	Sim.	No	Yes	Yes	No	N.A.	No (3D, Aerial)
[35] (**BC+DA**)	Sim.	Yes	Yes	Yes	No	No	No (3D, Aerial)
[15] (**DAgger+DR**)	Sim	No	Yes	No	Yes	N.A.	Yes (3D, Aerial)
[29] (**DAgger**)	Real	Yes	Yes	No	No	N.A.	Yes (2D, Ground)
[34] (**DAgger+DA**)	Real	Yes	No	Yes	No	No	Yes (2D, Ground)
[20] (**BC+DA**)	Real	Yes	No	Yes	No	No	Yes (2D, Ground)
[33] (**BC+DA**)	Real	Yes	No	Yes	No	No	Yes (3D, Aerial)
[36] (**SPARTAN**)	Real	Yes	Yes	Yes	No	No	Yes (3D, Arm)
Tube-NeRF (proposed)	Real	Yes	Yes	Yes	Yes	Yes	Yes (3D, Aerial)

heuristics to select relevant views, without explicitly accounting for uncertainties. Tube-NeRF uses properties of robust MPC to select relevant views and actions for DA, accounting for uncertainties. In addition, we incorporate available real-world images for DA, further reducing the sim-to-real gap.

Novel View Synthesis with Meshes. Triangle meshes are a widespread scene representation and can be used to efficiently generate photorealistic novel views from sparse images. While meshes can be obtained from readily available open-source 3D photogrammetry pipelines [87], they are often generated from data collected via specialized 3D scanning equipment, and novel view generations using meshes depend on the ability to correctly reconstruct the underlying 3D geometry of the environment, a task that may be challenging in texture-poor scenes, causing "gaps" and artifacts in the reconstruction.

Novel View Synthesis with NeRFs. NeRFs [88] enable efficient and photorealistic novel view synthesis by directly optimizing the photometric accuracy of the reconstructed images, in contrast to traditional 3D photogrammetry methods (e.g., for 3D meshes). This provides accurate handling of transparency, reflective materials, and lighting conditions, and therefore they constitute the method of choice for novel view synthesis employed in this Thesis. Recent work [88] has mitigated NeRFs' large computational requirements, making them an appealing approach for low *sim2real* gap image generation in visuomotor policy learning. Close to our work, [89] uses a NeRF to build a simulator that enables learning policies for robot control from RGB images, but their focus is on legged robots control trained via RL. In addition, they use a specialized camera for data collection, while our work employs images collected by the low-cost fisheye camera onboard the UAV. Ref. [90] uses a NeRF for estimation, planning, and control on a drone by querying the NeRF online, but this results in 1000× higher computation time[2] than our policy.

Output Feedback RTMPC. MPC [1] solves a constrained optimization problem that uses a model of the system dynamics to plan for actions that satisfy state and actuation constraints. RTMPC assumes that the system is subject to additive, bounded *process* uncertainty (disturbances, model errors) and employs an auxiliary (ancillary) controller that maintains the system within some known distance (cross-section of a tube) of the plan [4]. *Output feedback* RTMPC [4], [91] in addition accounts for the effects of *sensing* uncertainty (noise, estimation errors) by increasing the cross-section of the tube. Our method uses an output feedback RTMPC for data collection but bypasses its computational cost at deployment by learning a NN policy.

[2]Control and estimation in [90] require 6.0s on a NVIDIA RTX3090 GPU (10,496 CUDA cores, 24 GB VRAM), while ours requires 1.5ms on a much smaller Jetson TX2 GPU (256 CUDA cores, 8 GB shared RAM).

2.3 Rapid Adaptation

Adaptive Control. Adaptation strategies can be classified into two categories, direct and indirect methods. Indirect methods aim at explicitly estimating models or parameters, and these estimates are leveraged in model-based controllers, such as MPC [1]. Model/parameter identification include filtering techniques [92], [93], disturbance observers [94]–[96], set-membership identification methods [2], [37] or active, learning-based methods [40]. While these approaches achieve impressive performance, they often suffer from high computational cost due to the need of identifying model parameters online and, when MPC-based strategies are considered, solving large optimization problems online. Direct methods, instead, develop policy updates that improve a certain performance metric. This metric is often based on a reference model, while the updates involve the shallow layers of the DNN policy [97]–[99]. Additionally, policy update strategies can be learned offline using meta-learning [100], [101]. While these methods employ computationally-efficient DNN policies, they require extra onboard computation to update the policy, require costly offline training procedures, and/or do not account for actuation constraints. Parametric adaptation laws, such as $\mathcal{L}_1$ adaptive control [102], have been applied to the control inputs generated by MPC [103] [39], significantly improving MPC performance; however, these approaches still require solving onboard the large optimization problem associated with MPC, and [103] does not account for control limits. Our work leverages the inference speed of a DNN for computationally-efficient onboard deployment, training the policy using an efficient IL procedure (our previous work [62]) that uses a robust MPC capable of accounting for state and actuation constraints.

Rapid Motor Adaptation (RMA). RMA [41] has recently emerged as a high-performance, hybrid adaptive strategy. Its key idea is to learn a DNN policy conditioned on a low-dimensional (encoded) model/environment representation that can be efficiently inferred online using another DNN. The policy is trained using RL, in a simulation where it experiences different instances of the model uncertainties/disturbances. RMA policies have controlled a

wide range of robots, including quadruped [41], biped [42], hand-like manipulators [43] and multirotors [44], demonstrating rapid adaptation and generalization to previously unseen disturbances. Our work takes inspiration from the adaptation strategy introduced by RMA, as we learn policies conditioned on a low-dimensional environment representation that is estimated online. However, unlike RMA, our learning procedure does not require the reward selection and tuning typically encountered in RL, as it leverages an efficient IL strategy from robust MPC. An additional difference to [44], where RMA is used to generate a policy for attitude control of multirotors of different sizes, is that our work focuses on the challenging task of learning an adaptive trajectory tracking controller, which compensates for the effects of uncertainties on its attitude *and* position.

2.4 Agile Flight at Insect and Larger Scales

Existing sub-gram MAVs have demonstrated promising agile flight capabilities [60], [104], [105], but none of the existing controllers deployed on sub-gram MAVs *explicitly* account for environment and model uncertainties, and for actuation constraints and usage. An agile trajectory tracking strategy that has found success on larger-scale MAVs (e.g., palm-sized quadrotors) consists of decoupling position and attitude control via a cascaded scheme, where a fast feedback loop (*inner*) controls the attitude of the MAV, while a potentially slower loop (*outer*) tracks the desired trajectory by generating commands for the attitude controller. An *outer loop* controller that enables agile, robust, actuation-aware trajectory tracking is MPC [1], [2], [5]–[9]. This strategy generates actions by minimizing an objective function that explicitly trades tracking accuracy for actuation usage, taking into account the state and actuation constraints. This is achieved by solving a constrained optimization problem online, where a model of the robot is employed to plan along a predefined temporal horizon by taking into account the effects of future actions. Robust variants of MPC, such as robust tube MPC (RTMPC) [2], [3], can additionally take into account uncertainties (disturbances,

model errors) when generating their plans and control actions. This is done by employing an auxiliary (ancillary) controller capable of maintaining the system within some distance ("cross-section" of a tube) from the nominal plan regardless of the realization of uncertainty.

Chapter 3

Trajectory Tracking

3.1 Overview

In this chapter, we address the problem of generating a robust DNN policy from MPC in a demonstration and computationally efficient manner by designing a computationally-efficient DA strategy that systematically compensates for the effects of covariate shifts that might be encountered during real-world deployment. Our approach, named Sampling Augmentation (SA) and depicted in Fig. 3.1, relies on a prior model of the perturbations/uncertainties encountered in a deployment domain, which is used to generate a robust version of the given MPC, called RTMPC, to collect demonstrations and to guide the DA strategy. The key idea behind this DA strategy consists in observing that the RTMPC framework provides: (a) information on the states that the robot may visit when subject to uncertainty. This is represented by a *tube* that contains the collected demonstration; the tube can be used to identify/generate extra relevant states for DA; and (b) an *ancillary controller* that maintains the robot inside the tube regardless of the realization of uncertainties; this controller can be used to generate extra actions. To numerically and experimentally validate our approach, we tailor SA to the task of efficiently learning robust policies for agile flight on a multirotor. Specifically, we demonstrate in experiments trajectory tracking capabilities with a policy

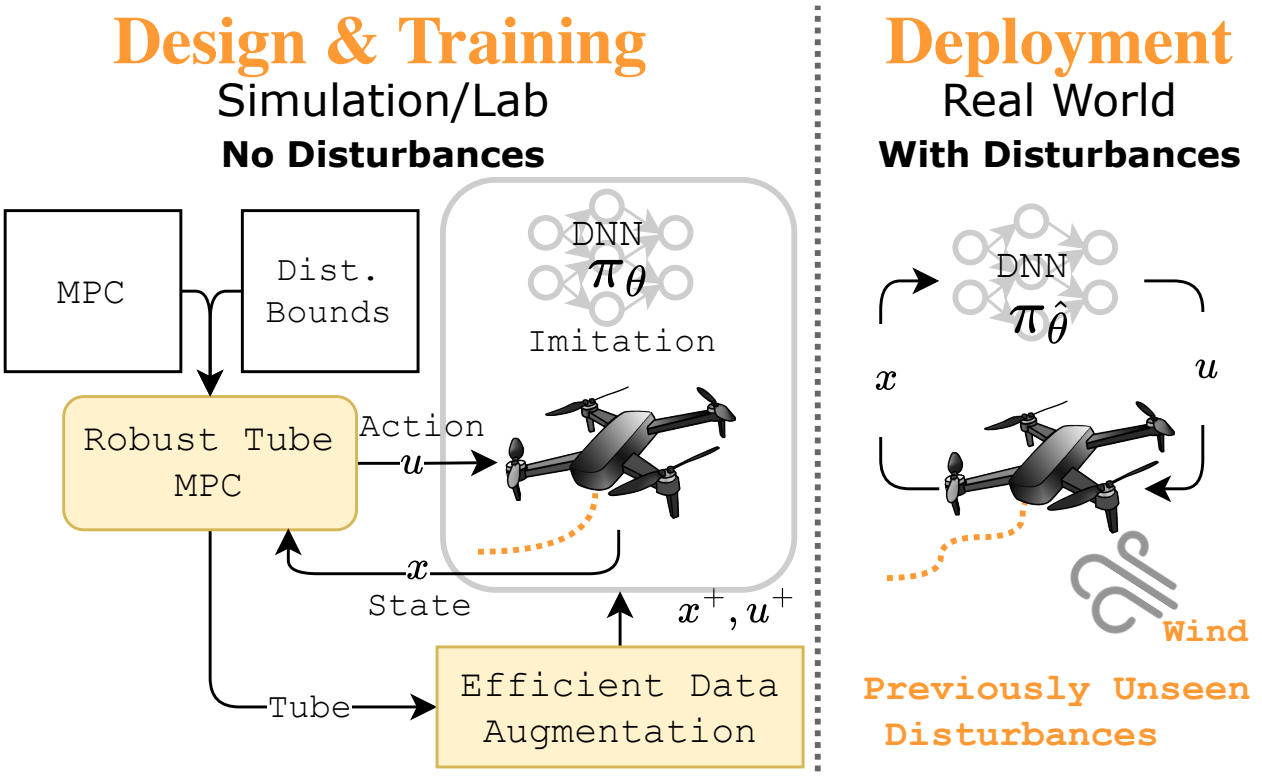


Figure 3.1: Overview of the proposed approach to generate a deep neural network-based policy π_θ from a computationally expensive Model Predictive Control in a demonstration and training-efficient way. We do so by generating a robust tube MPC using bounds of the disturbances encountered in the deployment domain. We use properties of the tube to derive a computationally efficient data augmentation strategy that generates extra state-action pairs (x^+, u^+), obtaining $\pi_{\hat{\theta}}$ via IL. Our approach enables zero-shot transfer from a single demonstration collected in simulation (*sim2real*) or a controlled environment (lab, factory, *lab2real*).

learned from a linear trajectory tracking RTMPC. The policy is learned from a *single* demonstration collected in simulation or directly on the real robot, and it is robust to previously-unseen wind disturbances.

3.2 Problem Formulation

This part describes the problem of learning a robust policy in a demonstration and computationally efficient way by imitating an MPC expert demonstrator. Robustness and efficiency are determined by the ability to design an IL procedure that can compensate for the covariate shifts induced by uncertainties encountered during real-world deployment while collecting demonstrations in an environment (the training environment) that presents only a subset of those uncertainty realizations. Our problem statement follows the one of robust IL (e.g., DART [24]), modified to use deterministic policies/experts and to account for the differences in uncertainties encountered in deployment and training environments. Additionally, we present a common approach employed to address the covariate shift issues caused by uncertainties, DR, highlighting its limitations.

3.2.1 Assumptions and Notation

System Dynamics. We assume the dynamics of the real system are Markovian and stochastic [106], and can be described by a twice continuously differentiable function $f(\cdot)$:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{w}_t, \tag{3.1}$$

where $\mathbf{x}_t \in \mathbb{X} \subseteq \mathbb{R}^{n_x}$ represents the state, $\mathbf{u}_t \in \mathbb{U} \subseteq \mathbb{R}^{n_u}$ the control input in the compact subsets $\mathbb{X}$, $\mathbb{U}$. $\mathbf{w}_t \in \mathbb{W}_{\mathcal{E}} \subset \mathbb{R}^{n_x}$ is an unknown state perturbation, belonging to a compact convex set $\mathbb{W}_{\mathcal{E}}$ containing the origin. Stochasticity in Eq. (3.1) is introduced by $\mathbf{w}_t$, sampled from a probability distribution having support $\mathbb{W}_{\mathcal{E}}$, under a (possibly unknown) probability

density function, capturing the effects of noise, approximation errors in the learned policy, model changes, and other disturbances acting on the system during training or under real-world conditions at deployment.

Sim2Real and Lab2Real Transfer Setup. Three different environments/domains[1] $\mathcal{E}$ are considered: a training environment based on a simulation $\mathcal{S}_{\text{sim}}$ (where $\mathcal{S}$ denotes *source*), a training environment based on the real robot in a controlled lab environment $\mathcal{S}_{\text{lab}}$, and a deployment environment $\mathcal{T}$ (*target*). Mathematically, the three environments differ in their transition probabilities induced by different uncertainty realizations/distributions. In all the environments, we do not assume knowledge on density function from which $\mathbf{w}_t$ is sampled, but we assume available prior knowledge of $\mathbb{W}_{\mathcal{T}}$, the support of the distribution (e.g, worst-case uncertainty realization, also assumed finite) at deployment. This is a common assumption in robust control [3], [27], where such knowledge can come from historical data, regulatory requirements, or can be assumed to match the physical limits of the robot. Additionally, we assume $\mathbb{W}_{\mathcal{S}_{\text{lab}}} \subset \mathbb{W}_{\mathcal{T}}$ and $\mathbb{W}_{\mathcal{S}_{\text{sim}}} \subset \mathbb{W}_{\mathcal{T}}$, representing the fact that training is usually performed in simulation or in a controlled/lab environment under some nominal model errors/disturbances, while at deployment a larger set of perturbations can be encountered. Note that for convenience we use $\mathcal{S}$ to denote both $\mathcal{S}_{\text{sim}}$ and $\mathcal{S}_{\text{lab}}$.

MPC Expert. We consider a tracking MPC expert demonstrator that plans along an $N+1$-steps horizon. The expert is given the current state $\mathbf{x}_t \in \mathbb{X}$, and a sequence of $N_{\text{des}}+1$ desired states $\mathbf{X}_t^{\text{des}} := \{\mathbf{x}_{0|t}^{\text{des}}, \dots, \mathbf{x}_{N_{\text{des}}|t}\}$[2] representing a desired trajectory to be followed. Note that $\mathbf{X}_t^{\text{des}} \in \underbrace{\mathbb{X}^{\text{des}} \times \cdots \times \mathbb{X}^{\text{des}}}_{N_{\text{des}}+1 \text{ times}} := (\mathbb{X}^{\text{des}})^{N_{\text{des}}+1}$ and $\mathbb{X}^{\text{des}} \subseteq \mathbb{R}^{n_x}$ (the desired trajectory can violate state constraints). Then, the MPC expert generates control actions by solving an

[1] Note that the term *domain* is borrowed from the domain adaptation and transfer learning literature [107], where it denotes different probability distributions $p(x, y)$ over the same feature-label space pair. In our context, a domain corresponds to a training or deployment environment characterized by distinct uncertainty distributions. These uncertainties in turn induce different state (feature) distributions. Throughout this work, we use *domain* interchangeably with *environment* (training, deployment) to highlight these similarities.

[2] An alternative and commonly employed notation used to represent a trajectory is $\mathbf{x}_{t:t+N_{\text{des}}}^{\text{des}}$.

Optimal Control (OC) problem of the form:

$$\begin{aligned} \bar{\mathbf{X}}_t^*, \bar{\mathbf{U}}_t^* \in \underset{\bar{\mathbf{X}}_t, \bar{\mathbf{U}}_t}{\operatorname{argmin}} \; & J_N(\bar{\mathbf{X}}_t, \bar{\mathbf{U}}_t, \mathbf{X}_t^{\text{des}}) \\ \text{subject to } & \bar{\mathbf{x}}_{0|t} = \mathbf{x}_t, \\ & \bar{\mathbf{x}}_{i+1|t} = f(\bar{\mathbf{x}}_{i|t}, \bar{\mathbf{u}}_{i|t}), \\ & \bar{\mathbf{x}}_{i|t} \in \mathbb{X}, \;\; \bar{\mathbf{u}}_{i|t} \in \mathbb{U}, \\ & \forall i = 0, ..., N-1. \end{aligned} \tag{3.2}$$

where J_N represents the cost to be minimized (where N denotes the dependency on the planning horizon), and $\bar{\mathbf{X}}_t = \{\bar{\mathbf{x}}_{0|t}, \ldots, \bar{\mathbf{x}}_{N|t}\}$ and $\bar{\mathbf{U}}_t = \{\bar{\mathbf{u}}_{0|t}, \ldots, \bar{\mathbf{u}}_{N-1|t}\}$ are sequences of states and actions along the planning horizon, where the notation $\bar{\mathbf{x}}_{i|t}$ indicates the planned state at the future time $t+i$, as planned at the current time t. At every timestep t, given $\mathbf{x}_t$, the control input applied to the real system is the first element of $\bar{\mathbf{U}}_t^*$, resulting in an implicit deterministic control law (policy) that we denote as $\pi_{\boldsymbol{\theta}^*} : \mathbb{X} \times (\mathbb{X}^{\text{des}})^{N+1} \to \mathbb{U}$.

DNN Student Policy. As for the MPC expert, we model the DNN student policy as a deterministic policy $\pi_{\boldsymbol{\theta}}$, with parameters $\boldsymbol{\theta}$, that does not necessarily belong to the same policy class as the expert. Indeed, while the policy class of the student is a neural network, the expert utilizes an MPC framework. When considering trajectory tracking tasks, the policy takes as input the current state and the desired reference trajectory segment, $\pi_{\boldsymbol{\theta}} : \mathbb{X} \times (\mathbb{X}^{\text{des}})^{N+1} \to \mathbb{U}$, matching the input/outputs of the MPC expert. We note that the initial state may be far from the desired trajectory, and the MPC expert can plan the maneuver to approach the desired trajectory from such initial state.

Transition Probabilities. We denote the state transition probability under $\pi_{\boldsymbol{\theta}}$ in an environment $\mathcal{E}$ for a given task as $p_{\pi_{\boldsymbol{\theta}},\mathcal{E}}(\mathbf{x}_{t+1}|\mathbf{x}_t)$. The probability of collecting a T-(state, action) pairs trajectory $\boldsymbol{\xi} = \{(\mathbf{x}_t, \mathbf{u}_t)_{t=0}^{T-1}\}$, given a policy $\pi_{\boldsymbol{\theta}}$, depends on the deployment environment $\mathcal{E}$:

$$p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{E}) = p(\mathbf{x}_0) \prod_{t=0}^{T-1} p_{\pi_{\boldsymbol{\theta}},\mathcal{E}}(\mathbf{x}_{t+1}|\mathbf{x}_t), \tag{3.3}$$

where $p(\mathbf{x}_0)$ represents the initial state distribution.

3.2.2 Robust Imitation Learning Objective

The objective of robust IL, following [24], is to find parameters $\boldsymbol{\theta}$ of $\pi_{\boldsymbol{\theta}}$ that minimize a distance metric $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi})$ from the MPC expert $\pi_{\boldsymbol{\theta}^*}$:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{T})} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi}). \tag{3.4}$$

This metric captures the differences between the actions generated by the expert $\pi_{\boldsymbol{\theta}^*}$ and the action produced by the student $\pi_{\boldsymbol{\theta}}$ across the distribution of trajectories induced by the student policy $\pi_{\boldsymbol{\theta}}$ in the perturbed environment $\mathcal{T}$, as denoted by $p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{T})$. The distance metric considered in this work is the Mean Squared Error (MSE) loss:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi}) = \frac{1}{T} \sum_{t=0}^{T-1} \|\pi_{\boldsymbol{\theta}} - \pi_{\boldsymbol{\theta}^*}\|_2^2. \tag{3.5}$$

Covariate Shift due to Sim2real and Lab2real Transfer. Because in practice we do not have access to the target environment, the goal of Robust IL is to try to solve Eq. (3.4) by finding an approximation $\hat{\boldsymbol{\theta}}$ of the expert policy using data from the source environment:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{S})} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi}). \tag{3.6}$$

The way this minimization is solved depends on the chosen IL algorithm. The performance of the learned policy in the target and source environments can be related via:

$$\begin{aligned} &\mathbb{E}_{p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{T})} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi}) = \\ &\underbrace{\mathbb{E}_{p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{T})} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi}) - \mathbb{E}_{p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{S})} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi})}_{\text{covariate shift due to transfer}} \\ &\qquad + \underbrace{\mathbb{E}_{p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{S})} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi})}_{\text{IL objective}}, \end{aligned} \tag{3.7}$$

which clearly shows the presence of a covariate shift[3] induced by the transfer. The last term corresponds to the objective minimized by performing IL in $\mathcal{S}$. Attempting to solve Eq. (3.4) by directly optimizing Eq. (3.6) (e.g., via BC [18]) offers no assurances of finding a policy with good performance in $\mathcal{T}$.

3.2.3 Shift Compensation via Domain Randomization.

A well-known strategy to compensate for the effects of covariate shifts between source and target environment is DR [22], which modifies the transition probabilities of the source $\mathcal{S}$ by trying to ensure that the trajectory distribution in the modified training environment $\mathcal{S}_{\text{DR}}$ matches the one encountered in the target environment: $p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{S}_{\text{DR}}) \approx p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{T})$. This is done by applying perturbations to the robot during demonstration collection, sampling perturbations $\mathbf{w} \in \mathbb{W}_{\text{DR}}$ according to some knowledge/hypotheses on their distribution $p_{\mathcal{T}}(\mathbf{w})$ in the target environment [22], obtaining the perturbed trajectory distribution $p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{S}, \mathbf{w})$. The minimization of Eq. (3.4) can then be approximately performed by minimizing instead:

$$\mathbb{E}_{p_{\mathcal{T}}(\mathbf{w})}[\mathbb{E}_{p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{S}, \mathbf{w})}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi})]. \tag{3.8}$$

This approach, however, requires the ability to apply disturbances/model changes to the system, which may be unpractical e.g., in the *lab2real* setting, and may require a large number of demonstrations due to the need to sample enough state perturbations $\mathbf{w}$.

3.3 Efficient Learning from Linear RTMPC

In this Section, we present the strategy to efficiently learn robust policies from MPC when the system dynamics in Eq. (3.1) can be well approximated by a linear model of the form:

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t. \tag{3.9}$$

[3]This definition of covariate shift is adapted from [24]

First, we present the Robust Tube variant of linear MPC, RTMPC, that we employ to collect demonstrations (Section 3.3.1). Then, we present a strategy that leverages information available from the RTMPC expert to compensate for the covariate shifts caused by uncertainties and mismatches between the training and deployment domains (Section 3.3.2). Our strategy is based on a DA procedure that can be combined with different IL methods (on-policy, such as DAgger [21], and off-policy, such as BC, [18]) for improved efficiency/robustness in the policy learning procedure. The RTMPC expert is based on [3] but with the objective function modified to track desired trajectories, as trajectory-tracking tasks will be the focus of the experimental evaluation of policies learned from this controller (Section 3.5).

3.3.1 Trajectory Tracking Robust Tube MPC Expert Formulation

RTMPC is a type of robust MPC that regulates the system in Eq. (3.9) while ensuring satisfaction of the state and actuation constraints $\mathbb{X}, \mathbb{U}$ regardless of the disturbances $\mathbf{w} \in \mathbb{W}_{\mathcal{T}}$.

Mathematical Preliminaries. Let $\mathbb{A} \subset \mathbb{R}^n$ and $\mathbb{B} \subset \mathbb{R}^n$ be convex polytopes, and let $\mathbf{C} \in \mathbb{R}^{m \times n}$ be a linear mapping. In this context, we establish the following definition:

a) Linear mapping: $\mathbf{C}\mathbb{A} \coloneqq \{\mathbf{C}\mathbf{a} \in \mathbb{R}^m \mid \mathbf{a} \in \mathbb{A}\}$

b) Minkowski sum: $\mathbb{A} \oplus \mathbb{B} \coloneqq \{\mathbf{a} + \mathbf{b} \in \mathbb{R}^n \mid \mathbf{a} \in \mathbb{A},\ \mathbf{b} \in \mathbb{B}\}$

c) Pontryagin difference: $\mathbb{A} \ominus \mathbb{B} \coloneqq \{\mathbf{c} \in \mathbb{R}^n \mid \mathbf{c} + \mathbf{b} \in \mathbb{A}, \forall \mathbf{b} \in \mathbb{B}\}$.

Optimization Problem. At each time step t, trajectory tracking RTMPC receives the current robot state $\mathbf{x}_t$ and a desired trajectory $\mathbf{X}_t^{\text{des}} = \{\mathbf{x}_{0|t}^{\text{des}}, \ldots, \mathbf{x}_{N|t}^{\text{des}}\}$ spanning $N+1$ steps as input. It then computes a sequence of reference ("safe") states $\bar{\mathbf{X}}_t = \{\bar{\mathbf{x}}_{0|t}, \ldots, \bar{\mathbf{x}}_{N|t}\}$ and actions $\bar{\mathbf{U}}_t = \{\bar{\mathbf{u}}_{0|t}, \ldots, \bar{\mathbf{u}}_{N-1|t}\}$ that ensure constraint compliance regardless of the realization of $\mathbf{w}_t \in \mathbb{W}_{\mathcal{T}}$. This is achieved by solving the following quadratic program (QP) (e.g., via the

solver [108]):

$$\begin{aligned}\bar{\mathbf{U}}_t^*, \bar{\mathbf{X}}_t^* = \underset{\bar{\mathbf{U}}_t, \bar{\mathbf{X}}_t}{\operatorname{argmin}} & \|\mathbf{e}_{N|t}\|_{\mathbf{P}_x}^2 + \sum_{i=0}^{N-1} \|\mathbf{e}_{i|t}\|_{\mathbf{Q}_x}^2 + \|\mathbf{u}_{i|t}\|_{\mathbf{R}_u}^2 \\ & \text{subject to } \bar{\mathbf{x}}_{i+1|t} = \mathbf{A}\bar{\mathbf{x}}_{i|t} + \mathbf{B}\bar{\mathbf{u}}_{i|t}, \\ & \bar{\mathbf{x}}_{i|t} \in \mathbb{X} \ominus \mathbb{Z}, \;\; \bar{\mathbf{u}}_{i|t} \in \mathbb{U} \ominus \mathbf{K}\mathbb{Z}, \\ & \mathbf{x}_t \in \mathbb{Z} \oplus \bar{\mathbf{x}}_{0|t}, \; \forall i = 0, \ldots, N-1\end{aligned} \tag{3.10}$$

where $\mathbf{e}_{i|t} = \bar{\mathbf{x}}_{i|t} - \mathbf{x}_{i|t}^{\text{des}}$ is the tracking error. The matrix $\mathbf{R}_u$ (positive definite) and $\mathbf{Q}_x$ (positive definite) define the trade-off between deviations from the desired trajectory and actuation usage, while $\|\mathbf{e}_{N|t}\|_{\mathbf{P}_x}^2$ is the terminal cost. $\mathbf{P}_x$ and $\mathbf{K}$ are obtained by formulating an infinite horizon optimal control LQR problem using $\mathbf{A}$, $\mathbf{B}$, $\mathbf{Q}_x$ and $\mathbf{R}_u$ and by solving the associated algebraic Riccati equation [109]. To achieve recursive feasibility, we ensure a sufficiently long prediction horizon is selected, as commonly practiced [110], while omitting the inclusion of terminal set constraints.

Tube and Ancillary Controller. A control input for the real system is generated by RTMPC via an *ancillary controller*:

$$\mathbf{u}_t = \bar{\mathbf{u}}_t^* + \mathbf{K}(\mathbf{x}_t - \bar{\mathbf{x}}_t^*), \tag{3.11}$$

where $\bar{\mathbf{u}}_t^* = \bar{\mathbf{u}}_{0|t}^*$ and $\bar{\mathbf{x}}_t^* = \bar{\mathbf{x}}_{0|t}^*$. As shown in Fig. 3.2, this controller ensures that the system remains inside a *tube* (with "cross-section" $\mathbb{Z}$) centered around $\bar{\mathbf{x}}_t^*$ regardless of the realization of the disturbances in $\mathbb{W}_\mathcal{T}$, provided that the tube contains the initial state of the system (constraint $\mathbf{x}_t \in \mathbb{Z} \oplus \bar{\mathbf{x}}_{0|t}$). The set $\mathbb{Z}$ is a disturbance invariant set for the closed-loop system $\mathbf{A}_K := \mathbf{A} + \mathbf{B}\mathbf{K}$, satisfying the property that $\forall \mathbf{x}_j \in \mathbb{Z}$, $\forall \mathbf{w}_j \in \mathbb{W}_\mathcal{T}$, $\forall j \in \mathbb{N}^+$, $\mathbf{x}_{j+1} = \mathbf{A}_K\mathbf{x}_j + \mathbf{w}_j \in \mathbb{Z}$ [3]. $\mathbb{Z}$ can be computed offline using $\mathbf{A}_K$ and the model of the disturbance $\mathbb{W}$ via ad-hoc analytic algorithms [1], [3], or can be learned from data [111]. Note that tracking aggressive trajectories may introduce large deviations from the operating points,

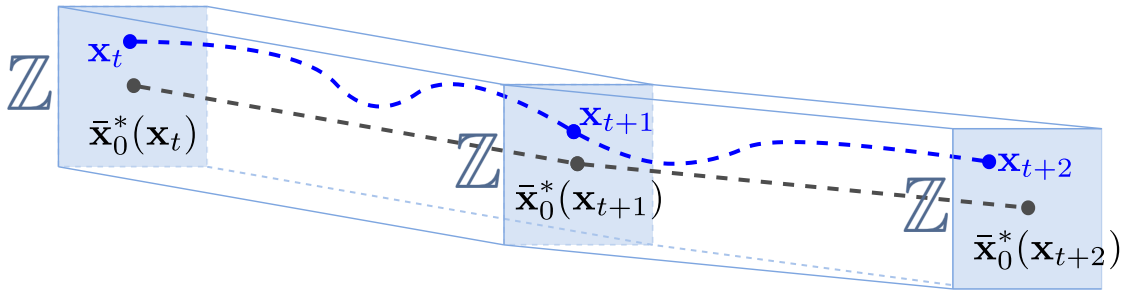


Figure 3.2: Illustration of the robust control invariant tube $\mathbb{Z}$ centered around the optimal reference $\bar{\mathbf{x}}_0^*(\mathbf{x}_t)$ computed by RTMPC at every state $\mathbf{x}$, for a system with state dimension $n_x = 2$.

resulting in linearization errors; these errors are treated as an additional source of process uncertainty when computing the tube. In addition, aggressive changes of the reference may result in infeasibility (e.g., when the terminal region is unreachable within the horizon, see [112]), which can be addressed, as typical in MPC, via an adequate choice of the planning horizon ($N = 20$ or $N = 30$ in our work).

3.3.2 Shift Compensation via Sampling Augmentation

Training a policy by collecting demonstrations in a controlled source domain $\mathcal{S}$, with the objective of deploying it in a perturbed target domain $\mathcal{T}$ introduces a sample selection bias [113], i.e., data is not collected around the distribution encountered in $\mathcal{T}$. Such bias is a known cause of distribution shifts [113], and can be mitigated by re-weighting collected samples based on their likelihood of appearing in the target domain $\mathcal{T}$ via importance-sampling [11]. Importance-sampling, however, does not apply in our case, since we do not have access to samples/demonstrations collected in $\mathcal{T}$.

In this work, distribution shifts are addressed by additionally utilizing the tube in RTMPC to obtain knowledge of the states that the system may visit when subjected to perturbations in $\mathcal{T}$. Given this information, we propose a tube-guided DA strategy, called Sampling Augmentation (SA), that samples states from the tube and *efficiently* computes

corresponding actions via the ancillary controller in RTMPC.

Tube as a Model of State Distribution Under Uncertainties. The key intuition of the proposed approach is the following. We observe that, although the density function of $p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}}, \mathcal{T})$ is unknown, an approximation of its support $\mathfrak{R}$, given a demonstration $\boldsymbol{\xi}$ collected in the source domain $\mathcal{S}$, is known and corresponds to the tube in RTMPC when collecting $\boldsymbol{\xi}$:

$$\mathfrak{R}_{\boldsymbol{\xi}^+|\pi_{\boldsymbol{\theta}^*},\boldsymbol{\xi}} = \{\bar{\mathbf{x}}_t^* \oplus \mathbb{Z}\}_{t=0}^{T-1}. \tag{3.12}$$

where $\boldsymbol{\xi}^+$ is a trajectory in the tube of $\boldsymbol{\xi}$. This is true thanks to the ancillary controller in Eq. (3.11), which ensures that the system remains inside Eq. (3.12) for every possible realization of $\mathbf{w} \in \mathbb{W}_{\mathcal{T}}$. The ancillary controller additionally provides a *computationally efficient* way to obtain the actions to apply for every state inside the tube. Let $\mathbf{x}_{t,j}^+ \in \bar{\mathbf{x}}_t^* \oplus \mathbb{Z}$, i.e., $\mathbf{x}_{t,j}^+$ is a state inside the tube computed when the system is at $\mathbf{x}_t$, then the corresponding robust control action $\mathbf{u}_{t,j}^+$ is:

$$\mathbf{u}_{t,j}^+ = \bar{\mathbf{u}}_t^* + \mathbf{K}(\mathbf{x}_{t,j}^+ - \bar{\mathbf{x}}_t^*). \tag{3.13}$$

For every timestep t in $\boldsymbol{\xi}$, extra state-action samples $(\mathbf{x}_{t,j}^+, \mathbf{u}_{t,j}^+)$, with $j = 1, \ldots, N_s$ collected from within the tube can be used to augment the dataset employed to train the policy, obtaining a way to approximate the expected risk in the domain $\mathcal{T}$ by only having access to demonstrations collected in $\mathcal{S}$:

$$\begin{aligned} &\mathbb{E}_{p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}},\mathcal{T})}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi}) \approx \\ &\quad \mathbb{E}_{p(\boldsymbol{\xi}|\pi_{\boldsymbol{\theta}},\mathcal{S})}[\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi}) + \mathbb{E}_{p(\boldsymbol{\xi}^+|\pi_{\boldsymbol{\theta}^*},\boldsymbol{\xi})}\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^*|\boldsymbol{\xi}^+)]. \end{aligned} \tag{3.14}$$

Tube Approximation and Sampling Strategies. In practice, the density $p(\boldsymbol{\xi}^+|\pi_{\boldsymbol{\theta}^*}, \boldsymbol{\xi})$ may not be available, making it difficult to establish which states to sample for DA. We consider an adversarial approach to the problem by sampling states that may be visited under worst-case perturbations. To efficiently compute those samples, we (outer) approximate the tube $\mathbb{Z}$ with an axis-aligned bounding box $\hat{\mathbb{Z}}$. Note that this approximation is also used in the

Input: $\mathbf{A}, \mathbf{B}, \mathbb{X}, \mathbb{U}, \mathbf{Q}_x, \mathbf{R}_u, \mathbb{W}_{\mathcal{T}}, \beta, \mathcal{S}, \mathbf{X}^{\text{des}}$
Output: Trained policy $\pi_{\hat{\boldsymbol{\theta}}_M}$

1: $\pi_{\boldsymbol{\theta}^*}, \mathbf{K}, \hat{\mathbb{Z}} \leftarrow \text{DesignRtmpc}(\mathbf{A}, \mathbf{B}, \mathbb{X}, \mathbb{U}, \mathbf{Q}_x, \mathbf{R}_u, \mathbb{W}_{\mathcal{T}})$
2: $\mathcal{D}, \pi_{\hat{\boldsymbol{\theta}}_0} \leftarrow \emptyset, \text{InitializePolicy}()$
3: **for** $i = 1$ **to** M **do**
4: $\quad \mathcal{D} \leftarrow \emptyset$ // optional
5: $\quad$ **for** $t = 0$ **to** $T - 1$ **do**
6: $\quad\quad \mathbf{u}_t^{\text{RTMPC}}, \bar{\mathbf{x}}_t^*, \bar{\mathbf{u}}_t^* \leftarrow \pi_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \mathbf{X}_t^{\text{des}})$ // Eq. (3.10) and Eq. (3.11)
7: $\quad\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_t, \mathbf{X}_t^{\text{des}}, \mathbf{u}_t^{\text{RTMPC}})\}$
8: $\quad\quad$ **for** $j = 1$ **to** N_{s} **do**
9: $\quad\quad\quad \mathbf{u}_{t,j}^+ = \bar{\mathbf{u}}_t^* + \mathbf{K}(\mathbf{x}_{t,j}^+ - \bar{\mathbf{x}}_t^*)$, $\mathbf{x}_{t,j}^+ \in \bar{\mathbf{x}}_t^* \oplus \hat{\mathbb{Z}}$
10: $\quad\quad\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_{t,j}^+, \mathbf{X}_t^{\text{des}}, \mathbf{u}_{t,j}^+)\}$
11: $\quad\quad \mathbf{u}_t \leftarrow \beta_i \mathbf{u}_t^{\text{RTMPC}} + (1 - \beta_i)\, \pi_{\hat{\boldsymbol{\theta}}_{i-1}}(\mathbf{x}_t, \mathbf{X}_t^{\text{des}})$ // DAgger/BC
12: $\quad\quad \mathbf{x}_{t+1} \leftarrow \text{StepSystem}(\mathbf{u}_t, \mathbf{x}_t, \mathcal{S})$ // Sim./Physical Robot
13: $\quad \pi_{\hat{\boldsymbol{\theta}}_i} \leftarrow \text{UpdatePolicy}(\mathcal{D}, \hat{\boldsymbol{\theta}}_{i-1})$

Algorithm 1: Sampling Augmentation (SA) for efficient learning from trajectory-tracking linear RTMPC.

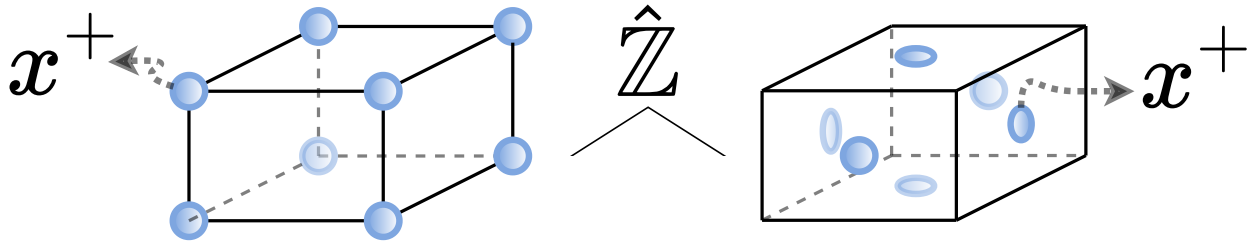


Figure 3.3: The possible strategies to sample extra state-action pairs from an axis-aligned bounding box approximation of the tube of the RTMPC expert: dense (left) and sparse (right). The tube is represented for a system with state dimension $n_x = 3$.

RTMPC (Eq. (3.10)) during demonstration collection. We investigate two strategies, shown in Fig. 3.3, to obtain state samples $\mathbf{x}_{t,j}^+$ at every state $\mathbf{x}_t$ in $\boldsymbol{\xi}$: i) dense sampling: sample extra states from the vertices of $\bar{\mathbf{x}}_t^* \oplus \hat{\mathbb{Z}}$. The approach produces $N_s = 2^{n_x}$ extra state-action samples. It is more conservative, as it produces more samples, but more computationally expensive. ii) sparse sampling: sample one extra state from the center of each *facet* of $\bar{\mathbf{x}}_t^* \oplus \hat{\mathbb{Z}}$, producing $N_s = 2n_x$ additional state-action pairs. It is less conservative and more computationally efficient.

Algorithm Summary. The procedure is summarized in Algorithm 1. First, SA designs the RTMPC expert according to the uncertainties in the target $\mathbb{W}_{\mathcal{T}}$ (line 1) and randomly initializes the student policy (line 2). Then, SA collects in the source domain $\mathcal{S}$ a demonstration,

using DAgger or BC, where β_i is an hyperparameter of DAgger controlling the probability of using actions from the expert and $\beta = 1$ corresponds to BC, storing state and actions in the dataset $\mathcal{D}$ (line 7). The safe plan from the expert is then used to generate extra data via Eq. (3.13) (line 9), and the policy is updated (line 13, Eq. (3.4) and Eq. (3.5) using the data in $\mathcal{D}$ and starting from the previous policy weights $\hat{\boldsymbol{\theta}}_{i-1}$). The data collection and training procedure can be repeated across M demonstrations.

3.4 Application to Agile Flight

In this Section, we tailor the proposed efficient policy learning strategies to agile flight tasks, as this will be the focus of our numerical and experimental evaluation. Specifically, in Section 3.4.1, we present the nonlinear model of the multirotor used to collect demonstrations in simulation in all our approaches, and used for control design. Then, in Section 3.4.2, we present a RTMPC expert for *trajectory tracking* based on a *linear* multirotor model and that will be used with the IL procedure described in Section 3.3. Because the considered trajectories require the robot to operate around a fixed, pre-defined condition (near hover), a hover-linearized model is suitable for the design of this controller.

3.4.1 Nonlinear Multirotor Model

We consider an inertial reference frame W attached to the ground, and a non-inertial frame B attached to the center of mass (CoM) of the robot. The translational and rotational dynamics of the multirotor are:

$$ {}_{\mathrm{W}}\dot{\boldsymbol{p}} = {}_{\mathrm{W}}\boldsymbol{v} \tag{3.15a} $$

$$ {}_{\mathrm{W}}\dot{\boldsymbol{v}} = m^{-1}(\boldsymbol{R}_{\mathrm{WB}}\, {}_{\mathrm{B}}\boldsymbol{t}_{\mathrm{cmd}} + {}_{\mathrm{W}}\boldsymbol{f}_{\mathrm{drag}} + {}_{\mathrm{W}}\boldsymbol{f}_{\mathrm{ext}}) - {}_{\mathrm{W}}\boldsymbol{g} \tag{3.15b} $$

$$ \dot{\boldsymbol{q}}_{\mathrm{WB}} = \frac{1}{2}\boldsymbol{\Omega}({}_{\mathrm{B}}\boldsymbol{\omega})\boldsymbol{q}_{\mathrm{WB}} \tag{3.15c} $$

$$ {}_{\mathrm{B}}\dot{\boldsymbol{\omega}} = \boldsymbol{I}_{\mathrm{mav}}^{-1}(- {}_{\mathrm{B}}\boldsymbol{\omega} \times \boldsymbol{I}_{\mathrm{mav}}\, {}_{\mathrm{B}}\boldsymbol{\omega} + {}_{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{cmd}} + {}_{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{drag}}) \tag{3.15d} $$

where $\boldsymbol{p}$, $\boldsymbol{v}$, $\boldsymbol{q}$, $\boldsymbol{\omega}$ are, respectively, position, velocity, attitude quaternion and angular velocity of the robot, with the prescript denoting the corresponding reference frame. The attitude quaternion $\boldsymbol{q} = [q_w, \boldsymbol{q}_v^\top]^\top$ consists of a scalar part q_w and a vector part $\boldsymbol{q}_v = [q_x, q_y, q_z]^\top$ and it is unit-normalized; the associated 3×3 rotation matrix is $\boldsymbol{R} = \boldsymbol{R}(\boldsymbol{q})$, while

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & -\boldsymbol{\omega}^\top \\ \boldsymbol{\omega} & \lfloor \boldsymbol{\omega} \rfloor_\times \end{bmatrix}, \tag{3.16}$$

with $\lfloor \boldsymbol{\omega} \rfloor_\times$ denoting the 3×3 skew symmetric matrix of $\boldsymbol{\omega}$. m denotes the mass, $\boldsymbol{I}_{\text{mav}}$ the 3×3 diagonal inertial matrix, and $\boldsymbol{g} = [0, 0, g]^\top$ the gravity vector. Aerodynamic effects are taken into account via $\boldsymbol{f}_{\text{drag}} = -c_{D,1}\boldsymbol{v} - c_{D,2}\|\boldsymbol{v}\|\boldsymbol{v}$ and isotropic drag torque $\boldsymbol{\tau} = -c_{D,3}\boldsymbol{\omega}$, capturing the parasitic drag produced by the motion of the robot. The robot is additionally subject to external force disturbances $\boldsymbol{f}_{\text{ext}}$, such as the one caused by wind or by an unknown payload. Last, $\boldsymbol{t}_{\text{cmd}} = [0, 0, t_{\text{cmd}}]^\top$ is the commanded thrust force, and $\boldsymbol{\tau}_{\text{cmd}}$ the commanded torque. These commands can be mapped to the desired thrust $f_{\text{prop},i}$ for the i-th propeller ($i = 1, \ldots, n_p$) via a linear mapping (*allocation* matrix) $\boldsymbol{\mathcal{A}}$:

$$\begin{bmatrix} t_{\text{cmd}} \\ \boldsymbol{\tau}_{\text{cmd}} \end{bmatrix} = \boldsymbol{\mathcal{A}} \begin{bmatrix} f_{\text{prop},1} \\ \vdots \\ f_{\text{prop},n_p} \end{bmatrix} = \boldsymbol{\mathcal{A}} \boldsymbol{f}_{\text{prop}}. \tag{3.17}$$

The attitude of the quadrotor is controlled via the geometric attitude controller in [114]. This controller generates desired torque commands ${}_{\text{B}}\boldsymbol{\tau}_{\text{cmd}}$ given a desired attitude $\boldsymbol{R}_{\text{WB}}^{\text{des}}$, angular

velocity ${}_{\mathrm{B}}\boldsymbol{\omega}^{\mathrm{des}}$ and acceleration ${}_{\mathrm{B}}\dot{\boldsymbol{\omega}}^{\mathrm{des}}$ via [114]:

$$
\begin{aligned}
{}_{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{cmd}} = &- \mathbf{K}_R\mathbf{e}_R - \mathbf{K}_\omega\mathbf{e}_\omega + {}_{\mathrm{B}}\boldsymbol{\omega} \times \mathbf{J}\,{}_{\mathrm{B}}\boldsymbol{\omega} \\
&- \mathbf{J}({}_{\mathrm{B}}\boldsymbol{\omega}^{\wedge}\boldsymbol{R}_{\mathrm{WB}}^{\top}\boldsymbol{R}_{\mathrm{WB}}^{\mathrm{des}}\,{}_{\mathrm{B}}\boldsymbol{\omega}^{\mathrm{des}} - \boldsymbol{R}_{\mathrm{WB}}^{\top}\boldsymbol{R}_{\mathrm{WB}}^{\mathrm{des}}\,{}_{\mathrm{B}}\dot{\boldsymbol{\omega}}^{\mathrm{des}}), \\
\mathbf{e}_R = &\frac{1}{2}(\boldsymbol{R}_{\mathrm{WB}}^{\mathrm{des}\,\top}\boldsymbol{R}_{\mathrm{WB}} - \boldsymbol{R}_{\mathrm{WB}}^{\top}\boldsymbol{R}_{\mathrm{WB}}^{\mathrm{des}})^{\vee}, \\
\mathbf{e}_\omega = &\,{}_{\mathrm{B}}\boldsymbol{\omega} - \boldsymbol{R}_{\mathrm{WB}}^{\top}\boldsymbol{R}_{\mathrm{WB}}^{\mathrm{des}}\,{}_{\mathrm{B}}\boldsymbol{\omega}^{\mathrm{des}}.
\end{aligned}
\tag{3.18}
$$

The diagonal matrices $\mathbf{K}_R, \mathbf{K}_\omega$ of size 3×3 are tuning parameters of the controller, while $\mathbf{e}_R$ denotes the attitude error, and $\mathbf{e}_\omega$ is its time derivative. The symbol $(\mathbf{r}^{\wedge})^{\vee} = \mathbf{r}$ denotes the operation transforming a 3×3 skew-symmetric matrix $\mathbf{r}^{\wedge}$ in a vector $\mathbf{r} \in \mathbb{R}^3$.

The position controllers designed in the next sections output setpoints for the attitude controller, and desired thrust t_{cmd}.

3.4.2 Linear Robust Tube MPC for Trajectory Tracking

The model employed by the linear RTMPC for trajectory tracking (Eq. (3.10)) is based on a simplified, hover-linearized model derived from Eq. (3.18), using the approach in [7], but modified to account for uncertainties. First, similar to [7], we express the model in a yaw-fixed, gravity-aligned frame I via the rotation matrix $\boldsymbol{R}_{\mathrm{BI}}$

$$
\begin{bmatrix} \phi \\ \theta \end{bmatrix} = \boldsymbol{R}_{\mathrm{BI}} \begin{bmatrix} {}_I\phi \\ {}_I\theta \end{bmatrix}, \; \boldsymbol{R}_{\mathrm{BI}} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix},
\tag{3.19}
$$

where the attitude has been represented, for interpretability, via the Euler angles yaw ψ, pitch θ, roll ϕ (*intrinsic* rotations around the z-y-x such that $\boldsymbol{R} = \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\phi)$, with $\boldsymbol{R}_l(\alpha)$ being a rotation of α around the l-th axis). Second, as in [7], we assume that the closed-loop attitude dynamics can be described by a first-order dynamical system that can be identified from experiments, replacing Eq. (3.15c), Eq. (3.15d). Last, different from [7], we assume ${}_{\mathrm{W}}\boldsymbol{f}_{\mathrm{ext}}$ in Eq. (3.15b) to be an unknown disturbance/model errors that capture

the uncertain parts of the model, such that ${}_{\mathrm{W}}\boldsymbol{f}_{\mathrm{ext}} \in \mathbb{W}$.

The controller generates tilt (roll, pitch) and thrust commands ($n_u = 3$) given the state of the robot ($n_x = 8$) consisting of position, velocity, and tilt, and given the reference trajectory. The desired yaw is fixed (and it is tracked by the cascaded attitude controller); similarly, ${}_{\mathrm{B}}\boldsymbol{\omega}^{\mathrm{des}}$ and ${}_{\mathrm{B}}\dot{\boldsymbol{\omega}}^{\mathrm{des}}$ are set to zero. We employ the nonlinear attitude compensation in [7].

The controller takes into account position constraints (e.g., available 3D flight space), actuation limits, and velocity/tilt limits via $\mathbb{X}$ and $\mathbb{U}$. The cross-section of the tube $\mathbb{Z}$ is a constant outer approximation based on an axis-aligned bounding box. It is estimated via Monte-Carlo sampling, by measuring the state deviations of the closed loop linear system $\mathbf{A}_K$ under the disturbances in $\mathbb{W}$.

3.5 Evaluation

We start by evaluating our policy learning approach for the task of trajectory tracking using the linear RTMPC expert.

3.5.1 Evaluation Approach and Details

Simulation Environment. Demonstration collection and policy evaluations are performed in a simulation environment implementing the nonlinear multirotor dynamics in Section 3.4.1, discretized at 400Hz, while the attitude controller runs at 200 Hz. The robot follows desired trajectories, starting from randomly generated initial states centered around the origin. Given the specified external disturbance magnitude bound $\mathbb{W}_{\mathcal{E}} = \{f_{\mathrm{ext}} \in \mathbb{R} | \underline{f}_{\mathrm{ext}} \leq f_{\mathrm{ext}} \leq \overline{f}_{\mathrm{ext}}\}$, disturbances are applied in the domain $\mathcal{E}$ by sampling ${}_{\mathrm{W}}\boldsymbol{f}_{\mathrm{ext}}$ via the spherical coordinates:

$$
{}_{\mathrm{W}}\boldsymbol{f}_{\mathrm{ext}} = f_{\mathrm{ext}} \begin{bmatrix} \cos(\phi)\sin(\theta) \\ \sin(\phi)\sin(\theta) \\ \cos(\theta) \end{bmatrix}, \quad \begin{aligned} f_{\mathrm{ext}} &\sim \mathcal{U}(\underline{f}_{\mathrm{ext}}, \overline{f}_{\mathrm{ext}}), \\ \theta &\sim \mathcal{U}(0, \pi), \\ \phi &\sim \mathcal{U}(0, 2\pi). \end{aligned} \tag{3.20}
$$

Linear RTMPC. The linear RTMPC expert demonstrator runs in simulation at 10 Hz, and its tube is designed assuming $\mathbb{W} = \{f_{\text{ext}} \in \mathbb{R} | 0 \leq f_{\text{ext}} \leq 0.35mg\}$, corresponding to the safe physical limit of the actuators of the robot. The reference fed to the expert is a sequence of desired positions and velocities for the next 3s, discretized with a sampling time of 0.1s; the expert uses a corresponding planning horizon of $N = 30$, (resulting in a reference being a 180-dim. vector).

Policy Architecture. The student policy is a 2-hidden layer, fully connected DNN, with $(32, 32)$ or $(64, 32)$ neurons/layer, and `ReLU` activation function. The total input dimension is 188 (matching the input of the expert, consisting of state and reference trajectory). The output dimension is 3 (desired thrust and tilt expressed in an inertial frame). We rotate the tilt output of the DNN in the body frame to avoid taking into account yaw, which is not part of the optimization problem [7], not causing any relevant computational cost. We additionally apply the nonlinear attitude compensation scheme as in [7].

Baselines and Training Details. We apply the proposed SA strategies to every demonstration collected via DAgger or BC, and we consider DAgger or BC without any augmentation/robustification approach (denoted **n.a.**), or combined with:

a) **DA (linear interpolation)**: a DA that groups the collected demonstrations based on the input reference trajectory (or reference position/time for the go-to-goal-position case that will be introduced in Chapter 4), and than randomly samples pairs of input-outputs in each cluster, linearly interpolating the state/action to obtain a new state-action pair.

b) **DA (expert neighborhood)**: a DA strategy that uniformly samples states from a region corresponding to 5% of the cross-section of the tube in RTMPC, centered around the current state of the robot. The corresponding actions are obtained using the ancillary controller. This baseline is useful at studying the importance of using the tube as a support of the sampling distribution.

c) **DR**: domain randomization.

During demonstration-collection in the source environment $\mathcal{S}$, we do not apply disturbances, setting $\mathbb{W}_{\mathcal{S}} = \{\emptyset\}$, with the exception for DR, where we sample disturbances from $\mathbb{W}_{\text{DR}} = \mathbb{W}_{\mathcal{T}}$. In all the methods that use DAgger we set the probability of using actions of the expert β (a hyperparameter of DAgger [21]) to be 1 at the first demonstration and 0 otherwise (as this was found to be the best-performing setup). The number of samples generated for the baseline DA methods is 16 per timesteps, matching the number used for SA-sparse, while SA-dense corresponds to 256 samples per timestep. Demonstrations are collected with a sampling time of 0.1s. After every collected demonstration, the policy is trained for up to 50 epochs[4] using all the data available so far with the ADAM [115] optimizer and a learning rate of 0.001, and we use early stopping, terminating the training if the validation loss (from 30% of the collected data) does not decreases within 7 epochs. The policy is then evaluated on the task for 10 times (episodes), starting from slightly different initial states centered around the origin, in both $\mathcal{S}$ and $\mathcal{T}$.

Evaluation Metrics: We monitor:

i) *Robustness (Success Rate)*, as the percentage of episodes where the robot never violates any state constraint;

ii) *Performance*, via either

   a) $C_{\boldsymbol{\xi}}(\pi_\theta) := \sum_{t=0}^{T} \|\mathbf{x}_t - \mathbf{x}_t^{\text{des}}\|^2_{\mathbf{Q}_x} + \|\mathbf{u}_t\|^2_{\mathbf{R}_u}$ tracking error along the trajectory (*MPC Stage Cost*); or

   b) $\|C_{\boldsymbol{\xi}}(\pi_{\theta^*}) - C_{\boldsymbol{\xi}}(\pi_{\hat{\theta}^*})\| / \|C_{\boldsymbol{\xi}}(\pi_{\theta^*})\|$ relative error between expert and policy tracking errors (*Expert Gap*);

iii) *Efficiency*

   (a) number of expert demonstrations (*Num. Demonstrations Used for Training*), and

   (b) wall-clock time to generate the policy (*Training Time* [5]).

[4] One epoch is one pass through the entire dataset

[5] Training time is the time to collect demonstrations and the time to train the policy, as measured by a

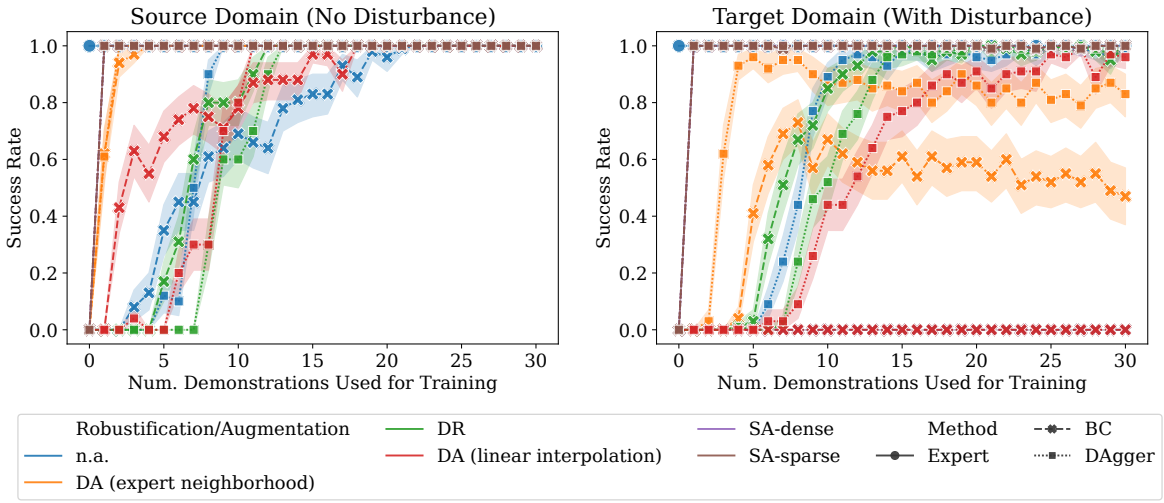


Figure 3.4: Robustness (*Success Rate*) in the task of flying along an eight-shaped, 7s long-trajectory, subject to wind-like disturbances (right, target domain $\mathcal{T}_1$) and without (left, source domain $\mathcal{S}$), starting from different initial states. Evaluation is repeated across 10 random seeds, 10 times per demonstration per seed. We additionally show the 95% confidence interval. The lines for the SA-based methods overlap.

3.5.2 Numerical Evaluation of Efficiency, Robustness, and Performance when Learning to Track a Single Trajectory

Tasks Description. Our objective is to generate a policy from linear RTMPC capable of tracking a 7s long (70 steps), figure eight-shaped trajectory. We evaluate the considered IL approaches in two different target domains, with wind-like disturbances ($\mathcal{T}_1$) or with model errors ($\mathcal{T}_2$). Disturbances in $\mathcal{T}_1$ are external force perturbations $\boldsymbol{f}_{\text{ext}}$ sampled from $\mathbb{W}_{\mathcal{T}_1} \approx \{f_{\text{ext}} | 0.25mg \leq f_{\text{ext}} \leq 0.3mg\}$. Model errors in $\mathcal{T}_2$ are applied via mismatches in the drag coefficients used between training and testing, representing uncertainties not explicitly considered during the design of the linear RTMPC.

Comparison with IL baselines. We start by evaluating the robustness in $\mathcal{T}_1$ as a function of the number of demonstrations collected in the source domain. The results are shown in Fig. 3.4, highlighting that: i) while all the approaches achieve robustness (full success rate)

wall-clock. In our evaluations, the simulated environment steps at its highest possible rate (in contrary to running at the same rate of the simulated physical system), providing an advantage to those methods that require a large number of environment interactions, such as the considered baselines.

Table 3.1: Comparison of the IL methods considered to learn a policy from RTMPC, highlight that the proposed SA-methods simultaneously achieve high robustness, demonstration efficiency and close performance to the expert, unlike the considered baselines which lack robustness, demonstration-efficiency or both. The task is tracking a 7s (70 steps) trajectory in a deployment domain with wind-like disturbances ($\mathcal{T}_1$), and one under model errors ($\mathcal{T}_2$, drag coefficient mismatch). At convergence (iteration 20-30 for non-SA methods, and 1-11 for our proposed SA-methods) we evaluate robustness (success rate) and performance (relative percent error between tracking error of expert and policy). Demonstration-Efficiency represents the number of demonstrations required to achieve, for the first time, full success rate. During data collection, an approach is considered easy if it does not require to apply disturbances/perturbations (e.g., in *lab2real* transfer, and safe if it does not execute actions that may cause state constraints violations (crashes). *Safe in our numerical evaluation, but not guaranteed as it requires executing actions of a policy that may be partially trained. Metrics color-coded from green/white (better) to red (worse).

Method		Data Collection		Robustness succ. rate (%)		Performance expert gap (%)		Demonstration Efficiency	
Robustification/ Augmentation	Imitation	Easy	Safe	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_1$	$\mathcal{T}_2$
n.a.	BC	Yes	Yes	0.0	100.0	22.8	37.2	-	18
	DAgger	Yes	No	97.6	100.0	13.6	2.9	-	9
DA (linear interpolation)	BC	Yes	Yes	0.0	99.9	23.3	36.7	-	16
	DAgger	Yes	No	92.9	100.0	26.9	3.7	-	12
DA (expert neighborhood)	BC	Yes	Yes	53.5	100.0	27.5	2.7	-	3
	DAgger	Yes	No	83.3	100.0	22.3	2.7	-	2
DR	BC	No	Yes	98.7	100.0	6.8	8.5	15	14
	DAgger	No	No	99.1	100.0	6.7	2.8	20	9
SA-Dense	BC	Yes	Yes	100.0	100.0	6.3	2.8	1	1
	DAgger	Yes	Yes*	100.0	100.0	6.3	2.8	1	1
SA-Sparse	BC	Yes	Yes	99.9	100.0	6.2	2.8	1	1
	DAgger	Yes	Yes*	100.0	100.0	6.3	2.8	1	1

in the source domain, SA achieves full success rate after only a single demonstration, being 3 times more sample efficient than the most demonstration-efficient baseline, DA (expert neighborhood), which however does not achieve full robustness in the target domain; ii) SA, instead, is also able to achieve full robustness in the target domain, while baseline methods do not fully succeed or converge at a much lower rate. These results emphasize the presence of a distribution shift between the source and target, which is not fully compensated for by baseline methods such as BC due to a lack of exploration and robustness.

The performance evaluation and additional results are summarized in Table 3.1. We highlight that in the target domain $\mathcal{T}_1$, SA achieves the performance that is closest to the

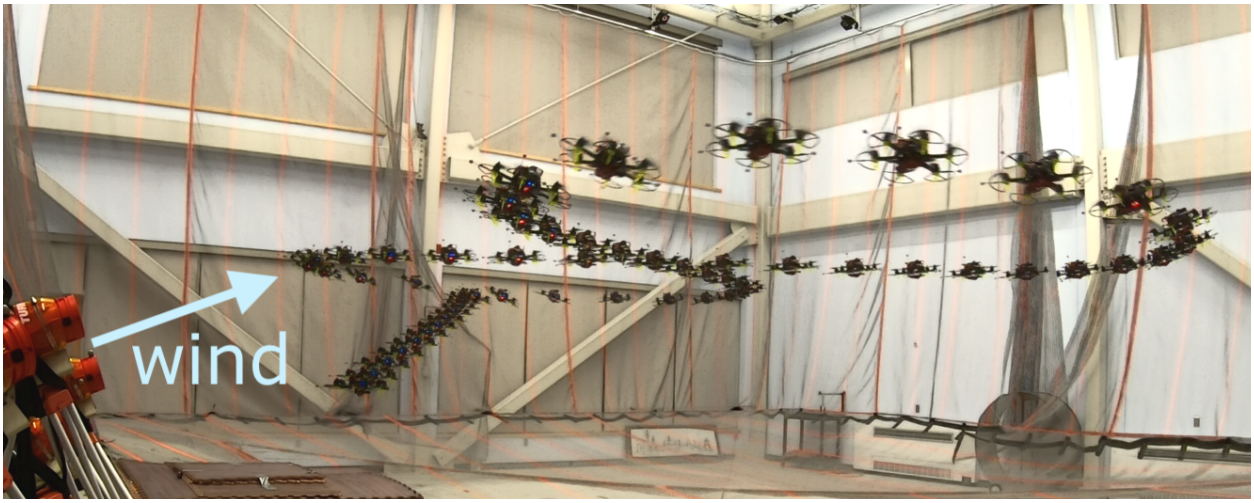
expert. Table 3.1 additionally presents the results for the target domain $\mathcal{T}_2$. Although this task is less challenging (i.e., all the approaches achieve full robustness), the proposed method (SA-sparse) achieves the highest demonstration-efficiency and among the lowest expert gap, with similar trends as in $\mathcal{T}_1$.

Training Time. Fig. 3.4 highlights that the best-performing baseline, DAgger+DR, requires about 10 demonstrations to learn to robustly track a 7s long trajectory, which corresponds to a total training time of 10.8s. Among the proposed approaches, DAgger+SA-sparse instead only requires 1 demonstration, corresponding to a training time of 3.8s, a 64.8% reduction in wall-clock time required to learn the policy. DAgger+SA-dense, instead, while requiring a single demonstration to achieve full robustness, necessitates 114s of training time due to the large number of samples generated. Because of its effectiveness and greater computational efficiency, we use SA-sparse, rather than SA-dense, for the rest of the work.

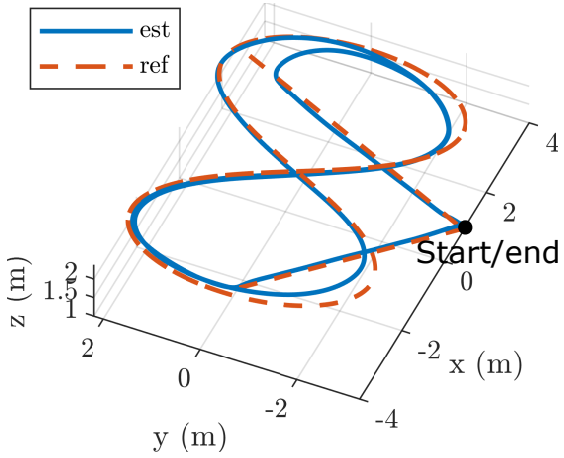
3.5.3 Hardware Evaluation for Tracking a Single Trajectory from a Single Demonstration

Sim2Real Transfers. We validate the demonstration-efficiency, robustness, and performance of the proposed approach by experimentally testing policies trained after a *single* demonstration collected in simulation using DAgger/BC (which operate identically since we use DAgger with $\beta = 1$ for the first demonstration), combined with SA-sparse. We use the MIT/ACL open-source snap-stack [116] for controlling the attitude of the MAV. The learned policy runs at 100Hz on the onboard `Nvidia Jetson TX2` (CPU), with the reference trajectory provided at 100Hz. State estimation is from a motion capture system, while the video listed in Table 1.1 includes additional experiments with onboard Visual-Inertial Odometry (VIO).

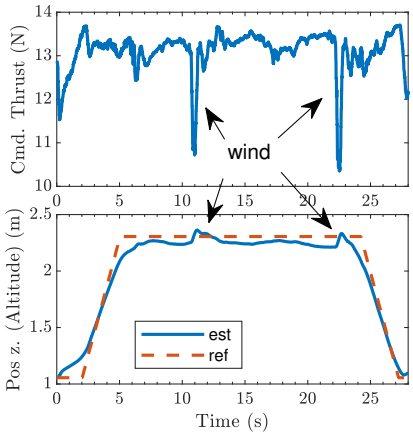
The task is to track a figure eight-shaped trajectory, with velocities up to 3.4m/s. We evaluate the robustness of the learned policy by applying a wind-like disturbance produced by an array of 3 leaf blowers (Fig. 3.5). The given position reference and the corresponding



(a) Time-lapse of an experiment showing the trajectory executed by the robot, and the leaf blowers used to generate disturbances



(b) Reference and actual trajectory



(c) Effects of wind

Figure 3.5: Experimental evaluation of a trajectory tracking policy learned from a *single* linear RTMPC demonstration collected in simulation, achieving *zero-shot* transfer. The multirotor is able to withstand previously unseen disturbances, such as the wind produced by an array of leaf-blowers, and whose effects are clearly visible in the altitude errors (and change in commanded thrust) in Fig. 3.5c. This demonstration-efficiency and robustness is enabled by Sampling-Augmentation (SA), our proposed tube-guided data augmentation strategy.

trajectory are shown in Fig. 3.5b. The effects of the wind disturbances are clearly visible in the altitude errors and changes in commanded thrust in Fig. 3.5b (at $t = 11$s and $t = 23$s). These experiments show that the learned policy can robustly track the desired reference, withstanding challenging perturbations unseen during the training phase.

Experimental Comparison. Table 3.2 reports an experimental comparison of SA (one demonstration) with MPC, RTMPC, and DAgger+DR (10 or 20 demonstrations) on the task of tracking different trajectories with a duration of 27 s, while the robot is subject to (1) wind speed up to 10m/s, (2) a slung load of 250 grams, and (3) a surface attached at the bottom of the robot that produces extra drag. The results confirm our finding in simulation, highlighting that SA-sparse achieves better or comparable performance and robustness than DAgger+DR, but under minimal training effort (one demonstration instead of 10-20). In addition, the evaluation highlights that RTMPC achieves larger tracking errors when the position constraints are tight (e.g., the reference trajectory is close to the position constraint), due to RTMPC's ability to maintain a safe distance from such constraints. However, this same property allows RTMPC to be safe (no constraint violation), unlike MPC which violates position constraints in the case of Slung Load + Wind. The learned policy runs at 500 Hz, while the RTMPC/and MPC run at their maximum rates (100 Hz, occupying the entire CPU).

Lab2Real Transfer. We evaluate the ability of the proposed method to learn from a single demonstration collected on a real robot in a controlled environment (lab) and generalize to previously unseen disturbances (real). We do so by collecting a RTMPC demonstration of a circular trajectory (velocity up to 3.5 m/s, with tight position constraints) with the multirotor, augmenting the collected demonstration with SA-sparse, and deploying the learned policy while we apply previously unseen disturbances (drag board, slung load). As shown in the sequence in Fig. 3.6, despite the large distribution shifts in velocity, the policy reproduces the expert demonstration and it is robust to previously unseen disturbances. Our video shows an additional example of wind disturbances.

			Lemniscate (27s, w/o tight position constr.)					**Circle** (27s, w/ tight position constr.)								**Constraints satisfied**
			# of	No Disturbance		Slung Load		# of	No Disturbance		Slung Load		Slung Load+Wind		Drag+Wind	
Method	**Agent**	**MAE type (m, ↓)**	Dem.	x,y	z	x,y	z	Dem.	x,y	z	x,y	z	x,y	z	x,y	z
Expert	MPC	Tracking	n.a.	**0.143**	0.145	**0.158**	**0.418**	n.a.	**0.151**	0.183	**0.207**	0.563	**0.193**	0.506	**0.219**	**0.284**	No
	RTMPC	Tracking	n.a.	0.144	**0.114**	**0.158**	0.423	n.a.	0.270	**0.161**	0.309	**0.561**	0.291	**0.433**	0.338	0.287	**Yes**
Student	DAgger+DR	Gap from RTMPC	10	0.062	0.124	0.091	**0.060**	20	**0.035**	**0.034**	**0.059**	0.103	0.043	**0.033**	**0.052**	**0.025**	**Yes**
	SA-Sparse	Gap from RTMPC	1	**0.057**	**0.121**	**0.081**	0.100	1	0.037	**0.034**	0.067	**0.090**	**0.036**	0.040	0.060	0.033	**Yes**

Table 3.2: Mean Absolute Error (MAE) in the experimental comparison of tracking a trajectory via MPC, RTMPC, and NN policies obtained with DAgger+DR (10 or 20 demonstrations) and SA-sparse (1 demonstration). The results highlight that 1) SA-sparse approximates the RTMPC expert in a comparable (or better) way than DAgger+DR, despite the decreased training efforts (1 demonstration vs 10 or 20), and it is robust to a variety of uncertainties, including wind (up to 10m/s), wind combined with a slung-load (0.2 Kg) and combined with an extra surface (0.8m^2) producing drag. In addition, it shows that RTMPC achieves tracking errors comparable to MPC when position constraints are not tight (Lemniscate, figure-8, max. vel. 3.0 m/s), while it modifies its trajectory to ensure sufficient distance from constraints when tight (Circle, max. vel. 3.5m/s). This leads to larger position errors than MPC in nominal conditions, but results in safe behaviors under disturbances, unlike MPC that fails to satisfy its position constraint in the run Slung Load+Wind. The Tracking MAE measures the distance of the robot from the reference, while the Gap from RTMPC measures the distance from the trajectory tracked by RTMPC (under the same type of disturbance). Results averaged across 3 Circles and 2 Lemniscates per agent.

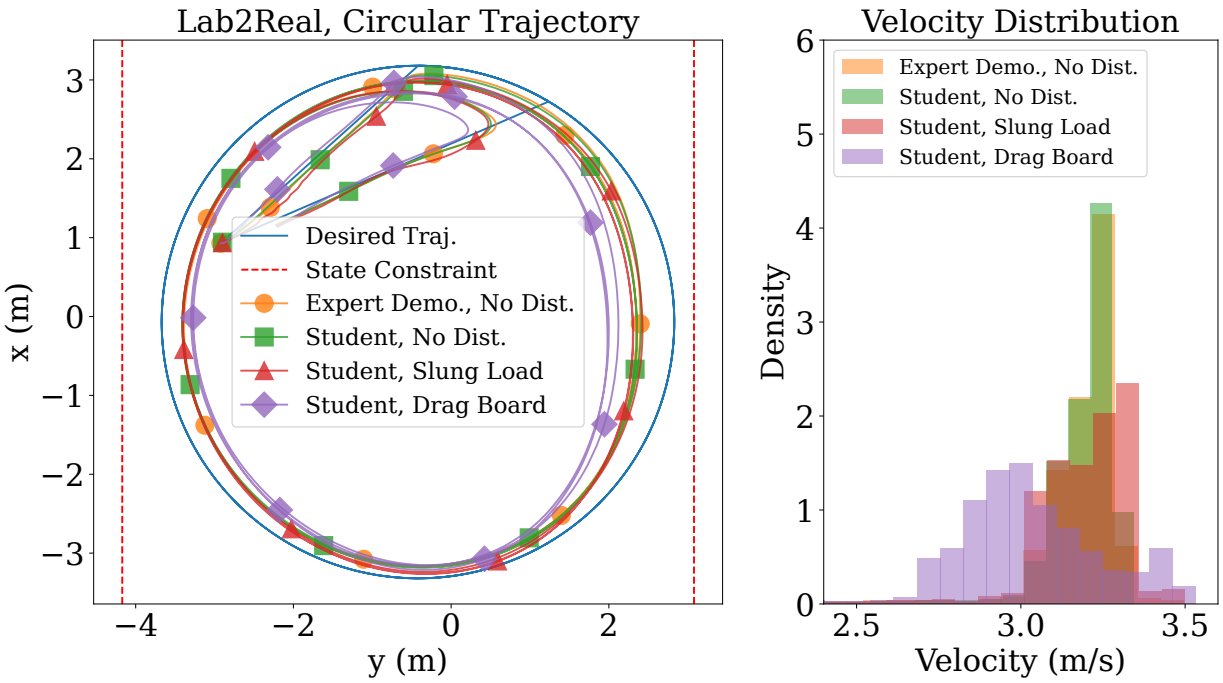


Figure 3.6: Example of *lab2real* transfer. One RTMPC demonstration (Expert Demo.) directly collected with the actual robot in a controlled (lab) environment is sufficient to train a policy that can reproduce the demonstration of the expert (Student, No Dist.) and that is robust to previously unseen disturbances from a slung load (Student, Slung Load, 0.25 kg) and a large surface producing drag (Student, Drag Board; the surface is shown in the video submission), despite the large velocity distribution shift. Note the safety distance from state constraints introduced by RTMPC. Policy runs onboard at 500 Hz.

Computation. Table 3.3 shows that the DNN policy is 230 times faster than the expert on the CPU of the onboard computer, an `Nvidia Jetson TX2`. Note that the computational cost of a traditional linear MPC is comparable to the one of its linear RTMPC variant [3], further highlighting the computational benefits of our approach when compared to traditional MPC.

3.5.4 Numerical and Hardware Evaluation for Learning and Generalizing to Multiple Trajectories

We evaluate the ability of the proposed approach to track multiple trajectories while generalizing to unseen ones. To do so, we define a training distribution of reference trajectories (circle, position step, eight-shape) and a distribution for these trajectory parameters (radius, velocity,

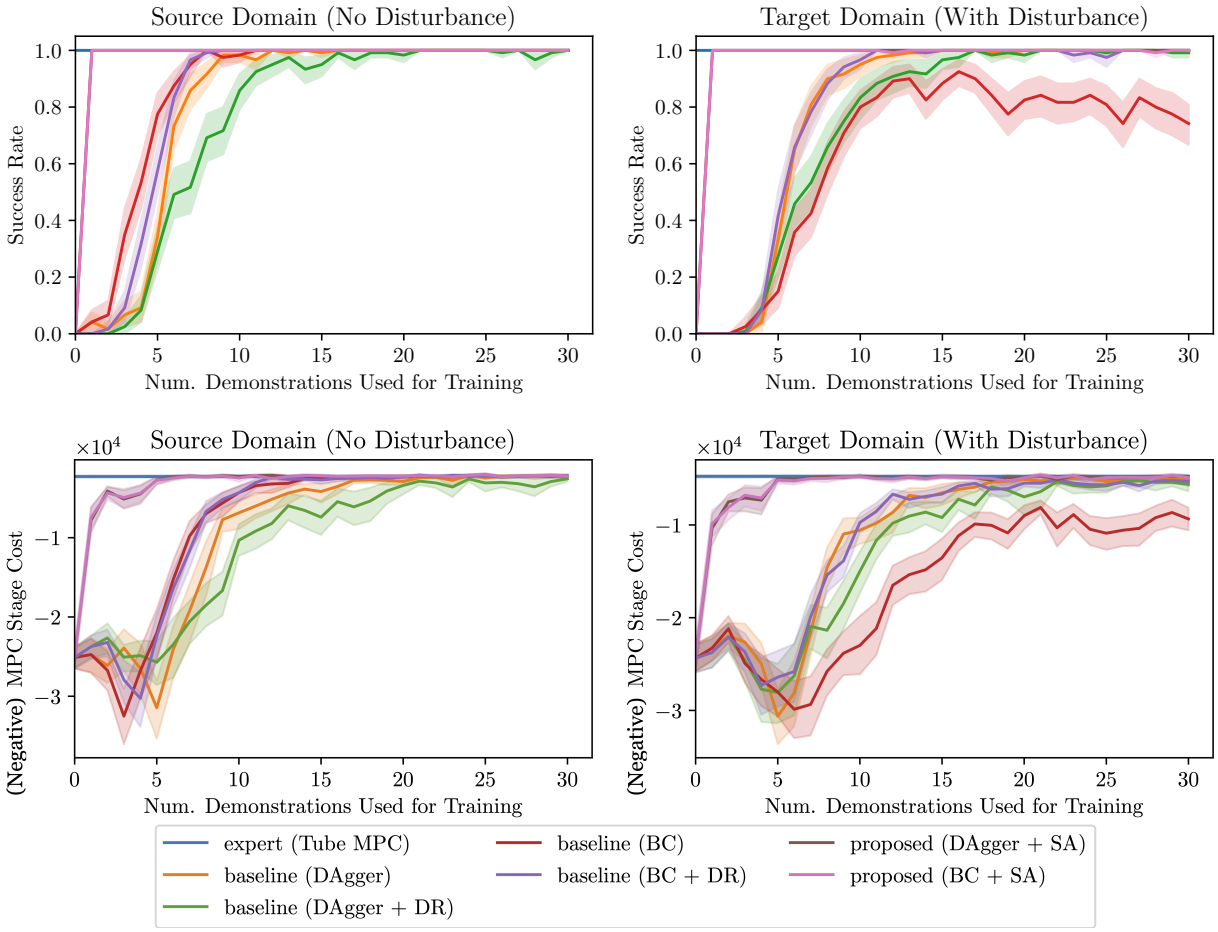


Figure 3.7: Robustness (*Success Rate*, top row) and performance (*negative MPC Stage Cost*, bottom row) of SA (with 95% confidence interval), as a function of the number of demonstrations used for training, for the task of learning to track previously unseen circular, eight-shaped and constant position reference trajectories with randomly sampled parameters, in the training domain (no disturbance, left column), and in the target domain, (wind-like disturbances, right column). The proposed RTMPC-guided SA-sparse strategy learns to track multiple trajectories and generalize to unseen ones requiring fewer demonstrations. The lines for SA-based methods overlap. Evaluation performed using 20 randomly sampled trajectories per demonstration, for 6 random seeds, with prediction horizon of $N = 20$.

Table 3.3: Comparison of the computation time (ms) required to generated a new action for the linear RTMPC (Expert (L-RTMPC)) and the DNN policy (Policy). **The DNN policy is 280 times faster than the optimization-based expert (onboard)**, and 25 times faster (offboard). The offboard computer for our numerical evaluation and training is an `Intel i9-10920` with two RTX 3090 GPUs. The onboard implementation (in C++, highly optimized for speed for both the policy and the expert, planning horizon $N = 30$ steps) uses an `NVIDIA Jetson TX2 CPU`. Note that our conference version [117] relied on Python/Pytorch (CPU), with average computation of 1.66ms.

			Time (ms)			
CPU	**Method**	**Setup**	**Mean**	**SD**	**Min**	**Max**
Offboard	Expert (L-RTMPC)	CVXPY [118]/OSQP [108]	4.28	0.39	4.21	16.66
	Policy	PyTorch	**0.17**	**0.00**	**0.17**	**0.22**
Onboard	Expert (L-RTMPC)	C++/CVXGEN [119]	8.4	1.4	4.5	15.9
	Policy	C++/Eigen	**0.03**	**0.01**	**0.02**	**0.24**

position). During training, we sample at random a desired, 7s long (70 steps) reference with randomly sampled parameters, collecting a demonstration and updating the proposed policy, while testing on a set of 20, 7s long trajectories randomly sampled from the defined distributions. We monitor the robustness and performance of the different methods, with force disturbances (from $\mathbb{W}_{\mathcal{T}_1}$) applied in the target domain. The results of the numerical evaluation, shown in Fig. 3.7, confirm that SA-sparse i) achieves robustness and performance comparable to the expert in a sample efficient way, requiring fewer than half the number of demonstrations needed for the baseline approaches; ii) simultaneously learns to generalize to multiple trajectories randomly sampled from the training distribution. Note that at convergence (from demonstration 20 to 30), DAgger+SA achieves the closest performance to the expert (2.7% *expert gap*), followed by BC+SA (3.0% *expert gap*). The hardware evaluation, performed with DAgger augmented via SA-sparse, is shown in Fig. 3.8. It confirms that the obtained policy is experimentally capable of tracking multiple trajectories under real-world disturbances/model errors.

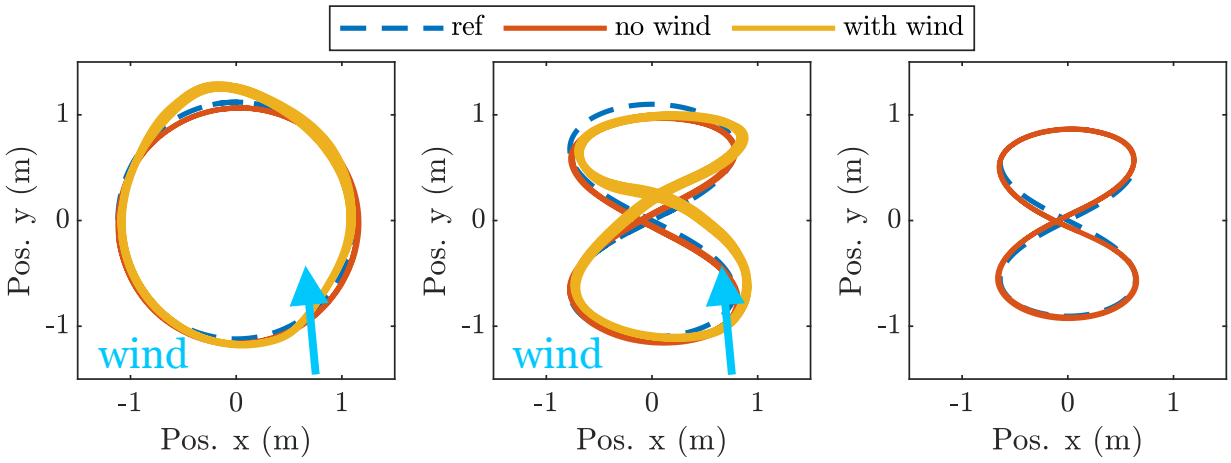


Figure 3.8: Examples of different, arbitrary chosen trajectories from the training distribution, tested in hardware experiments with and without strong wind-like disturbances produced by leaf blowers. The employed policy is trained with 10 demonstrations (when other baseline methods have not fully converged yet, see Fig. 3.7) using DAgger+SA (sparse). This highlights that sparse SA can learn multiple trajectories in a more sample-efficient way than other IL methods, retaining RTMPC's robustness and performance. The prediction horizon used is $N = 20$, and the DNN input size is adjusted accordingly.

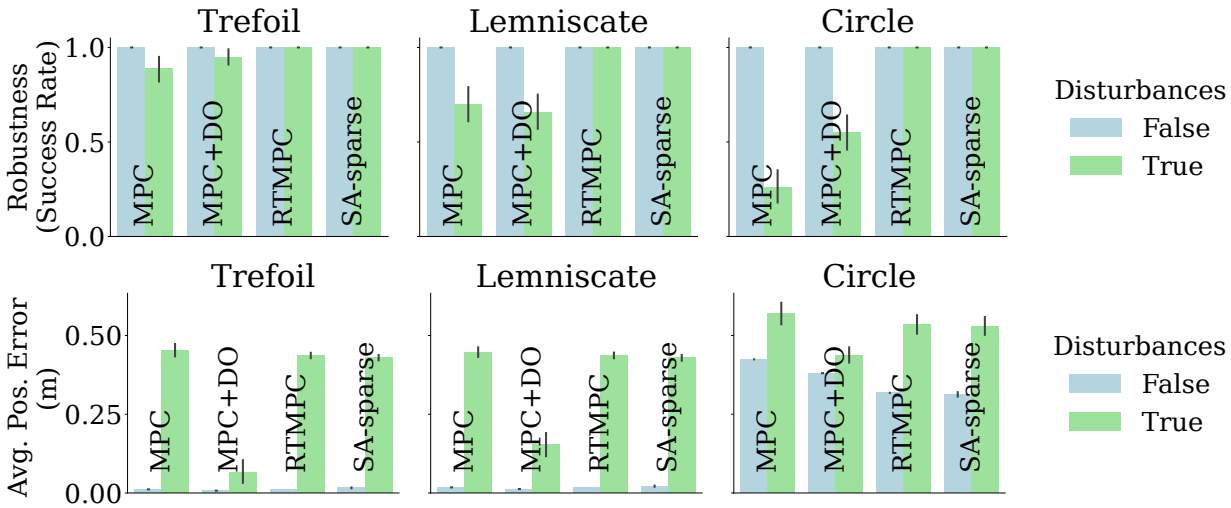


Figure 3.9: Comparison of performance and robustness of a traditional MPC, MPC combined with a disturbance observer (MPC+DO), robust tube MPC (RTMPC), and the policy learned from RTMPC (SA-sparse, one demonstration). The learned policy inherits the robustness properties of RTMPC, superior in robustness to all the other considered approaches, while achieving comparable or better performance than MPC. We consider two scenarios, with wind disturbances (True) or without (False).

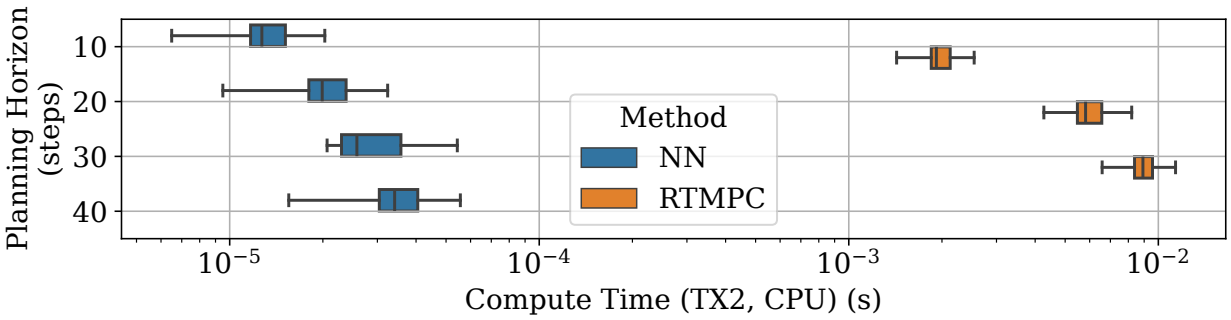


Figure 3.10: Computational cost of the neural network (NN) policy (two hidden layers, 32 neurons/layer, C++) and the onboard RTMPC expert (`CVXGEN`, C++) as a function of the length of the planning horizon. The NN provides more than 2-orders of magnitude computational improvement, freeing-up onboard computational resources, or enabling deployment of the controller on smaller and cheaper CPUs. Note that the computational cost of linear MPC (e.g., [7]) is comparable to the one of the considered RTMPC [3].

3.5.5 Extra Comparisons and Hyperparameter Study

Comparison with other Optimal Control approaches. SA-sparse (single demonstration) is compared in simulation with other optimal control approaches: 1. a linear trajectory tracking MPC, based on the one in [7] (denoted **MPC**), 2. the same MPC combined with a disturbance observer (Kalman filter) that estimates online additive force disturbances, updating the model used by the MPC [7] (denoted **MPC+DO**), and 3. RTMPC (expert). The considered trajectories include position, velocity, and actuation constraints, have velocities ranging in $2.0 - 3.5$m/s, and have 30s duration each. The results are presented in Fig. 3.9 and highlight that, while the adaptive variant of MPC (MPC+DO) achieves the lowest average tracking errors, RTMPC is superior in terms of robustness (constraint satisfaction), while the learned policy successfully inherits the robustness properties of RTMPC, with minimal trade-offs in terms of position errors.

Hyperparameter study. First, Fig. 3.10 studies the effects on onboard computation when varying the planning horizon, highlighting (1) two-orders-of-magnitude improvements in the onboard computation of the DNN policy compared to the RTMPC expert, and that (2) the computational benefits of the policy increase as the planning horizon increases. Second,

Table 3.4: Robustness and performance in simulation for policies learned using SA-sparse with a single demonstration, under different network architectures. The result highlights the low sensitivity of the approach to the choice of network architecture for the tasks considered. The evaluation is performed across the Trefoil, Lemniscate and Circle trajectories, with and without disturbances, using an expert with planning horizon $N = 20$ steps. The results are the average across 200 runs per trajectory (starting from 10 different initial states, repeated for 10 random seeds, with and without disturbances).

NN (Neurons/Layer)	Robustness Succ. Rate (%)		Performance Pos. Error (m)		Performance Expert Gap (%)		Computation (ms, TX2, CPU)	
	mean	std	mean	std	mean	std	mean	std
[32, 32]	99.9	3.3	0.29	0.19	5.4	5.3	0.021	0.01
[64, 32]	100.0	0.0	0.30	0.20	4.9	4.9	0.030	0.01
[64, 64]	99.9	2.4	0.30	0.20	4.8	5.0	0.033	0.01
[64, 64, 32]	99.6	6.2	0.29	0.19	5.3	6.2	0.039	0.01
[128, 128]	99.9	3.3	0.30	0.20	4.8	5.2	0.163	0.05

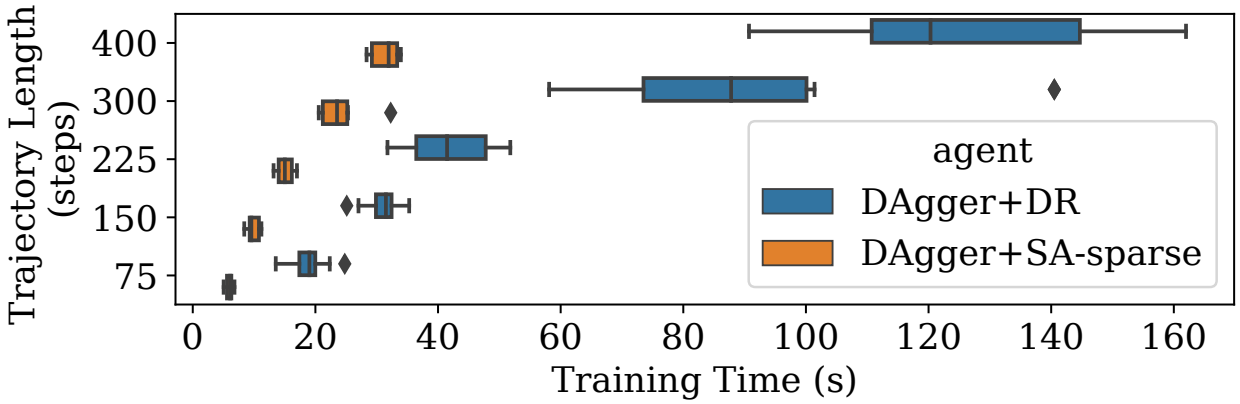


Figure 3.11: Time to generate a policy (data collection in simulation and training) compared to the complexity (number of time-steps in the trajectory) of the mission to be learned. The results highlight that SA-sparse enables learning of robust policies in significantly lower time than DAgger+DR, and it scales better as the length of the task increases. Note that one step corresponds to 0.1, therefore our method requires, i.e., less than 20s to learn to track a trajectory longer than 20s. The considered task consists in following an eight-shaped (Lemniscate) trajectory followed by a vertical circular trajectory, with velocities up to 3.5m/s, and the policy is considered robust if it achieves a success rate $> 95\%$ under wind-like disturbances (force up to 25% of the mass of the robot).

Table 3.4 studies robustness, performance, and onboard computation of the learned policy as a function of the size (layers, hidden neurons/layer) of DNN, highlighting that robustness and performance are minimally affected by these parameters. While onboard computation grows with the size of the network, it remains significantly lower than the onboard RTMPC expert. Last, Fig. 3.11 studies the time required to train a policy that achieves a success rate $> 95\%$ as a function of task complexity (length of the trajectory). This result highlights significant improvements in the scalability of our method when compared to the most robust baseline, DAgger+DR.

3.6 Summary

This Chapter has presented an IL strategy to efficiently train a robust DNN policy from MPC. Key ideas were to (a) leverage a Robust Tube variant of MPC, called RTMPC, to collect demonstrations using existing IL methods (DAgger, BC), and (b) augment the collected demonstrations with *efficiently-generated* extra state-and-actions samples *from the tube of the controller*, an approximation of the support of the state distribution that the learned policy will encounter when subject to uncertainties. Experimental results confirmed our numerical findings, showing trajectory tracking under wind-like disturbances on a multirotor via policy trained from a *single* demonstration, collected either in the real world or in simulation.

Chapter 4

Acrobatic Flights

4.1 Overview

In this Chapter, we design an IL and DA strategy, which is an extension of the one presented in Chapter 3, that enables robust and efficient policy learning from an MPC that employs nonlinear models of the form in Eq. (3.1). Different from Chapter 3, the focus here is on obtaining policies capable of reaching a desired goal state, as this will enable acrobatic maneuvers – the scenario considered in the evaluation of policies learned from this controller (Section 4.5). To accomplish this, first, we use a nonlinear version of RTMPC, based on [27], to collect demonstrations that account for the effects of uncertainties. This expert is summarized in Section 4.3.1. Second, we develop a computationally efficient tube-guided DA strategy leveraging the ancillary controller of the nonlinear RTMPC expert. Unfortunately, unlike in the linear RTMPC case, nonlinear RTMPC [27] uses Nonlinear Model Predictive Control (NMPC) as an ancillary controller. This limits the computational efficiency in DA, as the generation of extra state-action samples requires solving a large NLP associated with the ancillary NMPC (discussed in Section 4.3.2). We overcome this issue by presenting, in Section 4.3.3, a time-varying linear feedback law, approximation of the ancillary NMPC, that enables efficient generation of the extra data leveraging the sensitivity of the control input

to perturbations in the states visited during an initial demonstration collection procedure. Finally, in Section 4.3.4, we address the approximation errors introduced by the sensitivity-based DA by presenting strategies to mitigate the gap, in performance and robustness, between the learned policy and the RTMPC expert. We validate our procedures by demonstrating, in Section 4.5, the ability of the approach to generate a policy from a go-to-goal-state nonlinear RTMPC capable of performing acrobatic maneuvers, such as a 360 degrees flip. These maneuvers are performed under real-world uncertainties, using a policy obtained from only *two* demonstrations and in less than 100s of training time.

4.2 Problem Formulation

The problem formulation follows the one presented in Section 3.2, with the key difference that the expert in Eq. (3.2) now takes as input the current state $\mathbf{x}_t \in \mathbb{X}$ and a desired goal state $\mathbf{x}^{\text{des}} \in \mathbb{X}^{\text{des}}$ that needs to be reached. This results in an implicit deterministic control law (policy) that we denote $\pi_{\boldsymbol{\theta}^*} : \mathbb{X} \times \mathbb{X}^{\text{des}} \to \mathbb{U}$.

The student policy takes as input the current state, the desired goal state and the current timestep $t \in \mathbb{I}_{\geq 0}$, $\pi_{\boldsymbol{\theta}} : \mathbb{X} \times \mathbb{X}^{\text{des}} \times \mathbb{I}_{\geq 0} \to \mathbb{U}$. We note that the time-dependency was introduced to account for the fact that the solution of the ancillary NMPC used for DA depends on the time of the maneuver.

4.3 Approach

4.3.1 Nonlinear RTMPC Expert Formulation

Nonlinear RTMPC [27] ensures state and actuation constraint satisfaction while controlling a nonlinear, uncertain system of the form in Eq. (3.1). This controller operates by solving two Optimal Control Problems (OCPs), one to compute a nominal safe plan, and one to track the safe plan (ancillary NMPC).

Nominal Safe Planner. The first OCP, given an $N+1$-steps planning horizon, generates nominal safe state and action open-loop plans $\mathbf{Z}_{t_0} = \{\mathbf{z}_{0|t_0}, \dots, \mathbf{z}_{N|t_0}\}, \mathbf{V}_{t_0} = \{\mathbf{v}_{0|t_0}, \dots, \mathbf{v}_{N-1|t_0}\}$. The plans are open-loop because they are generated only at time t_0 when the desired state and action pair $\mathbf{X}^{\text{des}}_{t_0} = \{\mathbf{x}^e_{t_0}, \mathbf{u}^e_{t_0}\}$, equilibrium pair for the nominal system obtained from $\mathbf{x}^{\text{des}}$, changes. The nominal safe plan is obtained from:

$$\begin{aligned} \mathbf{V}^*_{t_0}, \mathbf{Z}^*_{t_0} = \underset{\mathbf{V}_{t_0}, \mathbf{Z}_{t_0}}{\text{argmin}}\ & J_{\text{RTNMPC}}(\mathbf{Z}_{t_0}, \mathbf{V}_{t_0}, \mathbf{X}^{\text{des}}_{t_0}) \\ \text{subject to}\ & \mathbf{z}_{i+1|t_0} = f(\mathbf{z}_{i|t_0}, \mathbf{v}_{i|t_0}), \\ & \mathbf{z}_{i|t_0} \in \bar{\mathbb{Z}},\ \ \mathbf{v}_{i|t_0} \in \bar{\mathbb{V}}, \\ & \mathbf{z}_{0|t_0} = \mathbf{x}_{t_0},\ \ \mathbf{z}_{N|t_0} = \mathbf{x}^e_{t_0}, \forall i = 0, \dots, N-1. \end{aligned} \tag{4.1}$$

$J_{\text{RTNMPC}} = \sum_{i=0}^{N-1} \|\mathbf{z}_{i|t_0} - \mathbf{x}^e_{t_0}\|^2_{\mathbf{Q}_z} + \|\mathbf{v}_{i|t_0} - \mathbf{u}^e_{t_0}\|^2_{\mathbf{R}_v}$, where $\mathbf{Q}_z$, $\mathbf{R}_v$ are positive definite. A key idea in this approach involves imposing modified state and actuation constraints $\bar{\mathbb{Z}} \subset \mathbb{X}$ and $\bar{\mathbb{V}} \subset \mathbb{U}$ so that the generated nominal safe plan is at a specific distance from state and actuation constraints. To be more precise, similar to the linear RTMPC case (Eq. (3.10)), the given state constraints $\mathbb{X}$ and actuation constraints $\mathbb{U}$ are tightened (made more conservative) by an amount that accounts for the spread of trajectories induced by the ancillary controller when the system is subject to uncertainties, obtaining $\bar{\mathbb{Z}} \subset \mathbb{X}$ and $\bar{\mathbb{V}} \subset \mathbb{U}$. Such spread of trajectories corresponds to state and action tubes $\mathbb{T}^{\text{state}} \subset \mathbb{R}^{n_x}, \mathbb{T}^{\text{action}} \subset \mathbb{R}^{n_u}$ that contain the current nominal safe state and action trajectories $\mathbf{z}^*_{t|t_0}$, $\mathbf{v}^*_{t|t_0}$. Different from the linear case, however, analytically computing the tightened constraints and the tubes is challenging. Fortunately, as highlighted in [27, Section 7], accurately computing these sets is not needed, and an outer approximation is sufficient. This approximation can be obtained via Monte-Carlo simulations [27] of the system under disturbances, or learned [111]; the procedure employed in our work is tailored to our application domain, and is described in details in Section 4.4.1 Last we note that, as in [27], Eq. (4.1) is assumed to be feasible.

Ancillary NMPC. The second OCP corresponds to a trajectory tracking NMPC, that acts

as an ancillary controller, to maintain the state of the uncertain system close to the reference generated by Eq. (4.1). The OCP is:

$$\begin{aligned} \bar{\mathbf{U}}_t^*, \bar{\mathbf{X}}_t^* = \underset{\bar{\mathbf{U}}_t, \bar{\mathbf{X}}_t}{\operatorname{argmin}} & \|\mathbf{e}_{N|t}\|_{\mathbf{P}_x}^2 + \sum_{i=0}^{N-1} \|\mathbf{e}_{i|t}\|_{\mathbf{Q}_x}^2 + \|\bar{\mathbf{u}}_{i|t} - \mathbf{v}_{i+t|t_0}^*\|_{\mathbf{R}_u}^2 \\ \text{subject to } & \bar{\mathbf{x}}_{i+1|t} = f(\bar{\mathbf{x}}_{i|t}, \bar{\mathbf{u}}_{i|t}) \\ & \bar{\mathbf{x}}_{0|t} = \mathbf{x}_t, \bar{\mathbf{u}}_{i|t} \in \mathbb{U}, \forall i = 0, \dots, N-1 \end{aligned} \tag{4.2}$$

where $\mathbf{e}_{i|t} = \bar{\mathbf{x}}_{i|t} - \mathbf{z}_{i+t-t_0|t_0}^*$ is the state tracking error. The positive definite matrices $\mathbf{Q}_x$ and $\mathbf{R}_u$ are tuning parameters and can differ from the ones in Eq. (4.1), while $\mathbf{P}_x$ defines a terminal cost. Note that the terminal cost can be set using the method in [27], or using the solution for the infinite horizon Riccati equation for the linearized system associated with the state at the end of the planning horizon. However, owing to the fact that the expert can use a sufficiently long planning horizon without affecting onboard computation of the learned policy, in our experiments we set the terminal cost to $\mathbf{Q}_x$, additionally demonstrating that our approach introduces opportunities to simplify control design. Eq. (4.2) is solved at each timestep using the current state $\mathbf{x}_t$, while the action applied to the robot is $\mathbf{u}_t = \bar{\mathbf{u}}_{0|t}^*$. We note that the ancillary NMPC can have different tuning parameters than Eq. (4.1), including the discretization time of the dynamics and the prediction horizon N, providing additional degrees of freedom to shape the response of the system under uncertainties.

A key result (presented in [27, Section 5]) of the employed nonlinear RTMPC [27] is that the ancillary NMPC in Eq. (4.2) maintains the trajectories of the uncertain system in Eq. (3.1) inside state and action tubes $\mathbb{T}^{\text{state}}, \mathbb{T}^{\text{action}}$ that contain the current nominal safe state and action trajectories $\mathbf{z}_{t|t_0}^*$, $\mathbf{v}_{t|t_0}^*$ from the OCP in Eq. (4.1). The state and action tubes are used to obtain the tightened state and actuation constraints $\bar{\mathbb{Z}}$, $\bar{\mathbb{V}}$, ensuring constraint satisfaction.

4.3.2 Solving the Ancillary NMPC

A large portion of the computational cost of deploying or collecting demonstrations from nonlinear RTMPC comes from the need to solve the OCP of the ancillary NMPC (Eq. (4.2)) at each timestep. In contrast, the OCP of the nominal safe plan (Eq. (4.1)) can be solved once per task (e.g., whenever the desired goal state $\mathbf{X}_{t_0}^{\text{des}}$ changes).

A state-of-the-art method to solve the optimization in Eq. (4.2) is sequential quadratic program (SQP), i.e., by repeatedly: i) linearizing the NLP around a given linearization point; ii) generating and solving a corresponding QP, obtaining a refined linearization point for the next SQP iteration. While capable of producing high-quality solutions, SQP methods incur large computational requirements due to the need of performing computationally-expensive system linearization and solving the associated QP one or more times per timestep.

4.3.3 Computationally-Efficient Data Augmentation using the Parametric Sensitivities

The tube $\mathbb{T}^{\text{state}}$ induced by the ancillary controller in Eq. (4.2) identifies relevant regions of the state space for DA, as it approximates the support of the state distribution under uncertainties, as discussed in Section 3.3.2. However, generating the corresponding extra action samples using Eq. (4.2) can be very computationally inefficient, as it requires solving the associated SQP for every extra state sample, making DA computationally impractical, and defeating our initial objective of designing *computationally efficient* DA strategies.

In this work, Sampling Augmentation (SA), is extended to efficiently learn policies from nonlinear RTMPC by employing a time-varying, linear approximation of the ancillary NMPC – enabling efficient generation of extra state-action samples. Specifically, we observe that Eq. (4.2) solves the implicit feedback law:

$$\mathbf{u}_t = \bar{\mathbf{u}}_{0|t}^*(\boldsymbol{\chi}_t) \coloneqq \kappa(\boldsymbol{\chi}_t),\ \boldsymbol{\chi}_t \coloneqq \{\mathbf{x}_t, t; \mathbf{V}_{t_0}^*, \mathbf{Z}_{t_0}^*\} \tag{4.3}$$

where the current inputs are denoted $\boldsymbol{\chi}_t$. Then, for each timestep of the trajectory collected during a demonstration in the source environment $\mathcal{S}$, with current ancillary NMPC input $\tilde{\boldsymbol{\chi}}_t = \{\tilde{\mathbf{x}}_t, \tilde{t}; \mathbf{V}^*_{t_0}, \mathbf{Z}^*_{t_0}\}$, we generate a local linear approximation of Eq. (4.3) by computing the first-order sensitivity of $\mathbf{u}_t$ to the initial state $\mathbf{x}_t$:

$$\mathbf{K}_{\tilde{\boldsymbol{\chi}}_t} \coloneqq \left.\frac{\partial \bar{\mathbf{u}}^*_{0|t}}{\partial \mathbf{x}_t}\right|_{\boldsymbol{\chi}_t = \tilde{\boldsymbol{\chi}}_t} = \left[\left.\frac{\partial \bar{\mathbf{u}}^*_{0|t}}{\partial [\mathbf{x}_t]_1}\right|_{\tilde{\boldsymbol{\chi}}_t}, \ldots, \left.\frac{\partial \bar{\mathbf{u}}^*_{0|t}}{\partial [\mathbf{x}_t]_{n_x}}\right|_{\tilde{\boldsymbol{\chi}}_t} \right]. \tag{4.4}$$

The sensitivity matrix $\mathbf{K}_{\tilde{\boldsymbol{\chi}}_t} \in \mathbb{R}^{n_u \times n_x}$, enables us to compute extra actions $\mathbf{u}^+_{t,j}$ from states inside the tube $\mathbf{x}^+_{t,j} \in \mathbb{T}^{\text{state}}$, with $j = 1, \ldots, N_s$, sampled from the tube:

$$\mathbf{u}^+_{t,j} = \bar{\mathbf{u}}^*_{0|t} + \mathbf{K}_{\tilde{\boldsymbol{\chi}}_t}(\mathbf{x}^+_{t,j} - \bar{\mathbf{x}}^*_{0|t}) \coloneqq \hat{\kappa}(\mathbf{x}^+_{t,j}, \tilde{\boldsymbol{\chi}}_t). \tag{4.5}$$

The DA procedure enabled by this approximation is computationally-efficient, as we do not need to solve an SQP for each extra state-action sample $(\mathbf{x}^+_{t,j}, \mathbf{u}^+_{t,j})$ generated for DA, and we only need to compute, once per timestep, the sensitivity matrix $\mathbf{K}_{\tilde{\boldsymbol{\chi}}_t}$. Note that the linearization points of Eq. (4.4) are based on the trajectory $\boldsymbol{\xi}$ executed during demonstration collection in the source environment $\mathcal{S}$. We remark, additionally, that the actions computed when collecting demonstrations are obtained by solving the entire SQP, and the sensitivity-based approximation is used only for DA.

Sensitivity Matrix Computation. As described in [120, §8.6], an expression to compute the sensitivity matrix in Eq. (4.4) (also called *tangential predictor*) can be obtained by re-writing the NLP in Eq. (4.2) in a parametric form $\mathfrak{p}([\mathbf{x}_t]_i)$, highlighting the dependency on scalar parameter representing the i-th component of the initial state $\mathbf{x}_t$ (part of $\boldsymbol{\chi}_t$). The parametric NLP $\mathfrak{p}_{\mathcal{X}_t}([\mathbf{x}_t]_i)$ is:

$$\begin{aligned} &\min_{\mathbf{y}} F_{\boldsymbol{\chi}_t}(\mathbf{y}) \\ &\text{subject to } G_{\boldsymbol{\chi}_t}([\mathbf{x}_t]_i, \mathbf{y}) = \mathbf{0} \\ &H(\mathbf{y}) \leq \mathbf{0}, \end{aligned} \tag{4.6}$$

where $\mathbf{y} \in \mathbb{R}^{n_\mathbf{y}}$ corresponds to the optimization variables in Eq. (4.2), and $F_{\boldsymbol{\chi}_t}(\cdot), G_{\boldsymbol{\chi}_t}(\cdot), H(\cdot)$ are, respectively, the objective function, equality, and inequality constraints in Eq. (4.2), given the current state and reference trajectory in $\boldsymbol{\chi}_t$. Additionally, we denote the solution of Eq. (4.6) at $\tilde{\boldsymbol{\chi}}_t$ (computed during the collected demonstration) as $(\tilde{\mathbf{y}}^*, \tilde{\boldsymbol{\lambda}}^*, \tilde{\boldsymbol{\mu}}^*)$, where $\tilde{\boldsymbol{\lambda}}^*, \tilde{\boldsymbol{\mu}}^*$ are, respectively, the Lagrange multipliers for the equality and inequality constraints at the solution found. Then, each i-th column of the sensitivity matrix (Eq. (4.4)) can be computed by solving the following QP ([120, Th. 8.16], and [121, Th. 3.4 and Remark 4]), denoted $\mathfrak{p}_{\mathcal{X}_t,L}([\mathbf{x}_t]_i)$:

$$\begin{aligned} \min_{\mathbf{y}} \; & F_{\boldsymbol{\chi}_t,L}(\mathbf{y};\tilde{\mathbf{y}}^*) + \frac{1}{2}(\mathbf{y}-\tilde{\mathbf{y}}^*)^\top \nabla^2_{\mathbf{y}} \mathscr{L}(\tilde{\mathbf{y}}^*, \tilde{\boldsymbol{\lambda}}^*, \tilde{\boldsymbol{\mu}}^*)(\mathbf{y}-\tilde{\mathbf{y}}^*) \\ \text{s.t.} \quad & G_{\boldsymbol{\chi}_t,L}([\mathbf{x}_t]_i, \mathbf{y};\tilde{\mathbf{y}}^*) = \mathbf{0} \\ & H_L(\mathbf{y};\tilde{\mathbf{y}}^*) \leq \mathbf{0} \end{aligned} \tag{4.7}$$

where $F_{\boldsymbol{\chi}_t,L}(\cdot;\tilde{\mathbf{y}}^*)$, $G_{\boldsymbol{\chi}_t,L}(\cdot;\tilde{\mathbf{y}}^*)$, $H_L(\cdot;\tilde{\mathbf{y}}^*)$ denote the respective functions in Eq. (4.6) linearized at the solution found. $\nabla^2_{\mathbf{y}}\mathscr{L}$ denotes the Hessian of the Lagrangian associated with Eq. (4.6), while the parameter is set to zero ($[\mathbf{x}_t]_i = 0$). The i-th column of the sensitivity matrix can be extracted from the entries of $\mathbf{y}^*$, solution of Eq. (4.7), at the position corresponding to $\bar{\mathbf{u}}_{0|t}$. We highlight that Eq. (4.7) can be computed efficiently, as it leverages the latest internal linearization of the Karush–Kuhn–Tucker (KKT) conditions performed in the SQP employed to solve Eq. (4.2), and therefore it does not require to re-execute the computationally expensive system linearization routines that are carried out at each SQP iteration. We note that this local approximation exists when the assumptions in [120, Th. 8.15] are satisfied, i.e., that the solution $(\tilde{\mathbf{y}}^*, \tilde{\boldsymbol{\lambda}}^*, \tilde{\boldsymbol{\mu}}^*)$ found during demonstration collection is a strongly regular KKT point, and satisfies strict complementary conditions. Last, extra samples are generated using Eq. (4.5) under the assumption that the set of active inequality constraints (i.e., the index set $p \in \{1, \ldots, n_H\}$ such that $[H(\tilde{\boldsymbol{y}}^*)]_p = 0$) does not change.

Generalized Tangential Predictor. A strategy that applies to the cases where strict

complementary conditions do not hold, or where the extra state samples cause a change in the active set of constraints, is based on the *generalized tangential predictor* [120, §8.9.1]. This predictor can be obtained by solving the QP in Eq. (4.7) with the set of equality constraints modified to be $G_{\boldsymbol{\chi}_t,L}(\mathbf{x}_{t,j}^+, \mathbf{y}; \tilde{\mathbf{y}}) = \mathbf{0}$ [120, Eq. 8.60]. Although this approach requires solving a QP to compute the action $\mathbf{u}_{t,j}^+$ corresponding to each state $\mathbf{x}_{t,j}^+$ sampled from the tube, it does not require re-generating the computationally expensive linearization performed at each SQP iteration (and other performance optimization routines, such as condensing [120]) nor solving the entire SQP for multiple iterations – resulting in a much more computationally-efficient procedure than solving the entire SQP *ex-novo* for every extra state-action sample. We remark that the linearization point in Eq. (4.7) is updated at every timestep when a full SQP is solved for demonstration-collection.

4.3.4 Robustness and Performance Under Approximate Samples

While the proposed sensitivity-based DA strategy enables the efficient generation of extra state-action samples, it introduces approximation errors that may affect the performance and robustness of the learned policy. Here, we discuss strategies to account for these errors, reducing the gaps between the nonlinear RTMPC expert and the learned policy in terms of robustness and performance.

Robustness. A key property of RTMPC is the ability to explicitly account for uncertainties, including the ones introduced by the proposed sensitivity-based DA framework, by further tightening state and actuation constraints for the nominal safe plan (Eq. (4.1)). The general nonlinear formulation of the dynamics in Eq. (3.1), however, makes it challenging to compute an *exact* additional tightening bound for state and actuation constraints. A possible avenue to establish a tightening procedure for the actuation constraints is to observe that the linear approximation of Eq. (4.3) introduces an error upper bounded by ([120, Th. 8.16]):

$$\|\kappa(\boldsymbol{\chi}_t) - \hat{\kappa}(\mathbf{x}_{t,j}^+, \boldsymbol{\chi}_t)\| \leq D\|\mathbf{x}_{t,j}^+ - \mathbf{x}_t\|^2 \tag{4.8}$$

where D may be obtained by considering the Lipschitz constant of the controller (e.g. [25]). However, estimating this constant may be difficult, or computationally expensive, for large-dimensional systems, as is the case herein. An alternative is to update the tubes as was done in Section 4.3.1, e.g., by employing Monte-Carlo simulations of the closed-loop system, starting from an initial (possibly conservative) tightening guess and by iteratively adjusting the cross-section (size) of the tube, or by directly learning the tubes from simulations or previous (conservative) real-world deployments [111]. These procedures, when performed using the learned policy, are particularly appealing in our context, as our efficient policy learning methodologies enable rapid training/updates of the learned policy, and the computational efficiency of the policy enables rapid simulations.

Performance Improvements via Fine-Tuning. In the context of learning policies from nonlinear RTMPC, we include in SA an (optional) fine tuning-step. This fine-tuning step consists in training the policy with additional demonstrations, without DA, therefore avoiding introducing further approximate samples, and having discarded the extra data used to train the policy after an initial demonstration. Therefore, tube-guided DA is treated as a methodology to efficiently generate an initial guess of the policy parameters.

Algorithm Summary. The SA procedure for nonlinear RTMPC with the fine-tuning step is summarized in Algorithm 2. It consists of the following:

1) Pre-compute the safe plan from the expert (line 2).

2) Collect a single ($M = 1$) task demonstration $\boldsymbol{\xi}$ that tracks the safe plan using the ancillary NMPC (line 6-14), while additionally storing the variables of the QP in Eq. (4.7).;

3) Perform DA using the parametric sensitivity (Section 4.3.3, line 10-12, shown for the case where no active change of constraints occurs and strict complementary conditions hold, else use Eq. (4.7)) and train the policy, obtaining the parameters $\hat{\boldsymbol{\theta}}_1$ (line 15);

4) Optional *fine-tuning* step (line 16):

   i) Discard the collected data so far, including the data generated by the DA (line 17);

Input: $f(\cdot), \mathbb{X}, \mathbb{U}, \mathbf{Q}_x, \mathbf{R}_u, \mathbb{W}_{\mathcal{T}}, \beta, \mathcal{S}, \mathbf{X}_{t_0}^{\text{des}}$
Output: Trained policy $\pi_{\hat{\boldsymbol{\theta}}_{M+L}}$

1: $\pi_{\boldsymbol{\theta}^*}, \mathbb{T}^{\text{states}} \leftarrow$ DesingNRtmpc$(f(\cdot), \mathbb{X}, \mathbb{U}, \mathbf{Q}_x, \mathbf{R}_u, \mathbb{W}_{\mathcal{T}}, \mathbf{X}_{t_0}^{\text{des}})$
2: $\mathbf{Z}_{t_0}^*, \mathbf{V}_{t_0}^* \leftarrow$ GetNominalSafePlan$(\pi_{\boldsymbol{\theta}^*})$ // Eq. (4.1)
3: $\kappa \leftarrow$ GetAncillaryNmpc$(\pi_{\boldsymbol{\theta}^*})$ // Eq. (4.2), Eq. (4.3)
4: **for** $i = 1$ **to** M **do**
5: **for** $t = 0$ **to** T **do**
6: $\mathcal{X}_t \leftarrow (\mathbf{x}_t, t; \mathbf{V}_{t_0}^*, \mathbf{Z}_{t_0}^*)$ // Current operating point
7: $\mathbf{u}_t^{\text{N-RTMPC}}, \mathfrak{p}_{\mathcal{X}_t, L} \leftarrow \kappa(\mathcal{X}_t)$ // Eq. (4.2), save QP Eq. (4.7)
8: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_t, \mathbf{X}^{\text{des}}, t, \mathbf{u}_t^{\text{N-RTMPC}})\}$
9: $\mathbf{K}_{\mathcal{X}_t} \leftarrow$ Sensitivity$(\mathfrak{p}_{\mathcal{X}_t, L})$ // Eq. (4.4)
10: **for** $j = 1$ **to** N_{s} **do**
11: $\mathbf{u}_{t,j}^+ = \bar{\mathbf{u}}_t^* + \mathbf{K}_{\mathcal{X}}(\mathbf{x}_{t,j}^+ - \bar{\mathbf{x}}_t^*)$, $\mathbf{x}_{t,j}^+ \in \bar{\mathbf{x}}_t^* \oplus \mathbb{T}^{\text{states}}$
12: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}_{t,j}^+, \mathbf{X}_t^{\text{des}}, \mathbf{u}_{t,j}^+)\}$
13: $\mathbf{u}_t \leftarrow \beta_i \mathbf{u}_t^{\text{N-RTMPC}} + (1 - \beta_i)\, \pi_{\hat{\boldsymbol{\theta}}_{i-1}}(\mathbf{x}_t, \mathbf{X}_t^{\text{des}})$ // DAgger/BC
14: $\mathbf{x}_{t+1} \leftarrow$ StepSystem$(\mathbf{u}_t, \mathbf{x}_t, \mathcal{S})$
15: $\pi_{\hat{\boldsymbol{\theta}}_i} \leftarrow$ UpdatePolicy$(\mathcal{D}, \hat{\theta}_{i-1})$
16: **if** FineTuning **then**
17: $\mathcal{D} \leftarrow \emptyset$
18: **for** $l = 1$ **to** L **do**
19: $\boldsymbol{\xi} = \{(\mathbf{x}_t, \mathbf{X}_{t_0}^{\text{des}}, \mathbf{u}_t^{\text{N-RTMPC}}))\}_{t=0}^{T-1}$
 $\leftarrow$ CollectDemo$(\kappa, \mathbf{Z}_{t_0}^*, \mathbf{V}_{t_0}^*, \pi_{\theta_{M+l-1}}, \beta_i, \mathcal{S})$// DAgger/BC
20: $\mathcal{D} \leftarrow \mathcal{D} \cup \{\boldsymbol{\xi}\}$
21: $\pi_{\hat{\boldsymbol{\theta}}_{M+l}} \leftarrow$ UpdatePolicy$(\mathcal{D}, \hat{\theta}_{M+l-1})$

Algorithm 2: Sampling Augmentation for efficient learning from Nonlinear RTMPC

ii) Collect new demonstrations using DAgger [21] and the pre-trained policy, or BC, line 21, and re-train the pre-trained policy (with parameters $\hat{\boldsymbol{\theta}}_1$) after every newly collected demonstration.

4.4 Application to Agile Flight

4.4.1 Nonlinear RTMPC for Acrobatic Maneuvers

In this Section we design a nonlinear RTMPC expert capable of performing a 360° flip in *near-minimum time* - a maneuver that demands exploitation of the full *nonlinear* dynamics of the multirotor, and that requires large and careful actuation usage; we rely on the IL procedure described in Section 4.1.

Ancillary NMPC. We start by designing the ancillary NMPC (Eq. (4.2)). The selected nominal model is the same used in the high-performance trajectory tracking NMPC for multirotors [23], [122]:

$$\begin{aligned} {}_{\mathrm{W}}\dot{\boldsymbol{p}} &= {}_{\mathrm{W}}\boldsymbol{v} \\ {}_{\mathrm{W}}\dot{\boldsymbol{v}} &= m^{-1}(\boldsymbol{R}_{\mathrm{WB}}\,{}_{\mathrm{B}}\boldsymbol{t}_{\mathrm{cmd}} + {}_{\mathrm{W}}\boldsymbol{f}_{\mathrm{drag}}) - {}_{\mathrm{W}}\boldsymbol{g} \\ \dot{\boldsymbol{q}}_{\mathrm{WB}} &= \frac{1}{2}\boldsymbol{\Omega}({}_{\mathrm{B}}\boldsymbol{\omega}_{\mathrm{cmd}})\boldsymbol{q}_{\mathrm{WB}}, \end{aligned} \tag{4.9}$$

where the rotational dynamics (Eq. (3.15d)) have been neglected, assuming that the cascaded attitude controller enables fast tracking of the desired angular velocity setpoint ${}_{\mathrm{B}}\boldsymbol{\omega}_{\mathrm{cmd}}$. The controller uses the state and control input:

$$\bar{\mathbf{x}} = [{}_{\mathrm{W}}\boldsymbol{p}^{\top}, {}_{\mathrm{W}}\boldsymbol{v}^{\top}, \boldsymbol{q}_{\mathrm{WB}}^{\top}]^{\top}, \quad \bar{\mathbf{u}} = [t_{\mathrm{cmd}}, {}_{\mathrm{B}}\boldsymbol{\omega}_{\mathrm{cmd}}^{\top}]^{\top}. \tag{4.10}$$

The feed-forward angular acceleration for the attitude controller ${}_{\mathrm{B}}\dot{\boldsymbol{\omega}}_{\mathrm{cmd}}$ is obtained via numerical differentiation. We do not explicitly generate an attitude setpoint (we set $\boldsymbol{R}_{\mathrm{WB}}^{\mathrm{des}} = \boldsymbol{R}_{\mathrm{WB}}$), so that Eq. (4.9) acts as a proportional body-rates controller with feed-forward accelerations.

Near-Minimum Time Safe Plan Generation. To compute safe nominal plans for acrobatic maneuvers (by solving the OCP in Eq. (4.1)), we employ an extended version of the full nonlinear dynamic model in Section 3.4.1. More specifically, we solve the OCP in Eq. (4.1) by using the following state $\tilde{\mathbf{z}} \in \tilde{\bar{\mathbb{Z}}}$ and control inputs $\tilde{\mathbf{v}} \in \tilde{\bar{\mathbb{V}}}$:

$$\tilde{\mathbf{z}} = [{}_{\mathrm{W}}\boldsymbol{p}^{\top}, {}_{\mathrm{W}}\boldsymbol{v}^{\top}, \boldsymbol{q}_{\mathrm{WB}}^{\top}, {}_{\mathrm{B}}\boldsymbol{\omega}^{\top}, {}_{\mathrm{B}}\boldsymbol{f}_{\mathrm{prop}}^{\top}]^{\top} \quad \tilde{\mathbf{v}} = {}_{\mathrm{B}}\dot{\boldsymbol{f}}_{\mathrm{prop}}, \tag{4.11}$$

where the state has been extended to include the thrust produced by each propeller $\boldsymbol{f}^{\mathrm{prop}}$ to ensure continuity in the reference thrust, accounting for the unmodeled actuators' dynamics. As for the linear case, uncertainties are modeled by ${}_{\mathrm{W}}\boldsymbol{f}_{\mathrm{ext}} \in \mathbb{W}$. The cost function captures

the near-minimum time objective:

$$\tilde{J}_{\text{RTNMPC}} = T_f + \alpha_1 \boldsymbol{v}^\top \boldsymbol{v} + \alpha_2 \boldsymbol{f}_{\text{prop}}{}^\top \boldsymbol{f}_{\text{prop}} + \alpha_3 \tilde{\mathbf{v}}^\top \tilde{\mathbf{v}} \tag{4.12}$$

where T_f is the total time of the maneuver, while the remaining terms act as a regularizer for the optimizer, with $\alpha_i \ll T_f$ (i.e., $\alpha_i \approx 10^{-2},\ \forall\ i$).

We note that $\tilde{J}_{\text{RTNMPC}}$ contains a non-quadratic term, therefore differing from the quadratic cost employed in the safe nominal planner in [27] (our Eq. (4.1)); such cost function was chosen to automate the selection of the prediction horizon N for the safe nominal plan. Our evaluation will demonstrate that the ancillary NMPC maintains the system within a tube from the generated reference, further highlighting the flexibility of the framework.

Additionally, we note that state and control input (Eq. (4.11)) have been extended compared to the ones (Eq. (4.10)) selected for the ancillary NMPC, as emphasized by our notation $\tilde{\cdot}$. For this reason, the optimal safe nominal plan $\tilde{\mathbf{Z}}_t^*$, $\tilde{\mathbf{V}}_t^*$ found using the extended state needs to be mapped to the reference trajectory for the ancillary NMPC, $\mathbf{Z}_t^*$ $\mathbf{V}_t^*$. This is done by simply selecting position, velocity and attitude from $\tilde{\mathbf{Z}}_t^*$ to obtain $\mathbf{Z}_t^*$. The thrust setpoint t_{cmd} in $\mathbf{V}_t^*$ is computed via $\boldsymbol{\mathcal{A}}$ in Eq. (3.17) from $\boldsymbol{f}_{\text{prop}}$ in $\tilde{\mathbf{Z}}_t^*$, while the angular velocity setpoint $\boldsymbol{\omega}_{\text{cmd}}$ is obtained by assuming it equal to the angular velocity $\boldsymbol{\omega}$ in $\tilde{\mathbf{Z}}_t^*$.

Constraints. The state constraint $\bar{\mathbf{x}}_t \in \mathbb{X}$ encodes the maximum safe linear velocity $\boldsymbol{v}$ and position boundaries $\boldsymbol{p}$ of the environment, while actuation constraints $\bar{\mathbf{u}}_t \in \mathbb{U}$ account for physical limits of the robot, restricting the nominal angular velocities $\boldsymbol{\omega}_{\text{cmd}}$ (to prevent saturation of the onboard gyroscope), and the maximum/minimum thrust force t_{cmd} produced by the propellers. We impose tightened constraints on the thrust force by constraining $\boldsymbol{f}_{\text{prop}}$ in $\tilde{\mathbf{z}} \in \tilde{\bar{\mathbb{Z}}}$. These constraints are obtained via a conservative approach, i.e. we require a minimal thrust to generate a trajectory feasible within our position and velocity constraints. Such feasible trajectory is found via an iterative tightening procedure for the thrust constraints, using the previously-obtained feasible trajectory as an initial guess for the subsequent

optimization under tightened thrust constraints. This procedure ensures that sufficient control authority is left to the ancillary NMPC to account for the presence of large unknown aerodynamic effects and mismatches in the mapping from commanded thrust/actual thrust. This cautious approach enabled successful real-world execution of the maneuver without further real-world tuning. We additionally leverage the further degrees of freedom introduced by the extended state $\tilde{\mathbf{z}}$ by shaping the safe plan through upper-bounding the thrust rates $\dot{\boldsymbol{f}}_{\text{prop}}$ via $\tilde{\bar{\mathbb{V}}}$, although this constraint will not be enforced by the ancillary NMPC. Last, using Monte-Carlo closed-loop simulations with disturbances sampled from $\mathbb{W}$, we verify that $\mathbb{X}$ and $\mathbb{U}$ are satisfied, and we generate a constant estimate (outer approximation, axis-aligned bounding box) of the cross-section of the tubes $\mathbb{T}^{\text{state}}$ and $\mathbb{T}^{\text{action}}$.

Tube and Data Augmentation with Attitude Quaternions. The normalized attitude quaternions part of the states $\bar{\mathbf{x}}$, $\tilde{\mathbf{z}}$ of nonlinear RTMPC, and part of the reference $\mathbf{Z}_t^*$ for the ancillary NMPC do not belong to a vector space, and therefore it is not trivial to describe its tube nor to generate extra samples for DA. In this work, we employ an attitude error representation $\boldsymbol{\epsilon} \in \mathbb{R}^3$ based on the Modified Rodriguez Parameters (MRP) [123] to generate a representation that can be treated as an element of a vector space. Specifically, we use

$$\boldsymbol{\epsilon}_t = \text{MRP}(\boldsymbol{q}_t \odot \boldsymbol{q}_t^{*-1}), \tag{4.13}$$

where $\boldsymbol{q}_t$ is the current attitude, $\boldsymbol{q}_t^*$ is the desired attitude (from the safe plan $\mathbf{z}_t^*$), $\text{MRP}(\cdot)$ maps a quaternion to the corresponding three-dimensional attitude representation, while $\odot$ denotes the quaternion product.

4.5 Evaluation

In this Section, we evaluate the ability of our method to efficiently learn acrobatic flight maneuvers using demonstrations collected from nonlinear RTMPC.

4.5.1 Evaluation Approach

Task Description. The goal is to perform a flip, i.e., a 360° rotation about the body-frame x-axis, in near-minimum time. This is a challenging maneuver, as it covers a large nonlinear envelope of the dynamics of the MAV, and the near-minimum time objective function, combined with the need to account for uncertainties, pushes the actuators close to their physical limits.

Simulation Environment. The simulation environment for training/numerical evaluations is the same as in Section 3.5, i.e., implements the nonlinear multirotor model in Section 3.4.1. In the training domain (source, $\mathcal{S}$), $\mathbb{W}_{\mathcal{S}} = \{\emptyset\}$, while in the deployment domain (target, $\mathcal{T}$) $\mathbb{W}_{\mathcal{T}} = \{f_{\text{ext}} | 0.001mg \leq f_{\text{ext}} \leq 0.3mg\}$, sampled according to Eq. (3.20).

Nonlinear RTMPC. We generate a safe nominal flip trajectory using `MECO-Rockit` [124]. Because this nominal trajectory happens in the plane spanned by the orthogonal vectors defining the y and z axis of the inertial reference frame W, for simplicity, we project the dynamics onto the y and z axes, resulting in a two-dimensional model of the MAV used to generate the nominal plan. The nominal flip trajectory can therefore be obtained by setting the initial rotation around the z to be 0, and the desired final attitude to be 2π, while the remaining initial/terminal states are all set to zero.

The ancillary NMPC is solved using the SQP solver `ACADOS` [125], and runs in simulation at 50Hz. Sensitivities for DA (Eq. (4.7)) are computed using the built-in sensitivity computation in the chosen solver, `HPIPM` [126]. We remark that the employed ancillary NMPC uses the full 3D multirotor model in Eq. (3.18), therefore performing 3D disturbance rejection – a critical requirement for real-world deployments. For a more challenging and interesting comparison to the considered IL baselines, the ancillary NMPC uses the `SQP_RTI` setting of `ACADOS`. This setting performs only a single SQP iteration per timestep, enabling significant speed-ups in the solver, and it is often employed in real-time, embedded implementations of NMPC. This setting creates an advantage, in terms of training time, to IL methods that require querying

Table 4.1: Parameters employed for the solution of the ancillary NMPC using the optimization package `ACADOS` [125].

Parameter(s)	Value(s)
Hessian Approximation	Gauss-Newton (flip), Exact (point-to-point)
QP solver	Partial Condensing `HPIPM` [126]
NLP/QP Tolerance	$10^{-8}/10^{-8}$
Levenberg-Marquardt	10^{-4}
Integrator Type	Implicit Runge-Kutta
Max. # Iterations QP Solver	100
Horizon (N, steps)/(time, seconds)	50/1.0

the expert multiple times (the baselines of our comparison), as it speeds-up the computation time of the expert. The other `ACADOS` parameters given in Table 4.1 were chosen as they enabled higher overall performance/accuracy in the selected acrobatic maneuver. Last, we introduce a discount factor $\gamma = 0.95$ in the stage cost of Eq. (4.2) to aid the convergence of the solver.

Student Policy. The student is a 2-hidden layers, fully connected DNN with $\{64, 32\}$ neurons/layer, and `ReLU` activation functions. The input has dimension 14, as it contains the current state ($n_x = 10$), time t, and a desired final position $\boldsymbol{p}^{\text{des}}$ (fixed to the origin). To simplify the learning and DA procedure, we enforce continuity to the quaternion input of the policy via [127, Eq. 3], avoiding the need to increase the training data/demonstrations at every timestep to account for the fact that $\boldsymbol{q}$ and $-\boldsymbol{q}$ encode the same orientation.

Baselines and Evaluation Metrics The baselines match those in Section 3.5 (DAgger, BC and their combination with DR, DA-N_s (linear interpolation) and DA-N_s (expert neighborhood), generating N_s samples per timestep. The monitored metrics (*Robustness*, *Performance* and *Training Time*) match those in Section 3.5, with the difference that performance is based on the stage cost of the ancillary NMPC Eq. (4.2).

Training Details. As in Section 3.5, training is performed by collecting demonstrations with the multirotor starting from slightly different initial states inside the tube centered around the origin. The nominal flip maneuver is pre-generated, as the goal state $\mathbf{x}^e_{t_0}$ (with $t_0 = 0$)

does not change, and only the ancillary NMPC is solved at every timestep. The resulting flip maneuver takes about $T_f = 2.5$s, and demonstrations are collected over an episode of length 3.0s, at 50Hz ($T = 150$ environment steps per demonstration).

Data collection. For SA-methods, we collect demonstrations one-by-one, and we implement the fine-tuning procedure described in Section 4.3.4 by performing DA with the first collected demonstration, while we do not perform DA for the following demonstrations. Because of its computational efficiency, we always use the sensitivity-based DA (i.e., Eq. (4.5), assuming no changes in active set of constraints). In addition, to study the effects of varying the number of samples used for DA, we introduce SA-N_S, a variant of SA where we sample uniformly inside the tube $N_S = \{25, 50, 100\}$ samples for every timestep. For the baselines, in order to speed-up the demonstration collection phase, and thereby avoid excessive re-training of the policy, we collect demonstrations in batches of 10, for 20 batches. An exception is made for DA-N_s (expert neighborhood) where, similar to SA, we collect demonstrations one-by-one to better study its sample efficiency, and the corresponding actions are obtained using the same sensitivity-based approximation of the ancillary NMPC employed by SA. Therefore, DA-N_s (expert neighborhood) constitutes an ablation of SA-N_s, where sampling is restricted to a smaller volume than the tube.

Evaluation. Each time the policy is updated, we evaluate it 20 times in source $\mathcal{S}$ and target $\mathcal{T}$ environments. The evaluations are averaged across 10 different random seeds. To further speed-up training of all the methods, we update the previously trained policy using only the newly collected batch of demonstrations (or single demonstration, for SA). All the policies are trained using the ADAM optimizer for up to 400 epochs, but we terminate training if the validation loss (from 30% of the data) does not decrease within 30 epochs.

4.5.2 Numerical Evaluation: Robustness, Performance, Efficiency

Comparison with Baselines. We start by evaluating the robustness and performance of the proposed approach as a function of the number of demonstrations collected in simulation,

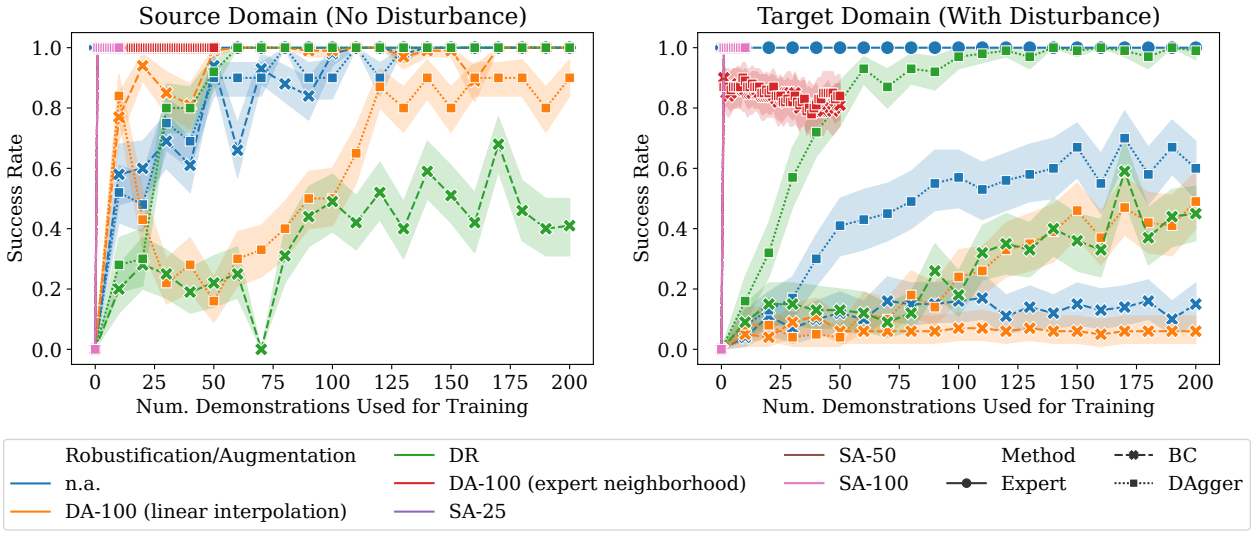


Figure 4.1: Robustness as a function of the number of training demonstrations. The IL methods that use the proposed DA strategy, Sampling-Augmentation (SA), overlap on the top-left part of the diagram, achieving full success rate in both the source and target domain of the training environment. Uncertainties in the target domains are applied in the form of a constant external force acting on the center of mass of the multirotor, representing large wind disturbances.

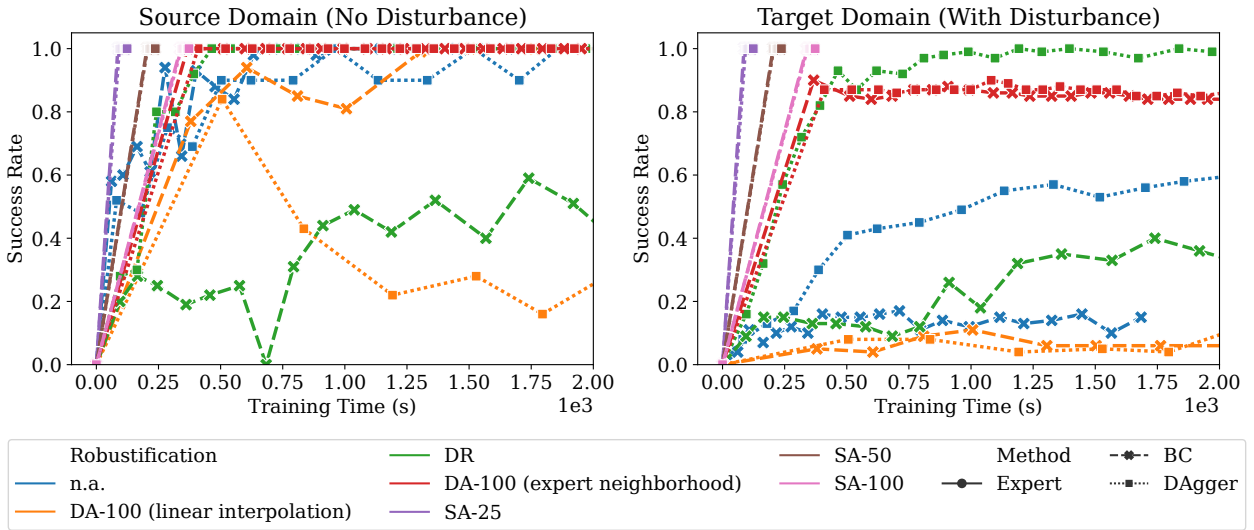


Figure 4.2: Robustness as a function of the training time. The IL methods that use the proposed DA and fine-tuning strategy, Sampling-Augmentation (SA), achieve full robustness under uncertainties in a fraction of the training time required by the best performing robust baseline, DAgger + DR.

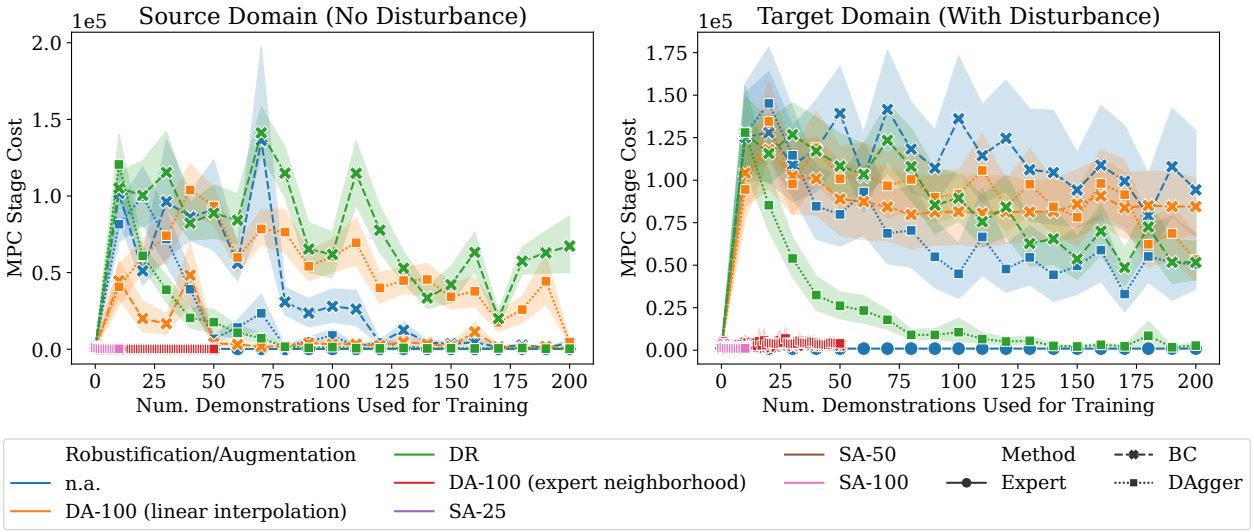


Figure 4.3: Performance as a function of the number of training demonstrations. The IL methods that use the proposed DA and fine-tuning strategy, Sampling-Augmentation (SA), achieve performance close to the expert in less than 10 demonstrations, while the best-performing baseline, SA-100 (expert neighborhood), achieves comparable performance, but is not robust in the target domain, as shown in Fig. 4.1.

and as a function of the training time.

Fig. 4.1 shows the robustness of the considered method as a function of the number of expert demonstrations. It reveals that SA-based approaches can achieve full success rate in the environment with disturbances (target, $\mathcal{T}$) and without disturbances (source, $\mathcal{S}$) after a single demonstration, while the best-performing baseline, DAgger+DR, requires about 60 demonstrations to achieve full robustness in $\mathcal{S}$, and more than 100 in $\mathcal{T}$. SA-based methods, therefore, enable more than one order of magnitude reduction in the number of demonstrations (interactions with the environment) compared to DAgger+DR. As previously observed in Section 3.5, DAgger alone is not robust. Additionally, BC methods fail to converge, potentially due to the lack of sufficiently meaningful exploration and the forgetting caused by the iterative training strategy employed. In addition, DA (expert neighborhood) confirms the importance of using the tube as a support of the sampling distribution, as the method achieves robustness and demonstration efficiency in the source domain, but fails to achieve robustness in the target domain, unlike SA. DA (linear interpolation) provides an

initial boost in demonstration efficiency, but struggles to achieve high robustness even in the source domain, potentially due to the far-from optimal action introduced.

Fig. 4.2 additionally shows the robustness as a function of the training time (recall, this includes demonstration collection and policy train). The results show that the demonstration-efficiency of SA-based methods translates into significant improvements in training time, as DAgger+DR requires more than 3 times the training time than SA-based approaches. These improvements are larger for the variants of SA that generate fewer extra samples (e.g., SA-25).

Last, Fig. 4.3 reports the *performance* as a function of the number of demonstrations. The results indicate that SA-based methods can achieve low tracking errors even after a single demonstration. Furthermore, employing a fine-tuning phase (after the initial demonstration) proves highly advantageous in further reducing this error, thereby reducing the performance gap between policies obtained via SA and the expert.

Comparison of Sampling Strategies. Table 4.2 provides a detailed comparison of performance, robustness, and training time of the different variants of SA methods, as a function of the number of demonstrations (1, 2 and 10), and compares those with the best-performing baselines, DAgger+DR, and methods based on sampling in the expert neighborhood. As expected, SA methods that require fewer samples obtain significant improvements in training time compared to DAgger+DR, while increasing the number of samples is beneficial in reducing the mean and the variance of the expert gap, both with and without disturbances, while DA (expert neighborhood) struggles to achieve high robustness and low expert gap in the target domain, despite the large number of samples used per timestep (100). Table 4.2 additionally highlights the benefits of fine-tuning, as even methods that use few samples (e.g., SA-sparse, SA-25) can obtain a significant performance improvement after a single fine-tuning demonstration (2 demonstrations in total), while there are diminishing returns for additional fine-tuning demonstrations (e.g., 10 demonstrations). In addition, in the data-sparse regime (e.g., SA-18), using DAgger for fine-tuning appears more beneficial

Method	Robustification/ Augmentation	# of Demonstr.	Robustness success rate (%, ↑) $\mathcal{S}$ mean	$\mathcal{S}$ std	$\mathcal{T}$ mean	$\mathcal{T}$ std	Performance expert gap (%, ↓) $\mathcal{S}$ mean	$\mathcal{S}$ std	$\mathcal{T}$ mean	$\mathcal{T}$ std	Efficiency training time (s, ↓) – mean	– std
BC	DA-100 (expert neighborhood)	1	100	0	90	30	9	6	562	1576	367	87
		2	100	0	85	36	6	4	312	671	512	90
		10	100	0	86	35	5	4	462	2932	1166	135
		50	100	0	81	39	5	2	323	838	4629	185
DAgger	DA-100 (expert neighborhood)	1	100	0	87	34	12	8	562	2375	409	90
		2	100	0	87	34	9	10	425	1478	520	97
		10	100	0	89	31	4	3	187	354	1149	107
		50	100	0	84	37	3	2	440	1623	4608	141
DAgger	DR	50	92	27	82	39	9094	20608	3096	5497	392	34
		100	100	0	97	17	634	711	1277	4947	810	104
		200	100	0	99	10	91	73	274	1247	1970	154
BC	SA-sparse (18)	1	100	0	100	0	553	462	211	228	84	9
		2	100	0	97	17	41	39	371	1808	88	10
		10	100	0	97	17	33	42	226	815	115	10
	SA-25	1	100	0	100	0	956	402	270	384	87	15
		2	100	0	100	0	148	140	107	150	90	14
		10	100	0	100	0	107	175	90	83	117	14
	SA-50	1	100	0	100	0	421	193	105	116	204	32
		2	100	0	100	0	76	31	66	56	207	31
		10	100	0	100	0	55	28	76	102	235	31
	SA-100	1	100	0	100	0	291	154	89	105	339	72
		2	100	0	100	0	57	20	76	85	342	72
		10	100	0	100	0	33	21	97	118	369	72
DAgger	SA-sparse (18)	1	100	0	100	0	747	705	319	879	85	6
		2	100	0	100	0	222	122	142	219	89	6
		10	100	0	100	0	29	24	114	150	117	6
	SA-25	1	100	0	100	0	579	224	160	168	92	12
		2	100	0	100	0	366	279	122	182	96	12
		10	100	0	100	0	110	115	100	120	124	12
	SA-50	1	100	0	100	0	361	161	78	91	206	28
		2	100	0	100	0	169	117	77	80	210	28
		10	100	0	100	0	56	77	82	107	237	28
	SA-100	1	100	0	100	0	309	133	92	105	342	29
		2	100	0	100	0	100	61	77	96	346	29
		10	100	0	100	0	30	28	90	109	373	29

Table 4.2: Performance, robustness and training time for SA-based methods after 1, 2, and 10 demonstrations, compared with the best performing baselines, DAgger+DR and sampling in the expert neighborhood (DA-100 (expert neighborhood)), in the environment without disturbances ($\mathcal{S}$, source), and with ($\mathcal{T}$, target). Robustness is color-coded from white (100%) to red (90% or below). Performance and training time are color-coded from green (fast training time, small expert gap) to red (long training time, large expert gap). The results highlight that SA-methods achieve high robustness and close to expert performance compared to DAgger+DR, even after a single demonstration, and their performance can be further improved via additional fine-tuning demonstrations. Methods based on DA-100 (expert neighborhood) are not robust and struggle to achieve high performance (low expert gap) in the target domain. We note that DAgger and BC-based approaches differ at one demonstration due to non-determinism in the training procedure.

Table 4.3: Computation time to obtain a new action for the ancillary NMPC and the safe planner of the nonlinear RTMPC expert (N-RTMPC) and the proposed DNN policy (Policy). **The policy is 180 times faster than the NMPC in [79] (on the same onboard computer)**. The offboard CPU is an `Intel i9-10920`, onboard is an `NVIDIA Jetson TX2` (CPU). Note that the faster inference time than the linear case is caused by the input dimension being smaller (14 vs 188).

			Time (ms)			
CPU	**Method**	**Setup**	**Mean**	**SD**	**Min**	**Max**
Offboard	N-RTMPC, ancillary NMPC	`ACADOS` [125]	7.28	0.15	7.05	8.00
	N-RTMPC, safe plan	`IPOPT`	5812	226	4828	6010
	Policy	PyTorch	**0.11**	**0.01**	**0.11**	**0.27**
Onboard	NMPC (from [79, Fig. 17])	`ACADO` [128]	2.7	n.a.	n.a.	n.a.
	Policy	C++/Eigen	**0.015**	0.005	0.006	0.101

than BC.

Computation. The computation time is reported in Table 4.3, highlighting that the average computation time of the policy on the onboard `Nvidia Jetsion TX2` is 0.015 ms, an 180-fold improvement compared to the value reported in [129] for a state-of-the-art NMPC for quadrotors. In addition, Fig. 4.4 reports the time to compute the sensitivity matrix, highlighting that it requires on average 2.28 ms, about only 32% of the time required to solve the full optimization problem. More importantly, this matrix is computed only once per timestep and can be used to draw many sampled at small additional cost (equivalent to solving a vector-matrix multiplication) instead of solving the full optimization problem for each sample. The average time to step the training environment is 2.1 ms.

4.5.3 Hardware Evaluation

We experimentally evaluate the ability of the policy to perform a flip on a real multirotor, under real-world uncertainties such as model errors (e.g., inaccurate thrust to battery voltage mappings, aerodynamic coefficients, moments of inertia) and external disturbances (e.g., ground effect). The tested policy is obtained using DAgger+SA-25 trained after 2 demonstrations (the first with DA, the second for fine-tuning), as the method represents a

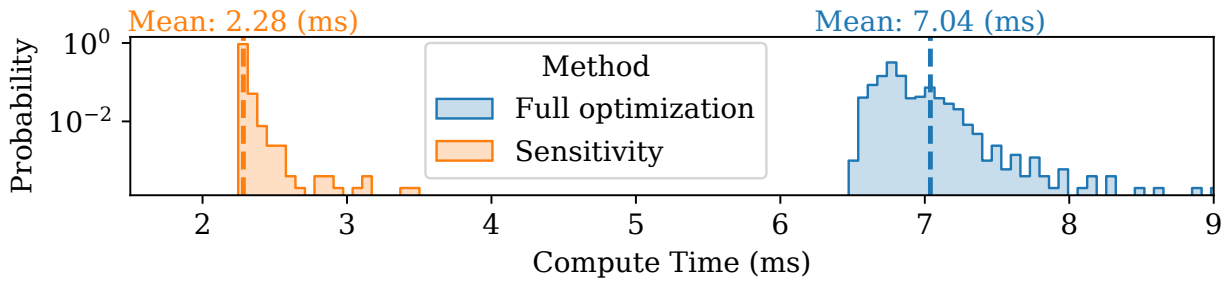


Figure 4.4: Distribution of the time to solve the full optimization problem for the ancillary NMPC (Eq. (4.2)), and the time required to additionally compute the sensitivity matrix (Eq. (4.7)). The sensitivity matrix can be computed efficiently, requiring only 32% of the time to solve the full optimization problem. Analysis performed on a `Intel i9-10920` using `ACADOS` with settings in Table 4.1. Note the log scale of the y axis.

good trade-off between performance, robustness and training time. As in Section 3.5, we deploy the learned policy on an onboard `Nvidia Jetson TX2`, where it runs at 100Hz. The maneuver includes a take-off/landing phase consisting of a 1m ramp on x-y-z in W and overall has a total duration of 6s. The maneuver is repeated 5 times in a row, to demonstrate repeatability, recording successful execution of the maneuver and successful landing at the designated location in all the cases. Fig. 4.5 shows a time-lapse of the different phases of the maneuver (excluding the ramp from and to the landing location). The 3D position of the robot, as well as the direction of its thrust vector, are shown for two runs in Fig. 4.6, highlighting the large distance and altitude traveled in a short time. Fig. 4.7 additionally shows some critical parameters of the maneuvers, such as the attitude and the angular velocity, as well as thrust and the vertical velocities. It highlights that the robot rotates at up to 11rad/s, and the overall 360° rotation takes about 0.5s. In addition, the maneuver is repeated under even more challenging uncertainties, obtained by attaching either (a) a slung-load or (b) a drag surface to the robot, as shown in Fig. 4.6, where we additionally deploy the policy onboard at 500Hz. The experiment is repeated 3 consecutive times per disturbances, achieving 100% success rate, and a resulting trajectory for each disturbance is shown in Fig. 4.6 [1]. Overall, these results validate our numerical analysis and highlight the robustness

[1]Note that the gains of the cascaded attitude controller were increased compared to the scenario without extra disturbances. This was done to account for the fact that the attitude controller is not explicitly robust/adaptive to these new uncertainties, unlike the learned policy.

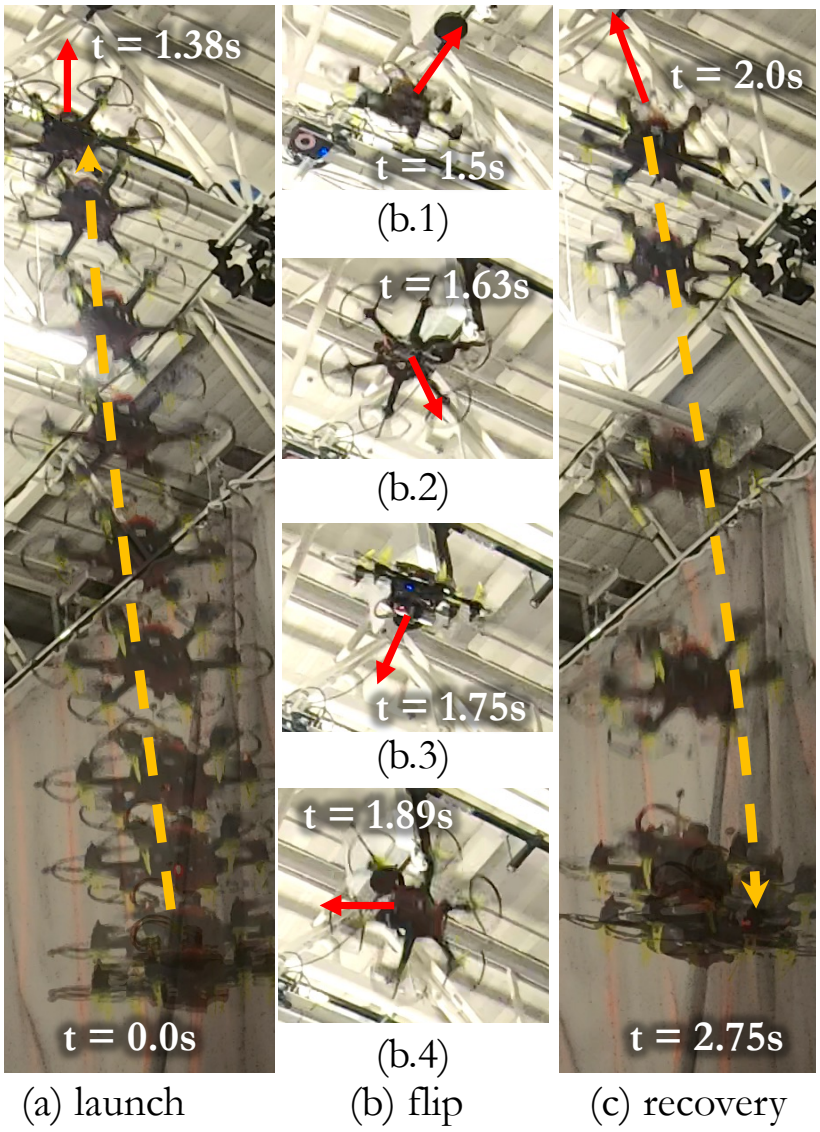


Figure 4.5: Time-lapse figure showing an acrobatic maneuver (flip) performed by our multirotor using a NN policy learned via our proposed approach. The policy is learned offboard in a computationally and data-efficient manner (**in only about** 100**s, requiring only two demonstrations in sim.**), and is deployed onboard (`Nvidia Jetson TX2`, CPU), where was tested at **up to** 500**Hz, with an average inference time as low as** 15 μ**s**). (a) shows the robot accelerating upwards until it reaches the optimal altitude found by MPC. The red arrow denotes the directions of the thrust vector (aligned with the body z-axis), while the yellow arrow denotes the approximate trajectory. (b) shows the robot performing a 360° rotation round its body x-axis. This phase takes only about 0.5s. (c) shows the robot successfully decelerating until it reaches a vertical velocity < 1 m/s, followed by its landing. This experiment demonstrates the ability of our approach to efficiently generate policies from MPC that can withstand real-world uncertainties and disturbances, even when leveraging models that operate in highly nonlinear regimes.

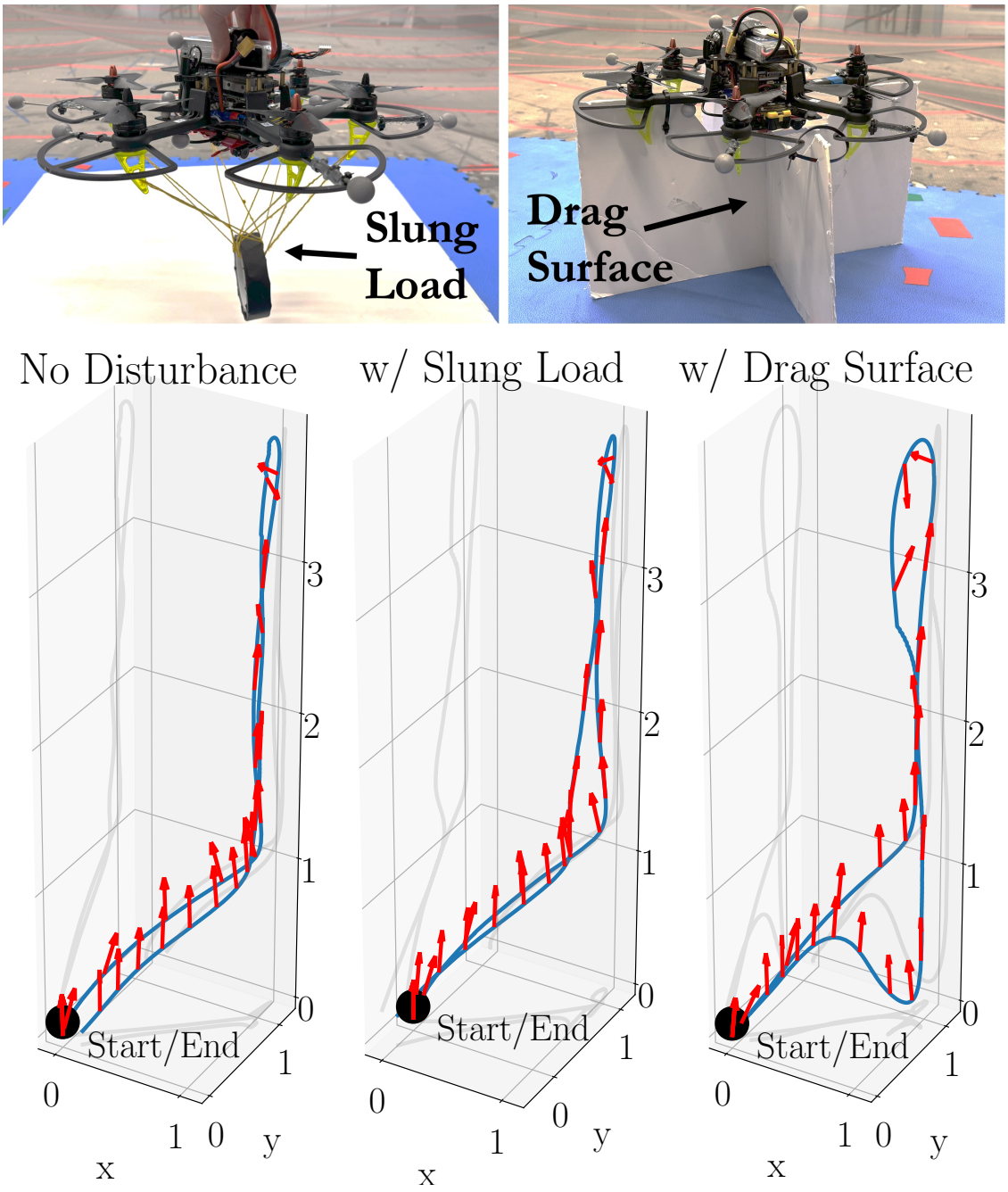


Figure 4.6: Acrobatic flip trajectory performed in experiments using a policy learned from a nonlinear Robust Tube MPC in about 100s. The policy runs onboard (TX2, CPU, tested up to 500Hz, average inference time during maneuver as low as 0.015ms) with and without disturbances (slung load of 0.18 Kg. drag surface of 0.13 kg). This highlights the real-world robustness and performance of the approach. Red arrows denote the direction of the thrust vector on the quadrotor, and they highlight that the flip occurs at the point of highest altitude of the maneuver. The plot shows the additional take-off/landing phase, also performed by the learned policy, taking the robot back and forth the origin and $(x, y, z) = (1, 1, 1)$, that is the point where the flip starts and ends. Units in (m).

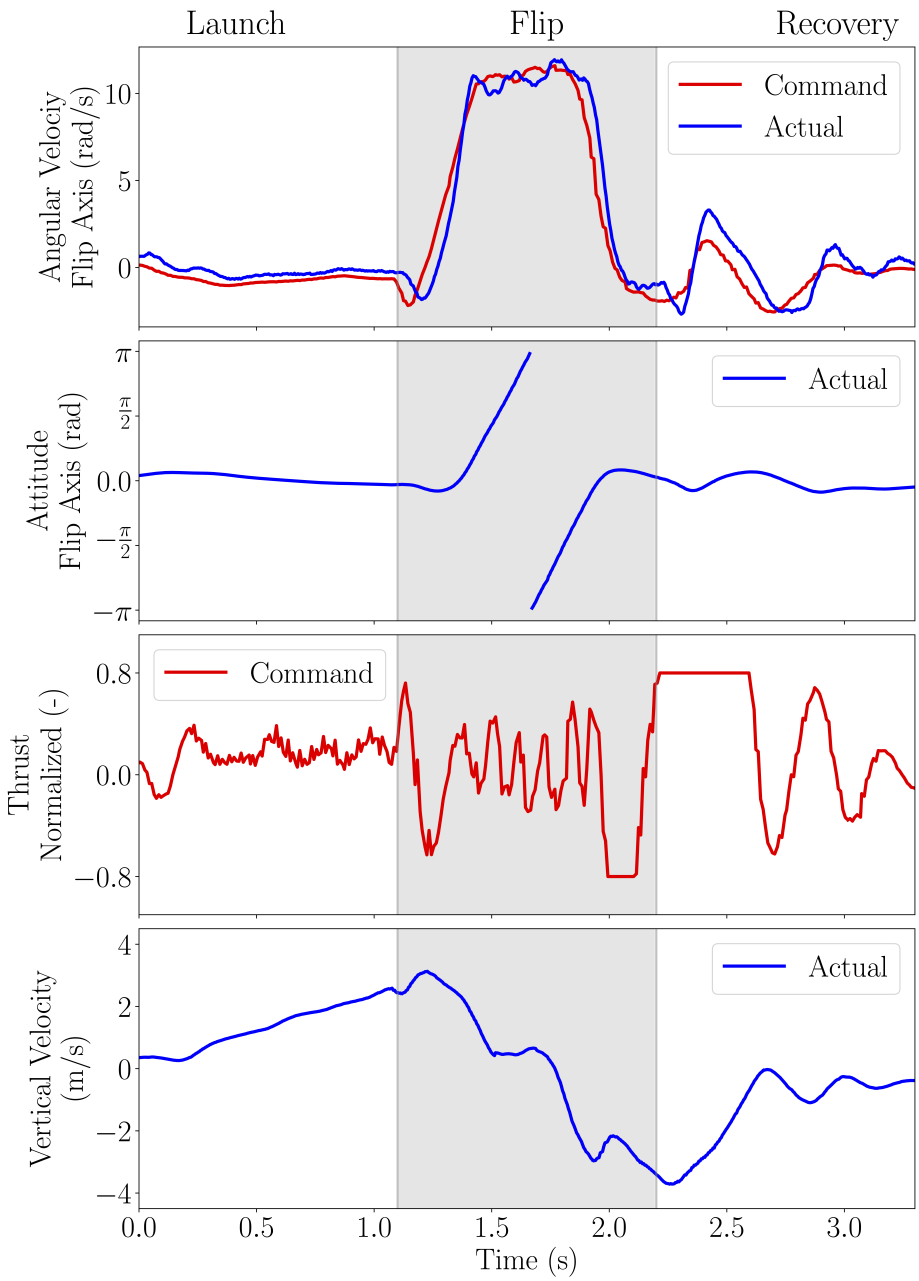


Figure 4.7: Control inputs generated by the learned policy and relevant states during the real–world acrobatic flip maneuver on a multirotor, where the robot is subject to large levels of uncertainties, caused primarily by model mismatches such as thrust-to-battery voltage mappings and hard-to-model aerodynamic effects. Despite the large level of uncertainties, that require the usage of the maximum thrust allowed, the maneuver is completed successfully, and the robot reaches the desired vertical velocity of $-1.0\,\mathrm{m\,s^{-1}}$ at the end of the recovery phase (and that corresponds to the initial velocity of the landing phase). We highlight that the flip is performed at an angular velocity of about $11.0\,\mathrm{rad/s}$. The actual thrust t_{cmd} can be related to the normalized thrust $\bar{t}_{\mathrm{cmd}}$ via $t_{\mathrm{cmd}} = mg(1 + \bar{t}_{\mathrm{cmd}})$, where mg is the weight force of the robot.

and performance of a policy efficiently trained from 2 demonstrations and about 100s of training time. Our video submission [130] includes an additional experiment demonstrating near-minimum time navigation from one position to another, starting and ending with velocity close to zero, using a policy trained with two demonstrations (DAgger+SA-25).

4.6 Summary

This Chapter has provided a strategy for efficient and robust policy learning from nonlinear RTMPC. While the linear ancillary controller in linear RTMPC employed in Chapter 3 provided extra data in a computationally efficient way, the same efficiency was challenging to achieve when leveraging nonlinear variants of RTMPC [27]. Therefore, this Chapter has efficiently performed tube-guided DA by leveraging a sensitivity-based approximation of the ancillary controller in NMPC, and introducing a fine-tuning phase to reduce the errors caused by the extra sampled obtained via the sensitivity-based approximation of the ancillary NMPC. Experimental evaluations on a multirotor have validated our numerical findings of efficiency and robustness, showing that a policy with 15 us of inference time trained with only two demonstrations (100 s of training time, 3 times more efficient than existing methods) can perform a flip under uncertainties.

Chapter 5

Vision-based Flight Control

5.1 Overview

This Chapter introduces Tube-Neural Radiance Field (NeRF) (Fig. 5.1), a novel DA framework for efficient, robust *visuomotor* policy learning from MPC that overcomes the robustness limitations in existing DA strategies, as presented in Problem Statement 3. Building on our prior DA strategy [62] (Chapter 3) that uses RTMPC for *efficiently* and *systematically* generate additional training data for motor control policies (actions from full state), Tube-NeRF enables learning of policies that directly use vision as input, relaxing the constraining assumption in [62] that full-state information is available at deployment.

Tube-NeRF, first, collects robust task demonstrations that account for the effects of process *and sensing* uncertainties via an output-feedback variant of RTMPC [4]. Then, it employs a photorealistic representation of the environment, based on a NeRF, to generate synthetic novel views for DA, using the tube of the controller to guide the selection of the extra novel views and sensorial inputs, and using the ancillary controller to efficiently compute the corresponding actions. Additionally, the tube is employed to generate queries from a database of real-world observations, reducing the synthetic-to-real gap when only NeRF images are used for DA. Lastly, we adapt our approach to a multirotor, training a *visuomotor* policy

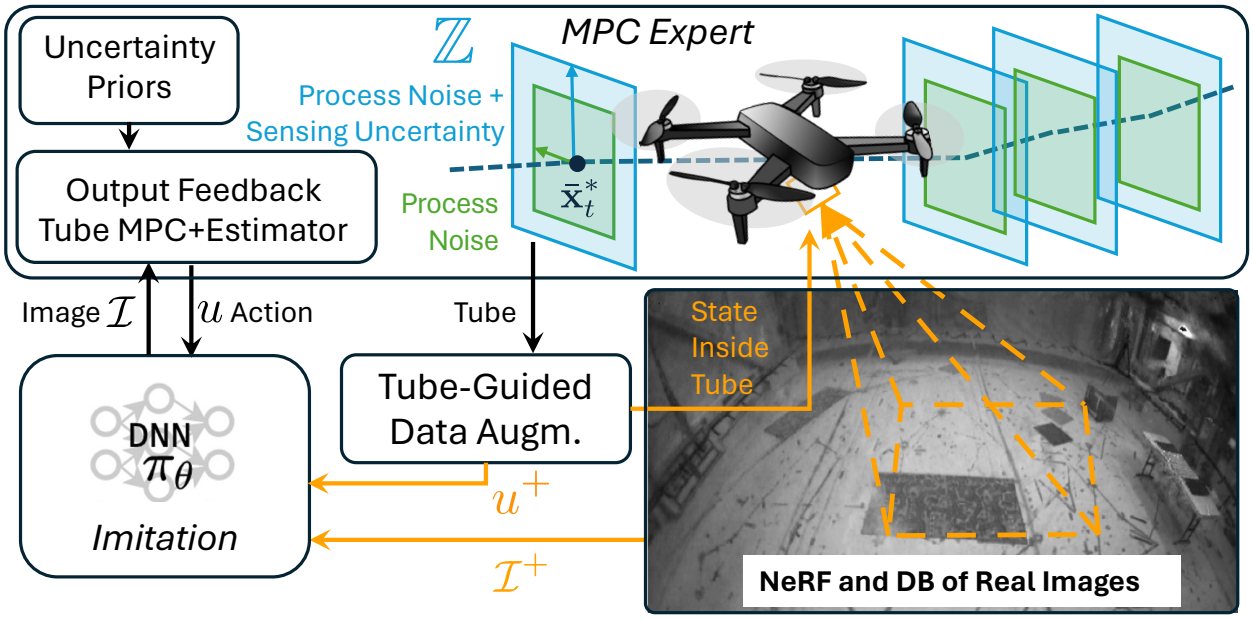


Figure 5.1: Tube-NeRF collects a real-world demonstration using output-feedback tube MPC, a robust MPC that accounts for process and sensing uncertainties through its tube cross-section $\mathbb{Z}$. Then, it generates a Neural Radiance Field (NeRF) of the environment from the collected images $\mathcal{I}_t$, and uses the tube's cross-section to guide the selection of synthetic views $\mathcal{I}_t^+$ from the NeRF for data augmentation, while corresponding actions are obtained via the *ancillary controller*, an integral component of the tube MPC framework.

for robust trajectory tracking and localization using onboard camera images and additional measurements of altitude, orientation, and velocity. The generated policy relies solely on images to obtain information on the robot's horizontal position, which is a challenging task due to (1) its high speed (up to 3.5 m/s), (2) varying altitude, (3) aggressive roll/pitch changes, (4) the sparsity of visual features in our flight space, and (5) the presence of a safety net that moves due to the down-wash of the propellers and that produces semi-transparent visual features above the ground.

5.2 Problem Formulation

The goal is to *efficiently* train a NN visuomotor policy π_θ (*student*), with parameters θ, that tracks a desired trajectory on a mobile robot (multirotor). π_θ takes as input images, which are needed to extract partial state information (horizontal position, in our evaluation), and

other measurements. The trained policy, denoted $\pi_{\hat{\theta}}$, needs to be robust to uncertainties encountered in the deployment domain $\mathcal{T}$. It is trained using demonstrations from a model-based controller (*expert*) collected in a source domain $\mathcal{S}$ that presents only a subset of the uncertainties in $\mathcal{T}$.

Student Policy. The student policy has the form:

$$\mathbf{u}_t = \pi_\theta(\mathbf{o}_t, \mathbf{X}_t^{\text{des}}). \tag{5.1}$$

It outputs deterministic, continuous actions $\mathbf{u}_t$ to track a desired $N+1$ steps ($N > 0$) trajectory $\mathbf{X}_t^{\text{des}} = \{\mathbf{x}_{0|t}^{\text{des}}, \ldots, \mathbf{x}_{N|t}^{\text{des}}\}$. $\mathbf{o}_t = (\mathcal{I}_t, \mathbf{o}_{\text{other},t})$ are noisy sensor measurements comprised of an image $\mathcal{I}_t$ from an onboard camera, and other measurements $\mathbf{o}_{\text{other,t}}$ (altitude, attitude, velocity, in our evaluation).

Expert. *Process model:* The considered robot dynamics are:

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t \tag{5.2}$$

where $\mathbf{x}_t \in \mathbb{X} \subset \mathbb{R}^{n_x}$ is the state, and $\mathbf{u}_t \in \mathbb{U} \subset \mathbb{R}^{n_u}$ the control inputs. The robot is subject to state and input constraints $\mathbb{X}$ and $\mathbb{U}$, assumed convex polytopes containing the origin [4]. $\mathbf{w}_t \in \mathbb{W}_{\mathcal{T}} \subset \mathbb{R}^{n_x}$ in (5.2) captures time-varying additive *process* uncertainties in $\mathcal{T}$, such as (i) disturbances (wind/payloads for a UAV) (ii) and model changes/errors (linearization errors and poorly known parameters). $\mathbf{w}_t$ is unknown, but the polytopic bounded set $\mathbb{W}_{\mathcal{T}}$ is assumed known [4]. *Sensing model:* The expert has access to (i) the measurements $\mathbf{o}_{\text{other},t}$, and (ii) a vision-based position estimator g_{cam} that outputs noisy measurements $\mathbf{o}_{\text{pos.xy}} \in \mathbb{R}^2$ of the horizontal position $\boldsymbol{p}_{xy,t} \in \mathbb{R}^2$ of the robot:

$$\mathbf{o}_{\text{pos,xy},t} = g_{\text{cam}}(\mathcal{I}_t) = \boldsymbol{p}_{xy,t} + \mathbf{v}_{\text{cam},t}, \tag{5.3}$$

where $\mathbf{v}_{\text{cam},t}$ is the associated sensing uncertainty. The measurements available to the expert

are denoted $\bar{\mathbf{o}}_t \in \mathbb{R}^{n_o}$, and they map to the robot state via:

$$\bar{\mathbf{o}}_t = \left[\mathbf{o}_{\text{pos.xy},t}^T, \mathbf{o}_{\text{other},t}^T\right]^T = \mathbf{C}\mathbf{x}_t + \mathbf{v}_t, \tag{5.4}$$

where $\mathbf{C} \in \mathbb{R}^{n_o \times n_x}$. $\mathbf{v}_t = [\mathbf{v}_{\text{cam},t}^T, \mathbf{v}_{\text{other},t}^T]^T \in \mathbb{V}_{\mathcal{T}} \subset \mathbb{R}^{n_o}$ is additive *sensing* uncertainty (e.g., noise, biases) in $\mathcal{T}$. $\mathbb{V}_{\mathcal{T}}$ is a known bounded set obtained via system identification, and/or via prior knowledge on the accuracy of g_{cam}.

State estimator. We assume the expert uses a state estimator:

$$\hat{\mathbf{x}}_{t+1} = \mathbf{A}\hat{\mathbf{x}}_t + \mathbf{B}\mathbf{u}_t + \mathbf{L}(\bar{\mathbf{o}}_t - \hat{\mathbf{o}}_t), \quad \hat{\mathbf{o}}_t = \mathbf{C}\hat{\mathbf{x}}_t, \tag{5.5}$$

where $\hat{\mathbf{x}}_t \in \mathbb{R}^{n_x}$ is the estimated state, and $\mathbf{L} \in \mathbb{R}^{n_x \times n_o}$ is the observer gain, set so that $\mathbf{A}-\mathbf{LC}$ is Schur stable. The observability index of the system $(\mathbf{A}, \mathbf{C})$ is assumed to be 1, meaning that full state information can be retrieved from a single noisy measurement. In this case, the observer plays the critical role of filtering out the effects of noise and sensing uncertainties. Additionally, we assume that the state estimation dynamics and noise sensitivity of the learned policy will approximately match the ones of the observer.

5.3 Methodology

Overview. Tube-NeRF collects trajectory tracking demonstrations in the source domain $\mathcal{S}$ using an output feedback RTMPC expert combined with a state estimator Eq. (5.5), and IL methods (DAgger or BC). The chosen output feedback RTMPC framework is based on [4], [91], with its objective function modified to track a trajectory (Section 5.3.1), and is designed according to the priors on process *and* sensing uncertainties at deployment ($\mathcal{T}$). Then, Tube-NeRF uses properties of the expert to design an efficient DA strategy, the key to overcoming efficiency and robustness challenges in IL (Section 5.3.2). The framework is then tailored to a multirotor leveraging a NeRF as part of the proposed DA strategy (Section 5.4).

5.3.1 Output Feedback RTMPC for Trajectory Tracking

Output feedback RTMPC for trajectory tracking regulates the system in Eq. (5.2) and Eq. (5.5) along a given *reference* trajectory $\mathbf{X}_t^{\text{des}}$, while satisfying state and actuation constraints $\mathbb{X}, \mathbb{U}$ regardless of sensing uncertainties ($\mathbf{v}$, Eq. (5.4)) and process noise ($\mathbf{w}$, Eq. (5.2)).

Preliminary (set operations): Given the convex polytopes $\mathbb{A} \subset \mathbb{R}^n, \mathbb{B} \subset \mathbb{R}^n$ and $\mathbf{M} \in \mathbb{R}^{m \times n}$, we define:

- Minkowski sum: $\mathbb{A} \oplus \mathbb{B} \coloneqq \{\mathbf{a} + \mathbf{b} \in \mathbb{R}^n \mid \mathbf{a} \in \mathbb{A},\ \mathbf{b} \in \mathbb{B}\}$
- Pontryagin difference: $\mathbb{A} \ominus \mathbb{B} \coloneqq \{\mathbf{c} \in \mathbb{R}^n \mid \mathbf{c} + \mathbf{b} \in \mathbb{A}, \forall \mathbf{b} \in \mathbb{B}\}$
- Linear mapping: $\mathbf{M}\mathbb{A} \coloneqq \{\mathbf{M}\mathbf{a} \in \mathbb{R}^m \mid \mathbf{a} \in \mathbb{A}\}$.

Optimization Problem. At every timestep t, the controller solves:

$$
\begin{aligned}
\bar{\mathbf{U}}_t^*, \bar{\mathbf{X}}_t^* = \underset{\bar{\mathbf{U}}_t, \bar{\mathbf{X}}_t}{\operatorname{argmin}} & \|\mathbf{e}_{N|t}\|_{\mathbf{P}}^2 + \sum_{i=0}^{N-1} \|\mathbf{e}_{i|t}\|_{\mathbf{Q}}^2 + \|\mathbf{u}_{i|t}\|_{\mathbf{R}}^2 \\
\text{subject to } & \bar{\mathbf{x}}_{i+1|t} = \mathbf{A}\bar{\mathbf{x}}_{i|t} + \mathbf{B}\bar{\mathbf{u}}_{i|t},\ \ \forall i = 0, \ldots, N-1 \\
& \bar{\mathbf{x}}_{i|t} \in \bar{\mathbb{X}},\ \ \bar{\mathbf{u}}_{i|t} \in \bar{\mathbb{U}}, \\
& \bar{\mathbf{x}}_{N|t} \in \bar{\mathbb{X}}_N,\ \ \hat{\mathbf{x}}_t \in \bar{\mathbf{x}}_{0|t} \oplus \begin{bmatrix} \mathbf{0}_{n_x}, & \boldsymbol{I}_{n_x} \end{bmatrix} \mathbb{S},
\end{aligned}
\tag{5.6}
$$

where $\mathbf{e}_{i|t} = \bar{\mathbf{x}}_{i|t} - \mathbf{x}_{i|t}^{\text{des}}$ represents the trajectory tracking error, $\bar{\mathbf{X}}_t = \{\bar{\mathbf{x}}_{0|t}, \ldots, \bar{\mathbf{x}}_{N|t}\}$ and $\bar{\mathbf{U}}_t = \{\bar{\mathbf{u}}_{0|t}, \ldots, \bar{\mathbf{u}}_{N-1|t}\}$ are reference state and action trajectories computed according to the nominal systems dynamics (also often denoted as "safe state and action plans", as they are at a sufficient distance from state and actuation constraints), and $N+1$ is the length of the planning horizon. The positive semi-definite matrices $\mathbf{Q}$ (size $n_x \times n_x$) and $\mathbf{R}$ (size $n_u \times n_u$) are tuning parameters, $\|\mathbf{e}_{N|t}\|_{\mathbf{P}}^2$ is a terminal cost (obtained by solving the infinite-horizon optimal control problem with $\mathbf{A}$, $\mathbf{B}$, $\mathbf{Q}$, $\mathbf{R}$) and $\bar{\mathbf{x}}_{N|t} \in \bar{\mathbb{X}}_N$ is a terminal state constraint.

Ancillary Controller. The control input $\mathbf{u}_t$ is obtained via:

$$\mathbf{u}_t = \bar{\mathbf{u}}_t^* + \mathbf{K}(\hat{\mathbf{x}}_t - \bar{\mathbf{x}}_t^*), \tag{5.7}$$

where $\bar{\mathbf{u}}_t^* \coloneqq \bar{\mathbf{u}}_{0|t}^*$ and $\bar{\mathbf{x}}_t^* \coloneqq \bar{\mathbf{x}}_{0|t}^*$, and $\mathbf{K}$ is computed by solving the LQR problem with $\mathbf{A}$, $\mathbf{B}$, $\mathbf{Q}$, $\mathbf{R}$. This controller maintains the system inside a set $\mathbb{Z} \oplus \bar{\mathbf{x}}_t^*$ ("cross-section" of a *tube* centered around $\bar{\mathbf{x}}_t^*$), regardless of the uncertainties.

Tube and Robust Constraints. Process and sensing uncertainties are taken into account by tightening the constraints $\mathbb{X}$, $\mathbb{U}$, obtaining $\bar{\mathbb{X}}$ and $\bar{\mathbb{U}}$ in Eq. (5.6). The amount by which $\mathbb{X}$, $\mathbb{U}$ are tightened depends on the cross-section of the tube $\mathbb{Z}$, which is computed (see [4]) by considering the closed-loop system formed by the ancillary controller Eq. (5.7), the nominal dynamics in Eq. (5.2) and the observer Eq. (5.5). *Sensing uncertainties* in $\mathbb{V}_\mathcal{T}$ and *process noise* in $\mathbb{W}_\mathcal{T}$ introduce two sources of errors in such system: a) the state estimation error $\boldsymbol{\xi}_t^{\text{est}} \coloneqq \mathbf{x}_t - \hat{\mathbf{x}}_t$, and b) the control error $\boldsymbol{\xi}_t^{\text{ctrl}} \coloneqq \hat{\mathbf{x}}_t - \bar{\mathbf{x}}_t^*$. These errors can be combined in a vector $\boldsymbol{\xi}_t = [\boldsymbol{\xi}_t^{\text{est}^T}, \boldsymbol{\xi}_t^{\text{ctrl}^T}]^T$ whose dynamics evolve according to (see [91]):

$$\boldsymbol{\xi}_{t+1} = \mathbf{A}_\xi \boldsymbol{\xi}_t + \boldsymbol{\delta}_t, \ \boldsymbol{\delta}_t \in \mathbb{D} \tag{5.8}$$

$$\mathbf{A}_\xi = \begin{bmatrix} \mathbf{A} - \mathbf{LC} & \mathbf{0}_{n_x} \\ \mathbf{LC} & \mathbf{A} + \mathbf{BK} \end{bmatrix}, \quad \mathbb{D} = \begin{bmatrix} \mathbf{I}_{n_x} & -\mathbf{L} \\ \mathbf{0}_{n_x} & \mathbf{L} \end{bmatrix} \begin{bmatrix} \mathbb{W}_\mathcal{T} \\ \mathbb{V}_\mathcal{T} \end{bmatrix}.$$

By design, $\mathbf{A}_\xi$ is a Schur-stable dynamic system, and it is subject to uncertainties from the convex polytope $\mathbb{D}$. Then, it is possible to compute the *minimal* Robust Positive Invariant (RPI) set $\mathbb{S}$, that is the *smallest* set satisfying:

$$\boldsymbol{\xi}_0 \in \mathbb{S} \implies \boldsymbol{\xi}_t \in \mathbb{S}, \ \forall \, \boldsymbol{\delta}_t \in \mathbb{D}, \ t > 0. \tag{5.9}$$

$\mathbb{S}$ represents the possible set of state estimation and control errors caused by uncertainties and is used to compute $\bar{\mathbb{X}}$ and $\bar{\mathbb{U}}$. Specifically, the error between the true state $\mathbf{x}_t$ and the

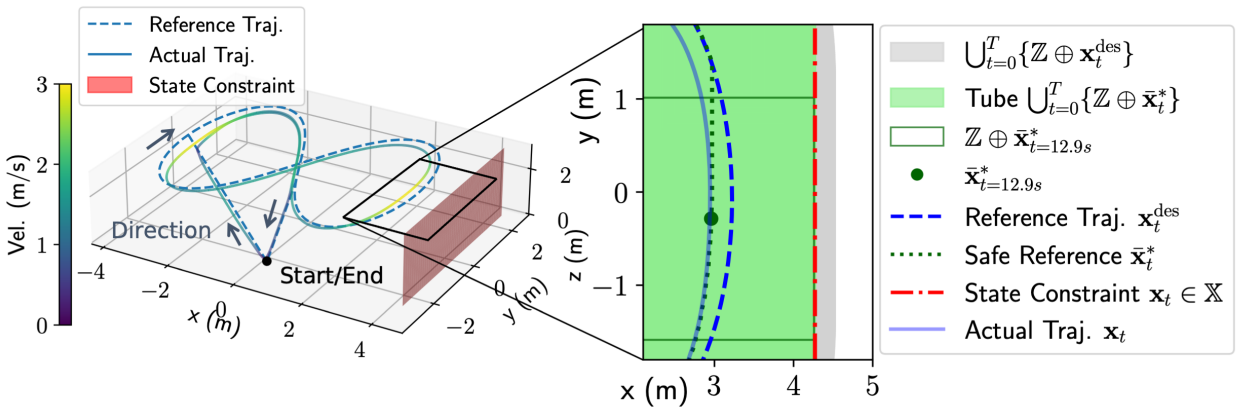


Figure 5.2: Output feedback RTMPC generates and tracks a safe reference to satisfy constraints.

reference state $\bar{\mathbf{x}}_t^*$ is: $\boldsymbol{\xi}^{\text{tot}} \coloneqq \mathbf{x}_t - \bar{\mathbf{x}}_t^* = \boldsymbol{\xi}^{\text{ctrl}} + \boldsymbol{\xi}^{\text{est}}$. As a consequence, the effects of noise and uncertainties can be taken into account by tightening the constraints of an amount:

$$\bar{\mathbb{X}} \coloneqq \mathbb{X} \ominus \mathbb{Z}, \ \mathbb{Z} = \begin{bmatrix} \boldsymbol{I}_{n_x} & \boldsymbol{I}_{n_x} \end{bmatrix} \mathbb{S}, \quad \bar{\mathbb{U}} \coloneqq \mathbb{U} \ominus \begin{bmatrix} \mathbf{0}_{n_x} & \mathbf{K} \end{bmatrix} \mathbb{S}. \tag{5.10}$$

$\mathbb{Z}$ (cross-section of a *tube*) is the set of possible deviations of the true state $\mathbf{x}_t$ from the safe reference $\bar{\mathbf{x}}_t^*$.

Computing $\mathbb{S}$. While accurately computing the minimal RPI set $\mathbb{S}$ for high-dimensional systems can be challenging [91], for simplicity, we efficiently obtain $\mathbb{S}$ from $\mathbb{W}_{\mathcal{T}}$ and $\mathbb{V}_{\mathcal{T}}$ via Monte Carlo simulations of Eq. (5.8), uniformly sampling instances of the uncertainties, and by computing an outer axis-aligned bounding box of the trajectories of $\boldsymbol{\xi}$. In addition, we treat linearization errors and future changes in the reference trajectory as an additional source of uncertainty, computing the tube based on an increased process uncertainty prior $\bar{\mathbb{W}}_{\mathcal{T}}$, and numerically validating that the resulting expert is robust. While this procedure is approximate, it was found computationally tractable and useful at estimating tubes with an adequate level of conservativeness.Fig. 5.2 shows an example of this controller for trajectory tracking on a multirotor, highlighting changes to the reference trajectory to respect state constraint under uncertainties.

5.3.2 Tube-guided Data Augmentation for Visuomotor Learning

IL objective

We denote the *expert* output feedback RTMPC in Eq. (5.6), Eq. (5.7) and the state observer in Eq. (5.5) as π_{θ^*}. The goal is to design an IL and DA strategy to efficiently learn the parameters $\hat{\theta}$ of the policy Eq. (5.1), collecting demonstrations from the expert π_{θ^*}. In IL, this objective consists in minimizing the MSE loss:

$$\hat{\theta} \in \arg\min_{\theta} \mathbb{E}_{p(\tau|\pi_\theta,\mathcal{T})}\left[\frac{1}{T}\sum_{t=0}^{T-1} \|\pi_\theta(\mathbf{o}_t, \mathbf{X}_t^{\text{des}}) - \pi_{\theta^*}(\mathbf{o}_t, \mathbf{X}_t^{\text{des}})\|_2^2\right]$$

where $\tau := \{(\mathbf{o}_t, \mathbf{u}_t, \mathbf{X}_t^{\text{des}})\,|t=0,\dots,T\}$ is a $T+1$ step (observation, action, reference) trajectory sampled from the distribution $p(\tau|\pi_\theta, \mathcal{T})$. Such a distribution represents all the possible trajectories induced by the *student* policy π_θ in the deployment environment $\mathcal{T}$. As observed in [24], [62], the presence of uncertainties in $\mathcal{T}$ makes IL challenging, as demonstrations are usually collected in a training domain ($\mathcal{S}$) under a different set of uncertainties ($\mathbb{W}_\mathcal{S} \subseteq \mathbb{W}_\mathcal{T}$, $\mathbb{V}_\mathcal{S} \subseteq \mathbb{V}_\mathcal{T}$) resulting in a different distribution of training data.

Tube and ancillary controller for DA

To overcome these limitations, we design a DA strategy that compensates for the effects of *process and sensing uncertainties* encountered in $\mathcal{T}$.

We do so by extending our previous approach [62] (Chapter 3), named SA, which provided a strategy to efficiently learn a control policy (i.e., $\pi_{\hat{\theta}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$) robust to *process* uncertainty ($\mathbb{W}_\mathcal{T}$). SA recognized that the tube in a RTMPC [3] represents a model of the states that the system may visit when subject to process uncertainties. SA used the tube in [3] to guide the selection of extra states for DA, while the ancillary controller in [3] provided a computationally efficient way to compute corresponding actions, maintaining the system inside the tube for every possible realization of the process uncertainty.

Tube-NeRF

Our new approach, named Tube-NeRF, employs the output feedback variant of RTMPC presented in Section 5.3.1. This has two benefits: (i) the controller appropriately introduces extra conservativeness during demonstration collection to account for sensing uncertainties (via tightened constraint $\bar{\mathbb{X}}$ and $\bar{\mathbb{U}}$ in Eq. (5.6)); and (ii) the tube $\mathbb{Z}$ in Eq. (5.10) additionally captures the effects of *sensing* uncertainty, guiding the generation of extra observations for DA. The new data collection and DA procedure is as follows.

Demonstration collection

We collect demonstrations in $\mathcal{S}$ using the output feedback RTMPC *expert* (Section 5.3.1). Each $T+1$-step demonstration $\bar{\tau}$ consists of:

$$\bar{\tau} = \{(\mathbf{o}_t, \mathbf{u}_t, \bar{\mathbf{u}}_t^*, \bar{\mathbf{x}}_t^*, \mathbf{X}_t^{\text{des}}, \hat{\mathbf{x}}_t) \mid t = 0, \dots, T\}. \tag{5.11}$$

Extra States and Actions for Synthetic Data Generation

For every timestep t in $\bar{\tau}$, we generate $N_{\text{synthetic},t} > 0$ (details on how $N_{\text{synthetic},t}$ is computed are provided in Section 5.3.2) extra (state, action) pairs $(\mathbf{x}_{t,j}^+, \mathbf{u}_{t,j}^+)$, with $j = 1, \dots, N_{\text{synthetic},t}$ by sampling extra states from the tube $\mathbf{x}_{t,j}^+ \in \bar{\mathbf{x}}_t^* \oplus \mathbb{Z}$, and computing the corresponding control action $\mathbf{u}_{t,j}^+$ using Eq. (5.7):

$$\mathbf{u}_{t,j}^+ = \bar{\mathbf{u}}_t^* + \mathbf{K}(\mathbf{x}_{t,j}^+ - \bar{\mathbf{x}}_t^*). \tag{5.12}$$

The resulting $\mathbf{u}_{t,j}^+$ is saturated to ensure that $\mathbf{u}_{t,j}^+ \in \mathbb{U}$.

Synthetic Observations Generation.

To generate the necessary data $\mathbf{o}_{t,j}^+ = (\mathcal{I}_{t,j}^+, \mathbf{o}_{\text{other},t,j}^+)$ input for the sensorimotor policy Eq. (5.1) from the selected states $\mathbf{x}_{j,t}^+$, we employ observation models Eq. (5.4) available for

the expert. In the context of learning a visuomotor policy, we generate synthetic camera images $\mathcal{I}^+_{t,j}$ using an inverse pose estimator $\hat{g}^{-1}_{\text{cam}}$, mapping camera poses $\mathbf{T}_{IC}$ to images $\mathcal{I}$ via $\mathcal{I}^+_{t,j} = \hat{g}^{-1}_{\text{cam}}(\mathbf{T}^+_{IC_{t,j}})$, where $\mathbf{T}_{IC}$ denotes a homogeneous transformation matrix from a world (inertial) frame I to a camera frame C. $\hat{g}^{-1}_{\text{cam}}$ is obtained by generating a NeRF of the environment (discussed in more details in Section 5.4) from the images $\mathcal{I}_0, \ldots, \mathcal{I}_T$ in the collected demonstration $\bar{\tau}$, and by estimating the intrinsic/extrinsic of the camera onboard the robot. The camera poses $\mathbf{T}^+_{IC_{t,j}}$ are obtained from the sampled states $\mathbf{x}^+_{t,j}$, which includes the robot's position and orientation. These are computed as $\mathbf{T}^+_{IC_{t,j}} = \mathbf{T}_{IB}(\mathbf{x}^+_{t,j})\hat{\mathbf{T}}_{BC_{t,j}}$, where $\mathbf{T}_{IB}$ is the transformation from the robot's body frame B to the reference frame I, and $\hat{\mathbf{T}}_{BC_{t,j}}$ are perturbed transformation of the nominal camera extrinsic, where perturbations are introduced to accommodate uncertainties and errors in the extrinsics. Last, the full observations $\mathbf{o}^+_{t,j}$ are obtained by computing $\mathbf{o}^+_{\text{other},t,j}$, using Eq. (5.4) and a selection matrix $\mathbf{S}$:

$$\mathbf{o}^+_{t,j} = (\mathcal{I}^+_{t,j}, \mathbf{o}^+_{\text{other},t,j}), \quad \mathbf{o}^+_{\text{other},t,j} = \mathbf{S}\mathbf{C}\mathbf{x}^+_{t,j}. \tag{5.13}$$

Tube-guided Selection of Extra Real Observations

Beyond guiding the generation of extra synthetic data, we employ the tube of the expert to guide the selection of real-world observations from demonstrations ($\bar{\tau}$) for DA. This procedure is useful at accounting for small imperfections in the NeRF and in the camera-to-robot extrinsic/intrinsic calibrations, further reducing the sim-to-real gap, and providing an avenue to "ground" the synthetic images with real-world data. This involves creating a database of the observations $\mathbf{o}$ in $\bar{\tau}$, indexed by the robot's estimated state $\hat{\mathbf{x}}$, and then selecting $N_{\text{real},t}$ observations at each timestep t inside the tube ($\hat{\mathbf{x}} \in \bar{\mathbf{x}}^*_t \oplus \mathbb{Z}$), adhering to the ratio $N_{\text{real},t}/N_{\text{samples}} \leq \bar{\epsilon}$, where $0 < \bar{\epsilon} \leq 1$ is a user-defined parameter that balances the maximum ratio of real images to synthetic ones, and N_{samples} is the desired number of samples (real and synthetic) per timestep. The corresponding action is obtained from the state associated with the image via the ancillary controller Eq. (5.12). The required quantity

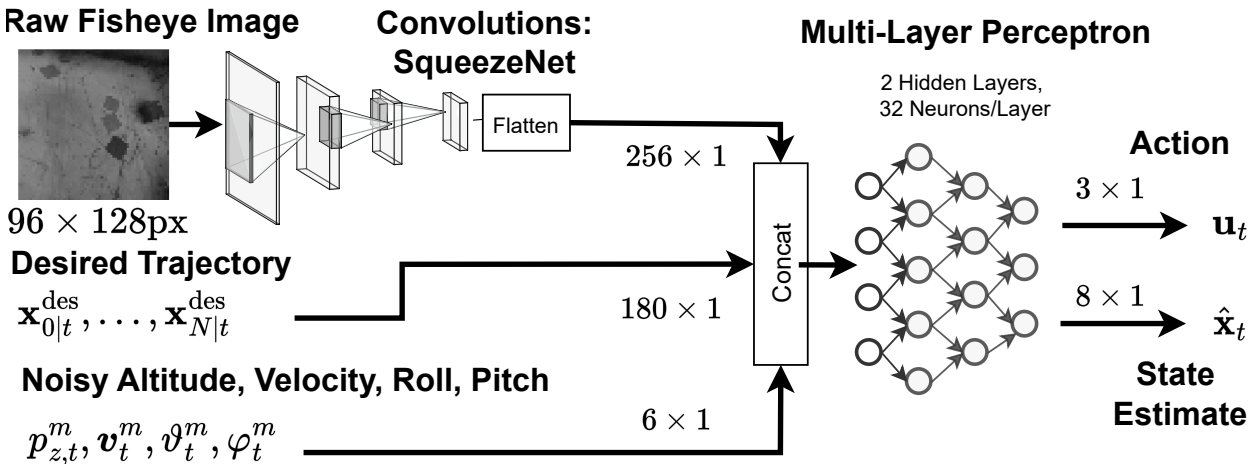


Figure 5.3: Architecture of the employed visuomotor student policy. The policy takes as input a raw camera image, a reference trajectory $\mathbf{x}^{\text{des}}_{0|t}, \ldots, \mathbf{x}^{\text{des}}_{N|t}$ and noisy measurements of the altitude p^m_z, velocity $\boldsymbol{v}_t$ and tilt (roll φ_t, pitch ϑ_t) of the multirotor. It outputs an action $\mathbf{u}_t$, representing a desired roll, pitch, and thrust set-points for the cascaded attitude controller. The policy additionally outputs an estimate of the state $\hat{\mathbf{x}}_t$, which was found useful to promote learning of features relevant to position estimation.

of synthetic samples to generate, as discussed in Section 5.3.2, is calculated using the formula $N_{\text{synthetic},t} = N_{\text{samples}} - N_{\text{real},t}$.

Robustification to Visual Changes

To accommodate changes in brightness and environment, we apply several transformations to both real and synthetic images. These include solarization, adjustments in sharpness, brightness, and gamma, along with the application of Gaussian noise, Gaussian blur, and erasing patches of pixels using a rectangular mask.

5.4 Application to Vision-based Flight

In this section, we provide details on the design of the expert, the student policy, and the NeRF for agile flight.

Task.

We apply our framework to learn to track a figure-eight trajectory (lemniscate, with velocity up to 3.15 m/s lasting 30s), denoted **T1**.

Robot model.

The expert uses the hover-linearized model of a multirotor [7], with state $\mathbf{x} = [{}_I\boldsymbol{p}^T, {}_I\,\boldsymbol{v}^T, {}_{\bar{I}}\,\varphi, {}_{\bar{I}}\,\vartheta]^T$, (position $\boldsymbol{p} \in \mathbb{R}^3$, velocity $\boldsymbol{v} \in \mathbb{R}^3$, roll $\varphi \in \mathbb{R}$, pitch $\vartheta \in \mathbb{R}$, $n_x = 8$), I is an inertial reference frame, while $\bar{I}$ a yaw-fixed frame [7]. The control input $\mathbf{u}_t$ ($n_u = 3$) is desired roll, pitch, and thrust, and it is executed by a cascaded attitude controller.

Measurements.

The multirotor is equipped with a fisheye monocular camera, tilted 45 deg, generating images $\mathcal{I}_t$ (size 128×96 pixels). In addition, we assume available onboard noisy altitude ${}_Ip_z^m \in \mathbb{R}$, velocity ${}_I\boldsymbol{v}^m \in \mathbb{R}^3$ and roll ${}_I\varphi^m$, pitch ${}_I\vartheta^m$ and yaw measurements. This is a common setup in aerial robotics, where noisy altitude and velocity can be obtained, for example, via optical flow and a downward-facing lidar, while roll, pitch, and yaw can be computed from an IMU with a magnetometer, using a complementary filter [131].

Student Policy.

The student policy Eq. (5.1), shown in Fig. 5.3, takes as input an image $\mathcal{I}_t$ from the onboard camera, the reference trajectory $\mathbf{X}_t^{\text{des}}$, and $\mathbf{o}_{\text{other}} \coloneqq [{}_Ip_z^m, {}_I\boldsymbol{v}^{mT}, {}_{\bar{I}}\,\varphi^m, {}_{\bar{I}}\,\vartheta^m]^T$, and it outputs $\mathbf{u}_t$. A Squeezenet [132] is used to map $\mathcal{I}_t$ into a lower-dimensional feature space; it was selected for its performance at a low computational cost. To promote learning of internal features relevant to estimating the robot's state, the output of the policy is augmented to predict the current state $\hat{\mathbf{x}}$ (or $\mathbf{x}^+$ for the augmented data), modifying the training loss accordingly. This output was not used at deployment time, but it was found to improve the performance.

Output Feedback RTMPC and Observer.

The expert uses the defined robot model for predictions, discretized with $T_s = 0.1$ s, and horizon $N = 30$ (3.0 s). $\mathbb{X}$ encodes safety and position limits, while $\mathbb{U}$ captures the maximum actuation capabilities. Process uncertainty in $\mathbb{W}_{\mathcal{T}}$ is assumed to be a bounded external force disturbance with magnitude between to $10 - 19\%$ of the weight of the robot and random direction, close to the physical limits of the platform, and the tube of the expert is computed assuming $\bar{\mathbb{W}}_{\mathcal{T}}$ equal to 20% of the weight of the robot. The state estimator Eq. (5.5) is designed by using as measurement model in Eq. (5.4): $\bar{\mathbf{o}}_t = \mathbf{x}_t + \mathbf{v}_t$ where we assume $\mathbf{v}_t \sim \mathcal{N}(\mathbf{0}_{n_x}, [\boldsymbol{\sigma}_{\text{cam}}^T, \boldsymbol{\sigma}_{\text{other}}^T]^T)$. We therefore set $\mathbb{V}_{\mathcal{T}} = \{\mathbf{v}_t \mid \|\mathbf{v}\|_\infty \leq 3 \left[\boldsymbol{\sigma}_{\text{cam}}^T, \boldsymbol{\sigma}_{\text{other}}^T\right]^T\}$, with $3\boldsymbol{\sigma}_{\text{cam}} = [0.6, 0.6]^T$ (units in m) and $3\boldsymbol{\sigma}_{\text{other}} = [0.4, 0.2, 0.2, 0.2, 0.05, 0.05]^T$ (units in m for altitude, m/s for velocity, and rad for tilt). These conservative but realistic parameters are based on prior knowledge of the worst-case performance of vision-based estimators in our relatively feature-poor flight space. The observer gain matrix $\mathbf{L}$ is computed by assuming fast state estimation dynamics, (poles of $\mathbf{A} - \mathbf{LC}$ at 30.0 rad/s).

Procedure to Generate the NeRF of the Environment

(i) **Dataset:** A NeRF of the environment, the MIT-Highbay, is generated from about 100 images collected during a single real-world demonstration of the figure-eight trajectory (**T1**) intended for learning, utilizing full-resolution images (640×480 pixels) from the fisheye camera onboard the Qualcomm Snapdragon Flight Pro board of our UAV.[1]

(ii) **Extrinsic/Intrinsic:** The extrinsic and intrinsic parameters of the camera are estimated from the dataset using structure-from-motion (COLMAP [133], RADTAN camera model).

(iii) **Frame Alignment:** The scale and homogeneous transformation aligning the reference frame used by the COLMAP with the reference frame used by the robot's state estimator

[1] https://developer.qualcomm.com/hardware/qualcomm-flight-pro

are determined via the trajectory alignment tool EVO [134]. This enables the integration of the NeRF as an image rendering tool in a simulation/DA framework.

(iv) **NeRF Training:** Instant-NGP [88] is utilized to train the NeRF. The scaling of the Axis-Aligned Bounding Box used by the Instant-NGP is manually adjusted to ensure that the reconstruction is photorealistic in the largest possible volume.

(v) **Images rendering for DA:** Novel images are rendered using the same camera intrinsics identified by COLMAP. The camera extrinsics, mapping from the robot's IMU to the optical surface, are determined via Kalibr [135], using an ad-hoc dataset. An example of an image from the NeRF is shown in Fig. 5.1.

5.5 Evaluation

5.5.1 Evaluation in Simulation

Here we numerically evaluate the efficiency (training time, number of demonstrations), robustness (average episode length before violating a state constraint, success rate) and performance (*expert gap*, the relative error between the stage cost $\sum_t \|\mathbf{e}_t\|^2_{\mathbf{Q}} + \|\mathbf{u}_t\|^2_{\mathbf{R}}$ of the expert and the one of the policy) of Tube-NeRF. Note that the number of demonstrations provides a metric useful not only to estimate real-world data collection efforts, but also the number of environment interactions required in simulation which, depending on the simulation environment considered, may be computationally costly (e.g., in fluid-dynamic simulations). We use PyBullet to simulate realistic full nonlinear multirotor dynamics [7], rendering images using the NeRF obtained in Section 5.4 – combined with realistic dynamics, the NeRF provides a convenient framework for training and numerical evaluations of policies. The considered task consists of following the figure-eight trajectory **T1** (lemniscate, length: 300 steps) used in Section 5.4, starting from $\mathbf{x}_0 \sim \text{Uniform}(-0.1, 0.1) \in \mathbb{R}^8$, without violating state constraints. The policies are deployed in two target environments, one with sensing

Table 5.1: Robustness, performance, and demonstration efficiency of IL methods for visuomotor policy learning. An approach is *easy* if it does not require disturbances during the demonstration collection, and it is *safe* if it does not violate state constraints (e.g., wall collision) during training. *Success Rate* is the percentage of trajectories that do no violate state constraints. *Performance* is the relative error between the cost of the trajectory generated by output feedback RTMPC (expert) and the ones from the policy. *Demonstration efficiency* is the number of demonstrations required to achieve at least 70% *success rate*. Performance and robustness of the baselines are evaluated after 150 demonstrations, while Tube-NeRF methods after *only* 2 demonstrations.

Method		Training		Robustness succ. rate (%)		Performance expert gap (%)		Demonstration Efficiency	
Robustif.	Imitation	Easy	Safe	Noise	Noise, Wind	Noise	Noise, Wind	Noise	Noise, Wind
-	BC	**Yes**	**Yes**	95.5	13.0	83.1	21.3	30	-
	DAgger	**Yes**	No	60.5	45.5	138.6	43.0	-	-
DR	BC	No	**Yes**	80.0	46.5	72.5	59.2	20	90
	DAgger	No	No	67.0	62.0	82.2	59.6	90	-
TN-100	BC	**Yes**	**Yes**	**100.0**	**100.0**	**8.3**	**9.1**	**1**	**1**
	DAgger	**Yes**	**Yes**	**100.0**	**100.0**	14.4	9.7	**1**	**1**
TN-50	BC	**Yes**	**Yes**	**100.0**	**100.0**	18.4	11.8	**1**	**1**
	DAgger	**Yes**	**Yes**	**100.0**	**100.0**	15.4	11.2	**1**	**1**

noise affecting the measurements $\mathbf{o}_{\text{other}}$ (the noise is Gaussian distributed with parameters as defined in Section 5.4) and one that additionally presents wind disturbances, sampled from $\mathbb{W}_{\mathcal{T}}$ (also with bounds as defined in Section 5.4).

Method and Baselines.

We apply Tube-NeRF to BC and DAgger, comparing their performance without any DA; Tube-NeRF$-N_{\text{samples}}$, with $N_{\text{samples}} = \{50, 100\}$, denotes the number of observation-action samples generated for every timestep by uniformly sampling states inside the tube. We additionally combine BC and DAgger with DR by applying, during demonstration collection, an external force disturbance sampled from $\mathbb{W}_{\mathcal{T}}$. We set β, the hyperparameter of DAgger controlling the probability of using actions from the expert instead of the learner policy, to be $\beta = 1$ for the first set of demonstrations, and $\beta = 0$ otherwise.

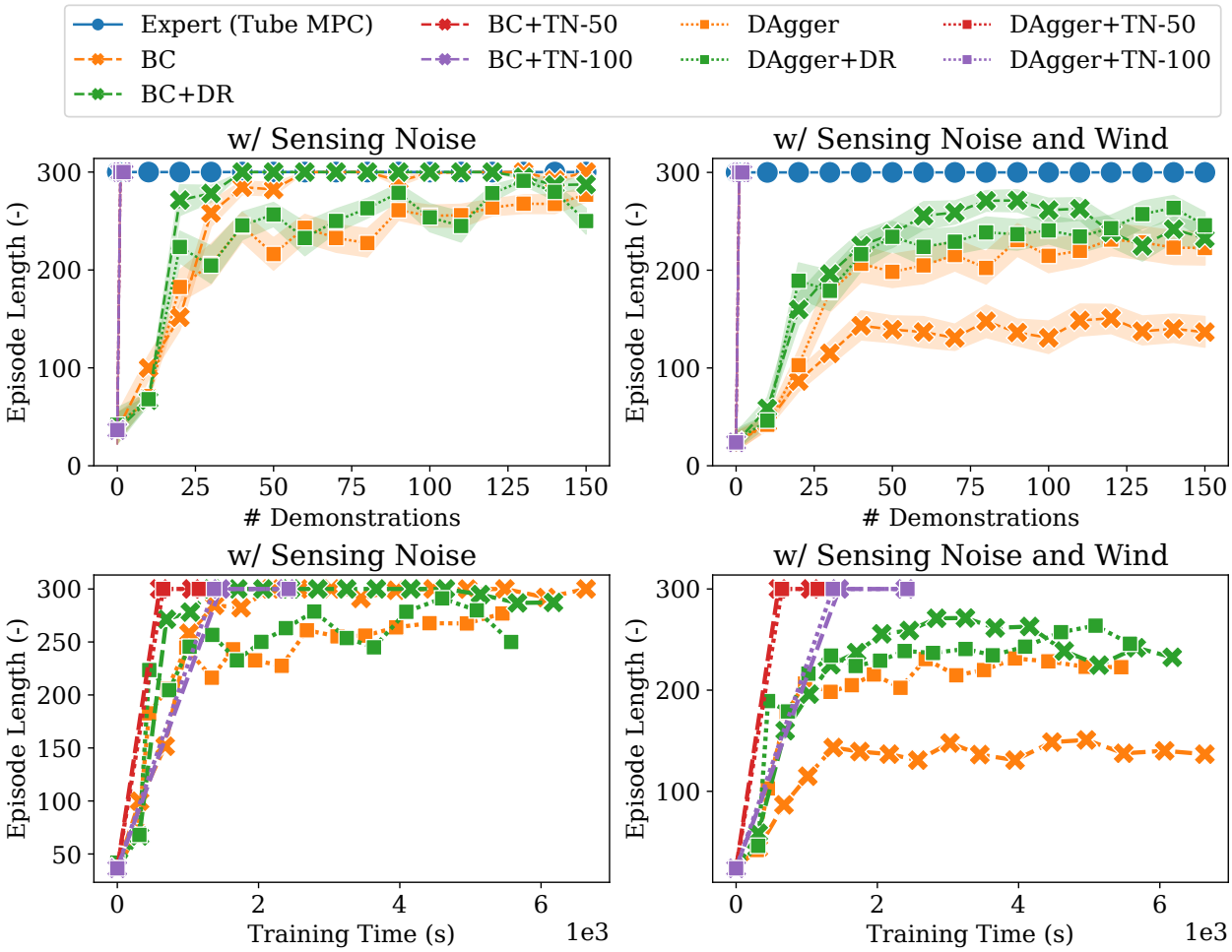


Figure 5.4: Episode length (timestep before a state constraint violation, up to 300) vs. number of demonstrations collected from the expert, and vs the training time (the time required to collect such demonstrations in simulation, and to train the policy). This shows that policies trained with Tube-NeRF (TN) archive full episode length after a single demonstration, and require less than half of the training time than the best-performing baselines (DR-based methods). Note that the lines of Tube-NeRF-based approaches vs the number of demonstrations overlap. Shaded areas are 95% confidence intervals. Note that to focus our study on the effects of process uncertainties and sensing noise, we do not apply visual changes to the environment, nor the robustification to visual changes (Section 5.3.2). Evaluations across 10 seeds, 10 times per seed.

Evaluation Details.

For every method, we: (i) collect K new demonstrations ($K = 1$ for Tube-NeRF, $K = 10$ otherwise) via the output feedback RTMPC expert and the state estimator; (ii) update a student policy using all the demonstrations collected so far. Note that policies are trained

for 50 epochs with the ADAM optimizer, learning rate 0.001, batch size 32, and terminating training if the loss does not decrease within 7 epochs. Tube-NeRF uses only the newly collected demonstrations and the corresponding augmented data to update the previously trained policy; (iii) evaluate the obtained policy in the considered target environments, for 10 times each, starting from different initial states; (iv) repeat from (i). Note that in our comparison the environment steps at its highest possible rate (simulation time is faster than wall-clock time), providing an advantage in terms of data collection time to those methods that require collecting a large number of demonstrations (our baselines).

Results.

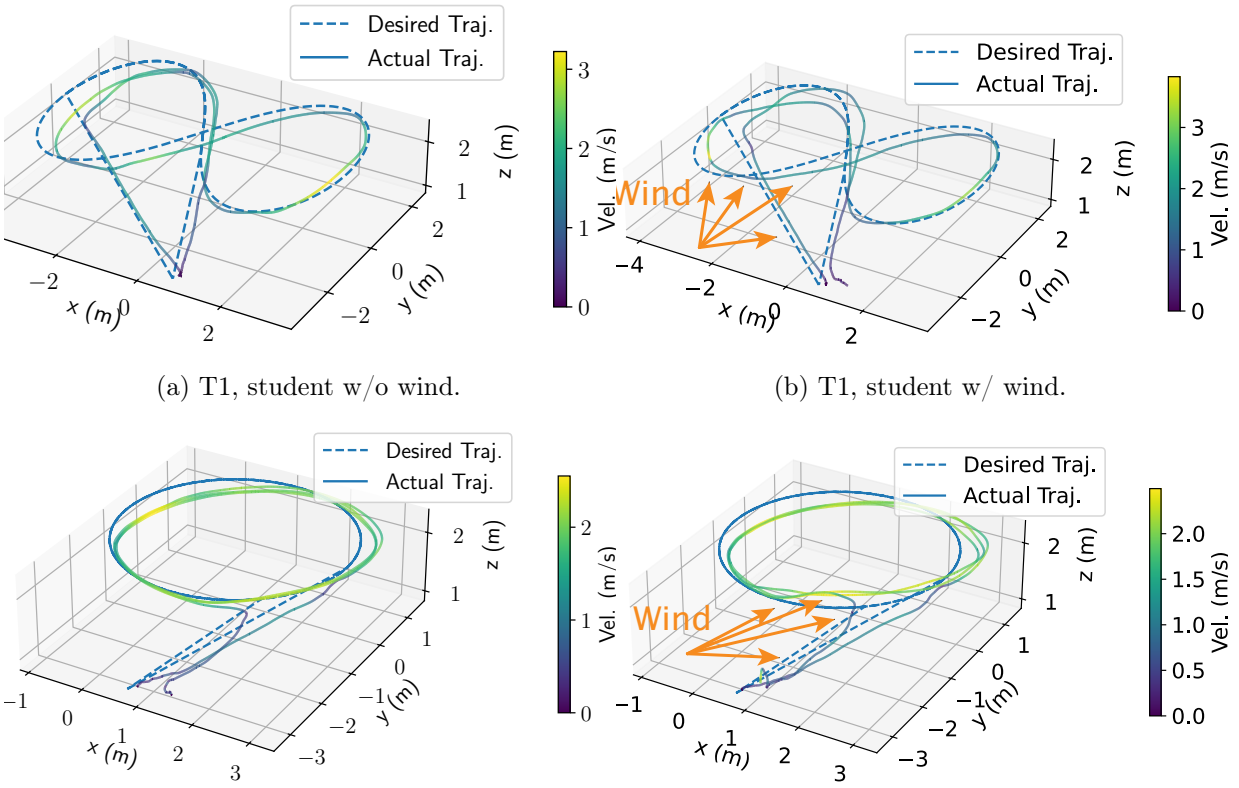
Fig. 5.4 highlights that all Tube-NeRF methods, combined with either DAgger or BC, can achieve complete robustness (full episode length) under combined sensing and process uncertainties after a single demonstration. The baseline approaches require 20-30 demonstrations to achieve a full episode length in the environment without wind, and the best-performing baselines (methods with DR) require about 80 demonstrations to achieve their top episode length in the more challenging environment. Similarly, Tube-NeRF requires less than *half* training time than DR-based methods to achieve higher robustness in this more challenging environment, and reducing the number of samples (e.g, Tube-NeRF-50) can further improve the training time. The time to generate the NeRF, not shown in Fig. 4, was approximately 5 minutes (20000 epochs on an `RTX 3090 GPU`). Even accounting for this time, TN is significantly faster than collecting real-world demonstrations (the 80 demonstrations required by DR correspond to 40 minutes of real-world data, followed by the time to train the policy). In addition, if the NeRF is combined with a simulation of the dynamics of the robot (creating a photo-realistic simulator), our DA strategy still provides benefits in terms of performance and training time.. Table 5.1 additionally highlights the small gap of Tube-NeRF policies from the expert in terms of tracking performance (*expert gap*), and shows that increasing the number of samples (e.g., Tube-NeRF-100) benefits performance.

Table 5.2: Position Root Mean Squared (RMS) Tracking Errors when following a figure-eight (lemniscate) trajectory (T1) and a circular trajectory (T2). This highlights the low errors of the sensorimotor policy, comparable to the ones of the expert whose position is estimated via a motion capture system. We apply wind disturbances with a leaf blower (With wind), and extra sensing noise in the altitude, velocity and orientation input of the policy (High noise). Note that T2 has been obtained without collecting corresponding images from a real-world demonstration, and heavily relies on NeRF data (see Fig. 5.7). Experiments repeated 3 times, except T2 with wind low noise (2 times), and slung-load (once).

	RMSE (m, ↓) for T1: Lemniscate (30s, real-world demo.)							**RMSE (m, ↓) for T2: Circle (30s, No real-world demo.)**			
	Expert + Motion Capture		**Student**					**Student**			
	No wind	With wind	No wind		With wind		Slung load	No wind		With wind	
	Low noise	Low noise	Low noise	High noise	Low noise	High noise		Low noise	High noise	Low noise	High noise
x	0.19 ± 0.00	0.20 ± 0.01	0.30 ± 0.01	0.33 ± 0.00	0.28 ± 0.02	0.40 ± 0.01	0.21	0.33 ± 0.01	0.34 ± 0.01	0.33 ± 0.00	0.34 ± 0.04
y	0.17 ± 0.01	0.21 ± 0.01	0.16 ± 0.01	0.21 ± 0.01	0.26 ± 0.01	0.22 ± 0.03	0.44	0.16 ± 0.01	0.27 ± 0.01	0.22 ± 0.03	0.28 ± 0.01
z	0.12 ± 0.01	0.13 ± 0.02	0.11 ± 0.01	0.20 ± 0.02	0.17 ± 0.01	0.17 ± 0.01	0.06	0.11 ± 0.08	0.15 ± 0.06	0.07 ± 0.00	0.14 ± 0.02

5.5.2 Flight Experiments

We now validate the data efficiency of Tube-NeRF highlighted in our numerical analysis by evaluating the obtained policies in real-world experiments. We do so by deploying them on an NVIDIA Jetson TX2 (at up to 200 Hz, TensorRT) on the MIT-ACL multirotor. The policies take as input the fisheye images generated at 30 or 60 Hz by the onboard camera. The altitude, velocity and roll/pitch inputs that constitute $\mathbf{o}_{\text{other},t}$ are, for simplicity, obtained from the onboard estimator (a filter fusing IMU with poses from a motion capture system), corrupted with additive noise (zero-mean Gaussian, with parameters as defined in Section 5.4) in the scenarios denoted as "high noise". We remark that no information on the horizontal position of the UAV is provided to the policy, and horizontal localization must be performed from images. We consider two tasks, tracking the lemniscate trajectory (**T1**), and tracking a new circular trajectory (denoted **T2**, velocity up to 2.0 m/s, duration of 30 s). No real-world images have been collected for **T2**, therefore this task is useful to stress-test the novel-view synthesis abilities of the approach, using the NeRF and the nonlinear simulated robot dynamics as a simulation framework. **Training.** We train one policy for each task, using a single task demonstration collected with `DAgger+Tube-NeRF-100` in our NeRF-based simulated environment. During DA, we try to achieve an equal amount between synthetic



(a) T1, student w/o wind.

(b) T1, student w/ wind.

(c) T2, student w/o wind.

(d) T2, student w/ wind.

Figure 5.5: Qualitative evaluation in experiments, highlighting the high velocity and the challenging 3D motion that the student policy can execute under uncertainties. The policy runs onboard the multirotors with an average inference time of about 1.5 ms. The eight-shaped trajectory (T1) is the same trajectory used to collect the real-world images and to generate the NeRF employed for Data Augmentation, while the circular trajectory is entirely trained without having collected in the real world a task demonstration. This demonstrates that our approach can be used to train vision-based policies (T2) for which no real-world demonstration has been collected, still achieving low tracking errors (comparable to T1).

images (from the NeRF) and real ones (from the database), setting $\bar{\epsilon} = 0.5$. Fig. 5.7 reports the number of sampled real images from the database, highlighting that the tube is useful at guiding the selection of real images, but that synthetic images are a key part of the DA strategy, as **T2** presents multiple parts without any real image available.

Performance under Uncertainties.

Fig. 5.5 and Table 5.2 show the trajectory tracking performance of the learned policy under a variety of real-world uncertainties. Those uncertainties include (i) model errors, such as poorly

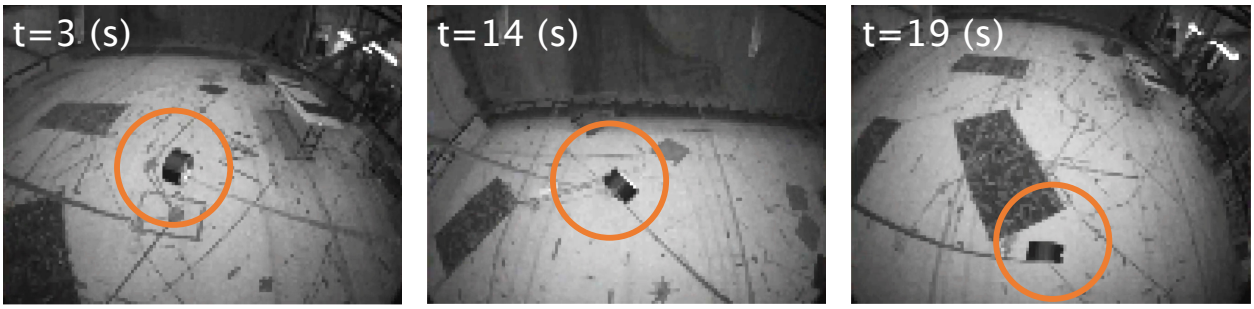


Figure 5.6: Robustness to visual uncertainties: we show images captured by the onboard camera while using the proposed sensorimotor policy for localization and tracking of **T1**, with a slung load (tape roll, 0.2 Kg) attached to the robot. The slung load (circled) repeatedly enters the field of view of the onboard camera, without however compromising the success of the maneuver. We hypothesize that randomly deleting patches of pixels during training (Sec. 5.3.2) contributes to achieving robustness to this disturbance.

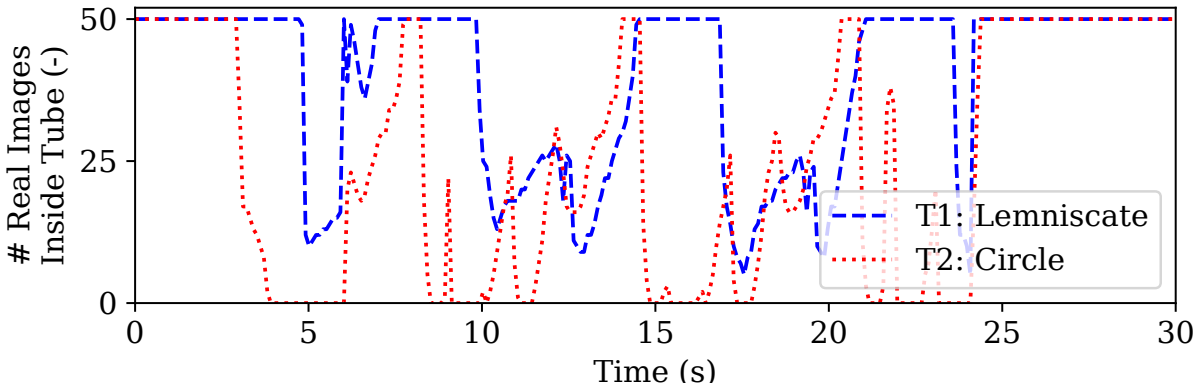


Figure 5.7: Number of real images sampled from the tube to perform data augmentation using `Tube-NeRF-100`, as a function of time in the considered trajectory. The considered trajectories are a Lemniscate trajectory (T1), which is the same as the one executed for real-world data collection, and a Circle trajectory (T2), which is different from the one executed for data collection. The results highlight that (1) the tube can be used to guide the selection of real-world images for data augmentation and (2) the synthetic images from the NeRF are a key component of our data augmentation strategy, as there are multiple segments of the circular trajectory (T2) where no real images are presents, but the sensorimotor policy successfully controls the robot in the real-world experiments.

known drag and thrust to voltage mappings; (ii) wind disturbances, applied via a leaf-blower, (iii) sensing uncertainties (additive Gaussian noise to the partial state measurements), and (iv) visual uncertainties, produced by attaching a slung-load that repeatedly enters the field of view of the camera, as shown in Fig. 5.6. The video linked in Table 1.1 shows additional experiments. These results highlight that (a) policies trained after a single demonstration collected in

Table 5.3: **Time (ms) required to generated a new action for the output feedback RTMPC only (Expert) and the proposed sensorimotor NN policy (Policy)**. Our policy is **9.8** faster (offboard) and **5.6** times faster (onboard) than the expert. We note that the computational cost of the expert is only based on computing actions from states, while the policy additionally performs localization (actions from images); therefore the reported data represents a **lower bound** on the cost reductions introduced by the policy. For a fair comparison, the onboard expert uses an highly-optimized C/C++ implementation. The offboard computer, used for our numerical evaluation and training, is an `AMD Threadripper 3960X` with two RTX 3090. The onboard implementation (optimized for speed for both the policy and the expert) uses an `NVIDIA Jetson TX2`.

			Time (ms)			
Computer	**Method**	**Setup**	**Mean**	**SD**	**Min**	**Max**
Offboard	Expert (MPC only)	CVXPY[118]/OSQP	30.3	41.5	4.8	244
	Policy (MPC+vision)	PyTorch	**3.1**	**0.0**	**0.8**	**1.0**
Onboard	Expert (MPC only)	CVXGEN [119]	8.4	1.4	4.5	15.9
	Policy (MPC+vision)	ONNX/TensorRT	**1.5**	**0.2**	**1.4**	**9.2**

our NeRF-based simulator using Tube-NeRF are robust to a variety of uncertainties while maintaining tracking errors comparable to the ones of the expert (Table 5.2, Fig. 5.2), while reaching velocities up to 3.5 m/s, and even though the expert localizes using a motion capture system, while the policy uses images from the onboard camera to obtain its horizontal position. In addition, (b) our method enables learning of vision-based policies for which no real-world task demonstration has been collected, effectively acting as a simulation framework, as shown by the successful tracking of **T2**, which relied entirely on synthetic training data for large portions of the trajectory (Fig. 5.7), and was obtained using a single demonstration in the NeRF-based simulator. Due to the limited robustness achieved in simulation (Table 5.1), we do not deploy the baselines on the real robot.

Efficiency at Deployment and Latency.

Table 5.3 shows that onboard the policy requires on average only **1.5** ms to compute a new action from an image, being at least **5.6** faster than a highly-optimized (C/C++) expert. Note that the reported computational cost of the expert is based on the cost of control only (no state estimation), therefore the actual computational cost reduction provided by

the policy is even larger. Online, image capture (independent of our method, nominal light conditions) has a latency of about 15ms, while image pre-processing and transfer to the TX2 takes less than 2ms.

Sensitivity to Uncertainties in the Visual Input.

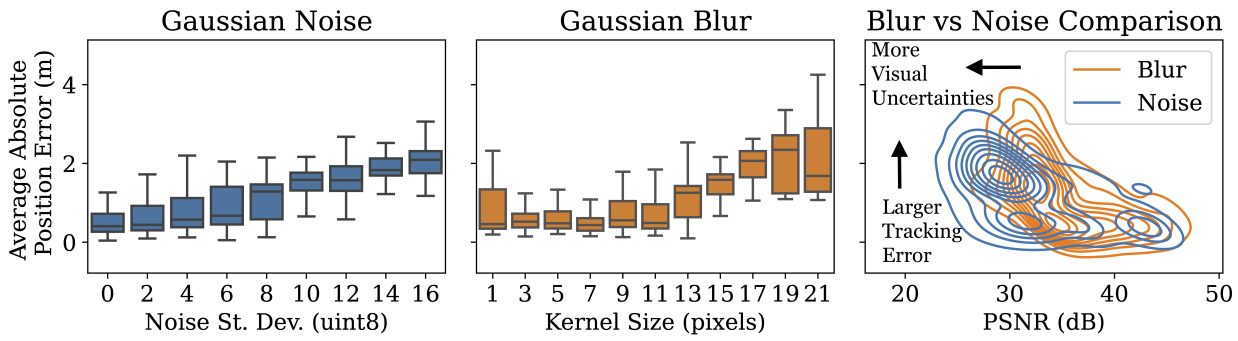


Figure 5.8: Position tracking error under wind disturbances for T1 in simulation as a function of different types and magnitude of noise in the visual input. PSNR is the Peak Signal to Noise Ratio, and lower PSNR denotes a larger amount of noise corrupting the image. The maximum noise level shown corresponds to to 0% success rate. Contour lines in the Comparison plot show the density of the errors.

We study the closed-loop performance of the policy under different types of uncertainties in the visual input by monitoring the position tracking error (in simulation, tracking T1, under wind) as a function of different types and magnitudes of noise applied to the images rendered by the NeRF and input to the policy. We consider 1) Gaussian noise, representing the presence of high-frequency disturbances/uncertainties such as new visual features in the environment, and 2) Gaussian blur, capturing visual changes that reduce high-frequency features (e.g., weather changes, and/or the effects of using a low-quality NeRF for training). The results in Fig. 5.8 highlight that 1) the visual input plays a key role in the output of the policy and, while our approach is not specifically designed for environments with rapidly changing visual appearance, it also shows 2) the overall robustness of the policy to visual uncertainties. Last, the comparison of the effects of the two types of noise highlights that 3) the policy has lower sensitivity to the presence of new high-frequency visual uncertainties,

as the position errors grow slower when more Gaussian noise is applied (lower PSNR) than for Gaussian blur. Note that the y axis of all the plots corresponds to the Average Absolute Position Error (m).

5.6 Summary

This Chapter presented Tube-NeRF, an efficient IL of robust end-to-end visuomotor policies that achieve real-world robustness in trajectory tracking from images on a multirotor. Tube-NeRF leveraged output feedback RTMPC to collect demonstrations that account for process and sensing uncertainties. In addition, properties of the controller guided a DA procedure that used a combination of a database of real-world images, a NeRF of the environment, and randomization procedures in image space to obtain novel relevant views. For each extra sensorial input, a corresponding action was *efficiently* computed using an ancillary controller, an integral part of the control framework. Tube-NeRF was tailored to localization and control of a multirotor, numerically outperforming IL baselines in robustness, data, and training-time efficiency. Real-world experiments validated the numerical finding, achieving accurate trajectory tracking with an onboard policy (1.5 ms average inference time) that relied entirely on images to infer the horizontal position of the robot, despite challenging 3D motion and uncertainties.

Chapter 6

Rapid Adaptation

6.1 Overview

In this Chapter we present Sampling Augmentation for Motion Adaptation (SAMA), a strategy to efficiently generate a *robust and adaptive* DNN policy using RTMPC, enabling the policy to compensate for large uncertainties and disturbances. The key idea of our approach, summarized in Fig. 6.1 and Fig. 6.2, is to extend the efficient IL strategy SA [62], presented in Chapter 3, with an adaptation scheme inspired by the recently proposed adaptive policy learning method RMA [41]. RMA uses RL to train in simulation a fast DNN policy whose inputs include a learned low-dimensional representation of the model/environment parameters experienced during training. At deployment, this low-dimensional representation is estimated online, triggering adaptation. RMA policies have demonstrated impressive adaptation and generalization performance on a variety of robots/conditions [41]–[44]. Similar to RMA, in our work we include to the inputs of the learned policy a low-dimensional representation of model/environment parameters that could be encountered at deployment. These parameters are experienced during demonstration collection from MPC and can be efficiently estimated online, enabling adaptation. Unlike RMA, however, we bypass the challenges associated with RL, such as reward selection and tuning, via an efficient IL procedure using our MPC-guided

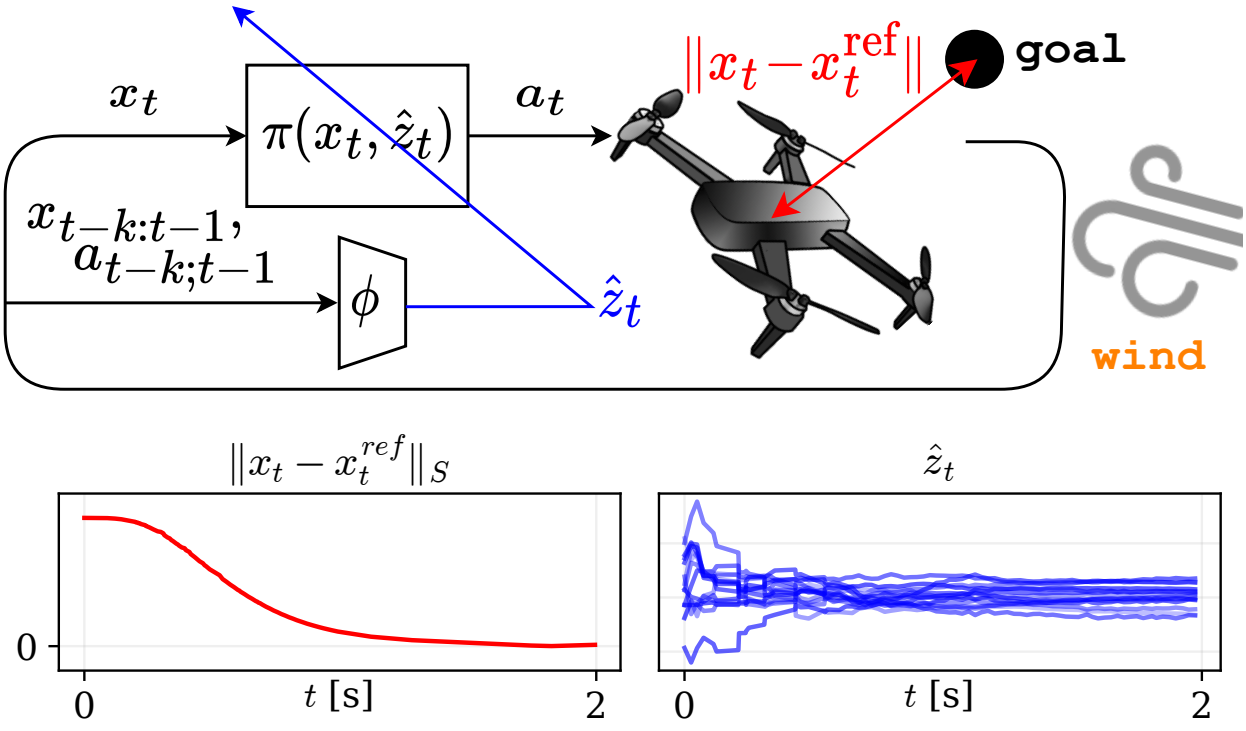


Figure 6.1: Diagram of our combined *position and attitude* controller, efficiently learned from Robust Tube MPC using Imitation Learning. The *adaptation module* ϕ uses a history of k states $x_{t-k:t-1}$ and actions $a_{t-k:t-1}$ to estimate the *extrinsics* $\hat{z}_t$, a low dimensional representation of the environment parameters. This estimate allows the *base policy* (controller) π to adapt to various changes in the system, reducing error in the face of model/environment uncertainties. Selection matrix S selects the position axis along the direction of the wind.

data augmentation strategy [62], presented in Chapter 3. We tailor the approach to the challenging task of *trajectory tracking* for a multirotor, designing a policy that controls both *position and attitude* of the robot and that is capable of adapting to uncertainties in the translational and rotational dynamics. Our evaluation, performed under challenging model errors and disturbances in a simulation environment, demonstrates rapid adaptation to in- and out-of-distribution uncertainties while tracking agile trajectories with top speeds of 3.2 m/s, using an adaptive policy that is learned in 1.3 hours. This differs from prior RMA work for quadrotors [44], where the focus of adaptation is only on *attitude* control during quasi-static trajectories. Additionally, SAMA shows comparable performance to RTMPC combined with a high-performance, state-of-the-art but significantly more computationally

expensive disturbance observer (DO).

6.2 Preliminaries

Our approach leverages and modifies two key algorithms, RTMPC [136] and RMA [41]. RTMPC has been introduced in Chapter 3, while RMA is summarized in the following parts for completeness.

6.2.1 Rapid Motor Adaptation (RMA)

RMA [41] enables learning of adaptive control policies in simulation using model-free RL. The key idea of RMA is to learn a policy composed of a base policy π and an adaptation module ϕ. The base policy is denoted as:

$$\boldsymbol{a}_t = \pi(\mathbf{x}_t, \boldsymbol{z}_t), \tag{6.1}$$

and takes as input the current state $\mathbf{x}_t \in \mathbb{R}^{n_x}$ and an *extrinsics* vector $\boldsymbol{z}_t \in \mathbb{R}^{n_z}$, outputting commanded actions $\boldsymbol{a}_t \in \mathbb{R}^{n_a}$. Key to this method is the extrinsics vector $\boldsymbol{z}_t$, a low-dimensional representation of an environment vector $\boldsymbol{e}_t \in \mathbb{R}^{n_e}$, which captures all the possible parameters/disturbances that may vary at deployment time (i.e., mass, drag, external disturbances, ...), and towards which the policy should be able to adapt. However, because $\boldsymbol{e}_t$ is not directly accessible in the real world, it is not possible to directly compute $\boldsymbol{z}_t$ at deployment time. Instead, RMA produces an estimate $\hat{\boldsymbol{z}}_t$ of $\boldsymbol{z}_t$ via an *adaptation module* ϕ:

$$\hat{\boldsymbol{z}}_t = \phi(\mathbf{x}_{t-k:t-1}, \boldsymbol{a}_{t-k;t-1}), \tag{6.2}$$

whose input is a history of the k past states $\mathbf{x}_{t-k:t-1}$ and actions $\boldsymbol{a}_{t-k:t-1}$ at deployment, enabling rapid adaptation.

Learning ϕ and π is divided in two *phases*.

Phase 1: Base Policy and Environment Factor Encoder

In *Phase 1*, RMA trains the base policy π and an intermediate policy, the environment factor encoder μ:

$$\boldsymbol{z}_t = \mu(\boldsymbol{e}_t) \tag{6.3}$$

which takes as input the vector $\boldsymbol{e}_t$ and produces the *extrinsics* vector $\boldsymbol{z}_t$. The two modules are trained using model-free RL (e.g., PPO [137]) in a simulation environment subject to instances $\boldsymbol{e}_t$ of the possible model/environment uncertainties, and leveraging a reward function that captures the desired control objective. Using this procedure, RL discovers policies that can perform well under disturbances/uncertainties.

Phase 2: Adaptation Module

The adaptation module ϕ is obtained by generating a dataset of state-action histories in simulation via

$$\hat{\boldsymbol{z}}_t = \phi(\mathbf{x}_{t-k:t-1}, \boldsymbol{a}_{t-k;t-1}), \tag{6.4}$$

$$\boldsymbol{a}_t = \pi(\mathbf{x}_t, \hat{\boldsymbol{z}}_t). \tag{6.5}$$

Because we have access to the ground truth environment parameters $\boldsymbol{e}_t$ in simulation, RMA can compute $\boldsymbol{z}_t$ at every timestep using Eq. (6.3), allowing us to train ϕ via supervised regression, minimizing the MSE loss $\|\boldsymbol{z}_t - \hat{\boldsymbol{z}}_t\|^2$. This is done iteratively, by alternating the collection of on-policy rollouts with updates of ϕ.

6.3 Approach

The proposed approach, summarized in Fig. 6.2, consists in a *three phase* policy learning procedure:

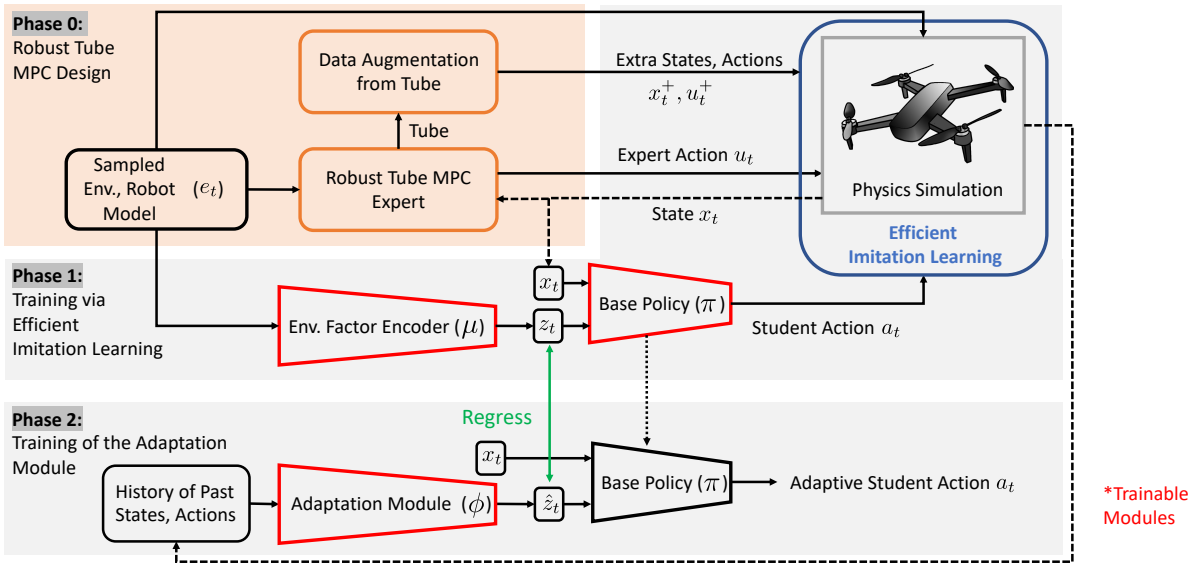


Figure 6.2: Schematic representation of Sampling Augmentation for Motion Adaptation (SAMA), our proposed approach for efficient learning of adaptive polices from MPC. The key idea of SAMA consists in leveraging an efficient Imitation Learning strategy, Sampling Augmentation (SA) [62], to collect demonstrations and perform data augmentation using a Robust Tube MPC. This efficiently generated data is used to train a student policy conditioned on a latent representation $\boldsymbol{z}_t$ of environment and robot parameters $\boldsymbol{e}_t$. Following the Rapid Motor Adaptation (RMA) [41] procedure, we then train an adaptation module that can produce an estimate $\hat{\boldsymbol{z}}_t$ of these environment parameters from a sequence of past states and actions. This approach enables efficient learning of a robust, adaptive policy from MPC without leveraging RL, avoiding any reward tuning and making use of available priors on the model of the robot.

6.3.1 Phase 0: Robust Tube MPC Design

As in RMA, we train our policies in a simulation environment implementing the full nonlinear dynamic model of the robot/environment, with parameters (model/environment uncertainties, disturbances...) captured by the environment parameter vector $\boldsymbol{e}$. At each timestep t, each entry in $\boldsymbol{e}$ may change with some probability p, with entries changing independently of each other (see Table 6.1 for more details on the distributions of $\boldsymbol{e}$ in the train and test environments). Whenever $\boldsymbol{e}$ changes, we update the RTMPC as follows. First, since the controller uses linear system dynamics, for a given environment parameter vector $\boldsymbol{e}_t$ at time t we compute a discrete-time linear system by discretizing and linearizing the full nonlinear system dynamics, obtaining:

$$\mathbf{x}_{t+1} = \mathbf{A}(\boldsymbol{e}_t)\mathbf{x}_t + \mathbf{B}(\boldsymbol{e}_t)\mathbf{u}_t. \tag{6.6}$$

The linearization is performed by assuming a given desired operating point; for our multirotor-based evaluation, this point corresponds to the hover condition.

Second, the feedback gain $\mathbf{K}_t$ for the ancillary controller is updated by solving the infinite horizon, discrete-time LQR problem using $(\mathbf{A}(\boldsymbol{e}_t), \mathbf{B}(\boldsymbol{e}_t), \mathbf{Q}_x, \mathbf{R}_u)$, leaving the tuning weights $\mathbf{Q}_x$, $\mathbf{R}_u$ fixed. Last, we compute the robust control invariant set $\mathbb{Z}_t$ employed by RTMPC from the resulting $\mathbf{K}_t$, $\mathbf{A}(\boldsymbol{e}_t)$, $\mathbf{B}(\boldsymbol{e}_t)$, and a given $\mathbb{W}$. Due to the computational cost of *precisely* computing $\mathbb{Z}_t$ (from $\mathbf{K}_t$, $\mathbf{A}(\boldsymbol{e}_t)$, $\mathbf{B}(\boldsymbol{e}_t)$, and $\mathbb{W}$), we generate an outer-approximation of $\mathbb{Z}_t$ via Monte Carlo simulation. This is done by computing the axis-aligned bounding box of the state deviations obtained by perturbing the closed loop system $\mathbf{A}_{\mathbf{K}_t}$ with randomly sampled instances of $\mathbf{w} \in \mathbb{W}$. The set $\mathbb{W}$ is designed to capture the effects of linearization and discretization errors, as well as errors that are introduced by the learning/parameter estimation procedure.

6.3.2 Phase 1: Base Policy and Environment Factor Encoder Learning via Efficient Imitation

We now describe the procedure to efficiently learn a base policy π and an environment factor encoder μ in simulation. Similar to RMA, our base policy takes as input the current state $\mathbf{x}_t$, an extrinsics vector $\boldsymbol{z}_t$ and, different from RMA, a reference trajectory $\mathbf{X}_t^{\text{des}}$ (as defined in Chapter 3). It outputs a vector of actuator commands $\boldsymbol{a}_t$. As in RMA, the extrinsics vector $\boldsymbol{z}_t$ represents a low dimensional encoding of the environment parameters $\boldsymbol{e}_t$, and it is generated in this phase by the environment factor encoder μ:

$$\begin{aligned} \boldsymbol{z}_t &= \mu(\boldsymbol{e}_t) \\ \boldsymbol{a}_t &= \pi(\mathbf{x}_t, \boldsymbol{z}_t, \mathbf{X}_t^{\text{des}}). \end{aligned} \tag{6.7}$$

We jointly train the base policy π and environment encoder μ end-to-end. However, unlike RMA, we do not use RL, but demonstrations collected from RTMPC in combination with DAgger [21], treating the RTMPC as an *expert*, and the policy in Eq. (6.7) as a *student*. More specifically, at every timestep, given the environment parameters vector $\boldsymbol{e}_t$, the current state of the robot $\mathbf{x}_t$, and the reference trajectory $\mathbf{X}_t^{\text{des}}$, the expert generates a control action $\mathbf{u}_t$ by first computing a *safe* reference plan $\bar{\mathbf{X}}_t^*, \bar{\mathbf{U}}_t^*$, and then by using the ancillary controller, as described in Chapter 3. The obtained control action is applied to the simulation with a probability β, otherwise the applied control action is queried from the student (Eq. (6.7)). At every timestep, we store in a dataset $\mathcal{D}$ the (input, output) pairs $(\{\mathbf{X}_t^{\text{des}}, \mathbf{x}_t, \boldsymbol{e}_t\}, \mathbf{u}_t)$.

Tube-guided Data Augmentation. We extend the data augmentation strategy presented in Chapter 3 to augment the collected demonstrations with extra data that accounts for the effects of the uncertainties in $\mathbb{W}$. This procedure leverages the idea that the tube $\mathbb{Z}_t$ centered around $\bar{\mathbf{x}}_{0|t}^*$, as computed by RTMPC, represents a model of the states that the system may visit when subject to the uncertainties captured by the additive disturbances $\mathbf{w} \in \mathbb{W}$, while the ancillary controller represents an efficient way to compute control actions that ensure the

system remains inside the tube. Therefore, at each timestep t, given the "safe" plan computed by the expert $\bar{\mathbf{X}}_t^*, \bar{\mathbf{U}}_t^*$, we compute extra state-action pairs $(\mathbf{x}_t^+, \mathbf{u}_t^+)$ by sampling states from inside the tube $\mathbf{x}_t^+ \in \bar{\mathbf{x}}_{0|t}^* \oplus \mathbb{Z}_t$, and computing the corresponding robust control action $\mathbf{u}_t^+$ using the ancillary controller:

$$\mathbf{u}_t^+ = \bar{\mathbf{u}}_{0|t}^* + \mathbf{K}(\mathbf{x}_t^+ - \bar{\mathbf{x}}_{0|t}^*). \tag{6.8}$$

In this way, we obtain extra (input, output) samples $(\{\mathbf{X}_t^{\text{des}}, \mathbf{x}_t^+, \boldsymbol{e}_t\}, \mathbf{u}_t^+)$ that are added to the training dataset $\mathcal{D}$. Last, the policy in Eq. (6.7) is trained end-to-end using the dataset $\mathcal{D}$, by finding the parameters of π and μ that minimize the following MSE loss: $\|\mathbf{u}_i - \pi(\mathbf{x}_i, \mu(\boldsymbol{e}_i), \mathbf{X}_i^{\text{des}})\|_2^2$, where i denotes the i-th datapoint in $\mathcal{D}$.

6.3.3 Phase 2: Learning the Adaptation Module

This step is performed as in RMA [41], and is described in Section 6.2.1 of this thesis.

6.4 Evaluation

6.4.1 Evaluation Approach

We evaluate the proposed approach in the context of trajectory tracking for a multirotor, by learning to track an 8 s long, heart-shaped trajectory (♡) and an 8 s long, eight-shaped trajectory (∞) with a maximum velocity of 3.2 m/s. All evaluation is performed on a desktop machine with Intel i9-10920X CPUs and Nvidia RTX 3090 GPUs.

Simulation Details. Learning and evaluation are performed in simulation, implemented by integrating a realistic nonlinear model of the dynamics of a multirotor (with 6 motors):

$$\begin{aligned} \dot{p} = v, \quad & m\dot{v} = f_{\text{cmd}} R_B(q) z - m g_W z + f_{\text{drag}} + f_{\text{ext}}, \\ \dot{q} = \frac{1}{2}\Omega(\omega) q, \quad & J\dot{\omega} = -\omega \times J\omega + \tau_{\text{cmd}} + \tau_{\text{drag}} + \tau_{\text{ext}}. \end{aligned} \tag{6.9}$$

Table 6.1: Robot/environment parameter ranges during training and testing. For most parameters, the testing range is twice as wide as the training range, allowing us to evaluate our methods under large out-of-distribution model errors and disturbances. The nominal values are the average of the training ranges. At each timestep, each entry in $\boldsymbol{e}_t$ changes with $p = 0.001$ in the training environment, and $p = 0.002$ in the test environment.

Parameters	**Units**	**Train Range**		**Test Range**	
Mass	kg	[1.0, 1.6]		[0.8, 1.8]	
Inertia (x_B, y_B)	kg m^2	[6.6, 9.8]	e-3	[4.9, 12.0]	e-3
Inertia (z_B)	kg m^2	[1.0, 1.5]	e-2	[0.8, 1.8]	e-2
Drag (translational)	Ns/m	[0.08, 0.12]		[0.06, 0.14]	
Drag (rotational)	Nms/rad	[8.0, 12.0]	e-5	[6.0, 14.0]	e-5
Arm length	m	[0.13, 0.20]		[0.10, 0.23]	
Ext. force (x_W, y_W, z_W)	N	[-2.6, 2.6]		[-3.2, 3.2]	
Ext. torque (x_B, y_B)	Nm	[-0.42, 0.42]		[-0.53, 0.53]	
Ext. torque (z_B)	Nm	[-4.2, 4.2]	e-2	[-4.2, 4.2]	e-2

Position and velocity $p, v \in \mathbb{R}^3$ are expressed in the world frame W, $q \in \mathrm{SO}(3)$ is the attitude quaternion, ω is the angular velocity, m is the mass and J is the inertia matrix assumed diagonal. The total torques τ_{cmd} and forces f_{cmd} produced by the propellers, as expressed in body frame B, are linearly mapped to the propellers' thrust via a mixer/allocation matrix (e.g., [94]). We assume the presence of isotropic drag forces and torques $f_{\text{drag}} = -c_{dv}v$ and $\tau_{\text{drag}} = -c_{d\omega}\omega$, with $c_{dv} > 0$, $c_{d\omega} > 0$, and the presence of external forces f_{ext} and torques τ_{ext}. The environment parameter vector $\boldsymbol{e}_t$ has size 13, and contains the robot/environment parameters in Table 6.1. We use the `acados` integrator [125] to simulate these dynamics with a discretization interval of 0.002 s. Note that during integration of the dynamics we normalize the attitude quaternion to make sure it remains with unit norm.

RTMPC for Trajectory Tracking on a Multirotor. The controller has state of size $n_x = 12$, consisting of position, velocity, Euler angles, and angular velocity. It generates thrust/torque commands ($n_u = 4$) mapped to the 6 motor forces via allocation/mixer matrix ($n_a = 6$). We use an adversarial heuristic to find a value for $\mathbb{W}$, and specifically we assume that it matches the external forces and torques used in training distribution (Table 6.1), as they are close to the physical actuation limits of the platform. The reference trajectory is a sequence of desired positions and velocities for the next 1 s, discretized with a sampling

time of 0.04 s (corresponding to a planning horizon of $N = 25$, and 300-dim vector). The controller takes into account state constraints (i.e., available 3D flight space, velocity limits, etc) and actuation limits, and is simulated to run at 500 Hz.

Student Policy Architecture. The base policy π is a 3-layer Multi-Layer Perceptron (MLP) with 256-dim hidden layers, which takes as input the current state $\mathbf{x}_t \in \mathbb{R}^{12}$ and extrinsics vector $\boldsymbol{z}_t \in \mathbb{R}^8$ and outputs motor forces $\boldsymbol{a}_t \in \mathbb{R}^6$. The environment encoder μ is a 2-layer MLP with 128-dim hidden layers, taking as input environment parameters $\boldsymbol{e}_t \in \mathbb{R}^{13}$. The adaptation module ϕ projects the latest 400 state-action pairs into 32-dim representations using a 2-layer MLP. Then, a 3-layer 1-D CNN convolves the representation across time to capture temporal correlations in the input. The input channel number, output channel number, kernel size, and stride for each layer is $[32, 32, 8, 4]$. The flattened CNN output is linearly projected to obtain $\hat{\boldsymbol{z}}_t$. Like the RTMPC expert, the student policy is simulated to run at 500 Hz.

6.4.2 Training Details and Hyperparameters

All policies are implemented in PyTorch and trained with the Adam optimizer, with learning rate 0.001 and default parameters.

Phase 1. We train μ and π by collecting 8 s long trajectories, with 1 s of hovering before and after the trajectories. The expert actions are sampled at 20 Hz, resulting in 200 expert actions per demonstration (when no additional samples are drawn from the tube). When drawing additional samples from the tube, we do so in two different ways. The first is to uniformly sample the tube for every demonstration we collect from the expert, extracting $N_{\text{samples}} = \{25, 50, 100\}$ samples per timestep; these methods are denoted as SAMA-N_{samples}. In the second way, we apply data augmentation (using $N_{\text{samples}} = 100$ samples per timestep) only to the first collected demonstration, while we use DAgger only (no data augmentation) for the subsequent demonstrations. This method is denoted as SAMA-100-FT (Fine-Tuning, as DAgger is used to fine-tune a good initial guess generated via data augmentation). These

different procedures enable us to study trade-offs between improving robustness/performance (more samples) or the training time (fewer samples). Across our evaluations, we always set the DAgger hyperparameter β to 1 for the first demonstration and 0 otherwise.

Phase 2. Similar to previous RMA-like approaches [41]–[44], we train ϕ via supervised regression.

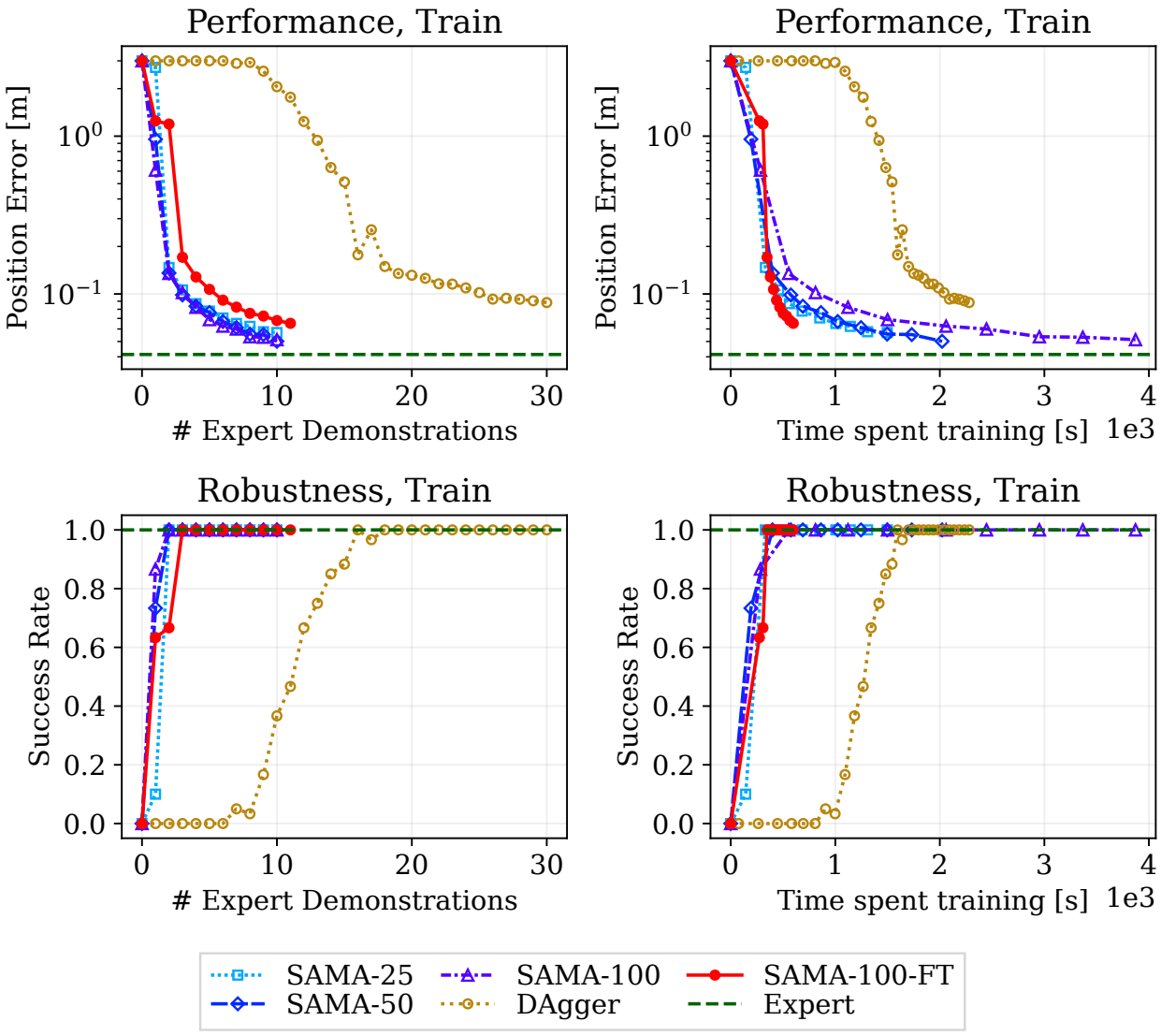


Figure 6.3: Performance and robustness in the **training environment** after *Phase 1*, as a function of number of demonstrations and training time. IL allows for the learning of effective *Phase 1* policies in one hour on a single core, as opposed to RL which has been reported to take two hours on an entire desktop machine [44]. This training time can be significantly shortened by using tube-guided data augmentation during training.

6.4.3 Efficiency, Robustness, and Performance

In this part, we analyze our approach on the task of performing *Phase 0 and 1*, as these are the parts where our method introduces key changes compared to prior RMA work, on the task of learning the heart-shaped trajectory (Fig. 6.6). We study the performance (average position error from the reference trajectory) and robustness (avoiding violation of state and actuation constraints) of our policy as a function of the total training time and the number of demonstrations used for training. Our comparison includes the RTMPC expert that our policy tries to imitate, as well as a policy trained only with DAgger, without any data augmentation. Each policy is evaluated on 10 separate realizations of the training/test environment. We repeat the procedure over 6 random seeds. All training in this part is done on a single CPU and GPU.

Figure 6.3 shows the evaluation of the policy under a new set of disturbances sampled from the same training distribution, defined in Table 6.1, highlighting that our tube-guided data augmentation strategy efficiently learns robust *Phase 1* policies. Compared to DAgger-only, our methods achieve full robustness in less than *half* the time, and using only 20% of the required expert demonstrations. Additionally, tube-guided methods achieve about half the position error of DAgger for the same training time, reaching an average of 5 cm in less than 10 minutes. Among tube-guided data augmentation methods, we observe that fine-tuning (SAMA-100-FT) achieves the lowest tracking error in the shortest time. Figure 6.4 repeats the evaluation under a challenging set of disturbances that are outside the training distribution (see Table 6.1). The analysis, as before, highlights the benefits of the data augmentation strategy, as SAMA methods achieve higher robustness and performance. The performance in this test environment confirms the trend that fine-tuning (SAMA-100-FT) achieves good trade-offs in terms of training time and robustness. Overall, these results highlight that our method can successfully and efficiently learn *Phase 1* policies capable of handling out-of-distribution disturbances.

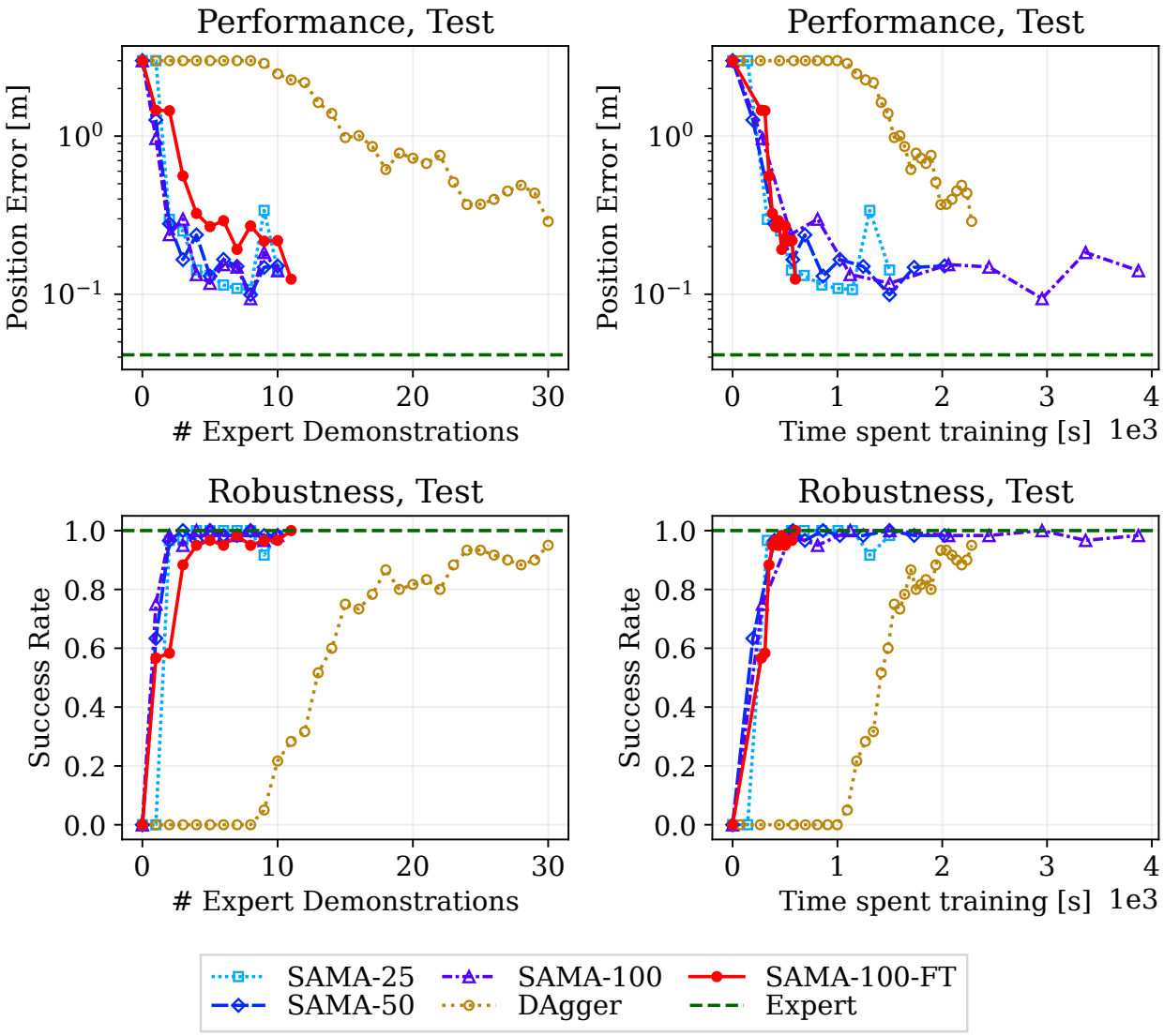


Figure 6.4: Performance and robustness in the **testing environment** after *Phase 1*, as a function of number of demonstrations and training time. The test environment presents a set of disturbances that the robot has never seen during training, as highlighted in Table 6.1. Methods that rely on our tube-guided data augmentation strategy (SAMA) generalize better than DAgger, achieving higher robustness and performance in lower time.

6.4.4 Adaptation Performance Evaluation

In this part, we analyze the adaptation performance of our approach after *Phase 2*. We consider the heart-shaped and the eight-shaped trajectory. For each trajectory, we train a μ and π in *Phase 1* using SAMA-100-FT, fine-tuning for 5 DAgger iterations, collecting 10

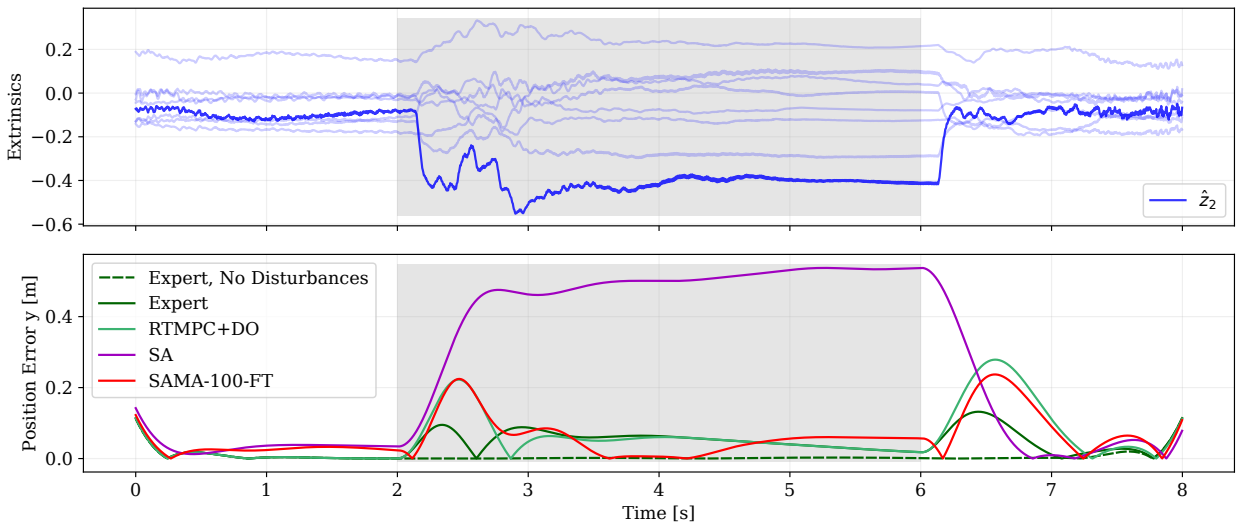


Figure 6.5: Performance while tracking the heart-shaped trajectory in Fig. 6.6. The robot is subject to an out-of-training-distribution wind-like force of 6 N (along positive y-axis, shown in shaded grey area) that is 36% larger than any external forces seen during training. Our method (SAMA-100-FT) is computationally efficient, robust, and adaptive, as shown by the changes in extrinsics ($\hat{\boldsymbol{z}}_2$) when the robot is subject to wind. Our method achieves 10% lower tracking error than RTMPC+DO, a model-based controller which is both robust and adaptive at the cost of being computationally expensive to run online (see Table 6.3), and which has been designed using a nominal model. We additionally maintain similar performance to the Expert, an RTMPC that has access to the ground truth model and disturbances and represents the best case performance of a model-based controller. This highlights improvements over our previous work SA [62], a learning-based controller which is robust and computationally efficient, but non-adaptive.

demonstrations per iteration during fine-tuning. Given a trained μ and π, we train ϕ via supervised regression in *Phase 2* (Section 6.4.2), conducting 20 iterations with 10 policy rollouts collected in parallel per iteration. On 10 CPUs and 1 GPU, *Phase 1* takes about 20 minutes and *Phase 2* takes about an hour. We note that the training efficiency of our proposed *Phase 1* compares favorably to the RL-based results in [44], where the authors report 2 hours of training time for *Phase 1*.

First, we evaluate the tracking performance of our adaptive controller in an environment subject to position-dependent winds, as shown in Fig. 6.6. The wind applies 6 N of force, a force 36% *larger* than any external force encountered during training. We compare our approach with SA, our previous non-adaptive robust policy learning method [62], and with

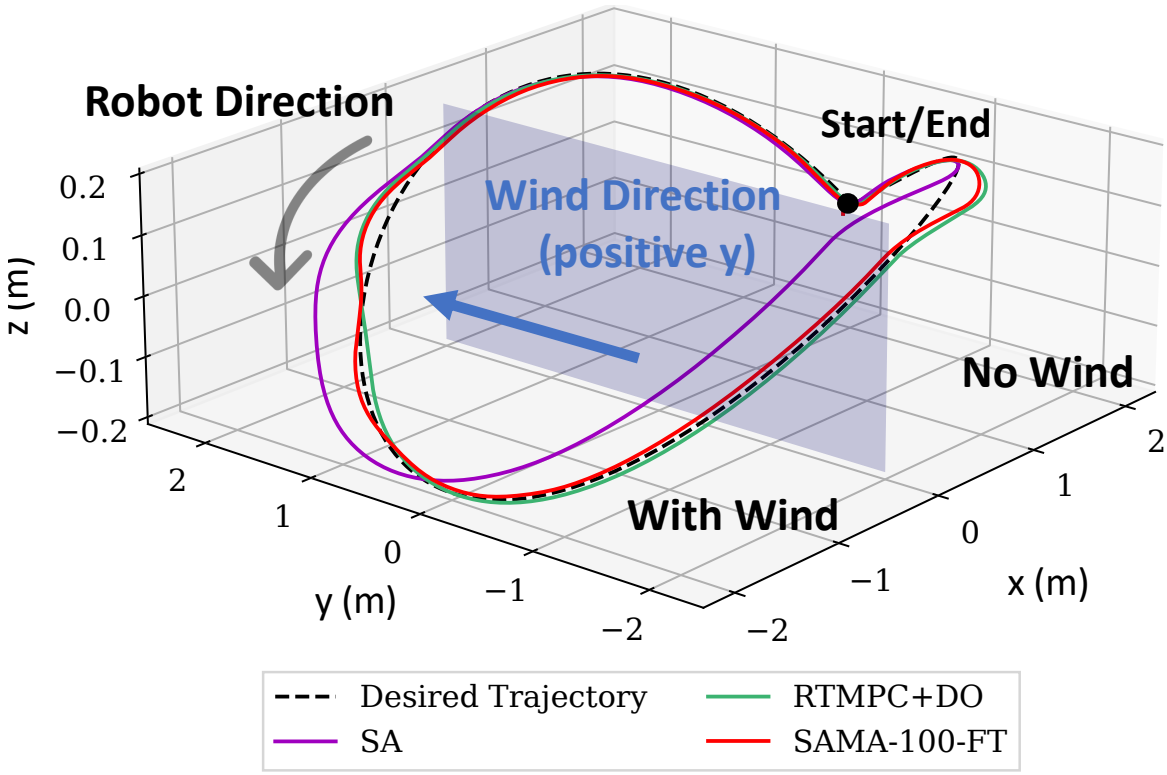


Figure 6.6: Tracking of a heart-shaped trajectory under strong, out-of-distribution wind-like external force disturbances. The blue plane $p_x = 0$ divides the environment into a part with wind ($p_x \leq 0$) and one without ($p_x > 0$). Our adaptive approach (SAMA-100-FT) demonstrates an improvement on our previous work (SA), which is robust but not adaptive, and it is able to match the performance of robust MPC combined with a disturbance observer (RTMPC+DO), but at a fraction of its computational cost.

Table 6.2: Average Position Error (APE) while tracking 8 s-long trajectories ♡ and ∞ in the test environment. Each policy was evaluated over 30 realizations of the test environment. Our approach (SAMA-100-FT) achieves successful adaptation, obtaining lower error than our previous, non-adaptive strategy (SA), and lower or comparable errors to RTMPC+DO.

Method	**APE** ♡ [m]	**APE** ∞ [m]
Expert	0.058	0.104
RTMPC+DO	0.125	0.163
SA (not adaptive) [62]	0.278	0.322
SAMA-100-FT (Ours)	0.110	0.175

the RTMPC expert that has access to $\boldsymbol{e_t}$ (the true value of the wind). We also consider an RTMPC whose state has been augmented with external force/torques estimated via a state-of-the-art nonlinear disturbance observer (RTMPC+DO) based on an Unscented Kalman filter (UKF) [95], a method that has access to the nominal model of the robot (matching the one used in this experiment) and ad-hoc external force/torque disturbance estimation. The results are presented in Fig. 6.5 and Fig. 6.6. The shaded section of Fig. 6.5, corresponding to the windy regions, highlights that SAMA-100-FT is able to adapt to a large, previously unseen force-like disturbance, obtaining a tracking error of less than 10 cm at convergence, unlike the corresponding non-adaptive variant (SA), which instead suffers from a 50 cm tracking error. Fig. 6.5 additionally highlights changes in the extrinsics, which do not depend on changes in reference trajectory but rather on the presence of the wind, confirming the successful adaptation of the policy. Table 6.2 reports a 10% reduction in tracking error compared to RTMPC+DO. Second, we repeat the evaluation on the challenging eight-shaped trajectory, with the robot achieving speeds of up to 3.2 m/s, where the robot is subject to a large set of out-of-distribution model errors: twice the nominal mass and arm length, ten times the nominal drag coefficients, and an external torque of 2.0 Nm. Table 6.2 and Fig. 6.7 highlight the adaptation capabilities of our approach, which performs comparably to the more computationally expensive (Table 6.3) RTMPC+DO.

Efficiency at Deployment. Table 6.3 reports the time to compute a new action for each method. On average, our method (SAMA-100-FT) is 12 times faster than the expert, and 24

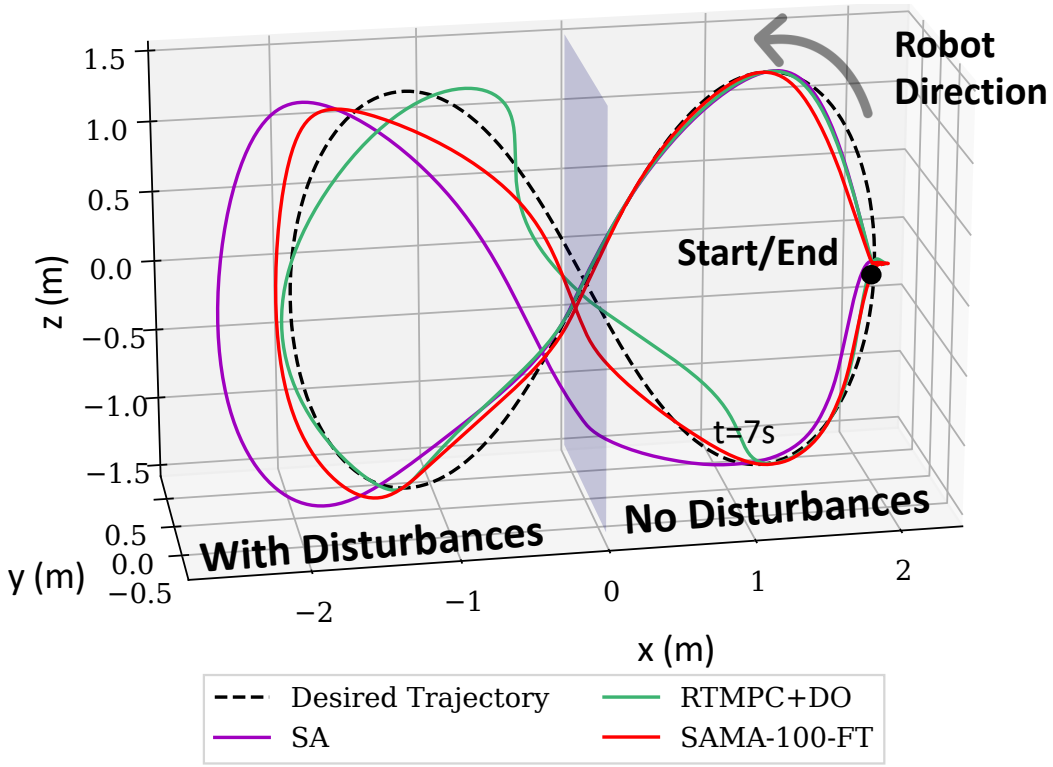


Figure 6.7: Tracking of an eight-shaped trajectory under out-of-distribution disturbances and model errors, where mass and arm length are twice the nominal values, the drag is 10 times the nominal, and there is a 2 Nm external torque disturbance. The blue plane $p_x = 0$ divides the environment into a part with model errors ($p_x \leq 0$) and without ($p_x > 0$). Our adaptive approach (SAMA-100-FT) can adapt during agile flight, reaching top speeds of 3.2 m/s, while maintaining performance comparable to RTMPC+DO.

Table 6.3: Time required to generate a new action; all times reported in milliseconds (ms). Our approach (SAMA-100-FT) is on average 12× faster than the optimization-based Expert, and 24× faster than an optimization-based approach with disturbance observer (RTMPC+DO). While our previous work (SA) [62] achieves a faster inference time than our method, it lacks adaptation, which our method adds with minimal computational cost.

Method	Setup	Mean	SD	Min	Max
Expert	CVXPY	9.51	6.16	5.04	62.2
RTMPC+DO	CVXPY	19.0	14.0	13.5	937
SA [62]	PyTorch	0.491	7.55e-2	0.458	1.54
SAMA-100-FT (Ours)	PyTorch	0.772	3.68e-2	0.669	1.58

times faster than RTMPC+DO.

6.5 Hardware Evaluation

In this section, we experimentally evaluate SAMA by deploying the obtained policies onboard a multirotors subject to challenging disturbances. The base policy and environment encoder are implemented in C++, and were run at 200Hz on the GPU of the onboard `Nvidia Jetson TX2`. Each module required less than 0.4 ms of inference time.

6.5.1 Performance at Hover under Multiple Simultaneous Disturbances

In this part, we demonstrate the real-world robustness and performance of the proposed approach when subject to simultaneous disturbances and failures. We do so by (i) attaching a slung load (tape roll, mass approximately 0.2 kg) to the hexarotor, while (ii) turning off one of the propellers, while (iii) applying wind disturbances with a leaf blower (wind approximately 4.0 m/s). Such disturbances/failures are applied while the robot hovers at the origin. The results of the experiment are reported in Fig. 6.8, demonstrating that SAMA can successfully withstand all the disturbances applied *simultaneously*, achieving tracking errors below 1 meter. We note that during training we did not explicitly simulate/collect

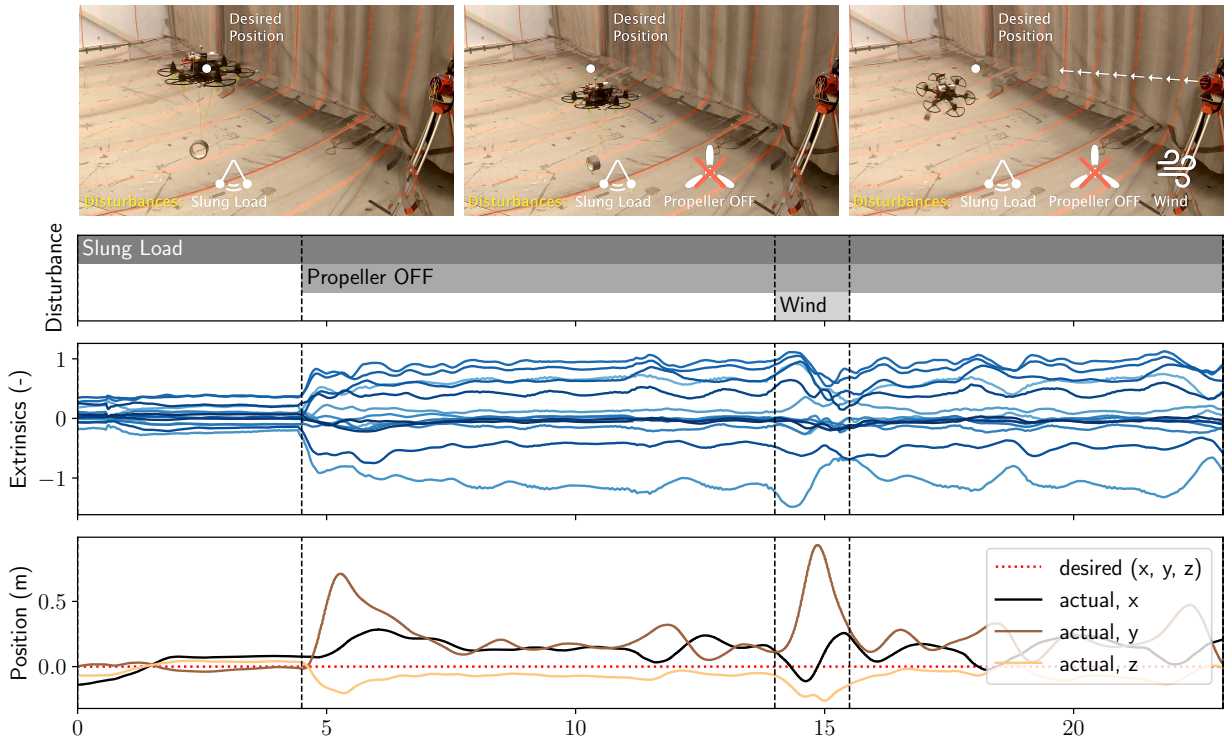


Figure 6.8: Real-world robustness and performance of SAMA under the *simultaneous* presence of different disturbances: (1) a slung load, (2) turning off one propeller, (3) a wind gust. The result highlights the robustness of SAMA to multiple disturbances which have not explicitly applied during the training phase, showcasing low tracking errors.

data under propeller failures, neither we simulated a slung-load. This result, therefore, additionally demonstrates the generalization of the approach to scenarios not experienced during demonstration collection and training.

6.5.2 Propeller Failure During Trajectory Tracking

In this part, we demonstrate the trajectory tracking capabilities of SAMA under a sudden propeller failure. The results are shown in Fig. 6.9, where the UAV tracks a Lemniscate (eight-shaped) trajectory (velocity up to 3.0m/s), when one propeller is suddenly turned off *(top)*. SAMA rapidly adapts to the new flight conditions, avoiding a failure *(middle)*. Last, the robot continues tracking the trajectory despite the failed propeller *(bottom)*. We note that no propeller failures were simulated/applied during data collection, therefore this result demonstrates robustness to out of distribution scenarios even during trajectory tracking.

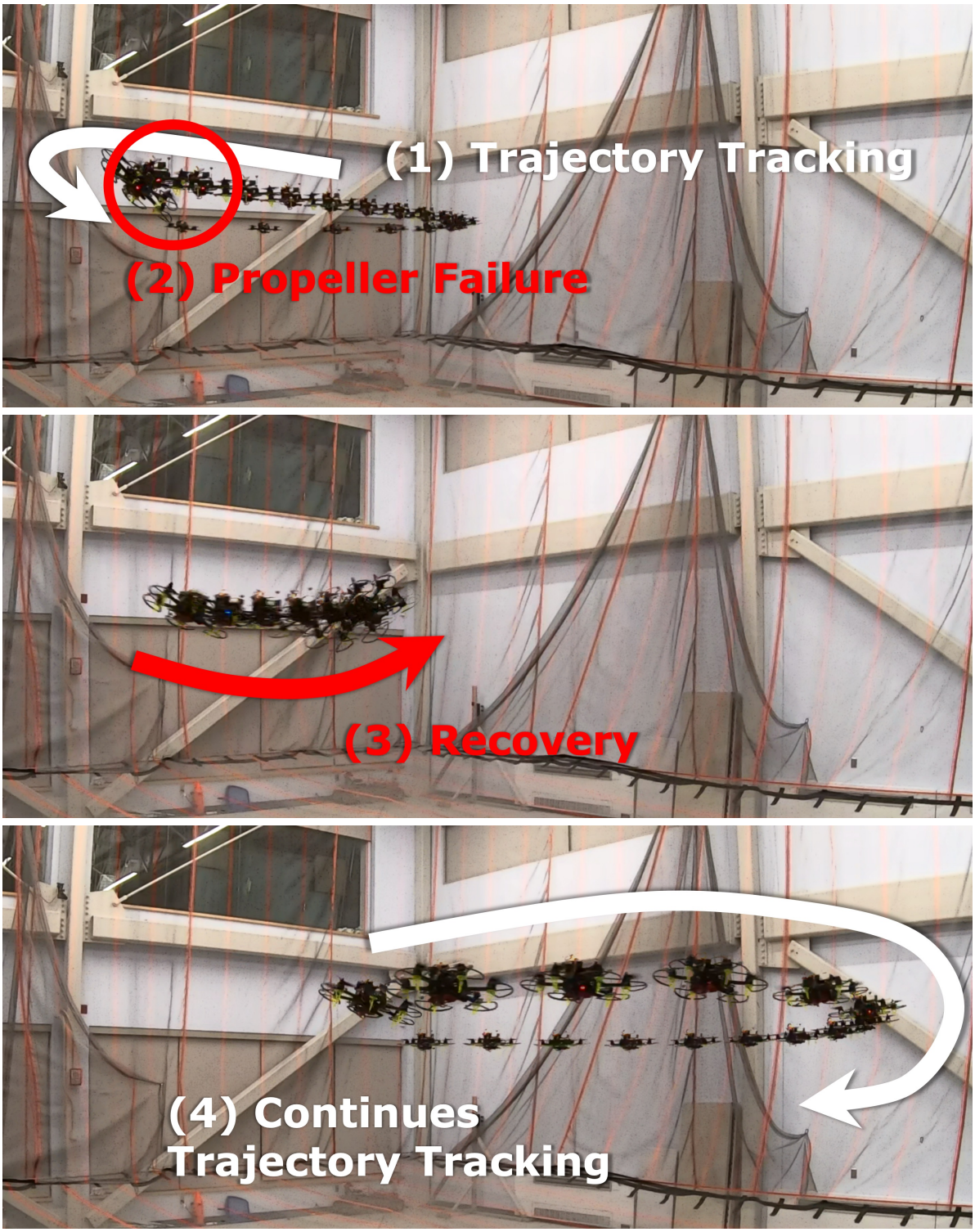


Figure 6.9: Composited image demonstrating trajectory tracking capabilities of SAMA under a sudden propeller failure. The UAV tracks a Lemniscate trajectory (velocity up to 3.0m/s), when one propeller is suddenly turned off *(top)*. SAMA rapidly adapts to the new flight conditions, avoiding a failure *(middle)*. Last, the robot continues tracking the trajectory despite the failed propeller *(bottom)*.

6.6 Summary

This Chapter presented SAMA, a strategy to enable adaptation in policies efficiently learned from a Robust Tube MPC expert using Imitation Learning and SA, the data augmentation strategy presented in Chapter 3. We did so by leveraging an adaptation scheme inspired by the recent RMA work [41]. Different from [41], the resulting procedure (1) avoids the challenges of reward design and tuning in RL, leveraging imitation combined with data augmentation, and (2) enables adaptation on both the translational and rotational dynamics of a UAV, unlike existing work that only focuses on adaptive attitude control [44]. Our evaluation in simulation and in real-world experiments has demonstrated successful learning of an adaptive, robust policy that can handle strong out-of-training distribution disturbances while controlling the position and attitude of a multirotor.

Chapter 7

Deployments on a Sub-Gram, Flapping-Wing Aerial Robot

7.1 Overview

In this Chapter, we present and experimentally demonstrate a computationally efficient method for accurate, robust, MPC-based flight control on sub-gram-MAVs.

First, we present a strategy for trajectory tracking leveraging the results presented in Chapter 3. The method uses a cascaded control scheme, where the attitude is controlled via the geometric attitude controller in [68], which presents a large region of attraction (initial attitude error should be <180 deg). This controller is additionally modified with a parametric adaptation scheme, where a torque observer estimates and compensates for the effects of slowly varying torque disturbances. Agile, robust, and real-time implementable trajectory tracking is achieved by employing a computationally efficient deep-NN policy, trained to imitate the response of a RTMPC, given a desired trajectory and the current state of the robot. The NN policy is obtained using our recent IL method [62], presented in Chapter 3, which uses a high-fidelity simulator and properties of the controller to generate training data. A key benefit of our method [62] is the ability to train a computationally efficient policy in a

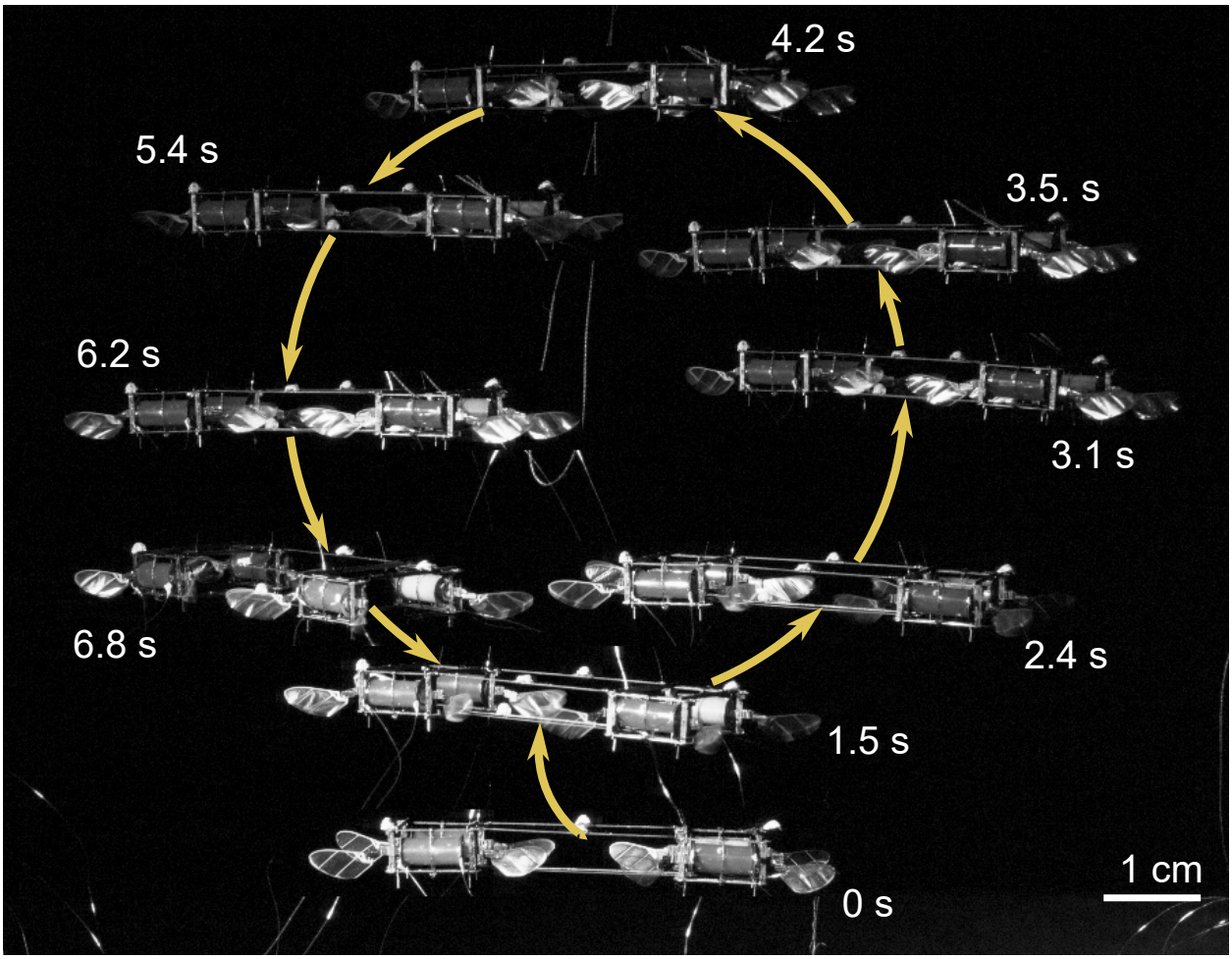


Figure 7.1: Composite image showing a 7.5-second flight where the MIT SoftFly [52], a soft-actuated, insect-scale MAV, follows a vertical circle with 5 cm radius. The robot is controlled by a neural network (NN) policy, trained to reproduce the response of a robust model predictive controller. Thanks to its computational efficiency, the NN controls the robot at 2 kHz while running on a small offboard computer.

computationally efficient way (e.g., a new policy can be obtained in a few minutes), greatly accelerating the tuning phase of the NN controller.

Second, we present a strategy for aggressive flight control, leveraging the results presented in Chapter 4.

The proposed robust, agile approaches are experimentally evaluated on the MIT sub-gram-MAV SoftFly [60], [69]. First, we show that our method can consistently achieve low position tracking error on a variety of trajectories, which include a circular trajectory (Fig. 7.1) and a ramp, while running at 2 kHz on a Baseline Target Machine, SpeedGoat

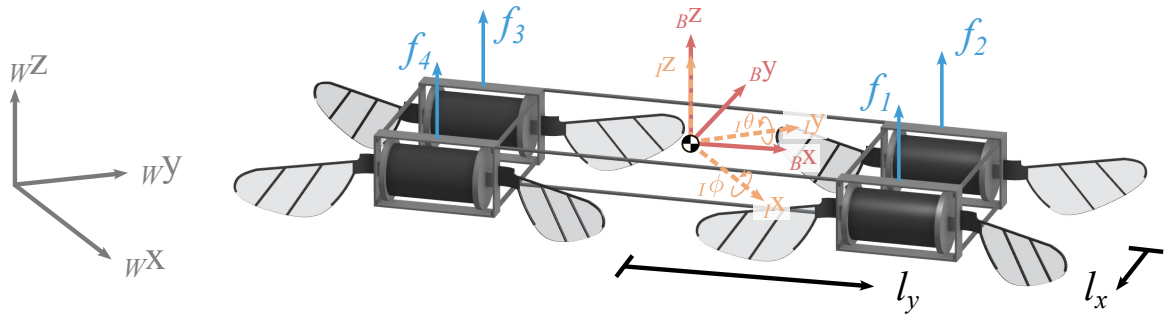


Figure 7.2: CAD model of sub-gram MAV SoftFly that consists of four soft artificial muscles (DEAs). We additionally show the following reference frames: inertial W (grey), body-fixed B (red), yaw-fixed frame I (orange) used for control.

offboard computer. Second, we demonstrate that our strategy is robust to large external disturbances, intentionally applied while the robot tracks a given trajectory. Last, we report high-speed flights, achieving velocities four times higher than existing work [69].

7.2 Robot Design and Model

Reference frames. We consider an inertial reference frame $W = \{{}_W\mathbf{x}, {}_W\mathbf{y}, {}_W\mathbf{z}\}$ and a body-fixed frame $B = \{{}_B\mathbf{x}, {}_B\mathbf{y}, {}_B\mathbf{z}\}$ attached to the CoM of the robot, as shown in Fig. 7.2.

Mechanical design. The sub-gram flapping-wing robot (Figure 7.2) consists of four individually-controlled DEAs [52], [61] with newly-designed enhanced-endurance wing hinges [138]. Unlike natural flying insects that actively control wing stroke and pitch motions [139], our robot leverages system resonance (400 Hz) and passive fluid-wing interaction to generate lift forces and support flight. [52], [61]. This design allows each of the four robot modules to generate lift forces without producing significant torques.

Actuation model. The voltage inputs to the actuator are controlled to produce desired time-averaged lift forces, and a linear voltage-to-lift-force mapping, $f_i = \alpha_i v_i + \beta_i$, is implemented as previously shown in [52], [61]. The time-averaged lift force f_i produce by each actuator i, with $i = 1, \ldots, 4$, is utilized in the model for control purpose since the wing inertia is orders of magnitude smaller than that of the robot thorax. These forces can be mapped to the total

torque produced by the actuators $\tau_{\text{cmd},x}$ and $\tau_{\text{cmd},y}$ (around ${}_B\mathbf{x}$ and ${}_B\mathbf{y}$, respectively) and f_{cmd} as the total thrust force on ${}_B\mathbf{z}$ via a linear mapping (*mixer* or *allocation* matrix) $\mathcal{A}$:

$$\begin{bmatrix} f_{\text{cmd}} \\ \tau_{\text{cmd},x} \\ \tau_{\text{cmd},y} \end{bmatrix} = \mathcal{A} \begin{bmatrix} f_1 \\ \vdots \\ f_4 \end{bmatrix}, \mathcal{A} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -l_y & l_y & l_y & -l_y \\ -l_x & -l_x & l_x & l_x \end{bmatrix}, \tag{7.1}$$

where l_x, l_y represent the distance of the actuators from the CoM of the robot, as shown in Fig. 7.2. This configuration (actuators' placement and near-resonant-frequency operating condition) does not generate controlled torques with respect to the body z-axis.

Translational and rotational dynamics. The MAV is modeled as a rigid body with six degrees of freedom, with mass m and diagonal inertia tensor $\mathbf{J}$, subject to gravitational acceleration g. The following set of Newton-Euler equations describes the robot's dynamics:

$$\begin{aligned} m\,{}_{\text{W}}\dot{\mathbf{v}} &= f_{\text{cmd}}\mathbf{R}\,{}_B\mathbf{z} - mg\,{}_W\mathbf{z} + {}_W\mathbf{f}_{\text{drag}} + {}_W\mathbf{f}_{\text{ext}}, \\ \mathbf{J}\,{}_{\text{B}}\dot{\boldsymbol{\omega}} &= -\,{}_{\text{B}}\boldsymbol{\omega} \times \mathbf{J}\,{}_{\text{B}}\boldsymbol{\omega} + {}_{\text{B}}\boldsymbol{\tau}_{\text{cmd}} + {}_{\text{B}}\boldsymbol{\tau}_{\text{drag}} + {}_{\text{B}}\boldsymbol{\tau}_{\text{ext}}, \\ {}_{\text{W}}\dot{\mathbf{p}} &= {}_W\mathbf{v}, \\ \dot{\mathbf{R}} &= \mathbf{R}\,{}_{\text{B}}\boldsymbol{\omega}^{\wedge}. \end{aligned} \tag{7.2}$$

Position $\mathbf{p} \in \mathbb{R}^3$ and velocity $\mathbf{v} \in \mathbb{R}^3$ are expressed in W; a rotation matrix $\mathbf{R} \in SO(3)$ defines the attitude, and the angular velocity ${}_{\text{B}}\boldsymbol{\omega}$ is expressed in B; $\boldsymbol{\omega}^{\wedge}$ denotes the skew-symmetric matrix of $\boldsymbol{\omega}$. We assume that the dynamics are affected by external force and torque ${}_W\mathbf{f}_{\text{ext}} \in \mathbb{R}^3$ and ${}_{\text{B}}\boldsymbol{\tau}_{\text{ext}} \in \mathbb{R}^3$, capturing the effects of unknown disturbances, such as the forces/torques applied by the power tethers, imperfections in the assembly and mismatches of model parameters (e.g, mass). Assuming no wind in the environment, we also include an isotropic drag force ${}_W\mathbf{f}_{\text{drag}} = -c_{Dv}\,{}_W\mathbf{v}$ and torque ${}_{\text{B}}\boldsymbol{\tau}_{\text{drag}} = -c_{D\omega}\,{}_{\text{B}}\boldsymbol{\omega}$, with $c_{Dv} > 0, c_{D\omega} > 0$.

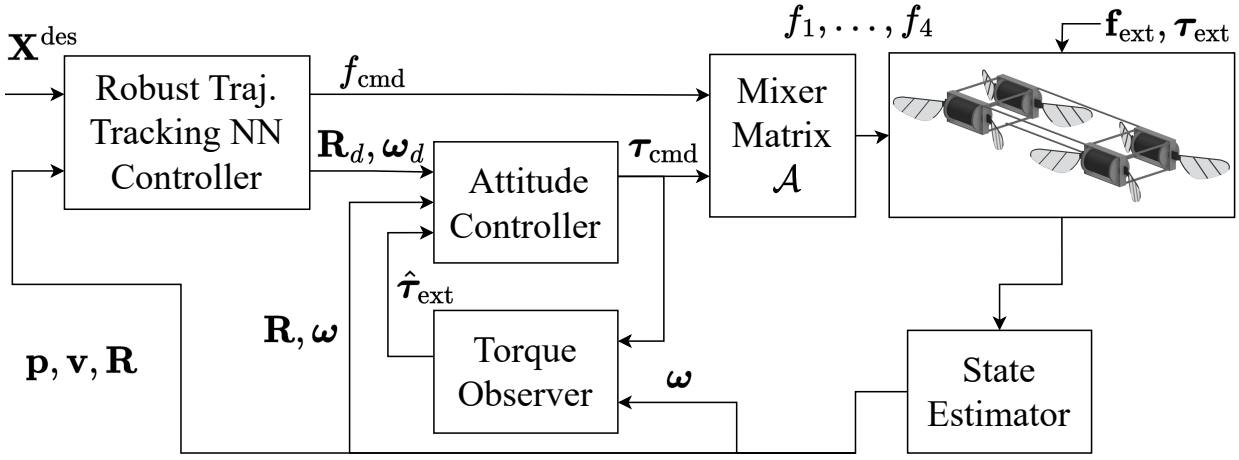


Figure 7.3: Cascaded architecture for the proposed robust trajectory tracking control strategy. The robust trajectory tracking NN controller constitutes the *outer* loop, and its task is to track a desired trajectory $\mathbf{x}_0^{\text{des}}, \ldots, \mathbf{x}_N^{\text{des}}$ by generating setpoints $\mathbf{R}_d$, $\boldsymbol{\omega}_d$ for the cascaded attitude controller. The attitude controller and the torque observer constitute the *inner* loop. Thanks to the robustness and adaptive properties of the *outer* and *inner* loops, our approach can withstand the external force/torque disturbances $\mathbf{f}_{\text{ext}}, \boldsymbol{\tau}_{\text{ext}}$.

7.3 Flight Control Strategy for Trajectory Tracking

We decouple trajectory tracking and attitude control via a cascaded scheme, as shown in Fig. 7.3. Given an $N+1$-step reference trajectory $\mathbf{X}^{\text{des}}$, a trajectory tracking controller generates desired thrust f_{cmd}, attitude $\mathbf{R}_d$ and angular velocity $\boldsymbol{\omega}_d$ setpoints. A nested attitude controller then tracks the attitude commands by generating a desired torque $\boldsymbol{\tau}_{\text{cmd}}$ and by leveraging the estimated torque disturbance $\hat{\boldsymbol{\tau}}_{\text{ext}}$ provided by a torque observer. The commands $\boldsymbol{\tau}_{\text{cmd}}, f_{\text{cmd}}$ are converted (in the Mixer Matrix) to desired mean lift forces $f_1, \ldots, f_4$ using the Moore-Penrose inverse $\mathcal{A}^\dagger$ of Eq. (7.1).

In the following paragraphs, we describe, first, the adaptive attitude controller (Section 7.3.1). Then, we present the computationally expensive trajectory-tracking RTMPC (Section 7.3.2) and, last, the computationally efficient procedure to generate the computationally efficient, robust NN tracking policy (Section 7.3.3) used to control the real robot.

7.3.1 Attitude Control and Adaptation Strategy

Attitude control law. The control law employed to regulate the attitude of the robot is based on the Geometric attitude controller in [68]:

$$\begin{aligned} {}_{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{cmd}} = &- \mathbf{K}_R\mathbf{e}_R - \mathbf{K}_\omega\mathbf{e}_\omega + {}_{\mathrm{B}}\boldsymbol{\omega} \times \mathbf{J}\,{}_{\mathrm{B}}\boldsymbol{\omega} \\ &- \mathbf{J}(\boldsymbol{\omega}^\wedge\mathbf{R}^\top\mathbf{R}_d\boldsymbol{\omega}_d - \mathbf{R}^\top\mathbf{R}_d\,{}_{\mathrm{B}}\dot{\boldsymbol{\omega}}_d) - \hat{\boldsymbol{\tau}}_{\mathrm{ext}}, \end{aligned} \tag{7.3}$$

where $\mathbf{K}_R, \mathbf{K}_\omega$ of size 3×3 are diagonal matrices, tuning parameters of the controller, and the attitude error $\mathbf{e}_R$ and its time derivative $\mathbf{e}_\omega$ are defined as in [68]:

$$\mathbf{e}_R = \frac{1}{2}(\mathbf{R}_d^\top\mathbf{R} - \mathbf{R}^\top\mathbf{R}_d)^\vee,\ \mathbf{e}_\omega = {}_{\mathrm{B}}\boldsymbol{\omega} - \mathbf{R}^\top\mathbf{R}_d\,{}_{\mathrm{B}}\boldsymbol{\omega}_d. \tag{7.4}$$

The symbol $(\mathbf{r}^\wedge)^\vee = \mathbf{r}$ denotes the operation transforming a 3×3 skew-symmetric matrix $\mathbf{r}^\wedge$ in a vector $\mathbf{r} \in \mathbb{R}^3$. Different from [68], we assume ${}_{\mathrm{B}}\dot{\boldsymbol{\omega}}_d = \mathbf{0}_3$. While this could result in larger tracking errors under very aggressive attitude changes, it avoids taking derivatives of potentially discontinuous angular velocity commands, reducing actuation noise. We additionally augment Eq. (7.3) with the adaptive term $\hat{\boldsymbol{\tau}}_{\mathrm{ext}}$, which is computed via a torque observer. We note that only the first two components of ${}_{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{cmd}}$ are used for control, as the actuators cannot produce torque $\tau_{\mathrm{cmd},z}$.

Torque observer. We compensate for the effects of uncertainties in the rotational dynamics by estimating torque disturbances ${}_{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{ext}}$ via a steady state (linear) Kalman filter. These disturbances are assumed to be slowly varying when expressed in the body frame B. The state of the filter is $\mathbf{x}_o = [{}_{\mathrm{B}}\boldsymbol{\omega}^\top, {}_{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{ext}}^\top]^\top$. We assume that the rotational dynamics, employed to compute the prediction (*a priori*) step, evolve according to:

$${}_{\mathrm{B}}\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}(\mathbf{u}_o + {}_{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{ext}}) + \boldsymbol{\eta}_\omega,\ \dot{\boldsymbol{\tau}}_{\mathrm{ext}} = \boldsymbol{\eta}_\tau, \tag{7.5}$$

where $\boldsymbol{\eta}_\omega$, $\boldsymbol{\eta}_\tau$ are assumed to be zero-mean Gaussian noise, whose covariance is a tuning

parameter of the filter. Additionally, the prediction step is performed assuming the control input $\mathbf{u}_o = {}_{\mathrm{B}}\boldsymbol{\tau}_{\mathrm{cmd}} - {}_{\mathrm{B}}\boldsymbol{\omega} \times \mathbf{J}\, {}_{\mathrm{B}}\boldsymbol{\omega}$, which enables us to take into account the nonlinear gyroscopic effects ${}_{\mathrm{B}}\boldsymbol{\omega} \times \mathbf{J}\, {}_{\mathrm{B}}\boldsymbol{\omega}$ while using a computationally efficient linear observer. The measurement update (a *posteriori*) uses angular velocity measurements $\mathbf{z}_o = {}_{\mathrm{B}}\boldsymbol{\omega}_m + \boldsymbol{\eta}_m$, assumed corrupted by an additive zero mean Gaussian noise $\boldsymbol{\eta}_m$, whose covariance can be identified from data or adjusted as a tuning parameter.

7.3.2 Robust Tube MPC for Trajectory Tracking

In this part, first we present the hover-linearized vehicle model employed for control design (Section 7.3.2) and the compensation schemes to account for the effects of linearization (Section 7.3.2). The optimization problem solved by a linear RTMPC is the same as the one described in Chapter 3.

Linearized Model

We linearize the dynamics Eq. (7.2) around hover using a procedure that largely follows [7], [110]. The key differences are highlighted in the following.

First, for interpretability, we represent the attitude of the MAV via the Euler angles yaw ψ, pitch θ, roll ϕ (*intrinsic* rotations around the z-y-x). The corresponding rotation matrix can be obtained as $\mathbf{R} = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi)$, where $\mathbf{R}_j(\alpha)$ denotes a rotation of α around the j-th axis. Additionally, we express the dynamics Eq. (7.2) in a yaw-fixed frame I, so that ${}_I\mathbf{x}$ is aligned with ${}_W\mathbf{x}$. The roll ${}_I\phi$ and pitch ${}_I\theta$ angles (and their first derivative ${}_I\varphi$, ${}_I\vartheta$) expressed in I can be expressed in B via the rotation matrix $\mathbf{R}_{BI}$:

$$\begin{bmatrix} \phi \\ \theta \end{bmatrix} = \mathbf{R}_{BI} \begin{bmatrix} {}_I\phi \\ {}_I\theta \end{bmatrix}, \mathbf{R}_{BI} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix}. \tag{7.6}$$

The state $\mathbf{x}$ of the linearized model is chosen to be:

$$\mathbf{x} = [{}_W\mathbf{p}^\top, {}_W\mathbf{v}^\top, {}_I\phi, {}_I\theta, {}_I\delta\phi_{\text{cmd}}, {}_I\delta\theta_{\text{cmd}}]^\top, \tag{7.7}$$

where ${}_I\delta\phi_{\text{cmd}}, {}_I\delta\theta_{\text{cmd}}$ denote the linearized commanded attitude. We chose the control input $\mathbf{u}$ to be:

$$\mathbf{u} = [{}_I\varphi_{\text{cmd}}, {}_I\vartheta_{\text{cmd}}, \delta f_{\text{cmd}}]^\top, \tag{7.8}$$

where δf_{cmd} denotes the linearized commanded thrust, and ${}_I\varphi_{\text{cmd}}$ and ${}_I\vartheta_{\text{cmd}}$ are the commanded roll and pitch rates. The choice of $\mathbf{x}$ and $\mathbf{u}$ differs from [7], [62] as the control input consist in the roll, pitch rates rather than $\theta_{\text{cmd}}, \phi_{\text{cmd}}$; this is done to avoid discontinuities in $\theta_{\text{cmd}}, \phi_{\text{cmd}}$, and to feed-forward angular velocity commands to the attitude controller.

The linear translational dynamics are obtained by linearization of the translational dynamics in Eq. (7.2) around hover. Linearizing the closed-loop rotational dynamics is more challenging, as they should include the linearization of the attitude controller. Following [110], we model the closed-loop attitude dynamics around hover expressed in I as:

$$\begin{aligned} {}_I\dot{\theta} &= \frac{1}{\tau_\theta}(k_\theta\, {}_I\theta_{\text{cmd}} - {}_I\theta), \;\; {}_I\dot{\theta}_{\text{cmd}} = {}_I\vartheta_{\text{cmd}}, \\ {}_I\dot{\phi} &= \frac{1}{\tau_\phi}(k_\phi\, {}_I\phi_{\text{cmd}} - {}_I\phi), \;\; {}_I\dot{\phi}_{\text{cmd}} = {}_I\varphi_{\text{cmd}}, \end{aligned} \tag{7.9}$$

where k_ϕ, k_θ are gains of the commanded roll and pitch angles, while τ_ϕ, τ_θ are the respective time constants. These parameters can be obtained via system identification.

Last, we model the unknown external force disturbance ${}_W\mathbf{f}_{\text{ext}}$ in Eq. (7.2) as a source of bounded uncertainty, assumed to be $\|\mathbf{f}_{\text{ext}}\|_\infty < \bar{f}_{\text{ext}}$. This introduces an additive bounded uncertainty $\mathbf{w} \in \mathbb{W}$, with:

$$\mathbb{W} := \{\mathbf{w} = [\mathbf{0}_3^\top, {}_W\mathbf{f}_{\text{ext}}^\top, \mathbf{0}_4^\top]^\top \mid \|\mathbf{f}_{\text{ext}}\|_\infty < \bar{f}_{\text{ext}}\}. \tag{7.10}$$

Via discretization with sampling period T_c, we obtain the following linear, uncertain state

space model:

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t + \mathbf{w}_k, \tag{7.11}$$

subject to actuation and state constrains $\mathbb{U} = \{\mathbf{u} \in \mathbb{R}^3 | \mathbf{u}_{\min} \leq \mathbf{u} \leq \mathbf{u}_{\max}\}$, and $\mathbb{X} = \{\mathbf{x} \in \mathbb{R}^{10} | \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}\}$.

Compensation schemes and attitude setpoints

Following [110], we apply a compensation scheme to the commands:

$$f_{\text{cmd}} = \frac{\delta f_{\text{cmd}} + g}{\cos(\phi)\cos(\theta)}, \quad \begin{bmatrix} \phi_{\text{cmd}} \\ \theta_{\text{cmd}} \end{bmatrix} = \frac{g}{f_{\text{cmd}}} \begin{bmatrix} \delta\phi_{\text{cmd}} \\ \delta\theta_{\text{cmd}} \end{bmatrix}. \tag{7.12}$$

This desired orientation θ_{cmd}, ϕ_{cmd} is converted to a desired rotation matrix $\mathbf{R}_d$, setpoint for the attitude controller, setting the desired yaw angle to the current yaw angle $\psi_{\text{cmd}} = \psi$. Last, the desired angular velocity $\boldsymbol{\omega}_d$ is computed from the desired yaw, pitch, roll rates $\dot{\mathbf{q}} = [\dot{\psi}_{\text{cmd}}, \vartheta_{\text{cmd}}, \varphi_{\text{cmd}}]^\top$ via [140] $\boldsymbol{\omega}_d = \mathbf{E}_{\psi,\theta,\phi}\dot{\mathbf{q}}$, with

$$\mathbf{E}_{\psi,\theta,\phi} = \begin{bmatrix} 0 & -\sin(\psi) & \cos(\psi)\sin(\theta) \\ 0 & \cos(\psi) & \sin(\psi)\cos(\theta) \\ 1 & 0 & -\sin(\theta) \end{bmatrix}, \tag{7.13}$$

and assuming the desired yaw rate $\dot{\psi}_{\text{cmd}} = 0$.

7.3.3 Robust Tracking Neural Network Policy

The procedure to generate a computationally efficient NN policy capable of reproducing the response of the trajectory tracking RTMPC in Section 7.3.2 is the one presented in Chapter 3, and it is summarized in Fig. 7.4.

Policy input-output. The deep NN policy that we intend to train is denoted as π_θ, with

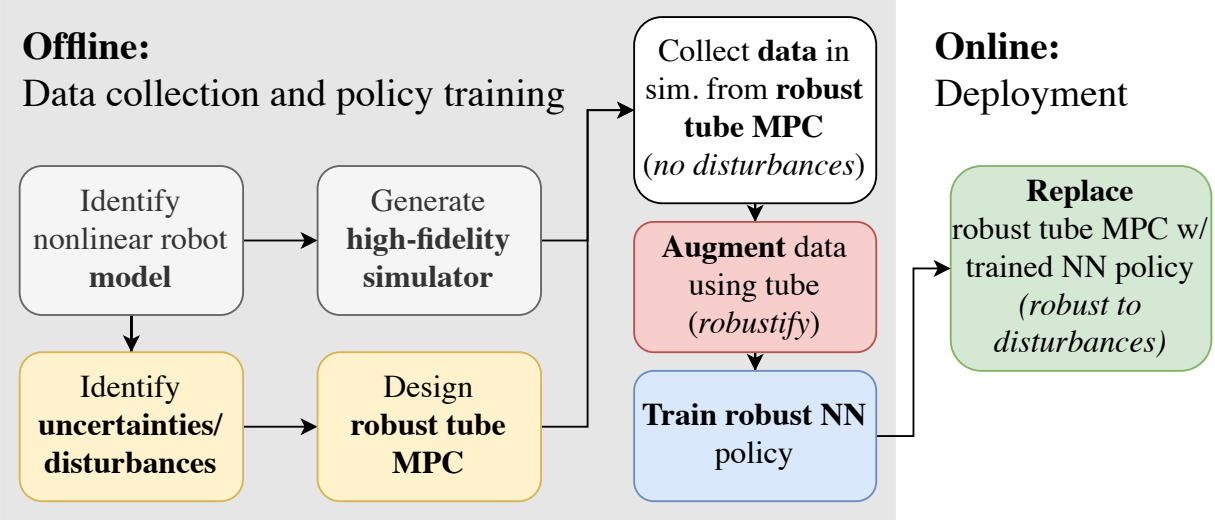


Figure 7.4: Imitation learning strategy employed to learn a robust NN from RTMPC. Key idea is to collect RTMPC demonstrations in simulation and to leverage properties of RTMPC to augment the collected demonstrations with data that improve the robustness of the trained policy.

parameters θ. Its input-outputs are the same as the ones of RTMPC in Section 7.3.2:

$$\mathbf{u}_t = \pi_\theta(\mathbf{x}_t, \mathbf{X}_t^{\text{des}}). \tag{7.14}$$

Policy training. Our approach, based on our previous work [62] presented in Chapter 3, consists in the following steps:

1) We design a high-fidelity simulator implementing a discretized model (discretization period T_s) of the nonlinear dynamics Eq. (7.2) and the control architecture consisting of the attitude controller (Section 7.3.1) and RTMPC (Section 7.3.2). In the simulator, we assume that the MAV is not subject to disturbances, setting $\mathbf{f}_{\text{ext}} = \mathbf{0}_3$ and $\boldsymbol{\tau}_{\text{ext}} = \mathbf{0}_3$, and therefore we do not simulate the torque observer.

2) Given a desired trajectory, we collect a $T+1$-step *demonstration* $\mathcal{T}$ by simulating the entire system controlled by RTMPC. At every timestep t of the demonstration, we store the inputs, output of RTMPC, with the addition of the safe plans $\bar{\mathbf{u}}_t^*, \bar{\mathbf{x}}_t^*$, obtaining $\mathcal{T} = \{(\mathbf{x}_0, \mathbf{u}_0, \bar{\mathbf{u}}_0^*, \bar{\mathbf{x}}_0^*, \mathbf{X}_0^{\text{des}}), \dots, (\mathbf{x}_T, \mathbf{u}_T, \bar{\mathbf{u}}_T^*, \bar{\mathbf{x}}_T^*, \mathbf{X}_T^{\text{des}})\}$.

3) We generate a dataset of (inputs, output) of the controller, using the data ob-

tained from the collected demonstration $\mathcal{T}$. The obtained dataset $\mathcal{D}$ has the form $\mathcal{D} = \{(\{\mathbf{x}_0, \mathbf{X}_0^{\text{des}}\}, \mathbf{u}_0), \ldots, (\{\mathbf{x}_T, \mathbf{X}_T^{\text{des}}\}, \mathbf{u}_T)\}$.

4) For every timestep t in $\mathcal{T}$, we augment the dataset $\mathcal{D}$ by generating N_{augm} extra (state, action) pairs $(\mathbf{x}_{i,t}^+, \mathbf{u}_{i,t}^+)$ by uniformly sampling extra states from the tube ($\mathbf{x}_{i,t}^+ \in \bar{\mathbf{x}}_t^* \oplus \mathbb{Z}$), and by computing the corresponding actions $\mathbf{u}_{i,t}^+$ with the ancillary controller:

$$\mathbf{u}_{i,t}^+ = \bar{\mathbf{u}}_t^* + \mathbf{K}(\mathbf{x}_{i,t}^+ - \bar{\mathbf{x}}_t^*). \tag{7.15}$$

The extra datasets $\mathcal{D}_t^+ = \{(\{\mathbf{x}_{i,t}^+, \mathbf{X}_t^{\text{des}}\}, \mathbf{u}_{i,t}^+)_{i=1}^{N_{\text{augm}}}\}$, $t = 0, \ldots, T$ are then combined with $\mathcal{D}$, obtaining the training dataset. This data augmentation procedure [62] generates data that can compensate for the effects of uncertainties in $\mathbb{W}$. This procedure has also the potential to reduce the time needed for the data collection phase over other existing IL methods, as Eq. (7.15) can be computed efficiently. We note that step $2-4$ should be repeated for every possible trajectory the policy needs to learn to follow, using IL methods such as BC or DAgger [21], [62]. However, as shown in Chapter 3 (our previous work [62]), if a set of sufficiently representative trajectories is collected during training, then the policy is capable to execute new slightly different trajectories, achieving generalization capabilities.

5) The optimal parameters θ^* for the policy Eq. (7.14) are then found by training the policy on the collected and augmented dataset, minimizing the MSE loss.

7.4 Flight Control Strategy for Aggressive, Near-Minimum Time Flight

The control approach and policy learning procedure for aggressive, near-minimum time flight follows the one detailed in Chapter 4, with the difference that we do not employ any cascaded attitude controller. We employ instead an ancillary NMPC that directly commands body torques and thrust ($n_u = 3$) that are directly mapped into voltage signals for the actuators

via Eq. (7.1); the ancillary NMPC uses the model of the robot presented in Section 7.2, with its state consisting of position, velocity, attitude and angular velocity ($n_x = 13$). The choice of not using an attitude controller was made to further simplify control design, avoiding the need to tune the attitude controller separately and avoiding the need to account for its dynamics.

7.5 Experimental Evaluation

The robustness and performance of the described flight controllers are experimentally evaluated on the soft-actuated MIT SoftFly [60]. Note that for the near-minimum time flights in Section 7.5.2 we employ the hardware variant described in [69]. Key differences between the two versions include inertia coefficients and arm lenghts ($l_x = l_y$ in hardware version [69]).

Experimental setup.

The flight experiments are performed in an environment equipped with 6 motion-capturing cameras (Vantage V5, Vicon). This system provides positions and orientations of the robot, and velocities are obtained via numerical differentiation. The controller runs on the Baseline Target Machine (Speedgoat) using the Simulink Real-Time operating system; its commands are converted to sinusoidal signals for flapping motion at 10 kHz. Voltage amplifiers ($677B$, Trek) are connected to the controller and produce amplified control voltages to the robot.

7.5.1 Trajectory Tracking

We consider two trajectories of increasing difficulty, a position ramp, where we additionally perturb the MAV with external disturbances, and a circular trajectory. The controller run at 2 kHz.

Table 7.1: Position Root Mean Squared Errors (RMSE) and Maximum Absolute Errors (MAE) when tracking a ramp (T1), a ramp with disturbances (T2) and a circle (T3). All the values are computed after takeoff ($t > 0.5$ s).

	T1: Ramp (3 runs, 2.5 s)						**T2 (1 run, 7.0 s)**		**T3: Circle (3 runs, 7.5 s)**					
	RMSE (cm, ↓)			MAE (cm, ↓)			RMSE	MAE	RMSE (cm, ↓)			MAE (cm, ↓)		
Axis	AVG	MIN	MAX	AVG	MIN	MAX	(cm, ↓)	(cm, ↓)	AVG	MIN	MAX	AVG	MIN	MAX
x	0.6	0.4	0.9	1.1	0.7	1.5	0.7	2.5	1.0	0.8	1.4	2.0	1.7	2.6
y	0.8	0.7	0.9	1.5	1.3	1.6	1.0	2.1	1.5	1.3	1.8	2.8	2.5	3.1
z	0.1	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3	0.6	0.5	0.8

Controller and training parameters.

The parameters for the controllers are obtained via system identification, also leveraging [141], [142]. The RTMPC has a 1-second long prediction horizon, with $N = 50$ and $T_c = 0.02$; we set $\bar{f}_{\text{ext}}$ to correspond to 15% of the weight force acting on the robot. State and actuation constraints capture safety and actuation limits of the robot (e.g., max actual/commanded roll/pitch $< 25\,\text{deg}$, $\|\delta f_{\text{cmd}}\| < 80\% mg$). The employed policy is a feed-forward, fully connected NN with 2-hidden layers, 32 neurons per layer (as in our previous work [62], where this size has been shown to be capable to learn multiple trajectories). Its input size is 310 (current state, and reference trajectory containing desired position, velocity across the prediction horizon N), and its output size is 3. During the experiments, we slightly tune the parameters $\mathbf{Q}_x, \mathbf{R}_u, \mathbf{K}_R, \mathbf{K}_\omega$ to study how sensitive is our approach to tuning changes, without observing large performance variations. We train a policy for each type of trajectory (ramp, circle), using the ADAM optimizer, with a learning rate $\eta = 0.001$, for 15 epochs. Data augmentation is performed by using $N_{\text{augm}} = 200$. Training each policy takes about 1 minute (for $T = 350$ steps) on a Intel i9-10920 (12 cores) with two Nvidia RTX 3090 GPUs.

Task 1: Position Ramps

First, we consider a position ramp of 3 cm along the three axes of the W, with a duration of 1 s. The total flight time in the experiment is of 3 s, and we repeat the experiment three times. This is a task of medium difficulty, as the robot needs to simultaneously roll, pitch

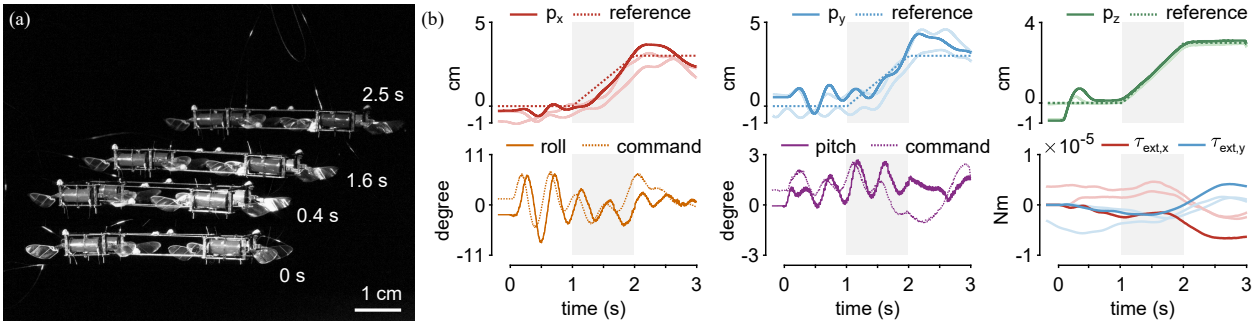


Figure 7.5: Performance of the proposed flight control strategy in Task 1 (T1), where the robot follows a 3.5 s ramp trajectory on x-y-z. The experiment is repeated three times, and the results are included in the shaded lines, with the exception of the roll and pitch angles, for clarity. These results highlight the accuracy of the proposed trajectory tracking error, which achieves a 0.6 cm position tracking RMSE on x-y, and 0.2 cm on z.

and accelerate along z, but the maneuver covers a small distance (5.2 cm). Fig. 7.5 (a) shows a time-lapse of the maneuver. Table 7.1 reports the position tracking Root Mean Squared Error (RMSE) (computed after take-off, starting at $t_0 = 0.5$), and shows that we can consistently achieve sub-centimeter RMSE on all the axes, with a maximum absolute error (MAE) smaller than 1.6 cm. This is a 60% reduction over the 4.0 cm MAE on x-y reported in [60] for a hover task. Remarkably, the altitude MAE is only 0.2 cm, with a similar reduction (60%) over [60] (0.5 cm). Fig. 7.5 shows the robot's desired and actual position and attitude across multiple runs (shaded lines), demonstrating repeatability. Fig. 7.5 additionally highlights the role of the torque observer, which estimates a position-dependent disturbance, possibly caused by the forces applied by the power cables, or by the safety tether.

Task 2: Rejection of Large External Disturbances

Next, we increase the complexity of the task by intentionally applying strong disturbances with a stick to the safety tether (Fig. 7.6 (a)), while tracking the same ramp trajectory in Task 1. This causes accelerations > 0.25 g. Fig. 7.6 reports position, attitude, and estimated disturbances, where the contact periods have been highlighted in orange. Table 7.1 reports RMSE and MAE. From Fig. 7.6 (b) we highlight that a) the position of the robot remains close to the reference ($<$ 2.5 cm MAE on x-y) and, surprisingly, the altitude is almost

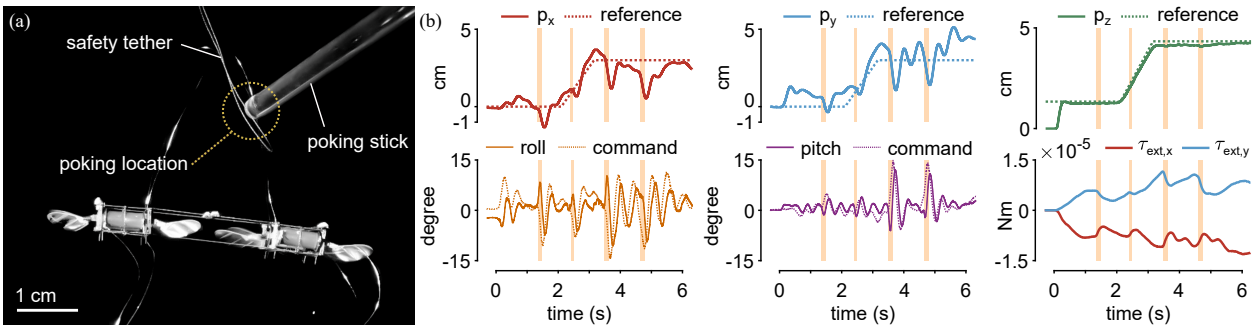


Figure 7.6: Robustness of the proposed flight control strategy when applying large force/torque disturbances to the MAV while tracking a 6.0 s ramp trajectory on x-y-z, our Task 2 (T2). These results highlight that the robot is not destabilized by the large disturbances, achieving an impressive 0.3 cm MAE on the z axis. The fast attitude dynamics additionally enable the robot to recover in only 200 ms.

unperturbed (0.3 cm MAE). b) the robot, thanks to its small inertia, recovers quickly from the large disturbances, being capable to permanently reduce (after impact, until the next impact) its acceleration by 50% in less than 200 ms; c) despite the impulse-like nature of the applied disturbance, the torque observer detects some of its effects.

Task 3: Circular Trajectory

Last, we track a circular trajectory along the x-z axis of W for three times. The trajectory has a duration of 7.5 s (including takeoff), with a desired velocity of 5.2 cm/s, and the circle has a radius of 5.0 cm. A composite image of the experiment is shown in Fig. 7.1. Fig. 7.7 provides a qualitative evaluation of the performance of our approach, while Table 7.1 reports the RMSE of position tracking across the three runs. These results highlight that: a) in line with Task 1, our approach consistently achieves mm-level accuracy in altitude tracking (3 mm average RMSE on z), and cm-level accuracy in x-y position tracking (RMSE < 1.8 cm, MAE < 3.1 cm); b) the external torque observer plays an important role in detecting and compensating external torque disturbances, which corresponds to approximately 30% of the maximum torque control authority around ${}_B x$.

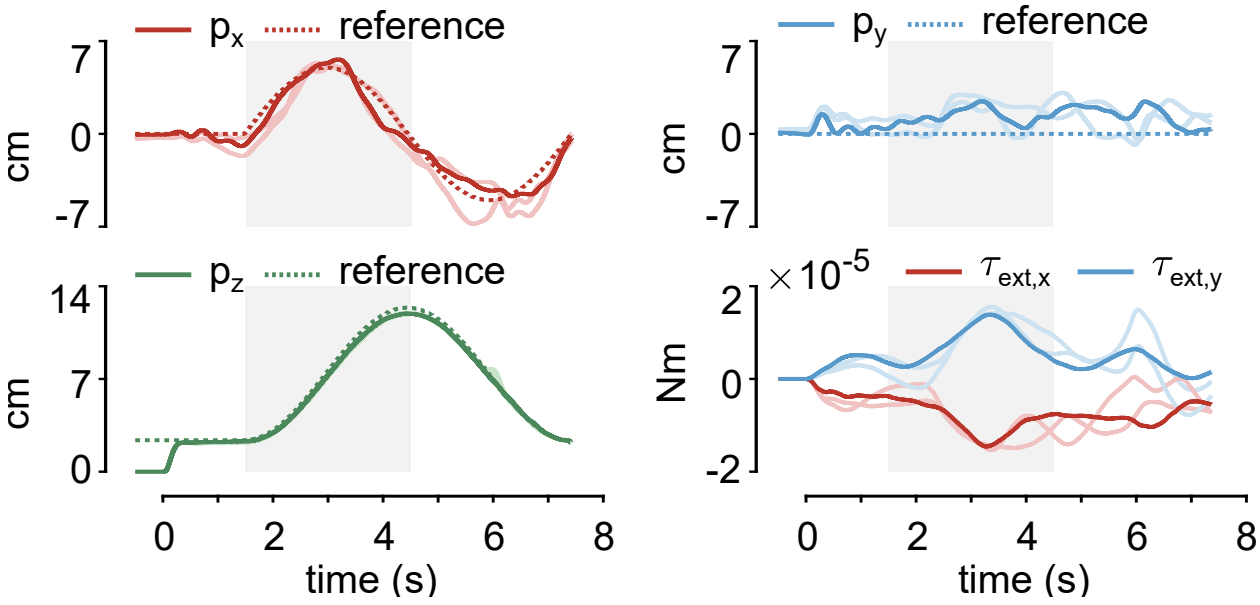


Figure 7.7: Performance in Task 3 (T3), where the robot tracks a long (7.5 s) circular trajectory, with a wide (5.0 cm) radius. The experiment has been repeated three times, and the results are included in the shaded lines. These results highlight the high and repeatable accuracy during agile flight of the RTMPC-like NN policy controlling the robot, as well as the role of the torque observer, which is capable to estimate and compensate for the effects of large rotational uncertainties. A composite image of the trajectory is shown in Fig. 7.1.

7.5.2 Aggressive, Near-Minimum Time Flight

In this part, we demonstrate the capabilities of the controller designed for acrobatic flights and discussed in Section 7.4. The controller runs at 1 kHz.

Controller Design

The Ancillary NMPC has a 0.5-second long prediction horizon, with $N = 100$ and $T_c = 0.005$; we set $\bar{f}_{\text{ext}}$ to correspond to 30% of the weight force acting on the robot.

Task: Near-Minimum Time Point-To-Point

The task consists of reaching a desired position (horizontal movement of 30 cm) in near-minimum time, while starting and terminating at hover (close-to-zero initial and final velocities and accelerations). This task is challenging because it requires rapid exploitation of the

coupling between translational dynamics (as the goal of the task consists in a translation) and rotational dynamics (as accelerating and breaking requires large and fast rotations of the thrust vector). The maneuver is repeated five times to demonstrate repeatability. The learned policy (two hidden layers, 128 neurons/layer, input size 17 consisting of position, velocity, attitude quaternion, angular velocity, desired end-position and time; output of size 3 consisting of thrust along ${}_B z$ and torque around ${}_B x$, ${}_B y$) ran offboard at 1 kHz, and was trained from 4 demonstrations collected in simulation (one with data augmentation, 100 samples per timestep, and the remaining three demonstrations for fine-tuning). Take-off and landing are also performed via the learned policy.

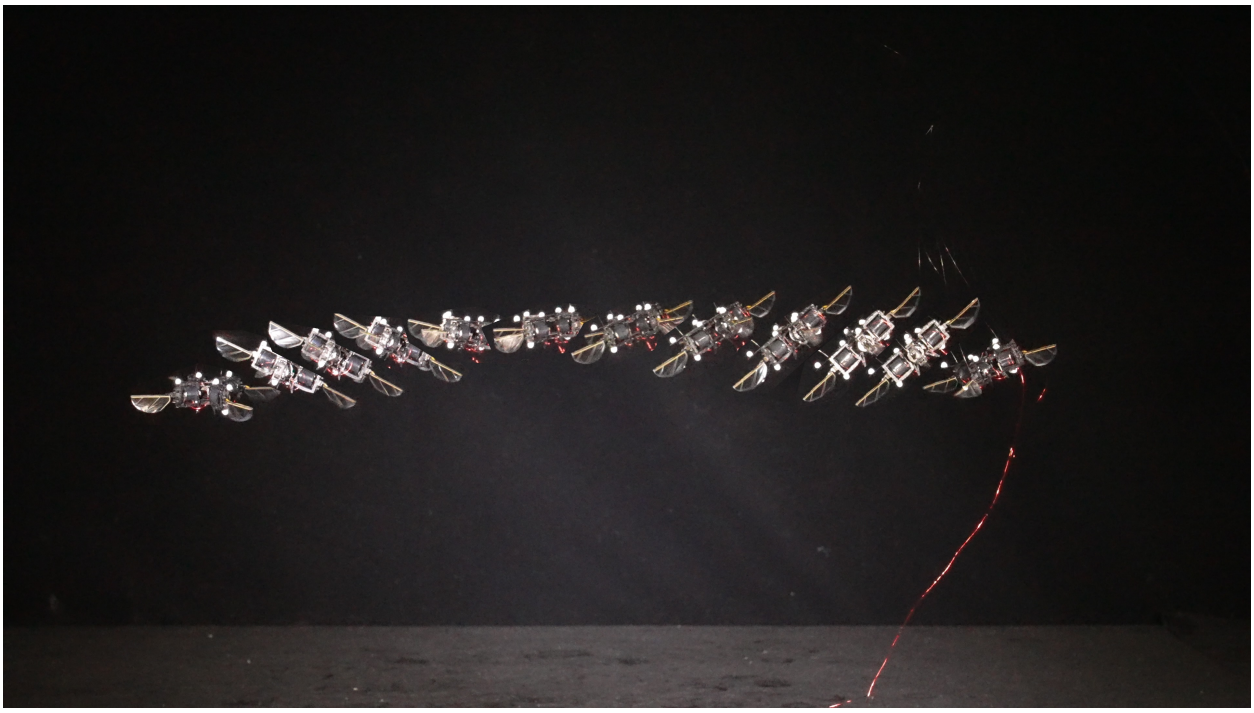


Figure 7.8: Composite image of the point-to-point (30 cm) maneuver performed in near-minimum time by the MIT SoftFly. The robot moves from the right to the left of the image ($-x$), reaching a velocity above 1.2 m/s and a pitch angle of 50 deg. This velocity is four times the maximum value reported in the literature of soft-actuated, insect-scale aerial robots. The policy runs at 1 kHz on a small offboard computer. This demonstrates the computational efficiency, performance, and robustness of the approach, which can effectively control a sub-gram, insect-scale, soft-actuated aerial robot. The safety tether and power cables were removed in post-processing, except for the first frame (*far right*).

The results are shown in Fig. 7.8 and Fig. 7.9. We highlight that:

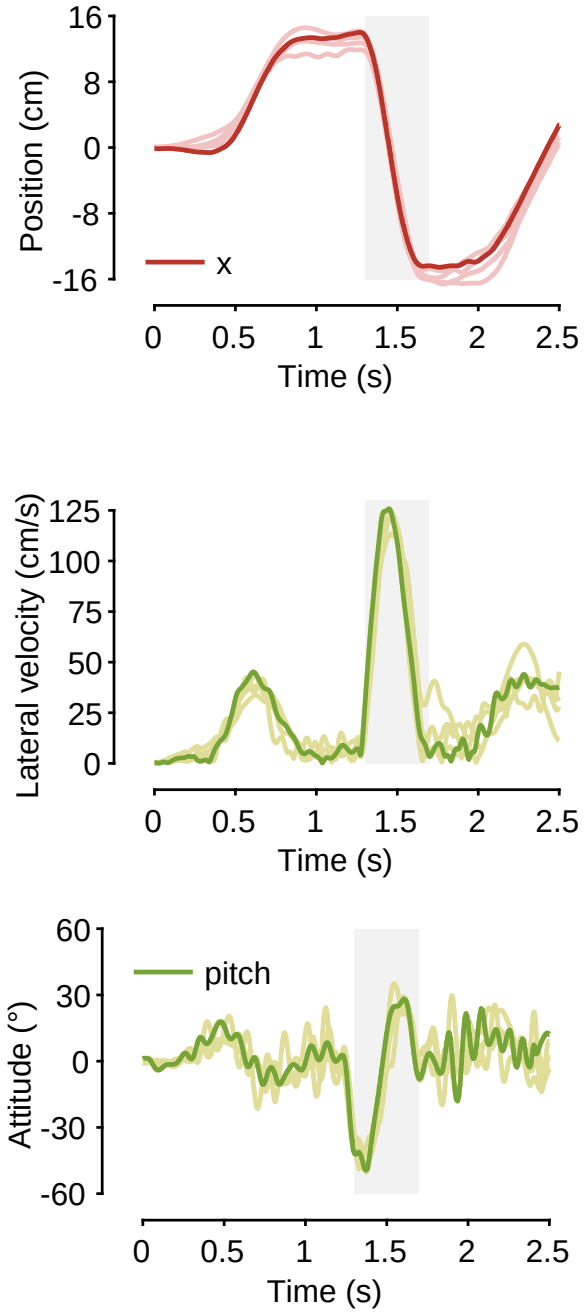


Figure 7.9: Velocity and attitude during the near-minimum time point-to-point maneuver. The robot reaches a velocity of about 1.2 m/s and a pitch angle of 50 deg. The maneuver is repeated 5 times (fastest run in solid color, other runs are in the shaded lines), demonstrating repeatability. The part of the maneuver shown in Fig. 7.8 corresponds to the gray-shaded area.

- The robot moves of 30 cm horizontally in about 0.5 s, reaching a peak velocity above 120 cm/s. This velocity is four times larger than the maximum horizontal velocity ever reported in the literature of soft-actuated, insect-scale aerial robots (30 cm/s [69]).
- During the maneuver, the robot reaches a pitch angle of about 50 deg, demonstrating the controller's aggressiveness and the platform's agility. We remark that achieving such aggressive flight was possible thanks to the approach presented in Chapter 4, which leverages nonlinear models at time-varying operating points.
- The maximum horizontal acceleration is above 1.2 g.

These results demonstrate that the proposed approach generalizes to an insect-scale aerial robot, achieving repeatable, low-computation, high-performance, and robust, agile flights with an efficiently trained policy.

7.6 Summary

This Chapter has presented the first robust MPC-like neural network policy capable of experimentally controlling a sub-gram MAV [60]. In our experimental evaluations, the proposed policy achieved high control rates (up to 2 kHz) on a small offboard computer, while demonstrating small (< 1.8 cm) RMS tracking errors, and the ability to withstand large external disturbances.

In addition, we presented the first strategy to achieve aggressive and fast flights on a new variant of a sub-gram MAV [69]. The presented control approach repeatably achieved an horizontal velocity above 120 cm/s, being four-times faster than existing work [69], using a policy learned from 4 demonstrations and that run at 1 kHz.

Chapter 8

Conclusions

8.1 Summary of Contributions

This thesis has presented strategies for efficient imitation learning of computationally efficient robust vision-based, adaptive policies from robust MPC, demonstrating their deployment onboard UAVs of different scales.

Chapter 3 and Chapter 4 have demonstrated that it is possible to generate motor policies from MPC that are fast and robust in the real-world, while requiring (a) few queries to expensive controller, (b) few environment interactions, and (c) short training times. Evaluations have validated the performance and data/computation efficiency, additionally showing that increasing the number of samples in DA or introducing a fine-tuning procedure can further improve performance. These findings have broad applicability beyond the MPC and IL communities. For example, our method can serve as an efficient policy pre-training procedure, using model and uncertainty priors, for subsequent fine-tuning via model-free RL, reducing inefficient random exploration in RL or simplifying reward design.

Chapter 5 has relaxed the constraining assumption in Chapter 3 and Chapter 4 that full state information is always available onboard, introducing a strategy for efficient vision-based policy learning. Key to this achievement was the design of a DA strategy enabling efficient

(demonstrations, training time) learning of a *sensorimotor* policy from MPC grounded in the output feedback RTMPC framework theory, unlike previous DA methods that rely on handcrafted heuristics, and used a NeRF to generate synthetic images. This approach was deployed in real-time (on the onboard Nvidia Jetson TX2, achieving average inference time of only 1.5 ms), demonstrating successful agile trajectory tracking with policies learned from a single demonstration. The policy used onboard fisheye images to infer the horizontal position of a multirotor despite aggressive 3D motion, and subject to a variety of sensing and dynamics disturbances.

Chapter 6 has provided SAMA, an adaptation strategy for the approach in Chapter 3, mitigating the performance degradation experienced due to uncertainties. Key to the approach was to leverage the performance of an RMA-like [41] adaptation scheme, but without relying on RL, therefore avoiding reward selection and tuning. The developed methodology was applied to the task of adaptive position and attitude control for a multirotor, demonstrating for the first time RMA-like adaptation to uncertainties that cause position and orientation errors, unlike previous work [44] that only focuses on adaptive attitude control. Hardware demonstrations have validated SAMA's robustness and performance, showcasing the ability of the approach to withstand a propeller failure even during trajectory tracking, a failure not explicitly considered during training.

Last, Chapter 7 has presented the first computationally-efficient strategy for robust, MPC-like control of sub-gram MAVs. First, we have presented an approach that employs a deep-learned NN policy that is trained to reproduce a trajectory tracking RTMPC, leveraging the methodology in Chapter 3. Second, we have presented aggressive flights (with velocities 4 times higher than previously reported in the literature), leveraging the methodology in Chapter 4. Our evaluations were performed with two different sub-gram MAVs, therefore additionally demonstrating generalization of our methodologies to multiple types of sub-gram aerial robots.

8.2 Future Work

We acknowledge many exciting opportunities for future work.

In Chapter 3 and Chapter 4, while our methodology has demonstrated real-world robustness, in the future we would like to leverage DNN reachability tools [143]–[145] to provide robustness certificates, enabling the deployment on safety-critical systems. In addition, while easy-to-compute fixed-size approximations of the tube have been sufficient to guide our DA strategy, future work will focus on leveraging tubes with varying cross-sections, enabling even more aggressive expert demonstrations.

In Chapter 5, the observer employed by the expert is a dynamical system whose dynamics, when performing DA, are approximated with a non-dynamical system; this approximation is valid if the observer is tuned to have sufficiently fast dynamics without significant overshooting, as done in our work. In the future, this approximation could be further improved by back-feeding into the policy the state estimate generated by the policy, part of our multi-task learning setup. Alternatively, an interesting extension could consists in designing a data augmentation strategy for recurrent neural networks, for example leveraging forward/backward reachability of the closed loop uncertain system to obtain sequences of measurements to be used for data augmentation. Future work will additionally generalize the approach to large visual changes in the environments by further randomization in image/NeRF space (for example, using [146] for weather conditions), and use event-cameras to ensure performance in poorly-lit environments. Last, it will also be interesting to leverage experts that use nonlinear models, using the techniques developed in Chapter 4.

In Chapter 6, future work will perform a more direct comparison with RMA for quadrotors [44], once code becomes available, and will deploy the approach to control the the MIT Softfly [60]. Another interesting extension consists in modifying Phase 0 by leveraging the expert and the data augmentation strategies for agile control presented in Chapter 4, enabling rapid adaptation while performing acrobatic maneuvers.

In Chapter 7, the presented results enable novel and exciting opportunities for agile control of sub-gram MAVs. First, the demonstrated robustness and computational efficiency paves the way for onboard deployment under real-world uncertainties. Second, the newly-developed agile flight capabilities enable data collection at different flight regimes, for model discovery and identification [141], [142]. Last, as the computational cost of a learned NN policy grows favorably with respect to state size, we can use larger, more sophisticated models (e.g., based on NN trained from real-world data) for control design, further exploiting the nimble characteristics of sub-gram MAVs.

References

[1] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[2] B. T. Lopez, "Adaptive robust model predictive control for nonlinear systems," Ph.D. dissertation, Massachusetts Institute of Technology, 2019.

[3] D. Q. Mayne, M. M. Seron, and S. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, 2005.

[4] D. Q. Mayne, S. V. Raković, R. Findeisen, and F. Allgöwer, "Robust output feedback model predictive control of constrained linear systems," *Automatica*, vol. 42, no. 7, pp. 1217–1222, 2006.

[5] B. T. Lopez, J.-J. E. Slotine, and J. P. How, "Dynamic tube mpc for nonlinear systems," in *2019 American Control Conference (ACC)*, IEEE, 2019, pp. 1655–1662.

[6] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems.," in *ICINCO (1)*, Citeseer, 2004, pp. 222–229.

[7] M. Kamel, M. Burri, and R. Siegwart, "Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3463–3469, 2017.

[8] M. V. Minniti, F. Farshidian, R. Grandia, and M. Hutter, "Whole-body mpc for a dynamically stable mobile manipulator," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3687–3694, 2019.

[9] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 1433–1440.

[10] J. B. Rawling and P. Kumar, *Industrial, large-scale model predictive control with deep neural networks*, http://helper.ipam.ucla.edu/publications/lco2020/lco2020_16406.pdf, (Accessed on 05/18/2021).

[11] S. Levine and V. Koltun, "Guided policy search," in *International conference on machine learning*, PMLR, 2013, pp. 1–9.

[12] G. Kahn, T. Zhang, S. Levine, and P. Abbeel, "Plato: Policy learning using adaptive trajectory optimization," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 3342–3349.

[13] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search," in *2016 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2016, pp. 528–535.

[14] J. Carius, F. Farshidian, and M. Hutter, "Mpc-net: A first principles guided policy search," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2897–2904, 2020.

[15] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep drone acrobatics," *Robotics, Science, and Systems (RSS)*, 2020.

[16] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *2013 IEEE international conference on robotics and automation*, IEEE, 2013, pp. 1765–1772.

[17] A. Reske, J. Carius, Y. Ma, F. Farshidian, and M. Hutter, "Imitation learning from mpc for quadrupedal multi-gait control," *arXiv preprint arXiv:2103.14331*, 2021.

[18] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," Carnegie-Mellon Univ Pittsburgh PA Artificial Intelligence and Psychology, Tech. Rep., 1989.

[19] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *arXiv preprint arXiv:1811.06711*, 2018.

[20] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[21] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[22] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2018, pp. 3803–3810.

[23] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 1–14, 2019.

[24] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "Dart: Noise injection for robust imitation learning," in *Conference on robot learning*, PMLR, 2017, pp. 143–156.

[25] D. Krishnamoorthy, "A sensitivity-based data augmentation framework for model predictive control policy approximation," *IEEE Transactions on Automatic Control*, vol. 67, no. 11, pp. 6090–6097, 2022. DOI: 10.1109/TAC.2021.3124983.

[26] D. Krishnamoorthy, "An improved data augmentation scheme for model predictive control policy approximation," *arXiv preprint arXiv:2303.05607*, 2023.

[27] D. Q. Mayne, E. C. Kerrigan, E. Van Wyk, and P. Falugi, "Tube-based robust nonlinear model predictive control," *International journal of robust and nonlinear control*, vol. 21, no. 11, pp. 1341–1353, 2011.

[28] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[29] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots, "Imitation learning for agile autonomous driving," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 286–302, 2020.

[30] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, eabg5810, 2021.

[31] K. Lee, B. Vlahov, J. Gibson, J. M. Rehg, and E. A. Theodorou, "Approximate inverse reinforcement learning from vision-based imitation learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 10 793–10 799. DOI: 10.1109/ICRA48506.2021.9560916.

[32] R. Bonatti, R. Madaan, V. Vineet, S. Scherer, and A. Kapoor, "Learning visuomotor policies for aerial navigation using cross-modal representations," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 1637–1644.

[33] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2015.

[34] D. Sharma, A. Kuwajerwala, and F. Shkurti, "Augmenting imitation experience via equivariant representations," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 9383–9389.

[35] M. Muller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Teaching uavs to race: End-to-end regression of agile controls in simulation," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018.

[36] A. Zhou, M. J. Kim, L. Wang, P. Florence, and C. Finn, "Nerf in the palm of your hand: Corrective augmentation for robotics via novel-view synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 907–17 917.

[37] J. P. How, B. Lopez, P. Lusk, and S. Morozov, "Performance analysis of adaptive dynamic tube MPC," p. 0785, 2021.

[38] M. Bujarbaruah, X. Zhang, U. Rosolia, and F. Borrelli, "Adaptive MPC for iterative tasks," in *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, 2018, pp. 6322–6327.

[39] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear MPC for quadrotors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 690–697, 2021.

[40] A. Saviolo, J. Frey, A. Rathod, M. Diehl, and G. Loianno, "Active learning of discrete-time dynamics for uncertainty-aware model predictive control," *arXiv preprint arXiv:2210.12583*, 2022.

[41] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *Robotics: Science and Systems (RSS)*, 2021.

[42] A. Kumar, Z. Li, J. Zeng, D. Pathak, K. Sreenath, and J. Malik, "Adapting rapid motor adaptation for bipedal robots," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 1161–1168.

[43] H. Qi, A. Kumar, R. Calandra, Y. Ma, and J. Malik, "In-hand object rotation via rapid motor adaptation," in *Conference on Robot Learning*, PMLR, 2023, pp. 1722–1732.

[44] D. Zhang, A. Loquercio, X. Wu, A. Kumar, J. Malik, and M. W. Mueller, "Learning a single near-hover position controller for vastly different quadcopters," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 1263–1269.

[45] P. Liu, S. P. Sane, J.-M. Mongeau, J. Zhao, and B. Cheng, "Flies land upside down on a ceiling using rapid visually mediated rotational maneuvers," *Science advances*, vol. 5, no. 10, eaax1877, 2019.

[46] S. B. Fuller, A. D. Straw, M. Y. Peek, R. M. Murray, and M. H. Dickinson, "Flying drosophila stabilize their vision-based velocity controller by sensing wind with their antennae," *Proceedings of the National Academy of Sciences*, vol. 111, no. 13, E1182–E1191, 2014.

[47] V. M. Ortega-Jimenez, R. Mittal, and T. L. Hedrick, "Hawkmoth flight performance in tornado-like whirlwind vortices," *Bioinspiration & biomimetics*, vol. 9, no. 2, p. 025 003, 2014.

[48] L. Ristroph, A. J. Bergou, G. Ristroph, K. Coumes, G. J. Berman, J. Guckenheimer, Z. J. Wang, and I. Cohen, "Discovering the flight autostabilizer of fruit flies by inducing aerial stumbles," *Proceedings of the National Academy of Sciences*, vol. 107, no. 11, pp. 4820–4824, 2010.

[49] A. K. Dickerson, P. G. Shankles, N. M. Madhavan, and D. L. Hu, "Mosquitoes survive raindrop collisions by virtue of their low mass," *Proceedings of the National Academy of Sciences*, vol. 109, no. 25, pp. 9822–9827, 2012.

[50] A. M. Mountcastle and S. A. Combes, "Biomechanical strategies for mitigating collision damage in insect wings: Structural design versus embedded elastic materials," *Journal of Experimental Biology*, vol. 217, no. 7, pp. 1108–1115, 2014.

[51] K. Y. Ma, P. Chirarattananon, S. B. Fuller, and R. J. Wood, "Controlled flight of a biologically inspired, insect-scale robot," *Science*, vol. 340, no. 6132, pp. 603–607, 2013.

[52] Y. Chen, H. Zhao, J. Mao, P. Chirarattananon, E. F. Helbling, N.-s. P. Hyun, D. R. Clarke, and R. J. Wood, "Controlled flight of a microrobot powered by soft artificial muscles," *Nature*, vol. 575, no. 7782, pp. 324–329, 2019.

[53] X. Yang, Y. Chen, L. Chang, A. A. Calderón, and N. O. Pérez-Arancibia, "Bee+: A 95-mg four-winged insect-scale flying robot driven by twinned unimorph actuators," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4270–4277, 2019.

[54] Y. M. Chukewad, J. James, A. Singh, and S. Fuller, "Robofly: An insect-sized robot with simplified fabrication that is capable of flight, ground, and water surface locomotion," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 2025–2040, 2021.

[55] *Drones by the Numbers*, [Online; accessed 11. Sep. 2022], Sep. 2022. URL: https://www.faa.gov/uas/resources/by_the_numbers.

[56] *Flyability — Drones for indoor inspection and confined space*, [Online; accessed 11. Sep. 2022], Sep. 2022. URL: https://www.flyability.com.

[57] *Skydio 2+™ and X2™ – Skydio Inc.* [Online; accessed 11. Sep. 2022], Sep. 2022. URL: https://www.skydio.com.

[58] R. D'Andrea, "Guest editorial can drones deliver?" *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 3, pp. 647–648, 2014.

[59] *Voliro Airborne Robotics*, [Online; accessed 11. Sep. 2022], Sep. 2020. URL: https://voliro.com.

[60] Y. Chen, S. Xu, Z. Ren, and P. Chirarattananon, "Collision resilient insect-scale soft-actuated aerial robots with high agility," *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1752–1764, 2021.

[61] Z. Ren, S. Kim, X. Ji, W. Zhu, F. Niroui, J. Kong, and Y. Chen, "A high-lift micro-aerial-robot powered by low-voltage and long-endurance dielectric elastomer actuators," *Advanced Materials*, vol. 34, no. 7, p. 2 106 757, 2022.

[62] A. Tagliabue, D.-K. Kim, M. Everett, and J. P. How, "Demonstration-efficient guided policy search via imitation of robust tube mpc," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 462–468. DOI: 10.1109/ICRA46639.2022.9812122.

[63] A. Tagliabue and J. P. How, "Efficient deep learning of robust policies from mpc using imitation and tube-guided data augmentation," *arXiv preprint arXiv:2306.00286*, 2023.

[64] A. Tagliabue and J. P. How, "Output feedback tube mpc-guided data augmentation for robust, efficient sensorimotor policy learning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 8644–8651.

[65] A. Tagliabue and J. P. How, "Tube-nerf: Efficient imitation learning of visuomotor policies from mpc via tube-guided data augmentation and nerfs," *IEEE Robotics and Automation Letters*, 2024.

[66] T. Zhao, A. Tagliabue, and J. P. How, "Efficient deep learning of robust, adaptive policies using tube mpc-guided data augmentation," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2023, pp. 1705–1712.

[67] A. Tagliabue, Y.-H. Hsiao, U. Fasel, J. N. Kutz, S. L. Brunton, Y. Chen, and J. P. How, "Robust, high-rate trajectory tracking on insect-scale soft-actuated aerial robots with deep-learned tube mpc," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 3383–3389.

[68] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*, IEEE, 2010, pp. 5420–5425.

[69] S. Kim, Y.-H. Hsiao, Z. Ren, J. Huang, and Y. Chen, "Acrobatics at the insect-scale: A durable, precise, and agile micro-aerial-robot," *under review*, 2024.

[70] X. Zhang, M. Bujarbaruah, and F. Borrelli, "Near-optimal rapid mpc using neural networks: A primal-dual policy learning framework," *IEEE Transactions on Control Systems Technology*, 2020.

[71] S. W. Chen, T. Wang, N. Atanasov, V. Kumar, and M. Morari, "Large scale model predictive control with neural networks and primal active sets," *Automatica*, vol. 135, p. 109 947, 2022.

[72] P. Kumar, J. B. Rawlings, and S. J. Wright, "Industrial, large-scale model predictive control with structured neural networks," *Computers & chemical engineering*, vol. 150, 2021, ISSN: 0098-1354.

[73] D. H. Jacobson and D. Q. Mayne, *Differential dynamic programming*. Elsevier Publishing Company, 1970.

[74] H. Tsukamoto and S.-J. Chung, "Learning-based robust motion planning with guaranteed stability: A contraction theory approach," *IEEE Robotics and Automation Letters*, 2021.

[75] 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 *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 8973–8979.

[76] S. Desai, I. Durugkar, H. Karnan, G. Warnell, J. Hanna, P. Stone, and A. Sony, "An imitation from observation approach to transfer learning with dynamics mismatch," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[77] H. Tsukamoto and S.-J. Chung, "Neural contraction metrics for robust estimation and control: A convex optimization approach," *IEEE Control Systems Letters*, vol. 5, no. 1, pp. 211–216, 2020.

[78] H. Nguyen, M. Kamel, K. Alexis, and R. Siegwart, "Model predictive control for micro aerial vehicles: A survey," in *2021 European Control Conference (ECC)*, IEEE, 2021, pp. 1556–1563.

[79] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza, "A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3357–3373, 2022.

[80] P. Foehn, A. Romero, and D. Scaramuzza, "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, vol. 6, no. 56, eabh1221, 2021.

[81] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for time-optimal quadrotor flight," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3340–3356, 2022.

[82] T. Manzoor, H. Pei, Z. Sun, and Z. Cheng, "Model predictive control technique for ducted fan aerial vehicles using physics-informed machine learning," *Drones*, vol. 7, no. 1, p. 4, 2022.

[83] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit," *Proceedings of the Royal Society A*, vol. 474, no. 2219, p. 20 180 335, 2018.

[84] A. Saviolo, G. Li, and G. Loianno, "Physics-inspired temporal learning of quadrotor dynamics for accurate model predictive trajectory tracking," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 256–10 263, 2022.

[85] N. A. Spielberg, M. Brown, and J. C. Gerdes, "Neural network model predictive motion control applied to automated driving with unknown friction," *IEEE Transactions on Control Systems Technology*, vol. 30, no. 5, pp. 1934–1945, 2021.

[86] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, precision, and payloads: Adaptive nonlinear mpc for quadrotors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 690–697, 2022. DOI: 10.1109/LRA.2021.3131690.

[87] C. Griwodz, S. Gasparini, L. Calvet, P. Gurdjos, F. Castan, B. Maujean, G. De Lillo, and Y. Lanthony, "Alicevision meshroom: An open-source 3d reconstruction pipeline," in *Proceedings of the 12th ACM Multimedia Systems Conference*, 2021, pp. 241–247.

[88] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *arXiv preprint arXiv:2201.05989*, 2022.

[89] A. Byravan, J. Humplik, L. Hasenclever, A. Brussee, F. Nori, T. Haarnoja, B. Moran, S. Bohez, F. Sadeghi, B. Vujatovic, *et al.*, "Nerf2real: Sim2real transfer of vision-guided bipedal motion skills using neural radiance fields," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 9362–9369.

[90] M. Adamkiewicz, T. Chen, A. Caccavale, R. Gardner, P. Culbertson, J. Bohg, and M. Schwager, "Vision-only robot navigation in a neural radiance world," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4606–4613, 2022.

[91] J. Lorenzetti and M. Pavone, "A simple and efficient tube-based robust output feedback model predictive control scheme," in *2020 European Control Conference (ECC)*, IEEE, 2020, pp. 1775–1782.

[92] J. Svacha, J. Paulos, G. Loianno, and V. Kumar, "Imu-based inertia estimation for a quadrotor using newton-euler dynamics," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 3861–3867, 2020.

[93] V. Wüest, V. Kumar, and G. Loianno, "Online estimation of geometric and inertia parameters for multirotor aerial vehicles," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 1884–1890.

[94] A. Tagliabue, A. Paris, S. Kim, R. Kubicek, S. Bergbreiter, and J. P. How, "Touch the wind: Simultaneous airflow, drag and interaction sensing on a multirotor," in *RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1645–1652.

[95] A. Tagliabue, M. Kamel, R. Siegwart, and J. Nieto, "Robust collaborative object transportation using multiple MAVs," *The International Journal of Robotics Research*, vol. 38, no. 9, pp. 1020–1044, 2019.

[96] C. D. McKinnon and A. P. Schoellig, "Unscented external force and torque estimation for quadrotors," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 5651–5657.

[97] G. Joshi and G. Chowdhary, "Deep model reference adaptive control," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, IEEE, 2019, pp. 4601–4608.

[98] G. Joshi, J. Virdi, and G. Chowdhary, "Design and flight evaluation of deep model reference adaptive controller," in *AIAA Scitech 2020 Forum*, 2020, p. 1336.

[99] S. Zhou, K. Pereida, W. Zhao, and A. P. Schoellig, "Bridging the model-reality gap with lipschitz network adaptation," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 642–649, 2021.

[100] S. M. Richards, N. Azizan, J.-J. Slotine, and M. Pavone, "Adaptive-control-oriented meta-learning for nonlinear systems," *Robotics: Science and Systems (RSS)*, 2021.

[101] M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural-fly enables rapid learning for agile flight in strong winds," *Science Robotics*, vol. 7, no. 66, eabm6597, 2022. DOI: 10.1126/scirobotics.abm6597.

[102] N. Hovakimyan, C. Cao, E. Kharisov, E. Xargay, and I. M. Gregory, "$\mathcal{L}_1$ Adaptive control for safety-critical systems," *IEEE Control Systems Magazine*, vol. 31, no. 5, pp. 54–104, 2011.

[103] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, "L1-adaptive mppi architecture for robust and agile control of multirotors," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 7661–7666.

[104] N. Pérez-Arancibia, *Challenging path to creating autonomous and controllable microrobots*, [Online; accessed 13. Sep. 2022], Apr. 2021. URL: https://youtu.be/4cdz2OPo9SI.

[105] P. Chirarattananon, K. Y. Ma, and R. J. Wood, "Adaptive control for takeoff, hovering, and landing of a robotic fly," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 3808–3815.

[106] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[107] W. M. Kouw and M. Loog, "A review of domain adaptation without target labels," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 3, pp. 766–785, 2021, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2019.2945942.

[108] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. DOI: 10.1007/s12532-020-00179-2. URL: https://doi.org/10.1007/s12532-020-00179-2.

[109] K. J. Aström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2021.

[110] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart, "Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system," in *Robot operating system (ROS)*, Springer, 2017, pp. 3–39.

[111] D. D. Fan, A.-a. Agha-mohammadi, and E. A. Theodorou, "Deep learning tubes for tube mpc," *arXiv preprint arXiv:2002.01587*, 2020.

[112] D. Limón, I. Alvarado, T. Alamo, and E. F. Camacho, "Robust tube-based mpc for tracking of constrained linear systems with additive disturbances," *Journal of Process Control*, vol. 20, no. 3, pp. 248–260, 2010.

[113] W. M. Kouw and M. Loog, "An introduction to domain adaptation and transfer learning," *arXiv preprint arXiv:1812.11806*, 2018.

[114] T. Lee, "Geometric tracking control of the attitude dynamics of a rigid body on so(3)," in *Proceedings of the 2011 American Control Conference*, 2011, pp. 1200–1205. DOI: 10.1109/ACC.2011.5990858.

[115] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[116] Aerospace Controls Laboratory, *Snap-stack: Autopilot code and host tools for flying snapdragon flight-based vehicles*, https://gitlab.com/mit-acl/fsw/snap-stack, (Accessed on 02/23/2020).

[117] A. Tagliabue, D.-K. Kim, M. Everett, and J. P. How, "Demonstration-efficient guided policy search via imitation of robust tube MPC," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 462–468. DOI: 10.1109/ICRA46639.2022.9812122.

[118] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *Journal of Control and Decision*, vol. 5, no. 1, pp. 42–60, 2018.

[119] J. Mattingley and S. Boyd, "Cvxgen: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, pp. 1–27, 2012.

[120] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[121] M. Diehl, "Real-time optimization for large scale nonlinear processes," Ph.D. dissertation, Heidelberg University, 2001.

[122] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 3480–3486.

[123] M. D. Shuster *et al.*, "A survey of attitude representations," *Navigation*, vol. 8, no. 9, pp. 439–517, 1993.

[124] J. Gillis, B. Vandewal, G. Pipeleers, and J. Swevers, "Effortless modeling of optimal control problems with rockit," in *39th Benelux Meeting on Systems and Control, Date: 2020/03/10-2020/03/12, Location: Elspeet, The Netherlands*, 2020.

[125] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "Acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, Jul. 2021, ISSN: 1867-2957. DOI: 10.1007/s12532-021-00208-8. URL: https://doi.org/10.1007/s12532-021-00208-8.

[126] G. Frison and M. Diehl, "Hpipm: A high-performance quadratic programming framework for model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020.

[127] T. Kusaka and T. Tanaka, "Stateful rotor for continuity of quaternion and fast sensor fusion algorithm using 9-axis sensors," *Sensors*, vol. 22, no. 20, p. 7989, 2022.

[128] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[129] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza, "Reaching the limit in autonomous racing: Optimal control versus reinforcement learning," *Science Robotics*, vol. 8, no. 82, eadg1462, 2023.

[130] URL: https://youtu.be/aWRuvy3LviI.

[131] M. Euston, P. Coote, R. Mahony, J. Kim, and T. Hamel, "A complementary filter for attitude estimation of a fixed-wing uav," in *2008 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2008, pp. 340–345.

[132] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[133] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[134] M. Grupp, *Evo: Python package for the evaluation of odometry and slam.* https://github.com/MichaelGrupp/evo, 2017.

[135] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 1280–1286.

[136] D. Mayne, M. Seron, and S. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, 2005, ISSN: 0005-1098.

[137] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[138] Y.-H. Hsiao, S. Kim, Z. Ren, and Y. Chen, "Heading control of a long-endurance insect-scale aerial robot powered by soft artificial muscles," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*.

[139] M. H. Dickinson, F.-O. Lehmann, and S. P. Sane, "Wing rotation and the aerodynamic basis of insect flight," *Science*, vol. 284, no. 5422, pp. 1954–1960, 1999.

[140] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.

[141] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the national academy of sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[142] U. Fasel, J. N. Kutz, B. W. Brunton, and S. L. Brunton, "Ensemble-sindy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control," *Proceedings of the Royal Society A*, vol. 478, no. 2260, p. 20 210 904, 2022.

[143] M. Everett, G. Habibi, C. Sun, and J. P. How, "Reachability analysis of neural feedback loops," *IEEE Access*, vol. 9, pp. 163 938–163 953, 2021. DOI: 10.1109/ACCESS.2021.3133370.

[144] N. Rober, S. M. Katz, C. Sidrane, E. Yel, M. Everett, M. J. Kochenderfer, and J. P. How, "Backward reachability analysis of neural feedback loops: Techniques for linear and nonlinear systems," *IEEE Open Journal of Control Systems*, vol. 2, pp. 108–124, 2023. DOI: 10.1109/OJCSYS.2023.3265901.

[145] C. Sidrane, A. Maleki, A. Irfan, and M. J. Kochenderfer, "Overt: An algorithm for safety verification of neural network control policies for nonlinear systems," *The Journal of Machine Learning Research*, vol. 23, no. 1, pp. 5090–5134, 2022.

[146] Y. Li, Z.-H. Lin, D. Forsyth, J.-B. Huang, and S. Wang, "Climatenerf: Extreme weather synthesis in neural radiance field," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.