Sample-efficient reinforcement learning for quadcopter flight control

L. Koomen

TU Delft

Sample-efficient reinforcement learning for quadcopter flight control

by

L. Koomen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday June 25, 2020 at 09:30 AM.

Student number:	4098730	
Project duration:	May 1, 2018 – June 25, 2020	
Thesis committee:	Dr. ir. G.C.H.E. de Croon,	TU Delft, chair
	Dr. ir. E. van Kampen,	TU Delft, supervisor
	Dr. A. Sharpanskykh,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This thesis is the culmination of my studies at the TU Delft, which have been the main focus of my life for a long time now. While I have enjoyed most of my time here, I am looking forward to move on and apply what I have learned to solve problems in the real world (and get paid). First of all, I would like to thank my supervisor, Erik-Jan, for patiently guiding me through the thesis process. Secondly, special thanks to my parents, Nico and Myrjam, for their unwavering support during my studies and throughout life in general, without which I would certainly not be where I am right now. Also, thanks to my siblings, Mick and Annemarie, for lifting me up whenever I come to visit. Furthermore, thanks to the rest of my family, friends, and colleagues for being a part of my life. And last, but certainly not least, I want to thank my girlfriend, Pat, for being my greatest supporter and keeping me sane when the going got tough.

L. Koomen
Delft, June 2020

Contents

Acronyms vii

1 Introduction 1
1.1 Research questions . . . 2
1.2 Outline . . . 3

2 Research questions 5

I Scientific paper 11

II Literature review and Preliminary analysis 33

3 Reinforcement learning 35
3.1 Motivation . . . 35
3.2 Important concepts . . . 35
3.2.1 Framework . . . 35
3.2.2 Accumulated reward . . . 37
3.2.3 Value functions . . . 37
3.2.4 State-value functions . . . 37
3.2.5 Action-value functions . . . 38
3.2.6 Dynamic Programming . . . 39
3.2.7 Monte Carlo methods . . . 40
3.2.8 Temporal-Difference methods . . . 41
3.2.9 Approximate methods . . . 42
3.2.10 Policy-gradient methods . . . 42
3.3 State-of-the-art . . . 43

4 Learning from Demonstration 45
4.1 Definition and framework . . . 45
4.1.1 Definition . . . 45
4.1.2 Framework . . . 46
4.1.3 Important concepts . . . 48
4.2 History and state-of-the-art . . . 50
4.2.1 Programming by Demonstration . . . 50
4.2.2 Behavioral cloning . . . 51
4.2.3 Imitation learning . . . 51
4.2.4 Applications in reinforcement learning . . . 52

5 Guided Policy Search 55
5.1 Motivation . . . 55
5.2 Rough outline . . . 56
5.3 Mirror Descent Guided Policy Search . . . 57

6 Preliminary results 67
6.1 Methods . . . 67
6.1.1 Pendulum . . . 67
6.1.2 Point mass - full view . . . 68
6.1.3 Point mass - partial view . . . 68

6.2 Results . . . 69
6.3 Analysis . . . 72
6.3.1 Proof of concept . . . 72
6.3.2 Unobservable states . . . 73
6.3.3 Generalization to unseen states . . . 73
6.3.4 Failure of trajectory optimization . . . 74

III Additional results and Discussion 75

7 Additional results and discussion 77
7.1 Research question 5 . . . 77
7.2 Research question 6 . . . 79
7.3 Research question 7 . . . 80

IV Closure 83

8 Conclusion 85

9 Recommendations 89

Bibliography 91

V Appendices 97

A List of GPS applications 99

B Preliminary experiments neural networks 101
B.1 Point-mass . . . 101

C Default parameter settings 103
C.1 General . . . 103
C.2 Sub-policies . . . 104
C.3 Trajectory optimization . . . 104
C.4 Dynamics models . . . 104
C.5 Global-policy . . . 105

D Quadcopter model and controller 107
D.1 Dynamic model . . . 107
D.2 Controller . . . 108
D.2.1 System Simplification . . . 112
D.2.2 Simulation Results . . . 113

Acronyms

A

AC	actor-critic.
AggeVaTe	aggregate values to imitate.
AIS-GPS	adaptive importance sampled GPS (IS-GPS).
ANN	artificial neural network.
API	approximate policy iteration.
APID	approximate policy iteration with demonstration.

B

BADMM	Bregmann alternating direction method of multipliers.
BADMM-GPS	Bregmann alternating direction method of multipliers (BADMM)-GPS.
BC	behavioral cloning.

C

C-GPS	constrained GPS.
CNN	convolutional neural network.
CPG	central pattern generator.

D

DAgger	dataset aggregation.
DDP	differential dynamic programming.
DDPG	deep deterministic policy gradients.
DGD	dual gradient descent.
DP	dynamic programming.

E

ELU	exponential linear unit.

G

GAIL	generalized adversarial imitation learning.
GMM	Gaussian mixture model.
GMR	generative motor reflexes.
GPI	generalized policy iteration.
GPS	guided policy search.

H

HMM	hidden Markov model.

I

IL	imitation learning.
iLQG	iterative linear quadratic Gaussian.
iLQG-FLM	iterative linear quadratic Gaussian (iLQG) with fitted linear models.
IM-TRPO	trust-region policy optimization with imitation learning.
IRL	inverse reinforcement learning.

IS-GPS importance sampled GPS.

K

KL Kullback–Leibler.

L

LfD learning from demonstration.
LQG linear quadratic Gaussian.
LSPI least-squares policy iteration.

M

MAV micro air vehicle.
MC Monte Carlo.
MD mirror-descent.
MD-GPS mirror descent guided policy search.
MDP Markov decision process.
MPC model predictive control.
MSE mean squared error.

P

PbD programming by demonstration.

R

RAIL reduction-based active imitation learning.
RL reinforcement learning.
RLS recursive least-squares.
RNN recurrent neural network.

S

SGD stochastic gradient descent.
SMILe stochastic mixing iterative learning.
SOTA state-of-the-art.
SVM support vector machine.

T

TD temporal difference.
TL transfer learning.
TRPO trust region policy optimization.
TVLG time-varying linear Gaussian.

U

UAV unmanced aerial vehicle.

V

V-GPS variational GPS.

1 Introduction

Ever since the invention of fixed-wing flight, stabilization and control of an aircraft has been of utmost importance for safety and performance. To reduce pilot workload, automatic flight controllers were developed not long after the first successful powered flight by the Wright brothers. Lawrence Sperry and others invented a mechanical autopilot, which was used in the first artificially controlled flight[82]. This mechanical system was eventually followed by analog- and finally digital autopilots using fly-by-wire.

Flight control is very relevant for safety in modern aviation, as shown by a summary of jet airplane accidents; more than 25% of fatal commercial jet accidents occur because of loss of control in flight [8]. This makes it the most frequently occurring type of accident and signifies a need for more research into intelligent, adaptive, and fault-tolerant control. Additionally, the rise of unmanced aerial vehicle (UAV) and micro air vehicle (MAV) use has lead to an increased demand for autonomous control, because there is no on-board pilot and air-ground communication is limited. Finally, high-performance aircraft and MAVs alike are prone to highly nonlinear- and rapidly changing dynamics, which require much effort to analyze, model, and synthesize controllers for. These issues lead to a need for self-learning and flexible control schemes, without a need for human input.

The problem of adaptive control has been studied extensively, leading to many useful techniques. However, many methods require a controller-structure to be defined a priori, adapting only the parameters on-line either by means of a model-reference (model-reference adaptive control), stability criterion (direct Lyapunov-based adaptive control), or an identified model (model identification adaptive control). Also, direct Lyapunov-based control requires the system dynamics to be known and model identification based control requires the dynamics to be modeled accurately to achieve high performance.

Another paradigm which could provide autonomous adaptive flight control, while circumventing some of the issues mentioned previously, is reinforcement learning (RL). Inspired by nature, RL mimics how humans and animals learn from interactions with their environment. The continuously self-learning algorithm is inherently adaptive, being a form of adaptive- and optimal control[73]. Since it can learn model-free, it can be applied to arbitrary dynamics and problem-structures, making it very flexible. Also, the amount of human input is small as only a reward signal needs to be defined. Mainly focused on robotics, RL has been applied to flight control as well; aerobatic helicopter flight[4, 61], perched UAV landings[83], autonomous UAV navigation[37], and simulated business jet flight control[22] among others.

Although RL seems to hold great promise for creating more intelligent control algorithms, it is not without its flaws. For example, many RL algorithms are known for requiring an inordinate amount of data to learn effectively , often leading to experiments and applications where data is abundant, such as simulations[72, 86]. This flaw is called having low sample-efficiency and the main goal of this thesis will be improving it in a flight control application.

There are many ways of improving sample-efficiency, one of which is called learning from demonstration (LfD). LfD is an umbrella-term for many techniques which all try to jump-start learning by imitating a, often supposed to be expert, teacher. These methods rely on the idea that the teacher is able to more quickly guide the student towards regions of high reward in the space of trajectories, circumventing a substantial amount of trial-and-error. These techniques have been applied successfully to autonomously traverse rough terrain using a robotic vehicle[40], reproduce demonstrated trajectories using a robotic arm[81], and learn a controller for a multi-rotor UAV[15].

The main goal of this thesis is to:

Main goal

Investigate how learning from demonstration can be used to improve the sample-efficiency of reinforcement learning applied to flight control.

This leads to several research questions which are investigated and answered in this work, using both literature review and experiments on a simulated quadcopter system.

1.1. Research questions

This section will provide a short overview of the research questions answered in this work.

To reach the main goal, first, a specific RL algorithm needs to be chosen that fits well with a flight control application, and preferably has a relatively high baseline sample-efficiency. Therefore, the first research question is:

Research question 1

What is an RL algorithm that fits well with a flight control application, is a good fit to be combined with LfD, and preferably has high baseline sample-efficiency?

This initial research question is key in formulating the remaining ones posed in this thesis report and it will be answered by literature review and preliminary analysis, as presented in part II. To better introduce the remaining research questions, the result from this review and analysis is presented here; the RL algorithm that is used in this work is mirror descent guided policy search (MD-GPS)[56]. This choice leads to six more refined research questions, specific to MD-GPS, which are presented here to provide a central point at which all research questions are posed. Three of these questions are handled in the scientific paper presented at part I; research question 2, 3, and 4, while the rest are answered in the remainder of the thesis document at part III. For a more thorough explanation of all research questions, see chapter 2.

As it turns out, MD-GPS is an iterative optimization algorithm that can be initialized using demonstrations. The effect of this intervention on the sample-efficiency will be investigated by the second research question, which is answered in part I:

Research question 2

How does initializing the sub-policies of MD-GPS with high-quality demonstrations affect the sample efficiency?

Although demonstrations are usually provided by an expert teacher, in general they are not optimal. Therefore, it is necessary to investigate the robustness of MD-GPS to sub-optimal demonstration. The third research question investigates this robustness:

Research question 3

How robust is MD-GPS to sub-policies initialized using sub-optimal demonstrations?

The MD-GPS algorithm as specified in [56] has a certain training structure, which can be altered slightly to improve sample-efficiency. The specifics of this structure are explained more thoroughly in section 3 of part I. The fourth research question investigates the effect of alterations to this structure on the sample-efficiency:

Research question 4

How does the specific training structure affect the sample-efficiency of MD-GPS?

The remaining questions are treated in the remainder of the thesis report, with results and discussion presented in chapter 7. As it turns out, MD-GPS uses model-based trajectory-centric RL, based on local dynamic models. These models can be trained beforehand using data from demonstrations, the effect of this pre-training on the sample-efficiency is investigated by the fifth research question:

Research question 5

How does pre-training the dynamics prior and time-varying linear Gaussian (TVLG) models, using demonstrations, affect the sample efficiency of MD-GPS?

Inspired by the field of inverse RL, which reverses the RL paradigm and sets out to find a reward-function from (optimal) behavior, instead of inferring behavior from a reward-function, demonstrations can be used to define a custom cost-function. The effect on using such a cost-function on the sample-efficiency is investigated by the sixth research question:

Research question 6

How does utilizing demonstrations to create a cost-function affect the sample-efficiency of MD-GPS?

As is explained in section 2 of part I and section 5.3, MD-GPS uses a Kullback–Leibler (KL)-divergence constraint during optimization, which acts similar to a step size of an iterative optimization algorithm. The final research question investigates the robustness of MD-GPS to different values of this step size:

Research question 7

How robust is MD-GPS to different settings of the KL-divergence step size increase/decrease factor?

1.2. Outline

This section will outline the structure of the remainder of this thesis. First, the main scientific contribution is presented in part I. This is followed by a literature review and preliminary analysis in part II. The literature review starts with a presentation of the fundamentals of RL and an overview of the state-of-the-art (SOTA) in chapter 3. Secondly, the fundamentals of LfD and an overview of the history and SOTA is presented in chapter 4. The literature review is finalized by an in-depth explanation of the MD-GPS algorithm in chapter 5. The preliminary analysis is presented in chapter 6, and the preliminary conclusions are combined with the conclusions regarding the entire thesis report presented in chapter 8. In part III, additional results and analysis are presented. Finally, the thesis is concluded and recommendations for further research are provided in part IV, starting with the conclusion in chapter 8, followed by the recommendations in chapter 9.

2

Research questions

This chapter will present the sub-questions that will help to answer the main research question. Each posed sub-question will be followed by a hypothesis and a descriptions of the experiments that will be performed.

As stated in the introduction, the main goal of the thesis is to:

Main goal

Investigate how learning from demonstration can be used to improve the sample-efficiency of reinforcement learning applied to flight control.

Research question 1

To achieve this goal, a RL algorithm needs to be chosen first. This choice will be the base for the choice of the specific LfD methods will be used, and any, more specific, research questions. This leads to the following first question:

Research question 1

What is an RL algorithm that fits well with a flight control application, is a good fit to be combined with LfD, and preferably has high baseline sample-efficiency?

As mentioned already in chapter 1, this question will be answered by a literature review in combination with several preliminary experiments. The result of this analysis is the choice of MD-GPS as the RL algorithm, which leads to the more specific research questions that follow.

During the LfD literature review it is noted that there are several ways of utilizing the information provided by demonstrations in a RL context; state-action mapping, reward function learning, system identification, and feature extraction (see section 4.1.2). Ignoring the latter during this work, these points can be mapped to the MD-GPS algorithm as follows:

- State-action mapping
  - **Initialize sub-policies**: demonstrations can be used to construct an initial TVLG sub-policy, providing a high-quality starting point in trajectory space.
  - **Pre-train global-policy**: demonstrations can be used to pre-train the global-policy, leading to a better baseline performance at the beginning of the training process.
- Reward function learning
  - **Demonstration based cost-function**: demonstrations can be used to specify a specific cost-function that forces the agent to follow that particular trajectory.
- System identification

– **Pre-fit dynamics models**: demonstrations can be used to pre-fit the dynamics models, both the Gaussian mixture model (GMM) prior and the TVLG local models.

To determine the effect of these demonstration-based interventions, experiments will be performed and the results analyzed. In each experiment the goal is to change one (or at most a few) key-parameter(s), while keeping all others the same. This will provide a first-order approximation of the effect of parameters around the nominal values. These nominal values are referred to as the "default" values. They have been chosen largely by trial-and-error, and lead to reasonable performance of the algorithm on the quadcopter system, making them suitable as a baseline. The values of the default parameters are presented both in section 3.G in part I, and in more detail in appendix C.

Research question 2

The C-step of the MD-GPS algorithm uses iterative linear quadratic Gaussian (iLQG) with fitted linear models (iLQG-FLM); a local trajectory optimization algorithm which uses local models to incrementally improve upon its current solution. Therefore, it can get stuck in local optima. In previous research like [48], demonstrations have been used to initialize the sub-policies to improve training performance, improving sampling-efficiency and being required for convergence to a good solution. The second research question aims to determine the effect on sample-efficiency:

Research question 2

How does initializing the sub-policies of MD-GPS with high-quality demonstrations affect the sample efficiency?

It is expected that initializing the sub-policies using high-quality demonstrations allows the trajectory-optimization to converge to a good solution faster than normal. Depending on the complexity of the system, it might even be crucial for convergence. The effect would be a starting performance that is much better than when initializing the sub-policies (semi-)randomly, skipping ahead in the cost-iteration curve.

To answer this question, two experiments will be run with the only difference being the initialization of the sub-policies:

experiment code	**sub-policy initialization method**
RQ2-A (E1)	random
RQ-CTRL (E2)	high-quality demonstrations (default)

Note that behind the experiment codes used in this thesis report, the corresponding experiment codes used in the scientific paper in part I are shown in parentheses. High-quality demonstrations will be generated using the nonlinear quaternion-based controller as presented in appendix D.

Research question 3

Generally, demonstrations are not the optimal solution as judged by the cost-function; there must always be some amount of additional optimization. The sensitivity of the algorithm to different levels of demonstration sub-optimality, i.e. difference between the demonstration and the optimal solution, will be referred to as its robustness. The more robust the algorithm, the easier it is to obtain a certain level of performance even when using quite sub-optimal demonstrations. Since the main goal of the thesis is to improve sample-efficiency using demonstrations, the robustness of the algorithm to sub-optimal demonstrations is important. The third research question aims to assess this robustness:

Research question 3

How robust is MD-GPS to sub-policies initialized using sub-optimal demonstrations?

As mentioned before, iLQG-FLM is a local trajectory optimization method and analogous to local gradient-based optimization there will be a region of convergence around a local optimum. The question becomes whether the initial demonstration lies within the region of convergence of the optimal solution in trajectory

space. As long as the demonstration is reasonably close to the optimal trajectory, it is expected to converge to this solution.

To answer this question, five experiments will be run with differences in the quality of the demonstrations with which the sub-policies are initialized. Behind the experiment codes used in this thesis report, the corresponding experiment codes used in the scientific paper in part I are shown in parentheses:

experiment code	**sub-policy initialization method**	**global-policy pre-training target**
RQ-CTRL (E1)	high-quality demonstrations (default)	high-quality demonstrations (default)
RQ3-B (E3)	demonstrations with 0.25 m target offset	demonstrations with 0.25 m target offset
RQ3-C (E4)	demonstrations with 1.00 m target offset	demonstrations with 1.00 m target offset
RQ3-D (E5)	squiggly demonstrations with 7.5 noise variance	squiggly demonstrations with 7.5 noise variance
RQ3-E (E6)	squiggly demonstrations with 7.5 noise variance	squiggly demonstrations with 15.0 noise variance

High-quality demonstrations will be generated using the quaternion-based controller as presented in appendix D. The squiggly demonstration is generated by adding noise to output of the quaternion-based controller, while the demonstrations with target offset are created by using quaternion-based controller with a target-position that is offset from the actual target.

Research question 4

Due to the unstable nature of the quadcopter system, no stability guarantees provided by neural networks, and the fact that iLQG-FLM does not provide any way to robustly restore itself back to a previous solution, a global-policy with poor performance can completely derail the training process. The way a poor global-policy can affect training is either by being used to sample, or by being involved in the KL-divergence constraint. Therefore, it is expected to be vital that the global-policy leads to stable trajectories before it is involved in these ways.

To this end, the training process can be structured in a certain way. Where structure refers to the choices made concerning the exact training iterations during which the following actions are undertaken:

- constrain w.r.t. previous sub-policies or global-policy
- sample using sub-policies or global-policy
- start supervised learning of the global-policy

An additional choice is whether to pre-train the global-policy using demonstrations.

The fourth research question asks what the effect is of different structure-settings:

Research question 4

How does the specific training structure affect the sample-efficiency of MD-GPS?

It is expected that the best way of structuring training is to begin by both sampling and constraining w.r.t. sub-policies, moving to constraining sub-policies w.r.t. the global-policy when it has been trained for a number of iterations, while finally moving to using the global-policy for sampling. Supervised learning of the global-policy should take place only when the sub-policies show good behavior. By using demonstrations to initialize the sub-policies, supervised training could start at the first iteration. Also, using demonstrations to pre-train the global-policy would be expected to reduce the training iterations needed to get the global-policy to generate stable trajectories and be useful to sample from.

In the interest of time the experiments will all follow the same general progression: constrain w.r.t. sub-policy → constrain w.r.t. global-policy → sample using global-policy Keeping the decision whether to pre-train the global-policy as a separate choice. The settings for each experiment are as follows:

experiment code	pre-train global-policy?	constrain w.r.t. sub-policy	sample using sub-policies
RQ4-B (E7)	no	iteration 0-4	iteration 0-9
RQ-CTRL (E8)	yes (default)	iteration 0-4 (default)	iteration 0-9 (default)
RQ4-C (E9)	no	iteration 0-9	iteration 0-19
RQ4-F (E10)	yes	iteration 0-9	iteration 0-19
RQ4-A (E11)	no	-	-
RQ4-D (E12)	yes	-	-

Note that behind the experiment codes used in this thesis report, the corresponding experiment codes used in the scientific paper in part I are shown in parentheses.

Research question 5

Since MD-GPS utilizes a model-based trajectory optimization technique, it relies on good dynamical models. Demonstrations provide a dataset of world interactions, which are useful for pre-training the dynamic models. The fifth research question aims to measure the effect this has on sample-efficiency:

Research question 5

How does pre-training the dynamics prior and TVLG models, using demonstrations, affect the sample efficiency of MD-GPS?

Pre-fitting the dynamics prior and models is expected to improve training performance since normally the models still need to be fit during the first few iterations, leading to worse optimization steps. The effect is expected to be that the cost will more quickly improve during the first few iterations, leading to slightly faster convergence overall.

To answer this question, three experiments will be run with the only difference being the quality of the demonstrations with which the sub-policies are initialized:

experiment code	pre-train dynamics?
RQ5-A	no
RQ-CTRL	yes (default)

Research question 6

Although the final goal of RL is to obtain a global-policy that optimizes the cost-function, it is always possible to utilize an altered cost-function during training to help the agent learn, akin to curriculum learning. Demonstrations can be used to define a cost-function that is optimized when the agent exactly follows the demonstration trajectory, which might help the agent during training by providing a more informative cost-function. The aim of the sixth research question is to measure the effect of such a cost-function on sample-efficiency:

Research question 6

How does utilizing demonstrations to create a cost-function affect the sample-efficiency of MD-GPS?

The demonstration-based cost-function would specify a target-state for each time step based on the demonstration trajectories. This would in principle provide more useful information to the algorithm as compared to only specifying a constant target-position for every time step. This extra information is expected to more quickly lead to good performance, but of course the performance will be limited by the performance of the demonstration. Therefore, there should be a point at which the cost-function switches back to normal, so performance can be further improved.

To answer this question, three experiments will be performed; one control, and two utilizing a demonstration-based cost-function up to different iterations:

experiment code	demonstration-based cost
RQ-CTRL	- (default)
RQ6-B	iteration 0-5
RQ6-C	iteration 0-10

Research question 7

The trajectory-optimization algorithm used by MD-GPS, iLQG-FLM, incrementally improves upon solutions by optimizing an augmented cost-function that, in addition to the normal cost-function, penalizes deviation from the current solution as measured by the KL-divergence. Each iteration the algorithm updates the trajectories to a point in trajectory-space that is a certain KL-divergence away from the old solution; a step size. The step size is adapted on-line, it is in- or decreased by some factor depending on some heuristic. While the main goal of this thesis is to improve sample-efficiency using demonstrations, it is important to assess the sensitivity of these improvements to the adaptation factor. The seventh research question aims to determine the robustness of MD-GPS training performance with respect to this factor:

Research question 7

How robust is MD-GPS to different settings of the KL-divergence step size increase/decrease factor?

The learning-rate modification factor determines the speed at which the algorithm adapts to either good or bad optimization results. The higher the factor, the faster the adaptation and the better the optimization performance. However, as the modification factor is increased there comes a point where the algorithm is destabilized, similar to destabilization of gradient-based learning would when choosing a step size that is too large.

To answer this question different values for the modification factor will be compared. To reduce the number of parameters, the adaptation factor will be simplified by making the decrease factor equal to the inverse of the (increase) factor.

experiment code	learning-rate modification factor
RQ7-A	1.15
RQ-CTRL	1.33 (default)
RQ7-C	1.5
RQ7-D	2.0
RQ7-E	4.0

I

Scientific paper

Improving sample-efficiency of mirror-descent guided policy search for flight control using demonstrations

L. Koomen *
Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands

The combination of reinforcement learning and deep neural networks has the potential to train intelligent autonomous agents on high dimensional sensory inputs, with applications in flight control. However, the amount of samples needed by these methods is often too large to use real-world interaction. In this work, mirror-descent guided policy search is identified as a promising algorithm to train high-dimensional policies on real-world samples. Several experiments are conducted to investigate how the use of expert-demonstrations can further improve the sample-efficiency of this algorithm when applied to the control of a quadcopter in simulation. It is shown how demonstrations, when combined with certain alterations in the mirror descent guided policy search algorithm, can significantly reduce the amount of samples needed to achieve good performance. Additionally, it is shown how these improvements are robust to sub-optimal demonstrations.

Nomenclature

Symbol	Explanation
$\boldsymbol{x}_t$	Markovian state vector at time $t \in [0, T-1]$
$\boldsymbol{o}_t$	Observation vector at time $t \in [0, T-1]$
$\boldsymbol{u}_t$	Action vector at time $t \in [0, T-1]$
$\boldsymbol{\tau}$	Trajectory: $\boldsymbol{\tau} = (\{\boldsymbol{x}_t, \boldsymbol{o}_t, \boldsymbol{u}_t\}, \cdots, \boldsymbol{u}_{T-1}\})$, inclusion of observations depends on the context
$\ell(\boldsymbol{\tau})$	The cost of a trajectory
$p(\boldsymbol{x}_{t+1}\|\boldsymbol{x}_t, \boldsymbol{u}_t)$	Unknown system dynamics
$p_i(\boldsymbol{u}_t\|\boldsymbol{x}_t)$	Action distribution of the i^{th} trajectory distribution, also called the i^{th} sub-policy, which starts at $\boldsymbol{x}_0^i \sim p(\boldsymbol{x}_0)$
$p_i(\boldsymbol{\tau})$	The i^{th} trajectory distribution, induced by $p_i(\boldsymbol{u}_t\|\boldsymbol{x}_t)$ when starting at $\boldsymbol{x}_0^i$
$\boldsymbol{\tau}_i^j$	The j^{th} realization of $p_i(\boldsymbol{\tau})$
$p_i(\boldsymbol{x}_{t+1}\|\boldsymbol{x}_t, \boldsymbol{u}_t)$	Linearized system dynamics around $p_i(\boldsymbol{\tau})$, in the form of a linear-Gaussian model
$\pi_\theta(\boldsymbol{u}_t\|\boldsymbol{o}_t)$	Global policy, action distribution conditioned on the observation, parameterized by weight vector $\boldsymbol{\theta}$
$\pi_{\theta,i}(\boldsymbol{\tau})$	The trajectory distribution induced by the global policy, while starting at $\boldsymbol{x}_0^i$
$\bar{\pi}_{\theta,i}(\boldsymbol{\tau})$	The trajectory distribution induced by a policy which is the result of linearizing the global policy around $p_i(\boldsymbol{\tau})$, or $\pi_{\theta,i}(\boldsymbol{\tau})$, while starting at $\boldsymbol{x}_0^i$
$\mathcal{D}_i$	Set of sampled trajectories $\{\boldsymbol{\tau}_{i,j}\}$
ϵ	Kullback–Leibler (KL)-divergence step size

I. Introduction

In recent years, micro air vehicle (MAV) technology has become more affordable and accessible, which has lead to a surge of interest in adaptive, autonomous, and intelligent flight control systems. Simultaneously, the field of deep

*Graduate student, Faculty of Aerospace Engineering, Control and Simulation division, Delft University of Technology

learning has revolutionized computer vision. The combination of these fields promises intelligent flight control policies trained directly on high-dimensional observations.

An optimal control paradigm that shows promise in training expressive policies is reinforcement learning (RL). Inspired by nature, RL mimics how humans and animals learn from interactions with their environment. The continuously self-learning algorithm is inherently adaptive, being a form of adaptive- and optimal control[1]. Since it can learn model-free, it is applicable to arbitrary dynamics and problem-structures, making it very flexible. Also, the degree of required human input is small as only a reward signal needs to be defined. Mainly focused on robotics, RL has been applied to flight control as well; aerobatic helicopter flight[2, 3], perched unmanced aerial vehicle (UAV) landings[4], autonomous UAV navigation[5], and simulated business jet flight control[6] among others.

Although RL hods great promise for creating more intelligent control algorithms, it is not without its flaws. For example, many RL algorithms are known for requiring an inordinate amount of data to learn effectively, leading to a focus on experiments and applications where data is abundant, such as simulations and games[7, 8].

A way of improving sample-efficiency is learning from demonstration (LfD), an umbrella-term for many techniques which try to jump-start learning by imitating an, often supposed to be expert, teacher. These methods rely on the idea that the teacher is able to more quickly guide the student towards regions of high reward in the space of trajectories, circumventing a substantial amount of trial-and-error. These techniques have been applied successfully to autonomously traverse rough terrain using a robotic vehicle[9], reproduce demonstrated trajectories using a robotic arm[10], and learn a controller for a multi-rotor UAV[11].

A family of methods named guided policy search (GPS), originally devised by Levine and Koltun in [12], combines model-based RL in a low-dimensional state-space with supervised learning to efficiently train high-dimensional policies in continuous-state and action RL tasks. Because of this two-step approach, these methods are sample-efficient enough to train vision-based robotic-arm controllers using real-world interaction [13]. Because of its simplicity, ability to handle continuous state- and action-spaces, sample-efficiency, mirror descent GPS (MD-GPS) has been identified as a promising RL algorithm for a flight control application. Additionally, these methods are easily combined with expert demonstrations, by initializing the model-based trajectory optimization with them[12].

The main contribution of this paper is an investigation into the effect of utilizing expert demonstrations to improve the sample-efficiency of the MD-GPS algorithm. To this end, experiments will be conducted on a simulated quadcopter system, while expert demonstrations are generated using a nonlinear quaternion-based controller.

In the remainder of this paper, the MD-GPS algorithm will be explained in section II, followed by a presentation of the methods used in this paper in section III. In section IV, the results of the experiments are show and discussed in section IV. Finally, the paper is concluded in section V.

II. Foundations

The specific GPS algorithm which will be used during this research is MD-GPS, using iterative linear quadratic Gaussian (iLQG) with fitted linear models (iLQG-FLM) for trajectory optimization under unknown dynamics, while utilizing a nonlinear global policy, from [14]. However, a summary of the history of GPS methods in general is presented first, followed by a thorough explanation of the MD-GPS algorithm.

A. History

In [12], Levine and Koltun state that training policies directly can scale to high dimensional inputs and parameterization, but training requires an inordinate amount of samples and frequently falls into poor local optima. To remedy this problem they have created a family of off-policy RL methods under the name of GPS, which uses trajectory optimization to obtain samples that guide the policy search to regions of high reward. The trajectory optimization is a form of model-based (as opposed to random) exploration and can also be initialized using demonstrations. The importance sampled GPS (IS-GPS) algorithm can be simplified as having two steps, starting with the assumption of known dynamics, a differential dynamic programming (DDP) method called iLQG is used to provide a distribution over trajectories which optimize reward. These trajectories are generated by local time-varying linear Gaussian (TVLG) controllers. The second step is a supervised learning step which minimizes the Kullback–Leibler (KL)-divergence between the global policy and the local sub-policies. Importance sampling (IS) is applied to estimate the returns of the global policy using the trajectories generated by iLQG. Additionally, on-policy samples are added to the set of trajectories each iteration, which ensures higher IS weights.

In the same paper an extension of the algorithm is proposed which adapts the guiding trajectories to the global policy at each iteration by optimizing the log-probability of actions under said global policy. This method will be called

adaptive IS-GPS (AIS-GPS). It requires new trajectories to be generated at each iteration and is necessary when the global policy cannot replicate the initial guiding trajectories. Adapting guiding samples to the capabilities of the global policy also provides the ability of altering the information to which it has access. For instance the local policies might have access to full-state feedback, while the global one has access to either a subset or a different representation.

In [15], a new algorithm is proposed based on AIS-GPS. Instead of optimizing the expected reward, the optimization is posed as an inference problem. A binary random stochastic variable is introduced which indicates whether a state-action pair is optimal. A Bayesian network is defined which relates states, actions, and this optimality indicator. By setting the latter to 1 for all time steps, the maximum-likelihood value of the policy parameter vector can be determined using an objective function equal to the log-probability of the optimality indicator conditioned on the policy parameter vector. This objective function is decomposed into a variational lower bound and a KL divergence term, the first one being maximized w.r.t. the policy parameters and the latter minimized w.r.t. the guiding trajectories obtained by iLQG. The on-policy return is not needed in this algorithm, which removes the need for IS. According to the authors this leaves variational GPS (V-GPS) less prone to local optima than AIS-GPS, while also being less complex in its implementation.

In [16], the GPS algorithm is cast as a constrained optimization problem, called constrained GPS (C-GPS). The Lagrangian of the constrained problem is solved by using dual gradient descent (DGD), which amounts to alternating between two steps. First, the trajectories are optimized w.r.t. cost using iLQG. Secondly, the global policy is optimized to match the local policies. A difference with IS-GPS and AIS-GPS methods is that no IS is used, which leaves it less vulnerable to degeneracies. An additional difference lies in the second optimization step. Similar to V-GPS, it amounts to a minimization of the KL-divergence between the global- and sub-policies. However, where V-GPS optimizes the global policy by performing a moment (M-) projection of the sub-policies onto the set of distributions spanned by the global policy's parameterization, C-GPS performs an information (I-) projection. The former being moment-matching; leading to a policy which optimizes the exponent of the negative cost, while the latter is mode-seeking; leading an optimization of the expected cost. The results obtained by the I-projection are shown to be superior.

Previously mentions GPS methods have all assumed known dynamics, which are used during the trajectory optimization by iLQG. In [17] an on-line system-identification method is combined with C-GPS to handle unknown dynamics. This trajectory optimization approach will be referred to as iLQG-FLM. Each iteration, the dynamics are fitted to the samples of the previous off-policy trajectories. Dynamics are modeled to be TVLG and are fitted each iteration using linear regression. To reduce the amount of samples needed to obtain a good fit a global Gaussian mixture model (GMM) prior is used. Since the assumption of linear dynamics falls apart when straying too far away from the linearization point, the cost function used in the trajectory optimization step includes a term that maximizes the log-probability of actions under sub-policies from the previous iteration. This is equivalent to a trust-region approach using the KL-divergence between sub-policies from successive iterations.

In [13], an augmented Lagrangian method called Bregmann alternating direction method of multipliers (BADMM) is used for GPS. The Lagrangian is augmented with a Bregmann-divergence term, being the KL-divergence between the sub-policies and the global-policy. The general process can be summarized as alternating optimization of the primals (sub-policies and global policy), folled by incrementing the Lagragne multipliers. As it is an implementation of BADMM, it inherits all its convergence properties outlined in [18]. In contrast with C-GPS, BADMM-GPS (BADMM-GPS) is able to employ the usual iLQG update steps, which are much faster than the extra steps the former method has to undertake. The difference lies in the KL-divergence terms used in the cost functions of the trajectory optimization. BADMM-GPS always optimizes the divergence w.r.t. its first argument, which leads to a convex optimization problem which can be directly solved by iLQG.

An interesting aerospace application of BADMM-GPS is presented in [19]. Zhang et al. state that previous GPS methods are not suitable for safety-critical applications, such as controlling flying agents, when either the dynamics are unknown or the model is sufficiently bad. The problem is that trajectory optimization is performed offline in the case of known dynamics, or identified online in the case of unknown dynamics. Since the dynamics model can be inaccurate and online identification takes time to converge, both methods can potentially lead to unstable or unsafe behavior. As a solution to this problem, online model predictive control (MPC) is used to track trajectories generated offline, providing improved robustness to model errors.

In [14], previously mentioned GPS methods are shown to all be variants of approximate mirror descent and a new algorithm is proposed called MD-GPS. Improvements over older methods are a reduced number of hyperparameters and a simpler algorithm. Performance is compared to previous methods and is shown to be the same or better, while requiring substantially less tuning.

Aforementioned methods have all assumed a known reward function to optimize, even when trajectory optimization was paired with demonstration trajectories. However, as mentioned before, some tasks are too complex to design a cost

function for and inverse reinforcement learning (IRL) or inverse optimal control (IOC) can be used to infer the reward function. In [20], BADMM-GPS under unknown dynamics from [17] is combined with a nonlinear generalization of maximum entropy IOC to infer a cost function from demonstrations. The result is a method called guided cost learning, which simultaneously obtains a highly nonlinear, high dimensional cost function directly from human demonstrations, and a policy which optimizes said cost. The main contribution is the fact that there is no need for task-specific cost-features, the cost is obtained directly from demonstrations. This is an improvement over prior IOC and IRL methods which often use predefined features in a linear cost-model.

Every GPS method mentioned up to now has made use of iLQG to optimize local linear time-varying Gaussian controllers to act as sub-policies. However, there is no reason no other trajectory optimization methods can be used in case they are a better fit to the task. In [21], a model-free method based on path integral stochastic optimal control, called PI^2, is used in the MD-GPS method to learn only the feedforward parameters of linear Gaussian controllers (feedback and covariance parameters are initialized by human demonstrations and kept constant during training). It is shown that this method is better suited to discontinuous dynamics than iLQG. Also, it is shown that by using on-policy sampling a new task-instance can be used at each iteration, improving generalization performance. The authors recognize that sampling a freshly initialized artificial neural network (ANN) policy on a real physical system can be dangerous, which is why in the beginning of the training procedure off-policy sampling is used instead.

In [22], a major assumption of prior GPS algorithms is relaxed; the ability to reset the agent to a specified set of initial states each episode. The new algorithm builds on MD-GPS and extends the method to allow for stochastic initial states, which increases the ability to generalize to unseen parts of the state-space. The method is similar to the previously mentioned PI^2-based MD-GPS approach, since both use on-policy sampling to remove the need for consistent initial states across episodes. However, in this work iLQG is used and therefore an explicit dynamics model is needed. Dynamics are identified online using a linear Gaussian model assumption, using the method outlined in [17]. To reduce computational cost, on-policy trajectories are assigned to clusters; each cluster owns one fitted dynamics model and linearized version of the global policy, which are used in the MD-GPS algorithm. It is shown the new method performs better than prior GPS methods in learning speed and generalization to unseen task instances.

Most GPS algorithms use iLQG for model-based trajectory optimization, with the exception of the model-free PI^2-based approach presented in [21]. A hybrid method is proposed in [23], which combines iLQG-FLM and PI^2 to perform trajectory optimization under unknown dynamics; PI^2-iterative linear quadratic regulator (iLQR) (PILQR). The hybrid method uses the model-based iLQG* approach to optimize a modeled cost, while the difference between this approximation and the encountered sample-based cost is optimized by the model-free PI^2-based approach. The combination provides the speed of model-based methods and the flexibility of model-free methods in instances where the model assumptions are violated. The hybrid method is combined with MD-GPS to learn high-dimensional policies. Performance is compared to prior methods on several tasks which suffer from discontinuous dynamics which are hard to handle using iLQG-FLM alone. Results show the hybrid method reliably outperforms purely model-based methods in such tasks.

In prior GPS methods, generalization and robustness to unseen states is entirely obtained by relying on the generalization capacity of the global policy (usually an ANN) trained on a set of training trajectories. In [24], Ennen et al. state that prior GPS methods lack robustness when encountering states outside of their training set. To solve this issue, the authors propose a new method which combines MD-GPS with generative motor reflexess (GMRs), which provide stabilizing behavior even outside the set of training trajectories. In contrast with prior GPS methods, the global policy is not given by a ANN which generates actions directly from the state; actions are generated by GMRs with parameters determined by a ANN. First, a latent representation of the state-space is learned using a variational autoencoder (VAE), this provides robustness as it is able to generalize over multiple states. Secondly, a ANN is used to transform the latent space vector into parameters of the GMR. The GMR, which is actually a TVLG controller, outputs an action conditioned on its parameters and the uncompressed state. The complete global policy is therefore the combination of the VAE, ANN, and the GMR, and can be straightforwardly implemented into the MD-GPS method. Results show that the proposed method outperforms standard MD-GPS by standards of variance and robustness to unseen states and state disturbances.

*Although the name indicates otherwise, the actual method used is indeed iLQG and not iLQR. The difference being that the former explicitly assumes Gaussian noise, while the latter is derived by assuming deterministic actions. Both methods lead to the same equations to obtain the optimal sub-policy. In the relevant literature the terms are used almost interchangeably.

B. Algorithm

This section will explain the workings of the MD-GPS algorithm. The complete algorithm is shown in algorithm 1, and consists of 6 steps, which will be treated one by one. Additionally, a stylized graphical explanation of the algorithm is presented in fig. 1, which illustrates every step except for the last.

Algorithm 1 Mirror descent GPS (MD-GPS): unknown dynamics, nonlinear global policy[14].

0: **for** $k = 1$ to K **do**
1: Generate samples $\mathcal{D}_i = \{\boldsymbol{\tau}_{i,j}\}$ by running either $p_i(\boldsymbol{u}|\boldsymbol{x})$ or $\pi_{\theta,i}(\boldsymbol{u}|\boldsymbol{o})$
2: Fit linear-Gaussian dynamics $p_i(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ using samples in $\mathcal{D}_i$
3: Fit linearized global policy $\bar{\pi}_{\theta,i}(\boldsymbol{u}|\boldsymbol{o})$ using samples in $\mathcal{D}_i$
4: C-step: $p_i \leftarrow \arg\min_{p_i} \mathrm{E}_{p_i(\tau)}\left[\ell(\tau)\right]$ such that $D_{KL}(p_i(\tau) \,||\, \bar{\pi}_{\theta,i}(\tau)) \leq \epsilon$
5: S-step: $\pi_\theta \leftarrow \arg\min_\theta \sum_{t,i,j} D_{KL}(\pi_{\theta,i}(\boldsymbol{u}_t|\boldsymbol{x}_{t,i,j}) \,||\, p_i(\boldsymbol{u}_t|\boldsymbol{x}_{t,i,j}))$ via supervised learning
6: Adapt ϵ
7: **end for**

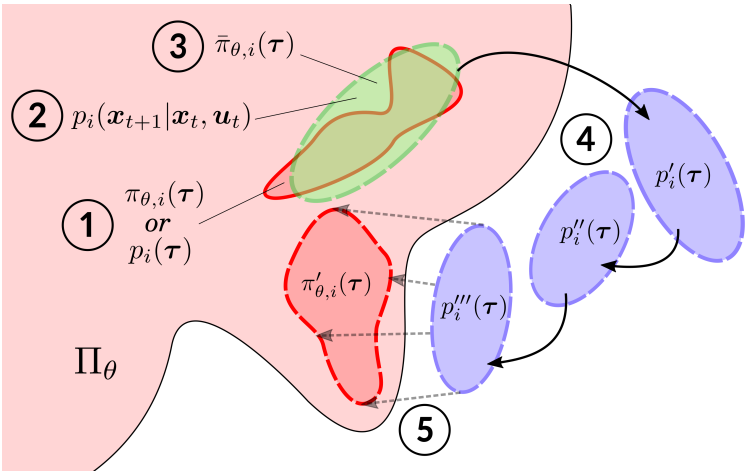


Fig. 1 Stylized representation of the MD-GPS algorithm, corresponding to algorithm 1. Trajectories sampled on the real system have a solid outline, Gaussian approximations are dashed and are ellipse-shaped. The light red area represents the constraint manifold of the global policy parameterization Π_Θ. Distances in this space can be roughly interpreted as a KL-divergence, as it is a measure of difference between two probability distributions.

Step 1: Sample trajectories

The first step is to sample trajectories on the real system, using either the sub-policies or global policy, of the previous iteration. By using multiple different, but consistent, initial states $\boldsymbol{x}_{0,i} \sim p(\boldsymbol{x}_0)$, generalization is improved[14]. The i^{th} initial state will be the starting point of a trajectory distribution from which multiple trajectories are sampled: $\boldsymbol{\tau}_i^j \quad \text{for} \quad j = [1, 2, 3, \cdots]$, stored in set $\mathcal{D}_i$. Depending on whether trajectories are sampled by sub-policies or the global policy, the distribution is $p_i(\tau)$ or $\pi_{\theta,i}(\tau)$, respectively. In fig. 1 the sampled trajectories presented as the oddly shaped dark-red area.

Step 2: Approximate dynamics

A TVLG model of the local dynamics around the nominal trajectory is estimated at each iteration. In fig. 1 this is denoted by the green dashed ellipse that roughly covers the sampled trajectories. Since the model is a linear approximation, the iLQG objective includes a term that penalizes distance from the nominal trajectory; the KL-divergence upper bound ϵ in eq. (2). Dynamics are approximated around each set of trajectories $\mathcal{D}_i$ separately, which have been sampled in the previous step. The dynamics models take the form of TVLGs:

$$p_i(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{f}_{xt}\boldsymbol{x}_t + \boldsymbol{f}_{ut}\boldsymbol{u}_t + \boldsymbol{f}_{ct}, \boldsymbol{F}_t) \tag{1}$$

Matrices $\boldsymbol{f}_{xt}$, $\boldsymbol{f}_{ut}$, $\boldsymbol{f}_{ct}$, and $\boldsymbol{F}_t$ will be estimated by fitting Gaussian distributions to the state transitions $(\boldsymbol{x}_t^{i,j}, \boldsymbol{u}_t^{i,j}, \boldsymbol{x}_{t+1}^{i,j})$ at every time step, using multiple samples of the trajectory distribution: $\boldsymbol{\tau}_i^j \in \mathcal{D}_i$, and then conditioning the distributions on $(\boldsymbol{x}_t^{i,j}, \boldsymbol{u}_t^{i,j})$ to obtain $p_i(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$.

This approach amounts to linear regression, which requires at least $N_x(N_x + N_u + 1) = 13 \cdot (13 + 4 + 1) = 234$ samples, which is prohibitively expensive. To combat this issue, a global model is fitted alongside the local dynamics, which will act as a prior and reduce the number of samples needed. A normal-inverse-Wishart distribution is used to combine the prior with new information.

Step 3: Linearize global policy

The objective optimized during the C-step, see line 4 of algorithm 1, includes a KL-divergence constraint of the sub-policies w.r.t. the global policy. This leads to a surrogate cost function for the iLQG trajectory optimization in the next step, as presented in eq. (10). Needed for the surrogate cost is the logarithm of the action distribution, induced by the global policy $\log \pi_{\theta,i}(\boldsymbol{u}_t|\boldsymbol{x}_t)$, which will be approximated by its second order Taylor expansion. In the case of highly nonlinear global policies, this approximation can be quite jagged during the beginning of the learning procedure[15], and prevent convergence[16]. To combat this, a TVLG approximation of the global policy, denoted $\bar{\pi}_{\theta,i}$, will be fitted for each sub-policy, in exactly the same way as for the dynamics as explained in section II.B. In fig. 1 this linearization is denoted by the green dashed ellipse, which roughly covers the sampled trajectories.

Step 4: C-step

At this point the trajectories have been sampled, the dynamics fitted, and the global policy approximated, the next step is the optimization of the TVLG sub-policies using the iLQG algorithm from [25]; the C-step. This iterative process is denoted in fig. 1 by the blue dashed ellipses. The objective of this optimization is shown on line 4 of algorithm 1 and will be repeated here for convenience:

$$p_i \leftarrow \arg\min_{p_i} \mathrm{E}_{p_i(\tau)}\big[\ell(\boldsymbol{\tau})\big] \quad \text{s.t.} \quad D_{KL}(p_i(\boldsymbol{\tau}) \,||\, \bar{\pi}_{\theta,i}(\boldsymbol{\tau})) \leq \epsilon \tag{2}$$

The objective is to minimize the expectation of some cost function over a distribution of finite trajectories induced by a sub-policy and corresponding initial state:

$$\min_{p_i} \mathrm{E}_{p_i(\tau)}\big[\ell(\boldsymbol{\tau})\big], \quad \text{where} \quad \ell(\boldsymbol{\tau}) = \sum_{t=0}^{T-1} \ell(\boldsymbol{x}_t, \boldsymbol{u}_t), \quad \text{and where} \quad \ell(\boldsymbol{x}_t, \boldsymbol{u}_t) = c(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{3}$$

In [26], iLQG is explained as follows: given a nominal trajectory:

$$\bar{\boldsymbol{\tau}} = \big(\{\bar{\boldsymbol{x}}_0, \bar{\boldsymbol{u}}_0\}, \{\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{u}}_1\}, \cdots, \{\bar{\boldsymbol{x}}_{T-1}, \bar{\boldsymbol{u}}_{T-1}\}\big) \tag{4}$$

define deviations as: $\hat{\boldsymbol{x}}_t = \boldsymbol{x}_t - \bar{\boldsymbol{x}}_t$, and $\hat{\boldsymbol{u}}_t = \boldsymbol{u}_t - \bar{\boldsymbol{u}}_t$. The deviation's dynamics and cost function are approximated as follows:

$$\tilde{\boldsymbol{x}}_{t+1} = \boldsymbol{f}_{ct} + \begin{bmatrix} \boldsymbol{f}_{xt} \\ \boldsymbol{f}_{ut} \end{bmatrix}^T \begin{bmatrix} \hat{\boldsymbol{x}}_t \\ \hat{\boldsymbol{u}}_t \end{bmatrix} \qquad \tilde{\ell}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) = \bar{\ell} + \begin{bmatrix} \boldsymbol{\ell}_{xt} \\ \boldsymbol{\ell}_{ut} \end{bmatrix}^T \begin{bmatrix} \hat{\boldsymbol{x}}_t \\ \hat{\boldsymbol{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \boldsymbol{\ell}_{xxt} & \boldsymbol{\ell}_{xut} \\ \boldsymbol{\ell}_{uxt} & \boldsymbol{\ell}_{uut} \end{bmatrix} \tag{5}$$

Where subscripts w.r.t. $\boldsymbol{x}$ and $\boldsymbol{u}$ denote Jacobians and Hessians, which can be obtained in multiple ways; finite differences, automatic differentiation, or analytical methods. It has already been explained how the dynamics are approximated in section II.B. Using the method explained there, the terms $\boldsymbol{f}_{ct}$, $\boldsymbol{f}_{xt}$, and $\boldsymbol{f}_{ut}$ are obtained. As for the cost approximation, it is obtained using finite differences.

The state- and action-value functions are approximated up to second order and are defined :

$$Q(\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}) = \bar{Q} + \begin{bmatrix} \boldsymbol{Q}_x \\ \boldsymbol{Q}_u \end{bmatrix}^T \begin{bmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{u}} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{u}} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{Q}_{xx} & \boldsymbol{Q}_{xu} \\ \boldsymbol{Q}_{ux} & \boldsymbol{Q}_{uu} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{u}} \end{bmatrix} \qquad V(\hat{\boldsymbol{x}}) = \bar{V} + \boldsymbol{V}_x^T \hat{\boldsymbol{x}} + \frac{1}{2} \hat{\boldsymbol{x}}^T \boldsymbol{V}_{xx} \hat{\boldsymbol{x}} \tag{6}$$

By starting at the final state and iterating backwards through time, each component of the state- and action-value functions is computed:

$$\begin{aligned} \boldsymbol{Q}_{xxt} &= \boldsymbol{\ell}_{xxt} + \boldsymbol{f}_{xt}^T \boldsymbol{V}_{xxt+1} \boldsymbol{f}_{xt} \\ \boldsymbol{Q}_{uut} &= \boldsymbol{\ell}_{uut} + \boldsymbol{f}_{ut}^T \boldsymbol{V}_{xxt+1} \boldsymbol{f}_{ut} \\ \boldsymbol{Q}_{uxt} &= \boldsymbol{\ell}_{uxt} + \boldsymbol{f}_{ut}^T \boldsymbol{V}_{xxt+1} \boldsymbol{f}_{xt} \end{aligned} \qquad \begin{aligned} \boldsymbol{Q}_{xt} &= \boldsymbol{\ell}_{xt} + \boldsymbol{f}_{xt}^T \boldsymbol{V}_{xt+1} \\ \boldsymbol{Q}_{ut} &= \boldsymbol{\ell}_{ut} + \boldsymbol{f}_{ut}^T \boldsymbol{V}_{xt+1} \end{aligned} \qquad \begin{aligned} \boldsymbol{V}_{xt} &= \boldsymbol{Q}_{xt} - \boldsymbol{Q}_{uxt}^T \boldsymbol{Q}_{uut}^{-1} \boldsymbol{Q}_u \\ \boldsymbol{V}_{xxt} &= \boldsymbol{Q}_{xxt} - \boldsymbol{Q}_{uxt}^T \boldsymbol{Q}_{uut}^{-1} \boldsymbol{Q}_{ux} \end{aligned} \tag{7}$$

When optimizing the objective shown in eq. (3), the locally optimal sub-policy can then be described by the following time-varying linear controller:

$$g(\boldsymbol{x}_t) = \bar{\boldsymbol{u}}_t + \boldsymbol{K}_t(\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t) + \boldsymbol{k}_t \qquad \boldsymbol{K}_t = -\boldsymbol{Q}_{uut}^{-1}\boldsymbol{Q}_{uxt} \qquad \boldsymbol{k}_t = -\boldsymbol{Q}_{uut}^{-1}\boldsymbol{Q}_{ut} \tag{8}$$

However, the C-step optimizes a slightly different objective from eq. (3); it includes a KL-divergence constraint as in eq. (2). It turns out this objective can be optimized using iLQG as well. To this end, the constrained problem is transformed to its Lagrangian:

$$\begin{aligned}\mathcal{L}(p_i, \eta) &= \mathrm{E}_{p_i(\tau)}\left[\ell(\boldsymbol{\tau})\right] + \eta(D_{KL}(p_i(\boldsymbol{\tau}) \mid\mid \bar{\pi}_{\theta i}(\boldsymbol{\tau})) - \epsilon) \\ &= \sum_{t=0}^{T-1} \mathrm{E}_{p_i(x_t,u_t)}\left[\ell(\boldsymbol{x}_t, \boldsymbol{u}_t) - \eta \log \bar{\pi}_{\theta i}(\boldsymbol{u}_t|\boldsymbol{x}_t)\right] - \eta\mathcal{H}(p(\boldsymbol{u}_t|\boldsymbol{x}_t)) - \eta\epsilon\end{aligned} \tag{9}$$

Where the second line follows from the assumption of conditional Gaussian policies, identical TVLG dynamics, and identical initial states, as in [14, 17]. This Lagrangian can be optimized by changing the cost function to the following:

$$\ell(\boldsymbol{x}_t, \boldsymbol{u}_t) = \frac{1}{\eta} c(\boldsymbol{x}_t, \boldsymbol{u}_t) - \log \bar{\pi}_{\theta i}(\boldsymbol{u}_t|\boldsymbol{x}_t) \tag{10}$$

The deterministic sub-policy in eq. (8) is adapted to be a TVLG. The optimal sub-policy becomes a TVLG controller, which stabilizes the state around the nominal trajectory:

$$\boldsymbol{C}_{t,i} = -\boldsymbol{Q}_{uut}^{-1} \qquad p_i(\boldsymbol{u}_t|\boldsymbol{x}_t) = \mathcal{N}(\bar{\boldsymbol{u}}_{t,i} + \boldsymbol{K}_{t,i}(\boldsymbol{x}_t - \bar{\boldsymbol{x}}_{t,i}) + \boldsymbol{k}_{t,i}, \boldsymbol{C}_{t,i}) \tag{11}$$

This way iLQG optimizes the following objective:

$$p_i = \arg\min_{p_i} \mathrm{E}_{p_i(x_t,u_t)}\left[\ell(\boldsymbol{x}_t, \boldsymbol{u}_t)\right] - \mathcal{H}(p_i(\boldsymbol{u}_t|\boldsymbol{x}_t)) \tag{12}$$

The Lagrangian is optimized using DGD, iteratively optimizing p_i using iLQG and incrementing the dual variable η. Since the dual function is convex and there is only one dual variable, a bracketed quadratic-fit line-search method can efficiently find the optimum[14]. To update η, the constraint violation is estimated, which requires the KL-divergence over trajectories in eq. (2). In the case of identical initial states and dynamics, as well as conditional Gaussian policies, the divergence reduces to an operation over the Gaussian policies directly[14, 17], and the KL-divergence is calculated efficiently.

As shown in eq. (11) and eq. (11), the optimal sub-policy depends on the inverse of Q_{uut}, which is the Hessian of the action-value function w.r.t. the input. Care must therefore be taken to ensure this matrix is invertible, which is done by increasing the dual variable η until Q_{uut} becomes positive definite for all t.

To conclude the explanation of the C-step, its pseudocode is presented in algorithm 2, explaining every step of this sub-routine.

Step 5: S-step

The final step of a MD-GPS iteration is the S- or supervised step, its goal is to optimize the global policy by minimizing the KL-divergence between the trajectory distributions induced by itself and the sub-policies. This step amounts to an approximate projection of the sub-policies onto the constraint manifold formed by the global-policy parameterization, and is denoted in fig. 1 by the dashed red blob.

By again assuming identical TVLG dynamics, conditional Gaussian policies, and identical initial states, the KL-divergence over trajectories reduces to a divergence over the action distributions of the policies[14, 17]:

$$\pi_\theta \leftarrow \arg\min_{\boldsymbol{\theta}} \sum_{t,i,j} D_{KL}(\pi_{\theta,i}(\boldsymbol{u}_t|\boldsymbol{x}_{t,i,j}) \mid\mid p_i(\boldsymbol{u}_t|\boldsymbol{x}_{t,i,j})) \tag{13}$$

This step amounts to an approximate projection of the sub-policies on the constraint manifold induced by the parameterization of the global policy. As explained previously in eq. (11), if the global-policy is chosen to be a (nonlinear) conditional Gaussian in the form of:

$$\pi_\theta(\boldsymbol{u}_t|\boldsymbol{o}_t) = \mathcal{N}(\mu^\pi(\boldsymbol{o}_t), \Sigma^\pi(\boldsymbol{o}_t)) \tag{14}$$

Algorithm 2 C-step pseudocode for the i^{th} sub-policy.

Require:

- Initial sub-policy $p_i(\boldsymbol{u}|\boldsymbol{x})$
- A set of trajectories $\mathcal{D}_i$ sampled by either $p_i(\boldsymbol{u}|\boldsymbol{x})$ or $\pi_{\theta,i}(\boldsymbol{u}|\boldsymbol{o})$
- A TVLG model of system dynamics, approximated around $\mathcal{D}_i$; $p_i(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$
- A TVLG model of the global policy, approximated around $\mathcal{D}_i$; $\bar{\pi}_{\theta,i}(\boldsymbol{u}|\boldsymbol{x})$
- A KL-divergence step size ϵ, obtained from eq. (17)

Initialize $\eta \leftarrow \eta_0$
while not converged **do**
  Set $\ell(\boldsymbol{x}, \boldsymbol{u})$ according to eq. (10)
  Approximate $\ell(\boldsymbol{x}, \boldsymbol{u})$ as in eq. (5)
  while not converged **do**
    iLQG backward pass to determine V- and Q-function components as in eq. (7), and local optimal sub-policy as in eq. (11)
    Use new sub-policy and the TVLG dynamics model to obtain new open-loop trajectory
  end while
  Update η using a bracketed quadratic-fit line-search
end while
return Sub-policy optimized as in eq. (2)

where μ^π and Σ^π are arbitrary functions (e.g. ANNs) that determine the mean and covariance of the Gaussian distribution, then, in combination with the TVLG sub-policies, there is an elegant equation for the KL-divergence in eq. (13), in terms of the means and covariances of the distributions:

$$\sum_{t,i,j} D_{KL}(\pi_\theta(\boldsymbol{u}_t|\boldsymbol{x}_t) \;||\; p_i(\boldsymbol{u}_t|\boldsymbol{x}_t)) = \cdots$$

$$\sum_{t,i,j} \mathrm{E}_{p_i(\boldsymbol{x}_t,\boldsymbol{o}_t)}\Big[\overbrace{\mathrm{Tr}[\boldsymbol{C}_{ti}^{-1}\Sigma^\pi(\boldsymbol{o}_t)] - \log|\Sigma^\pi(\boldsymbol{o}_t)|}^{\text{covariance terms}} + \underbrace{\big(\mu^\pi(\boldsymbol{o}_t) - \mu^p_{ti}(\boldsymbol{x}_t)\big)^T \boldsymbol{C}_{ti}^{-1}\big(\mu^\pi(\boldsymbol{o}_t) - \mu^p_{ti}(\boldsymbol{x}_t)\big)}_{\text{mean terms}} \Big] \quad (15)$$

Where $\mu^p_{ti}(\boldsymbol{x}_t)$ and $\boldsymbol{C}_{ti}$ are the mean and covariance matrix of the i^{th} sub-policy, at time step t (see eq. (11)).

Step 6: Adapt ϵ

The parameter ϵ in eq. (2) places a constraint on the maximum difference between the trajectory distributions of the sub-policy and the previous global policy (linearized around sampled trajectories).

To ensure fast convergence, the constraint is adapted online using one of two update rules, both based on differences in expected- and measured cost:

$$\ell^k_m = \sum_{t=0}^{T-1} \mathrm{E}_{p^k(x_t,u_t)}\big[\ell(\boldsymbol{x}_t, \boldsymbol{u}_t)\big] \qquad \ell^{k,\pi}_m = \sum_{t=0}^{T-1} \mathrm{E}_{\bar{\pi}^k_\theta(x_t,u_t)}\big[\ell(\boldsymbol{x}_t, \boldsymbol{u}_t)\big] \quad (16)$$

Where $p^k(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and $\bar{\pi}^k_\theta(\boldsymbol{x}_t, \boldsymbol{u}_t)$ are the marginals of the local policy and linearized global policy at iteration k, respectively, when subject to dynamics fitted at iteration m. Since the sub-policy, global policy, and dynamics model are all linear Gaussians, the expectations in eq. (16) can be efficiently calculated by propagating the Gaussian distributions forward in time[14].

The adaptation heuristics takes into account both the inaccuracies in the dynamics model and the inability of the global-policy to reproduce sub-policy trajectories:

$$\epsilon' = \frac{\epsilon}{2} \frac{\ell^k_{k-1} - \ell^{k-1,\pi}_{k-1}}{\ell^{k-1}_k - \ell^{k,\pi}_k} \quad (17)$$

III. Methods

This chapter describes the methods used during the experiments that have been conducted. First, a general outline of the conducted experiments is given in section III.A, followed by an explanation of the training structure used in this work (section III.B). In section III.C, the methods which are used in demonstration-based sub-policy initialization are explained, followed by a short section on the methods used to generate example trajectories in section III.D. In section III.E, the global-policy parameterization that is used in this work is presented, followed by a short section on the quadcopter model and controller in section III.F.

A. General experiment outline

In this work, the effect of expert demonstrations on the sample-efficiency of MD-GPS is investigated. To this end, experiments are conducted on a simulated quadcopter system. To reduce the effect of randomness, each experiment is run a total of 5 times, with different but consistent seeds. The experiments use 4 initial-conditions, which depend only on the random seed. Each run has 50 training iterations, 80 time steps, and a sampling time of 3×10^{-2} s. The cost function that is used is the mean of the square of the difference between actual- and target-position. A more thorough list of the default parameter settings used in this work is presented in section III.G.

A list of the different experiments is presented below:

- **E1: High-quality demonstrations**: This experiment initializes the sub-policies using high-quality demonstrations.
- **E2: Random initialization**: This experiment initializes the sub-policies randomly.
- **E3:** 0.25 m **target-offset**: This experiment initializes the sub-policies using the high-quality demonstrations, but the target during demonstrations has been off-set by 0.25 m.
- **E4:** 1.0 m **target-offset**: This experiment initializes the sub-policies using the high-quality demonstrations, but the target during demonstrations has been off-set by 1.0 m.
- **E5: Squiggly with** 7.5 **input-noise variance**: To generate the demonstrations with which the sub-policies are initialized, input-noise is added to the actions chosen by the nonlinear quaternion-based controller. This results in demonstrations that stochastically move around the nominal trajectory.
- **E6: Squiggly with** 15.0 **input-noise variance**: This experiment is the same as the above, but with an input-noise variance of 15.0.
- **E7: No global-policy pre-training - default**: This experiment uses the default settings, but without global-policy pre-training.
- **E8: With global-policy pre-training - default**: This experiment is equal to E1
- **E9: No global-policy pre-training - delayed**: This experiment uses no global-policy pre-training. Furthermore, the first 9 iterations sub-policies are constrained w.r.t. the previous sub-policies (up from 4) and the global-policy is used to sample from iteration 20 onward (up from 10).
- **E10: With global-policy pre-training - delayed**: Same as the above, but with global-policy pre-training.
- **E11: No global-policy pre-training - vanilla MD-GPS**: This experiment has no global-policy pre-training. Additionally, it does not have any iterations at which the optimization of the sub-policies is constrained w.r.t. the previous ones and the global-policy is used to sample from iteration 1 onward.
- **E12: With global-policy pre-training - vanilla MD-GPS**: Same as the above, but with global-policy pre-training.

B. Training structure

In MD-GPS, the global-policy is both used to sample trajectories, and constrain the trajectory optimization. However, due to the unstable nature of the quadcopter system, no stability guarantees provided by neural networks, and the fact that iLQG-FLM does not provide any way to robustly restore itself back to a previous solution, a global-policy with poor performance can completely derail the training process. In the case of sufficiently complex or unstable system dynamics, the algorithm might get stuck in a local-optimum, this has also been observed in [12] and Levine and Koltun advise to initialize sub-policies using demonstrations. In general, a poor global-policy can affect training either by being used to sample trajectories, or by being involved in the KL-divergence constraint. Therefore, it is vital that the global-policy generates stable trajectories before it is involved in those ways.

To this end, the training process can be structured in a certain way. Where structure refers to the choices made concerning the exact training iterations during which the following actions are undertaken:

- Constrain trajectory optimization w.r.t. previous sub-policies or global-policy?
- Sample trajectories using sub-policies or global-policy?
- At what iteration to start supervised learning of the global-policy?

An additional choice is whether to pre-train the global-policy using demonstrations, or not.

It is expected that the best way of structuring training is to begin by both sampling using- and constraining w.r.t. sub-policies, moving to constraining sub-policies w.r.t. the global-policy when it has been trained for a number of iterations, while finally moving to using the global-policy for sampling. Supervised learning of the global-policy should take place only when the sub-policies already show good behavior. By using demonstrations to initialize the sub-policies, supervised training could start at the first iteration. Also, using demonstrations to pre-train the global-policy would be expected to reduce the training iterations needed to get the global-policy to generate stable trajectories and be useful to sample from.

The default training structure is to pre-train the global-plicy, start supervised-learning of the global-policy at the first training iteration, constrain the trajectory optimization w.r.t. the previous sub-policies until the 5^{th} iteration, and start using the global-policy to sample trajectories on the 10^{th} training iteration. This structure is used in experiments `E1`, `E2`, `E3`, `E4`, `E5`, `E6`, `E7` (without global-policy pre-training), and `E8`. Experiments `E9` and `E10` keep constraining the trajectory optimization w.r.t. the previous sub-policies until the 10^{th} iteration, and delay the use of the global-policy to sample trajectories until iteration 20. Finally, experiments `E11` and `E12` use the global-policy to sample trajectories and constrain the trajectory optimization for the entire training process.

C. Demonstration-based sub-policies

This section explains how demonstration policies can be transformed into TVLG sub-policies, to be used as the starting point of the iLQG-FLM trajectory optimization. In this work, the focus will be on demonstration trajectories generated by a nonlinear quaternion-based controller inspired by [27].

Irrespective of the initial form of the demonstrations (trajectories, a controller, etc), to be used in the iLQG-FLM algorithm, they need to be in the form of a TVLG sub-policy, which can be parameterized as follows:

$$p_i(\boldsymbol{u}_t|\boldsymbol{x}_t) = \mathcal{N}(\bar{\boldsymbol{u}}_{t,i} + \boldsymbol{K}_{t,i}(\boldsymbol{x}_t - \bar{\boldsymbol{x}}_{t,i}), \boldsymbol{C}_{t,i}) \tag{18}$$

$$= \mathcal{N}(\boldsymbol{K}_{t,i}\boldsymbol{x}_t + \hat{\boldsymbol{k}}_{t,i}, \boldsymbol{C}_{t,i}) \tag{19}$$

Where i denotes the initial-condition and t the time step.

To create a TVLG sub-policy from the demonstration trajectories, parameters that determine the mean need to be estimated. Since the mean is a linear function of the state, this means the demonstration controller must be linearized. However, the linearization must take place along the trajectory that results when an agent follows the demonstration policy while interacting with the quadcopter simulation. This trajectory is obtained by simply initializing the agent at the initial-conditions and using the demonstration policy as a controller in a feedback-loop with the quadcopter simulation. The result is a trajectory with $(\bar{\boldsymbol{u}}_{t,i}, \bar{\boldsymbol{x}}_{t,i})$ tuples, such as in eq. (18).

The gain matrix in eq. (18) is simply the jacobian of the controller w.r.t. the state, which can be obtained in multiple ways: analytical-, numerical-, or automatic differentiation. However, in this work the choice was made to go with a fourth method: linear regression. To this end, multiple demonstration trajectories are needed that are close to the nominal trajectory. These are generated by injecting input noise in the feedback system, while storing the unmodified outputs of the controller alongside the encountered states. The additive input noise results in stochastic state-action trajectories that stay within a certain distance of the nominal trajectory, depending on the variance of the noise.

At every time step a total of 56 ($(N_x + 1) \cdot N_u = (13 + 1) \cdot 4 = 56$) parameters need to be estimated. Using simple linear regression, this would require a bare minimum of 56 trajectory samples per initial-condition. However, using Tikhonov regularization with a λ value of 1×10^{-6}, this number can be greatly reduced. The choice has been made to use a quite large number of 40 demonstration trajectories per initial-condition, since this improves the estimates and generating trajectories in simulation is quite fast. It is expected that this number could be reduced in a real-world application, with little loss in estimation quality.

Although the parameters that define the means of the TVLG sub-policies are estimated, the covariance matrices are initialized manually and are based on prior experimentation. Higher variance promotes exploration and can improve convergence rates, but it can also lead the agent outside of the stable region of the sub-policy. So care is taken to balance exploration with stability. Prior experiments indicate deviations of initial sub-policy trajectories w.r.t. their nominal trajectory increase with time. To counter this effect and equalize exploration over the initial sub-policy trajectory, the initial variance is scaled at each time step according to the following equation:

$$\begin{aligned} \hat{\sigma}_t^2 &= \frac{\sigma^2}{e^{(K-k)\,\delta}} \quad k = 1, 2, \cdots, K \\ \delta &= d_\sigma/(K-1) \end{aligned} \tag{20}$$

During the experiments there are three types of demonstration-based sub-policies, these are defined as follows:

- **High-quality**: This is the sub-policy that results from the nonlinear quaternion-based controller and is used as the default option. It is used in experiment `E1`, `E7`, `E8`, `E9`, `E10`, `E11`, and `E12`.
- **Random**: These sub-policies are randomly initialized and are used in experiment `E2`.
- **Target-offset**: This is equal to the high-quality demonstration, but the target provided to the controller is offset w.r.t. the real target with either 2.5×10^{-1} m or 1.0 m along each axis. This is used in experiment `E3` and `E4`.
- **Squiggly**: This is equal to the high-quality demonstration, but input noise with variance 7.5 or 15.0 is added in the feedback system to generate demonstration trajectories that stochastically move around the nominal trajectory of the high-quality demonstration. This is used in experiment `E5` and `E6`.

D. Generating pre-training examples

This section explains how pre-training examples are generated for both the dynamics- and global-policy pre-training, starting with the former.

To effectively pre-train the dynamics priors and TVLG models, a dataset of varied state-action-state tuples is needed, that are located around the nominal trajectory at the first iteration. To obtain this dataset the same process is used as described in section III.C: a generating policy can be used as a controller in a feedback-loop with the quadcopter simulation. To provide samples that are randomly spread around the nominal trajectory, input noise is added to the system.

To obtain a global-policy pre-training dataset, the same tactic is used as for the dynamics pre-training data.

To pre-train the global-policy most effectively, a dataset is needed with state-action tuples around the nominal trajectory that would be generated by the global-policy itself, which essentially amounts to solving the optimal control problem and cannot be done off-line. Instead, global-policy pre-training state-action tuples are generated by querying the policy-to-be-imitated (the imitation-policy) at states around the nominal trajectory generated by some generating policy plus input noise in the same way as explained above for the dynamics pre-training examples. The supervised learning method used to pre-train the global-policy is the same as the method used during normal training.

E. Global-policy parameterization

In this work, the global-policy is parameterized by a feedforward ANN. This choice is based on the relevant literature, and the fact that ANNs are expressive function approximators. Since the observation consists of the full state of the quadcopter, a simple, fully-connected, architecture is chosen, sporting an input layer of 13 neurons (one for each state variable), two hidden layers of 200 and 100 neurons, respectively. The output layer naturally consists of 4 neurons, equal to the length of the input vector. The activation functions are chosen to be exponential linear units (ELUs).

F. Quadcopter model and controller

The experiments in this work are conducted in simulation, using a quadcopter model with nonlinear dynamics, using a quaternion to parameterize the attitude to prevent gimbal-lock. It is assumed the quadcopter can generate any combination of moments and total thrust[27]. Therefore, propeller dynamics are ignored and the agent can directly control the force and moments in the body frame. This also allows the simulation time step to be increased significantly to 3×10^{-2} s, as the quadcopter body dynamics are much slower than its propellers. This leads to the following input vector:

$$\boldsymbol{u} = \begin{bmatrix} F & M_{b,x} & M_{b,y} & M_{b,z} \end{bmatrix}^T \tag{21}$$

The force and moments are clamped using the following maximum absolute values:

$$F_{max} = 8.36 \times 10^1 \text{ N} \qquad M_{b,x,max} = 1.88 \times 10^1 \text{ N·m} \qquad M_{b,y,max} = 1.88 \times 10^1 \text{ N·m} \qquad M_{b,z,max} = 3.20 \text{ N·m} \tag{22}$$

The mass and inertia parameters of the quadcopter model are presented in section III.F. Note that the the mass-moment of inertia tensor has zero-valued off-diagonal entries.

The controller used to generate expert demonstrations is inspired by [27]; quaternion-based tilt-prioritized control.

G. Default parameters

In table 4 the default settings of the MD-GPS algorithm used in this work are described, which are used as a nominal point during the experiments.

Parameter	Description	Value
m	mass	4.68×10^{-1} kg
I_{xx}	principal mass-moment of inertia around body x-axis	4.856×10^{-3} kg·m^2
I_{yy}	principal mass-moment of inertia around body y-axis	4.856×10^{-3} kg·m^2
I_{zz}	principal mass-moment of inertia around body z-axis	8.801×10^{-3} kg·m^2

Table 2 Quadcopter model parameters, which have been copied from [28].

IV. Results and discussion

To investigate the sample efficiency, the cost-iteration curves are plotted in fig. 2, for both the demonstration-based- and random sub-policy initialization. Sample efficiency can be defined as the amount of real-world samples are needed to achieve some level of cost. As the amount of real world samples are equal for both curves, they can be compared directly. From the plot it can be seen that the median cost-iteration curve of the random initialization lies almost two order of magnitude above the demonstration-based curve. The demonstration-based cost improves most in the first 10 iterations, after which improvements gradually taper off and seem to converge around the 20^{th} iteration. The random initialization cost shows a steep decrease in the first five iterations, but then bounces back up and starts to decrease very slowly over the next 45 iterations. However, the slope is so small that it would take an unpractical amount of samples for the random-initialization to obtain the cost-level of the demonstration-based method.

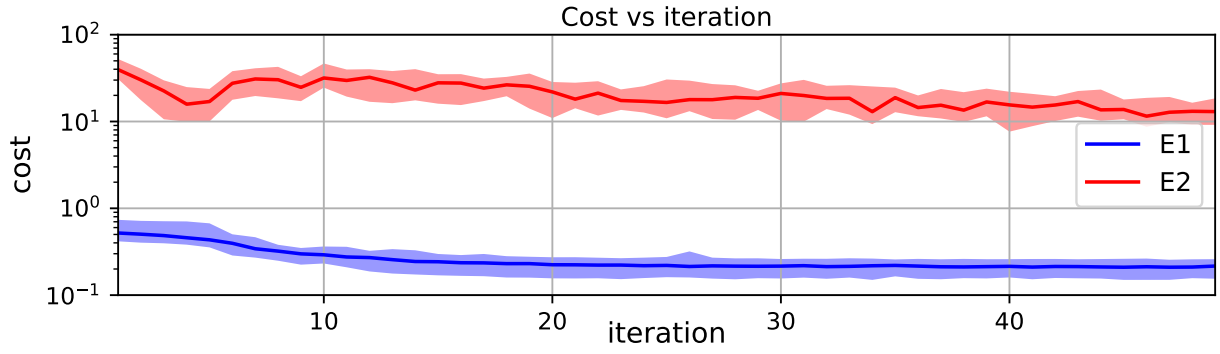


Fig. 2 A plot of the cost versus training iteration curves, for demonstration-based- (E1) and random initialization E2). The plots show a thick line, which signifies the median, and a shaded area which signifies the 68.27% **confidence interval. Take note of the log-scale on the y-axis.**

To investigate the stability of the agents, the percentage of trajectories that were unstable have been recorded and plotted in the bar-plot presented in fig. 3. The demonstration-based initialization has lead to zero unstable trajectories during training, while random initialization starts out on the first iteration with almost 100 % of its trajectories being unstable, which is reduced steadily during training to a value of 75 %.

Parameter	Value
Number of seeds	5
Number of training iterations	50
Number of initial-conditions	4
Number of trajectory samples	5
Number of time steps	80
Time step	3×10^{-2} s
Iteration at which to start supervised learning	1
Iterations at which to constrain trajectory-optimization w.r.t. sub-policies	1-4
Iterations at which to sample using sub-policies	1 - 9
Target-position	$\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$
Initial sub-policy variance	1×10^{-2}
Initial sub-policy variance discount	1.0
Initial KL-divergence step size	1×10^{-1}
KL-divergence adaptation factor	1.33
Perform dynamics pre-training?	`TRUE`
Pre-training example gathering policy	`High-quality demonstration`
Number of dynamics pre-training samples	30
Dynamics prior GMM maximum clusters	1000
Dynamics prior GMM minimum samples per cluster	20
Dynamics prior GMM maximum trajectory storage	10
Perform global-policy pre-training?	`TRUE`
Global-policy pre-training example gathering policy	`High-quality demonstration`
Global-policy pre-training imitation policy	`High-quality demonstration`
Global-policy pre-training samples	30
Global-policy prior GMM maximum clusters	1000
Global-policy prior GMM minimum samples per cluster	20
Global-policy prior GMM maximum trajectory storage	10

Table 4 Default parameters of the MD-GPS algorithm used in this work.

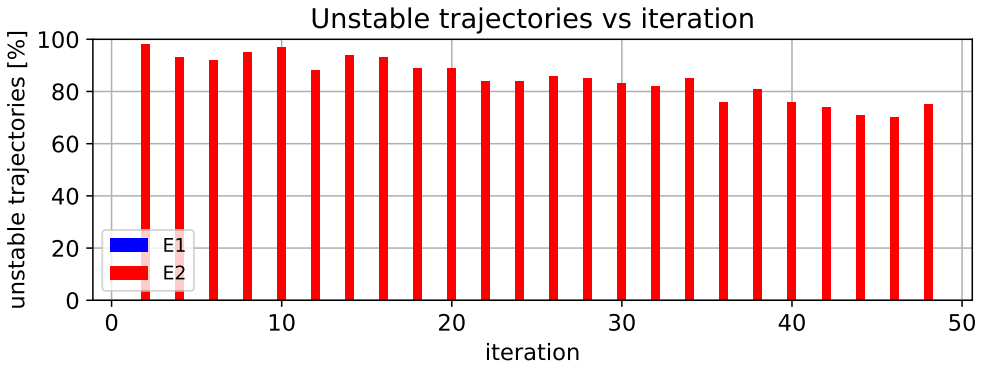


Fig. 3 A bar-plot showing the percentage of sampled trajectories that is "unstable", comparing demonstration-based- (`E1`) and random initialization (`E2`). A trajectory is unstable if at any time step the distance to the target is larger than 6 m**.**

To investigate the difference in the types of trajectories during both experiments, the target-distance is plotted versus time in fig. 4. This gives an indication of the difference in behavior of the agents during the training process. As can be seen, the demonstration-based initialization starts out at the first iteration with trajectories that are already quite performant; target-distance decreases quickly up to 1.5 s and then levels off. At the 15^{th} iteration the target-distance curve has slightly improved by reducing the time needed to reach a distance of 1 m and converging to a lower final value. At the 49^{th} iteration these aspects have been improved slightly more. In contrast, the random initialization leads to trajectories that almost immediately diverge from the target. During training the improvements seem to lie in the fact that the agent stays closer to its initial target-distance for longer, before diverging.

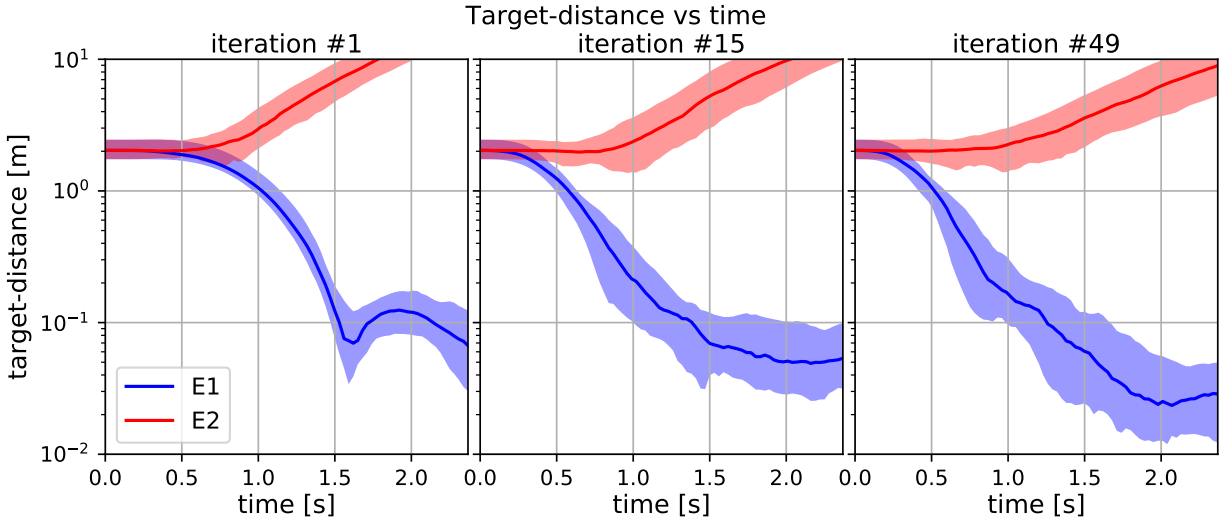


Fig. 4 Three plots of target-distance versus time plots at different iterations, that compare demonstration-based- (E1) with random initialization (E2). The plots show a thick line, which signifies the median, and a shaded area which signifies the 68.27% confidence interval. Take note of the log-scale on the y-axis.

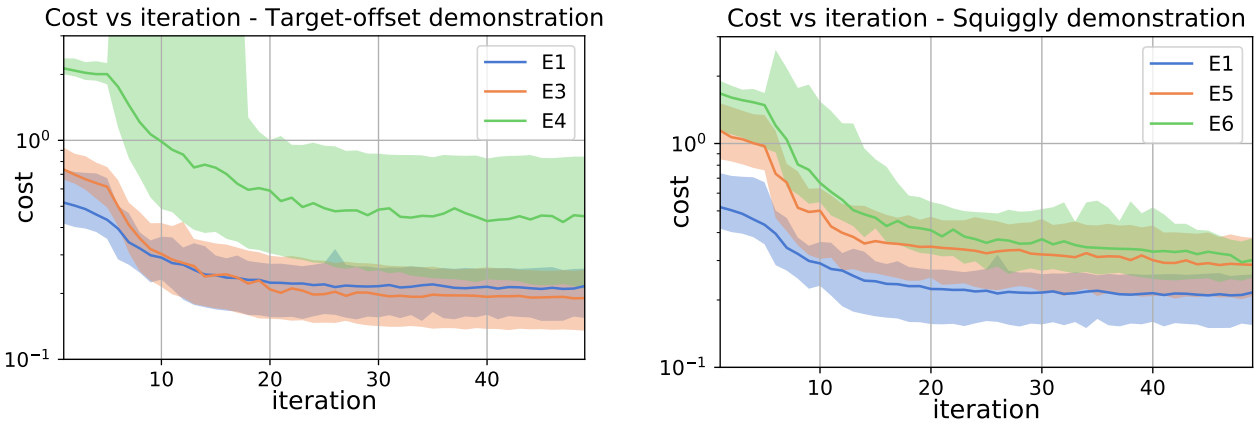
MD-GPS with demonstration-based initialization needs much fewer iterations to converge and obtains much lower cost than with random-initialization, even when given 50 iterations. Furthermore, the agents behave qualitatively different, demonstration-based initialization leading to stable-controller-like behavior, while random initialization leads to policies that diverge from the target and fail to converge.

To investigate the robustness of MD-GPS to sub-policy initialization with sub-optimal demonstrations, the cost-iteration curves for two different kinds and degrees of demonstration sub-optimality; target-offset and squiggly (explained in section III.C), are presented in fig. 5.

The curves that are the result of initializing the sub-policies using high-quality demonstrations will be used as a control during this analysis. As can be seen in fig. 5a, when initialized with a 0.25 m target offset the median cost-iteration curve starts out at a significantly higher cost than the control, but it quickly decreases and even overtakes it at iteration 15, converging to a slightly lower value. Cost variance is comparable to the control over the entire training process. When the target-offset is 1.00 m median cost improves quickly as well, but does not converge to the control value. Additionally, cost-variance is much higher from iteration 5 onward, peaking between iteration 5 and 17 to an upper bound that exceeds the boundaries of the plot.

The cost-iteration curves resulting from sub-policy initialization with squiggly demonstrations are shown in fig. 5b, using the high-quality demonstration curve as a control. As expected, initializing with squiggly demonstrations leads to higher initial cost, but in both cases decreases quickly. However, in neither case the cost converges to the control value within 50 iterations. Interestingly, the cost at the final iteration is almost equal for both values of input-noise variance. As for the cost-variance, the 15.0 input-noise variance curve shows a spike at the 5^{th} iteration, with seemingly increased variance up to the 17^{th} iteration.

To investigate the differences in the trajectories resulting from different sub-policy initializations, the target-distance versus time curves are plotted. As can be seen in fig. 6, the 0.25 m and 1.00 m target-offset demonstrations lead to a target-distance of 0.43 m and 1.73 m at the first iteration, respectively. This is to be expected since this is equal to



(a) Cost-iteration curves for high-quality- (E1), 0.25 m **target-offset- (E3), and** 1.00 m **target-offset demonstrations (E4).**

(b) Cost-iteration curves for high-quality- (E1), and squiggly demonstrations with an input-noise variance of both 7.5 **(E5) and** 15.0 **(E6).**

Fig. 5 The plots show a thick line and shaded area, which signify the median and 68.27% **confidence interval, respectively. Take note of the log-scale on the y-axes.**

$\|\boldsymbol{d}\|$, where $\boldsymbol{d}$ is the target-offset vector. After 15 iterations, the target-distance curve when using 0.25 m target-offset demonstrations has almost completely matched the high-quality demonstrations, a trend that continues into the 49^{th} iteration. The target-distance curve when using 1.00 m target-offset demonstrations improves much less, and converges to an offset of 0.3 m at iteration 49.

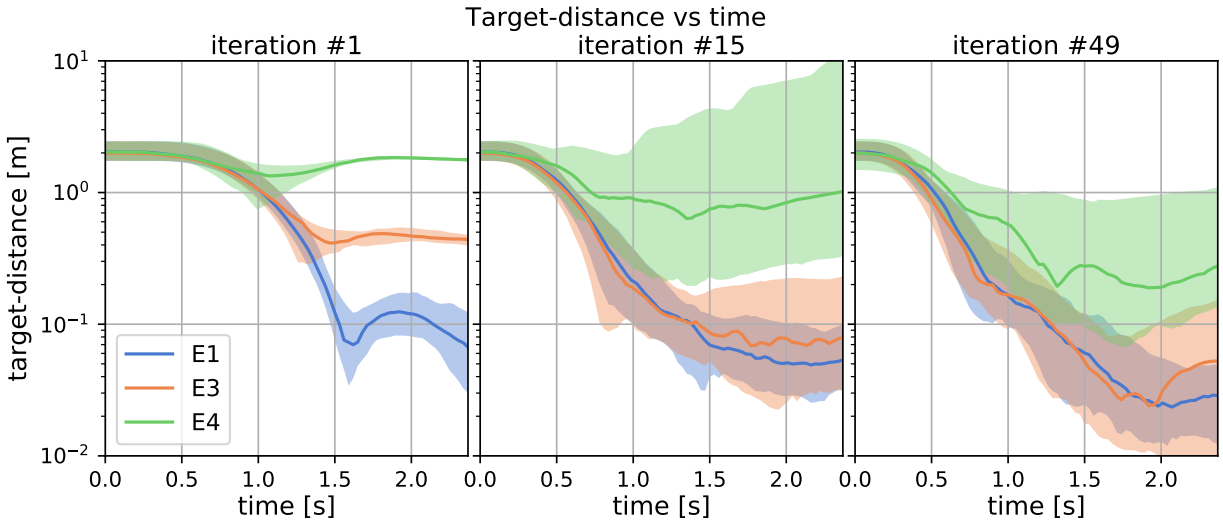


Fig. 6 Three plots of target-distance versus time plots for high-quality- (E1), 0.25 m **target-offset- (E3), and** 1.00 m **target-offset demonstrations (E4). The plots show a thick line, which signifies the median, and a shaded area which signifies the** 68.27% **confidence interval. Take note of the log-scale on the y-axis.**

The target-distance curves for the squiggly demonstrations can be seen in fig. 7. For both an input-noise variance of 7.5 and 15.0, the target-distance curves improve during the training process. However, the performance of neither matches the high-quality demonstrations, even at iteration 49. Interestingly, the target-distance curves at iteration 49 are of very similar performance.

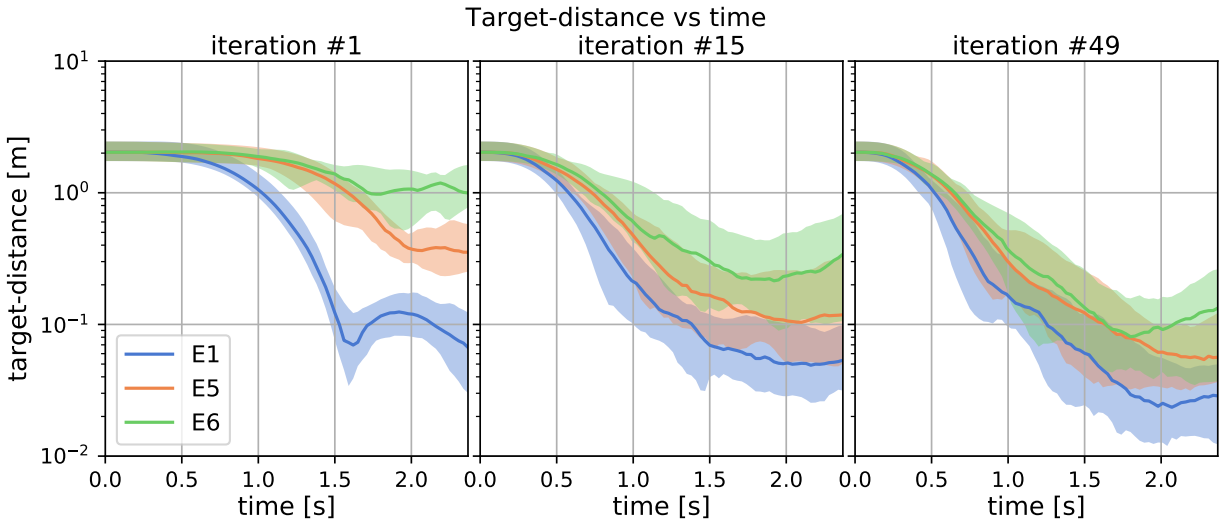


Fig. 7 Target-distance versus time plots for high-quality- (E1), and squiggly demonstrations with an input-noise variance of both 7.5 (E5) and 15.0 (E6). The plots show a thick line, which signifies the median, and a shaded area which signifies the 68.27% confidence interval. Take note of the log-scale on the y-axis.

As noted before, the variance of the cost spikes for both the 1.00 m target-offset and 15.0 input-noise variance demonstrations. The reason for this is likely an increase in the number of unstable trajectories, which can be observed in fig. 8. For these demonstrations with a higher degree of sub-optimality the number of unstable trajectories suddenly increases around iteration 6. This is probably linked to the fact that at iteration 5 the switch is made from constraining the trajectory optimization w.r.t. the previous sub-policy, to constraining w.r.t. the global-policy.

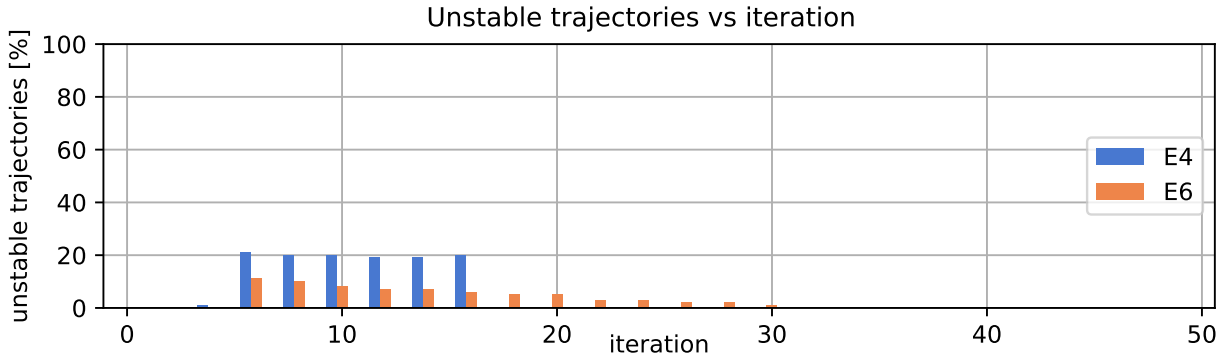


Fig. 8 A plot of the percentage of "unstable" trajectories, for both 1.00 m target-offset demonstrations (E4) and squiggly demonstrations with input-noise variance of 15.0 (E6). A trajectory is unstable if at any time step the distance to the target is larger than 6 m. For high-quality- (E1), 0.25 m target-offset- (E3), and squiggly demonstrations with input-noise variance of 7.5 (E5), no trajectories were unstable.

From this analysis, it can be concluded that MD-GPS is moderately robust to being initialized using sub-optimal demonstrations. The degree of sub-optimality seems dictates whether the solutions converge to the same cost, analogous to the region-of-attraction in control theory. This can be observed in fig. 5, where the least sub-optimal demonstrations lead to similar performance as high-quality demonstrations, while the 1.00 m target-offset does not. As for the squiggly demonstrations, both degrees of sub-optimality converge to similar cost values, which are slightly above the cost for the high-quality demonstrations.

To investigate the effect of different training structures on the sample-efficiency, the cost-iteration curves are plotted in fig. 9. When using default settings, pre-training the global-policy has the effect of very slightly reducing cost between

the 5^{th} and 10^{th} iteration. However, after iteration 10 the two curves are practically identical. Regarding the delayed settings, the pre-training has almost no discernible effect on the cost, except for the significant increase in cost-variance when omitting pre-training. Finally, while pre-training leads to a rapid decrease in cost at the start of training when using vanilla MD-GPS settings, the improvements stop around iteration 5 and cost starts to increase. In contrast, the cost-iteration curve when performing no pre-training converges to a much lower cost, which is higher but comparable to the level to which the default settings converge. However, both with- and without pre-training vanilla MD-GPS shows increased cost-variance as compared to the default settings.

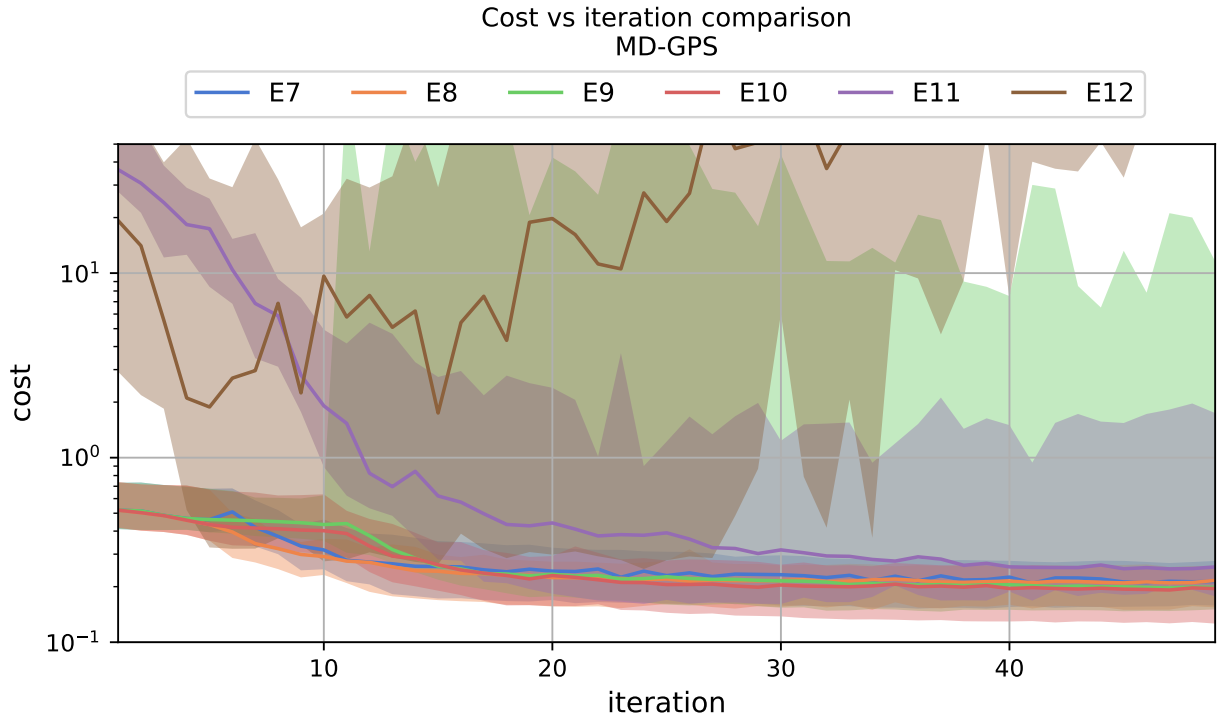


Fig. 9 Cost-iteration curves, for experiment `E7` (no global-policy pre-training, default), `E8` (with global-policy pre-training, default), `E9` (no global-policy pre-training, delayed), `E10` (with global-policy pre-training, delayed), `E11` (with global-policy pre-training, vanilla), and `E12` (with global-policy pre-training, vanilla). The plot shows thick lines, which signify the median, and shaded areas which signify the 68.27% confidence intervals. Take note of the log-scale on the y-axis.

In fig. 10, target-distance versus time plots are shown for different training structures. As can be seen, median target-distance curves of the default- and the delayed settings are comparable, both with- and without pre-training. However, when omitting pre-training with the delayed settings the target-distance variance is significantly increased. Regarding the vanilla MD-GPS settings, when omitting pre-training the target-distance curve diverges almost immediately, while the use of pre-training causes the target-distance to decrease sharply before diverging as well. At the 15^{th} iteration, the target-distance curves both with and without pre-training have improved, decreasing until around 1.0 s and then converging. However, the final target-distance is still much larger than for the default- and delayed-settings. At the 49^{th} iteration, the vanilla MD-GPS without pre-training has almost converged to the same level as the default- and delayed-settings, to a cost of 3.58×10^{-1}. With pre-training and using the default structure, this cost is obtained significantly earlier; at the eighth iteration. However, when utilizing global-policy pre-training with the vanilla MD-GPS settings, the target-distance curve almost immediately diverges.

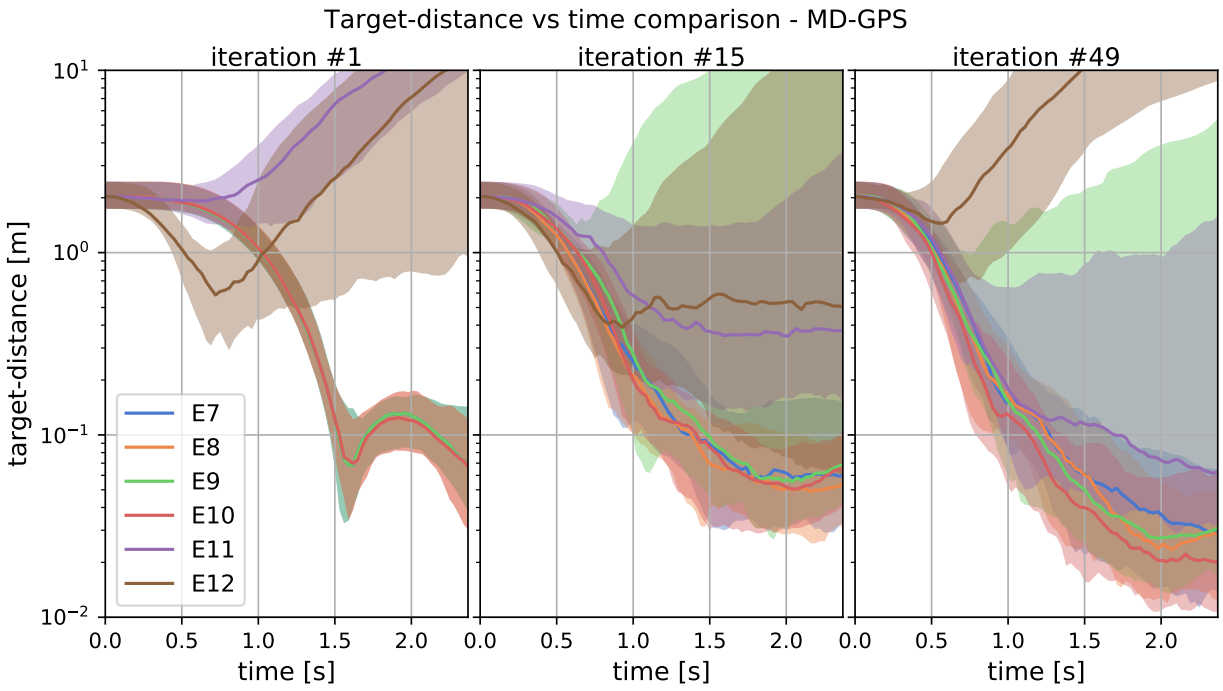


Fig. 10 Three plots of target-distance versus time plots at different iterations, for experiment `E7` (no global-policy pre-training, default), `E8` (with global-policy pre-training, default), `E9` (no global-policy pre-training, delayed), `E10` (with global-policy pre-training, delayed), `E11` (with global-policy pre-training, vanilla), and `E12` (with global-policy pre-training, vanilla). The plots show a thick line, which signifies the median, and a shaded area which signifies the 68.27% confidence interval. Take note of the log-scale on the y-axis.

In fig. 11, the percentage of unstable trajectories is shown for the experiments with differing training structures. As can be seen, the default-settings do not lead to any unstable trajectories, this is also true when utilizing pre-training with the delayed-settings. Without pre-training, the delayed-settings lead to a percentage of unstable trajectories around 18%, starting at the tenth iteration. This is probably linked to the fact that at the tenth iteration the trajectory-optimization constraint is switched from using the previous sub-policy, to the global-policy. Without pre-training the global-policy might still lead to unstable trajectories. When performing pre-training in combination with the vanilla MD-GPS settings, the percentage of unstable trajectories starts out around 50% and decreases up to the twelfth iteration, followed by a steady increase afterwards. Since vanilla MD-GPS does not use constraint switching, this cannot be explained as such. Interestingly, while the percentage of unstable trajectories when foregoing pre-training with the vanilla MD-GPS settings starts out around 90%, it steadily decreases over the next 15 iterations, converging to a value around 3%.

From this analysis, it can be concluded that pre-training of the global-policy can significantly improve the sample-efficiency of MD-GPS, provided that both constraining the trajectory optimization w.r.t. the global-policy and using the global-policy to sample trajectories is delayed as in `E8` and `E10`. This leads to significantly higher rates of convergence and lower rates of unstable trajectories. Of these experiments, `E10` converges to a slightly lower cost, but takes a bit longer to converge. Without these delays, pre-training the global-policy leads to instability issues from which the algorithm cannot recover. Compared to `E11`, which eventually converges to a low cost of 3.58×10^{-1}, `E8` and `E10` are significantly more sample-efficient, requiring only 8 and 13 iterations to achieve this level, respectively.

V. Conclusion

In this work, it is investigated how the sample-efficiency of MD-GPS can be further improved by making use of expert demonstrations, by conducting experiments on a simulated quadcopter.

It is shown that by initializing the trajectory-optimization with demonstration-based sub-policies, both the initial- and final cost are reduced significantly. As a matter of fact, with random sub-policy initialization the algorithm is not able to find a policy that converges to the target-position at all and never even reaches the initial cost-value corresponding to

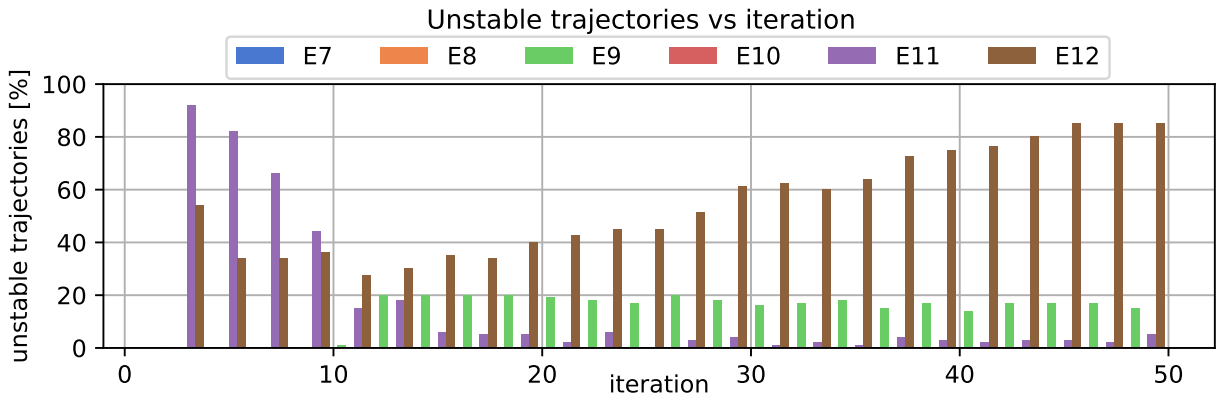


Fig. 11 A bar-plot showing the percentage of sampled trajectories that is "unstable", for experiment `E7` (no global-policy pre-training, default), `E8` (with global-policy pre-training, default), `E9` (no global-policy pre-training, delayed), `E10` (with global-policy pre-training, delayed), `E11` (with global-policy pre-training, vanilla), and `E12` (with global-policy pre-training, vanilla). A trajectory is unstable if at any time step the distance to the target is larger than 6 m**.**

demonstration-based sub-policy initialization. To find a controller that stabilizes the quadcopter system, it is necessary to initialize the sub-policies using demonstrations. Also, it is shown that, for a small target-offset in the demonstrations (0.25 m), the global-policy converges to a similar solution as when initialized by a high-quality demonstration. While, for the tested noisy (squiggly) demonstration initialization, the resulting solution has slightly worse cost.

Results show that without any alterations and while omitting global-policy pre-training, MD-GPS converges to a solution. However, by delaying the point at which the global-policy is used for the KL-divergence constraint to the fifth iteration, and global-policy sampling to the tenth iteration, the final cost value is slightly lowered and the sample-efficiency is greatly improved; using roughly six times fewer samples to obtain the final cost value achieved when using MD-GPS without any alterations. Finally, performing pre-training while omitting the alterations, causes the algorithm to diverge.

Summarizing, it is shown that by combining MD-GPS with demonstrations and making certain alterations in training structure, sample-efficiency is improved significantly and robustly. This opens up the way for more applications in UAV flight control. Reducing the need for samples allows a policy to be trained using real-world interactions, which circumvents the issue of having to accurately model the system dynamics and the associated reality-gap.

References

[1] Sutton, R. S., Barto, A. G., and Williams, R. J., *Reinforcement Learning Is Direct Adaptive Optimal Control*, Vol. 12, 1992. https://doi.org/10.1109/37.126844.

[2] Bagnell, J., and Schneider, J., "Autonomous Helicopter Control Using Reinforcement Learning Policy Search Methods," *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, Vol. 2, 2001, pp. 1615–1620. https://doi.org/10.1109/ROBOT.2001.932842.

[3] Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E., "Autonomous Inverted Helicopter Flight via Reinforcement Earning," *Springer Tracts in Advanced Robotics*, Vol. 21, 2006, pp. 363–372. https://doi.org/10.1007/11552246_35.

[4] Waldock, A., Greatwood, C., Salama, F., and Richardson, T., *Learning to Perform a Perched Landing on the Ground Using Deep Reinforcement Learning*, 2017. https://doi.org/10.1007/s10846-017-0696-1.

[5] Imanberdiyev, N., Fu, C., Kayacan, E., and Chen, I.-M., "Autonomous Navigation of UAV by Using Real-Time Model-Based Reinforcement Learning," *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Vol. 2016, 2016, pp. 1–6. https://doi.org/10.1109/ICARCV.2016.7838739.

[6] Ferrari, S., and Stengel, R. F., "Online Adaptive Critic Flight Control," *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 5, 2004, pp. 777–786. https://doi.org/10.2514/1.12597.

[7] Yu, Y., "Towards Sample Efficient Reinforcement Learning $_*$," *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 5739–5743.

[8] Spector, B., and Belongie, S., "Sample-Efficient Reinforcement Learning through Transfer and Architectural Priors," , No. 3, 2018.

[9] Kadous, M., Sammut, C., and Sheh, R., "Autonomous Traversal of Rough Terrain Using Behavioural Cloning," *the 3rd International Conference on Autonomous Robots and Agents*, , No. June, 2006.

[10] Vakanski, A., Mantegh, I., Irish, A., and Janabi-Sharifi, F., "Trajectory Learning for Robot Programming by Demonstration Using Hidden Markov Model and Dynamic Time Warping," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 42, No. 4, 2012, pp. 1039–1052. https://doi.org/10.1109/TSMCB.2012.2185694.

[11] Choi, S., Kim, S., and Jin Kim, H., "Inverse Reinforcement Learning Control for Trajectory Tracking of a Multirotor UAV," *International Journal of Control, Automation and Systems*, Vol. 15, No. 4, 2017, pp. 1826–1834. https://doi.org/10.1007/s12555-015-0483-3.

[12] Levine, S., and Koltun, V., "Guided Policy Search," *International Conference on Machine Learning*, 2013, pp. 1–9.

[13] Levine, S., Finn, C., Darrell, T., and Abbeel, P., "End-to-End Training of Deep Visuomotor Policies," *arXiv:1504.00702 [cs]*, 2015.

[14] Montgomery, W., and Levine, S., "Guided Policy Search as Approximate Mirror Descent," *arXiv:1607.04614 [cs]*, 2016.

[15] Levine, S., and Koltun, V., "Variational Policy Search via Trajectory Optimization," *Advances in Neural Information Processing Systems*, 2013, pp. 207–215.

[16] Levine, S., and Koltun, V., "Learning Complex Neural Network Policies with Trajectory Optimization," *International Conference on Machine Learning*, 2014, pp. 829–837.

[17] Levine, S., and Abbeel, P., "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics," *Advances in Neural Information Processing Systems*, 2014, pp. 1071–1079.

[18] Wang, H., and Banerjee, A., "Bregman Alternating Direction Method of Multipliers," *arXiv:1306.3203 [cs, math, stat]*, 2013.

[19] Zhang, T., Kahn, G., Levine, S., and Abbeel, P., "Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search," *arXiv:1509.06791 [cs]*, 2015.

[20] Finn, C., Levine, S., and Abbeel, P., "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization," *International Conference on Machine Learning*, 2016, pp. 49–58.

[21] Chebotar, Y., Kalakrishnan, M., Yahya, A., Li, A., Schaal, S., and Levine, S., "Path Integral Guided Policy Search," *arXiv:1610.00529 [cs]*, 2016.

[22] Montgomery, W., Ajay, A., Finn, C., Abbeel, P., and Levine, S., "Reset-Free Guided Policy Search: Efficient Deep Reinforcement Learning with Stochastic Initial States," *arXiv:1610.01112 [cs]*, 2016.

[23] Chebotar, Y., Hausman, K., Zhang, M., Sukhatme, G., Schaal, S., and Levine, S., "Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning," *arXiv:1703.03078 [cs]*, 2017.

[24] Ennen, P., Bresenitz, P., Vossen, R., and Hees, F., "Learning Robust Manipulation Skills with Guided Policy Search via Generative Motor Reflexes," *arXiv:1809.05714 [cs]*, 2018.

[25] Todorov, E., and Weiwei Li, "A Generalized Iterative LQG Method for Locally-Optimal Feedback Control of Constrained Nonlinear Stochastic Systems," *Proceedings of the 2005, American Control Conference, 2005.*, IEEE, Portland, OR, USA, 2005, pp. 300–306. https://doi.org/10.1109/ACC.2005.1469949.

[26] Han, W., Levine, S., and Abbeel, P., "Learning Compound Multi-Step Controllers under Unknown Dynamics," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Hamburg, Germany, 2015, pp. 6435–6442. https://doi.org/10.1109/IROS.2015.7354297.

[27] Mueller, M. W., "Multicopter Attitude Control for Recovery from Large Disturbances," *arXiv:1802.09143 [cs]*, 2018.

[28] Luukkonen, T., "Modelling and Control of Quadcopter," *Independent research project in applied mathematics, Espoo*, Vol. 22, 2011.

II

This part of the thesis report has already been graded for the course AE4020.

3

Reinforcement learning

This chapter will present a general explanation and exploration of the SOTA, regarding RL. First, the motivation for RL as a research field is presented in section 3.1, followed by an explanation of multiple important concepts in section 3.2, and finally an exploration of the SOTA in section 3.3.

3.1. Motivation

One of the quintessential problems in the world is how to act optimally, meaning; how to act in a way that maximizes some performance criterion. This criterion can be anything; how to invest money in such a way to optimize profit? What moves to play in a game of chess to ensure victory? How to improve social standing in a group? But also; how to control a vehicle in such a way to optimize fuel efficiency, travel time, or safety.

Problems like these have been studied extensively throughout the years in fields like optimal control theory, linear programming, and dynamic programming (DP). However, these approaches either require an accurate model of the problem dynamics, or expert knowledge and design. These requirements impose a significant burden; an accurate mathematical model of the problem may be unavailable and many problems are too complex for humans to design optimal solutions. Additionally, nature shows these biases are unnecessary; humans and animals can learn how to act near-optimally without external help.

Inspired by animal and human learning, RL aims to learn to act optimally by interacting with the environment and receiving feedback in the form of reward. Over time, in a process of trial-and-error, the system learns how to act in a way that optimizes the cumulative reward. One can think of a dog that receives a treat when it fetches a stick or sits when told to do so; over time it will perform these actions on command as it has learned that they are followed by a reward. Interestingly, animal brain structures have been found that operate analogous to a RL approach called temporal difference (TD) learning[71].

3.2. Important concepts

This section explains multiple important concepts in the field of RL, starting with the framework in which it operates, followed by an explanation of accumulated reward, value functions, state-value functions, dynamic-programming, Monte-Carlo methods, temporal-difference methods, and finally approximate methods.

3.2.1. Framework

This section explains the mathematical framework through which we will view the RL problem, laying the foundation for further analysis. This section relies heavily on the works of Sutton and Barto in [75].

Reinforcement learning can be seen as a sequential decision making process and can be cast in a finite Markov decision process (MDP) framework. MDPs are a classical formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those future rewards [75]. An MDP consists of two separate entities, the agent and environment, which interact sequentially using three signals; state, action, and reward. These interactions occur in a fixed order; the environment sends a state and scalar reward to the agent at time t, followed by the agent sending an action to the environment[1]. The receiving of the action by the environment leads to a subsequent state and

[1] The concept of sending a state, reward, or action must not be taken literally, but rather be seen in a systems-context.

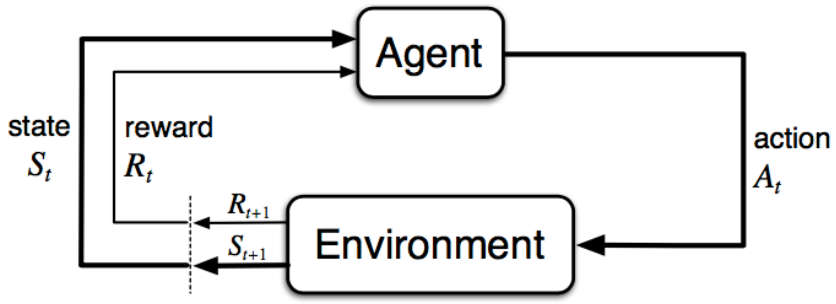


Figure 3.1: A graphical representation of the interactions in an MDP.[75]

reward at time $t+1$, starting the sequence anew. This process is graphically explained by fig. 3.1.

The three signals can be explained as follows:

- **State**
  The state, denoted by S, describes the environment at each time step. An important assumption of MDPs concerning the state is: *the future is independent of the past given the present.* This is known as the Markov assumption and can be taken to mean the state must hold all relevant information about the environment. This allows the entire environment to be described by only the present state, while past states can be ignored.

- **Action**
  An action, denoted by A, is a signal the agent sends to the environment based on the current state and its policy, influencing the transition to the next state. The goal of RL algorithms is to find a way to pick actions such that the accumulated reward is maximized.

- **Reward**
  A reward, denoted by R, is a scalar valued signal the agent receives from the environment to indicate whether the agent is performing optimally. Rewards are specified by the researcher and implicitly define the task the agent is supposed to complete. Although not strictly a feature of the environment in the usual sense, in this framework it is seen as such, since the agent should usually not be able to change rewards and therefore the task it is burdened with. Also, the reward due to a certain action and state at time t will be received one time step later, at time $t+1$. Both decisions follow the conventions of [75].

The two interacting entities, agent and environment, are described as follows:

- **Agent**
  The agent is the learner and decision maker [75] and governs the following variable and function:
  - An action set $\mathscr{A}$, which contains the actions available to the agent. This may be a function of the state, since some actions may be unavailable in certain states.
  - A policy π, which maps states to probabilities over the action set. This defines the agent's behavior and is described mathematically as follows:

$$\pi(a|s) = \Pr\{A_t = a|S_t = s\} \tag{3.1}$$

    The ultimate goal of RL algorithms is to find a policy that maximizes the accumulated reward.

- **Environment**
  The environment is everything outside direct control of the agent, described by the following variables and function:
  - A state set $\mathscr{S}$, which contains all possible states.
  - A reward set $\mathscr{R}$, which contains all rewards.

– Transition probabilities p, which give the probability of encountering subsequent states and rewards, given the current state and action. This is described mathematically as follows[75]:

$$p(s', r|s, a) \doteq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \tag{3.2}$$

The transition probabilities fully describe the dynamics of the environment, and implicitly the task.

3.2.2. Accumulated reward

As mentioned before, informally, the goal of RL algorithms is to find a policy that maximizes the accumulated reward. This is called the return, denoted by G, and is some function of future rewards. In the simplest case a straightforward total sum of rewards may be used[75]:

$$\begin{aligned} G_t &= R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \\ &= R_{t+1} + G_{t+1} \end{aligned} \tag{3.3}$$

Note how the return can be written as a recursive relation, this is an important feature of RL algorithms and DP.

To ensure the simple sum converges the sequence has to be finite, ending at some time T. This definition of return therefore naturally lends itself to tasks with a well defined ending, called episodic tasks. These kinds of tasks have states where the sequence ends, called terminal states. Natural examples of this are games, like chess, tic-tac-toe, or soccer.

In contrast, tasks without a clear ending are called continuing tasks. These are tasks that last for so many time steps it is no longer feasible to use eq. (3.3). Without a clear end, the final time step might be at $T = \infty$, risking a divergent sum. There exist many ways of handling the return of a task without a clearly defined ending, usually involving some kind of weighting of the different elements of the reward sequence. Simple exponential discounting of rewards is used the most, described by:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \quad 0 \leq \gamma \leq 1 \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{3.4}$$

The parameter γ is called the discount rate and controls to which extend the agent takes into account more temporally distant rewards in a smooth way. It can be used to interpolate between a myopic agent, which cares only aboute immediate reward ($\gamma \to 0$), and an agent which cares equally about immediate and future reward ($\gamma \to 1$). The latter ends up being equal to the simple sum from eq. (3.3).

In general, the more far away in time an event will take place, the more uncertain it is. Discounting rewards can therefore be thought of as a simplified way of representing the uncertainty about the future. In practice the optimal value of γ depends on the problem, being a trade-off between going for the more certain, but small, immediate reward, or the more uncertain, but large, long-term gain.

3.2.3. Value functions

As mentioned in section 3.2.2, RL algorithms try to optimize the return. To this end, information is gathered by interacting with the environment regarding rewards. This information is often stored in so-called value functions; functions which provide an estimate of the expected return. Since the value of a state depends heavily on the policy which is followed, value functions are always defined w.r.t. a policy. Two types of value functions exist, state-value function, which are a function of state only, and action-value functions, which are a function of state and action as well.

3.2.4. State-value functions

The state-value function for policy π; $v_\pi(s)$, is an estimate of the expected return when following policy π, starting in state s. It is defined as follows:

$$v_\pi(s) \doteq \mathbb{E}_\pi\big[G_t|S_t = s\big] = \mathbb{E}_\pi\big[R_{t+1} + \gamma G_{t+1}|S_t = s\big], \quad \forall s \in \mathcal{S} \tag{3.5}$$

Note how the recursive relation of the return leads to a recursive state-value function, again, this is an important property.

Using the policy and transition probabilities the expectation in eq. (3.5) can be written out explicitly:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma v_\pi(s')\big], \quad \forall s \in \mathscr{S} \tag{3.6}$$

The summations in this equation are implicitly over the action set in state s; $\mathscr{A}(s)$, the entire state set $\mathscr{S}$, and reward set $\mathscr{R}$.

3.2.5. Action-value functions

The action-value function for policy π; $q_\pi(s,a)$, is an estimate of the expected return when, while in state s, first taking action a and subsequently following policy π. This is formally defined as follows:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\big[G_t|S_t = s, A_t = a\big] = \mathbb{E}_\pi\big[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a\big], \quad \forall s \in \mathscr{S}, \forall a \in \mathscr{A}(s) \tag{3.7}$$

Unsurprisingly, state- and action-values are closely related, since after the initial action a and transition from s to s', policy π is simply followed. The expected return of following policy π from a state is simply the value function, leading to the following relation:

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}_\pi\big[G_t|S_t = s, A_t = a\big] \\ &= \mathbb{E}_\pi\big[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a\big] \\ &= \mathbb{E}\big[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a\big], \quad \forall s \in \mathscr{S}, \forall a \in \mathscr{A}(s) \end{aligned} \tag{3.8}$$

Again, using the transition probabilities, the expectation in eq. (3.7) can be written out explicitly:

$$q_\pi(s,a) = \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma v_\pi(s')\big], \quad \forall s \in \mathscr{S}, \forall a \in \mathscr{A}(s) \tag{3.9}$$

Note that the only difference, when comparing to eq. (3.6), is the summation over actions and weighting by the policy has disappeared.

Optimality

Informally, the optimal policy is the one which acquires the largest return. More specifically, a policy is better than or equal to another if the expected return from any state is greater than or equal[75]. This is formally defined using value functions:

$$\pi \geq \pi' \iff v_\pi(s) \geq v_{\pi'}(s) \quad \forall s \in \mathscr{S} \tag{3.10}$$

The optimal policy, π_*, is simply the policy of which the value function is greater than or equal to all others:

$$\pi = \pi_* \iff v_\pi(s) \geq v_{\pi'}(s) \quad \forall s \in \mathscr{S}, \forall \pi' \tag{3.11}$$

There is at least one optimal policy, but they all share the same optimal value function.

The optimal value function, v_*, is defined as follows:

$$v_*(s) \doteq \max_\pi v_\pi(s), \quad \forall s \in \mathscr{S} \tag{3.12}$$

The intuition that the optimal policy should, at each state, chose the action with greatest expected return leads to the Bellman optimality equations for the state-value function:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}_{\pi_*}\big[G_t|S_t = s, A_t = a\big], \quad \forall s \in \mathscr{S} \\ &= \max_a \mathbb{E}_{\pi_*}\big[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a\big], \quad \forall s \in \mathscr{S} && (3.13) \\ &= \max_a \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma v_*(s')\big], \quad \forall s \in \mathscr{S} && (3.14) \end{aligned}$$

The optimal action-value function is defined as follows:

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \quad \forall s \in \mathscr{S}, \forall a \in \mathscr{A}(s) \tag{3.15}$$

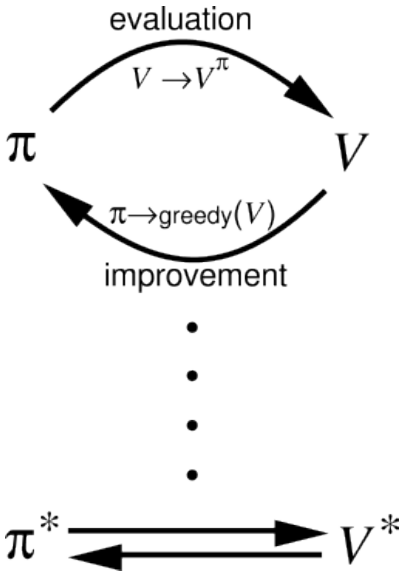
It provides the expected return after, while in state s, choosing action a and subsequently following an optimal policy. The Bellman equations for the action-values are as follows:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a\big], \quad \forall s \in \mathscr{S}, \forall a \in \mathscr{A}(s) && (3.16) \\ &= \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma \max_{a'} q_*(s', a')\big], \quad \forall s \in \mathscr{S}, \forall a \in \mathscr{A}(s) && (3.17) \end{aligned}$$

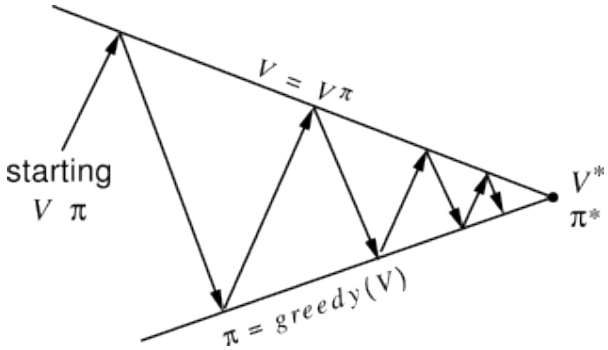
3.2.6. Dynamic Programming

DP here refers to an umbrella term for methods that use perfect knowledge of the MDP's transition probabilities to iteratively find value functions and improve policies. Although this is almost never the case in practical RL problems, the general ideas and methods still are useful to understand as most RL algorithms can be traced back to them.

DP uses a process called generalized policy iteration (GPI) to iteratively estimate value functions and improve policies. Generalized policy iteration consists of two separate processes which are applied sequentially; policy evaluation, and policy improvement. This is explained graphically by fig. 3.2a.



(a) A graphical representation of GPI.[75]



(b) A graphical representation of how GPI converges to the optimal value function and policy.[75]

Policy evaluation tries to estimate the value function for a certain policy. This is done by turning the Bellman equations (see eqs. (3.13), (3.14), (3.16) and (3.17)) into update equations. The algorithm will initialize the value function arbitrarily, go through each state in $\mathcal{S}$ and by using perfect knowledge of both policy and transition probabilities ensure the estimate will eventually converge to the value function of the policy. The Bellman update equation for the state- and action-value function estimates are the following, respectively:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}\big[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s\big], \quad \forall s \in \mathcal{S} \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma v_k(s')\big], \quad \forall s \in \mathcal{S} \end{aligned} \tag{3.18}$$

$$\begin{aligned} q_{k+1}(s,a) &= \mathbb{E}\big[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') | S_t = s, A_t = a\big], \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s) \\ &= \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma \max_{a'} q_k(s', a')\big], \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s) \end{aligned} \tag{3.19}$$

The iterative process of improving the value function estimate eventually converges to the one corresponding to the followed policy. After converging, the policy can be improved by making use of its value function; the process of policy improvement.

The policy improvement theorem states that when taking a different action then the policy dictates improves the value function, the policy can be improved. Fortunately, the action-value function can answer this question straightforwardly:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \implies v_{\pi'}(s) \geq v_\pi(s), \quad where \quad \pi'(x) = \pi(x) \quad \forall x \neq s \tag{3.20}$$

This result leads to the two greedy update equations for improving policies based on value functions. When the state-value function is known it is a straightforward maximization over actions to improve the policy, shown in eq. (3.21). The policy improvement equation by using state-values is obtained by expanding the action-values using eq. (3.9), shown in eq. (3.22). As can be seen an improved policy can be obtained from

action values directly without using the transition probabilities, this simplification is paid for having to find values for every combination of state and action.

$$\pi'(s) = \arg\max_a q_\pi(s,a), \quad \forall s \in \mathcal{S} \tag{3.21}$$

$$= \arg\max_a \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma v_\pi(s')\big], \quad \forall s \in \mathcal{S} \tag{3.22}$$

By alternating between policy evaluation and improvement, the policy is gradually improved, abstractly shown in fig. 3.2b. Policy evaluation is guaranteed to converge after infinitely many steps, however in practice some stopping criterion is used on the maximum difference between successive value estimates. Waiting for the value function to converge in this sense is called policy iteration, however, it turns out this is not necessary to find the optimal policy. In fact, a single policy evaluation iteration over the state set works as well, this is called value iteration. In this case the evaluation and improvement steps can be combined into one neat formula for state-value improvement:

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}\big[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a\big] \\ &= \max_a \sum_{s'} \sum_r p(s', r|s, a)\big[r + \gamma v_k(s')\big], \quad \forall s \in \mathcal{S} \end{aligned} \tag{3.23}$$

3.2.7. Monte Carlo methods

In section 3.2.6 it was shown how, when transition probabilities are known, DP methods can be used to find value functions and improve policies. However, in most practical RL problems this knowledge is unavailable. There are multiple techniques able to learn exclusively from experience, of which Monte Carlo (MC) methods will be explained in this section.

Although MC methods learn from experience, the principle of GPI introduced in section 3.2.6 can still be applied. Again, policies are optimized by alternating between policy evaluation and improvement.

Policy evaluation and improvement

MC methods learn value functions by sampling experience from an MDP, this can be the actual process, but a simulation works as well. In the case of a simulated MDP a reasonable accurate model is needed, but explicit transition probabilities are unnecessary. Instead, the model must only provide samples, which is usually easier.

State-value function estimation is done by starting in some state and following the to-be-evaluated policy thereafter. If a state occurs at least once during the episode, the return is added to a list of returns for that state. After each episode the set of returns is averaged and stored in the value-function.

State-values are useful, but since transition probabilities are unavailable it is not possible perform the policy update from eq. (3.22). Because of their straightforward policy update (see eq. (3.21)), it is more convenient to estimate action-values. Fortunately, the algorithm mentioned above can still be used, averaging returns per state-action pair instead of state.

Usually, instead of rolling out episodes until the action-values converge completely, policy evaluation is cut short after one episode. This is somewhat akin to the concept of value iteration introduced in section 3.2.6. Even though the true value function is not reached after one episode, the policy still improves. After each episode the action-values are used to update the policy using eq. (3.21).

Exploration

To find the optimal policy, every state-action pair needs to be visited continually, a requirement which works against the concept of acting according to the optimal policy. This dichotomy is inherent to RL problems and has been dubbed "exploration vs. exploitation", also called the dual control problem.

Exploration in MC methods can be handled in several ways:

- **Exploring starts**
  Exploring starts explore the state-action space by starting each episode with a random state-action pair. After this one exploring action the policy is followed.

- **Soft policies**
  Soft policies are stochastic policies which have nonzero probability for every state-action pair, ensuring that they are all visited eventually. The most common example is ϵ-greedy, which has a chance of $1-\epsilon$ the actual policy is followed, and an ϵ chance a random action is chosen.

Off- and on-policy learning

A problem with using a soft policies is the fact that exploration becomes part of the policy which is evaluated, even though its exploratory actions are known to be suboptimal and are not part of the policy we are actually interested in. In RL the policy which generates behavior is called the behavior policy, while the one being of interest, the one being learned about, is called the target policy. The problem mentioned above is a problem of so-called on-policy learning; the behavior policy is equal to the target policy. To circumvent this problem, off-policy learning can be used; RL algorithms which are able to learn about a policy different from the one generating behavior.

Off-policy- are more powerful and general than on-policy methods, but usually have higher variance and are slower to converge[75]. The only thing off-policy requires is that the behavior policy has has nonzero probability of choosing action a in state s where the target policy does. This is called the assumption of coverage, formally defined as follows:

$$\pi(a|s) > 0 \implies b(a|s) > 0, \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s) \tag{3.24}$$

Where b is the behavior policy.

A common way of doing off-policy learning is to weight returns by the relative ratio of the probability of their trajectory following either behavior- or target-policy, this is called importance sampling. When there is a lower chance of encountering some trajectory if the target policy would be followed instead of the behavior policy, the return is given low weight and vice-versa. This relative probability of the trajectory from time step t to $T-1$ is determined by the following equation:

$$\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{3.25}$$

Since, except for the assumption of coverage, the target policy has no dependence on the behavior, off-policy methods are able to learn about multiple target policies at the same time, making much better use of the available experience. This opens up the ability of learning from data gathered using experts or conventional controllers, learning extensive predictive models of the world, and increasing sample-efficiency significantly[74, 75].

3.2.8. Temporal-Difference methods

Another method which is able to learn directly from experience without requiring a model of the environment is TD learning. Where MC methods wait until the end of an episode to make updates to the value function, TD methods perform value updates immediately after receiving rewards using a technique called bootstrapping; making value updates partly based on other estimated values. Just like DP and MC methods, TD learning makes use of GPI; alternating value function estimation and greedy policy improvement.

TD methods perform iterative value updates by calculating a TD target; a new value function estimate made by integrating received rewards with an old estimate. The value function is updated by interpolating between the old value and the TD target based on a step-size parameter α, the difference between the old- and new estimate is called the TD error. The update rule of n-step TD for state-value learning is as follows:

$$V_{t+n}(S_t) \longleftarrow V_{t+n-1}(S_t) + \alpha\big[\overbrace{G_{t:t+n}}^{\text{TD target}} - \underbrace{V_{t+n-1}(S_t)}_{\text{TD error}}\big] \tag{3.26}$$

Where the subscript of the state-value function denotes the time step at which the function is evaluated and the n-step return estimate $G_{t:t+n}$ is defined as follows:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \tag{3.27}$$

Naturally, TD methods can be used to learn action-values as well, which is more convenient for finding optimal policies. Slightly different from the state-value update shown in eq. (3.26), the general idea of incrementally moving the value function towards a bootstrapped estimate still applies.

As can be seen TD target of the state-value of a state at time t is determined using n rewards and the state-value estimate at time $t+n-1$, therefore, n determines the amount of new information incorporated into each update. If set to 1, the algorithm becomes one-step TD, if set to ∞ one obtains an MC method.

Since the n-step update for a state at time t requires information from a later time, it is delayed by n steps. This is called a forward view algorithm and in a control setting this delay is unwanted; new information should be incorporated sooner rather than later. To solve this problem (and offer a computational advantage[75]) eligibility traces can be used, offering a backward view version of n-step TD.

Eligibility traces refer to a trace vector, which estimates which states are eligible for a value update due to newly received rewards. The eligibility of each state decays exponentially over time, using the trace-decay parameter λ, valued between 0 and 1. Analogous to the parameter n in n-step TD, λ can be used to interpolate between one-step TD and MC methods. TD methods which incorporate eligibility traces and the trace-decay parameter are called TD(λ).

3.2.9. Approximate methods

The methods mentioned before have all been so-called tabular methods; the state- and action-spaces are finite and discrete. Tabular methods are useful to showcase the basics of RL, but are not convenient for flight control because of the continuous state- and action-space. Although it is possible to simply discretize every state and use tabular methods, the amount of states and the resolution needed make it infeasible. Instead, methods can be used which use function approximation to parameterize policies and value functions; approximate methods.

Approximate methods usually apply stochastic gradient-based optimization techniques to find value functions and optimize policies, this requires the parameterizations to be differentiable w.r.t. the parameters. Except for this requirement, any parameterization can be used; linear models, artificial neural networks (ANNs), decision trees, etcetera.

Gradient-based methods for finding value functions require a performance metric which specifies how well the estimated value function approximates the actual values, for which the mean squared error (MSE) of the actual and approximated value function is an obvious choice. However, the real value function is not known in advance, which is why an approximation will be used; $U(s)$. The resulting objective function to be minimized is as follows:

$$\bar{V}(\boldsymbol{w}) = \frac{1}{2} \sum_{s \in \mathscr{S}} \big[U(s) - \hat{v}_\pi(s, \boldsymbol{w}) \big]^2 \tag{3.28}$$

Where $\boldsymbol{w}$ is a parameter vector, and $\hat{v}_\pi$ is the state-value function estimate given by the parameterized model.

This performance metric leads to the following one-step stochastic gradient descent (SGD) update of the state-value function:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \big[U_t(S_t) - \hat{v}_\pi(S_t, \boldsymbol{w}_t) \big] \nabla_{\boldsymbol{w}} \hat{v}_\pi(S_t, \boldsymbol{w}_t) \tag{3.29}$$

Where α is a step size parameter, and $\nabla \hat{v}_\pi(S_t, \boldsymbol{w}_t)$ is a vector of partial derivatives of $\hat{v}$ w.r.t. $\boldsymbol{w}_t$.

The same can be done to estimate action-values, leading to the following update equation:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \big[U_t(S_t, A_t) - \hat{q}_\pi(S_t, A_t, \boldsymbol{w}_t) \big] \nabla_{\boldsymbol{w}} \hat{q}_\pi(S_t, A_t, \boldsymbol{w}_t) \tag{3.30}$$

The value approximations $U(s)$ and $U(s,a)$ can be any value approximation. When using observed- or bootstrapped returns this results in MC- and TD-like methods, respectively. However, when using a bootstrapped estimate, the method becomes a so-called semi-gradient method and does not converge as robustly. Still, there are benefits to using semi-gradient methods, such as faster learning and a natural application to continuing tasks, which is why they are widely used.

3.2.10. Policy-gradient methods

The techniques mentioned before all learn optimal policies by applying some form of GPI, which alternates between finding value functions and optimizing the policy. However, it is also possible to skip value functions and optimize the policy directly using the gradient and returns. These methods are called policy-gradient methods.

By using the policy-gradient theorem, an equation for updating the policy, parameterized by $\boldsymbol{\theta}$, based only on the observed return and the policy itself is obtained:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha G_t \nabla_{\boldsymbol{\theta}} \ln \pi(A_t | S_t, \boldsymbol{\theta}_t) \tag{3.31}$$

These parameter updates are used by the REINFORCE algorithm and although value functions are not necessary for the policy to be optimized this way, the algorithm can be improved by subtracting a baseline from the return; the state-value estimate.

Other methods which use both policy-gradient and value functions are actor-critic (AC) methods. The difference with the aforementioned method being that instead of using the full return over an episode, the returns are bootstrapped like in TD methods. The state-value function is used in the bootstrapped return estimate and is learned alongside the policy improvement, following the GPI paradigm.

The principle of AC methods is general and there are many variations, for instance the policy update equation of one-step AC is as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\big[R_{t+1} + \gamma\,\hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t)\big]\nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta}_t) \tag{3.32}$$

With the following update equation for the state-value function estimate:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\big[R_{t+1} + \gamma\,\hat{v}(S_{t+1}, \boldsymbol{w}_t) - \hat{v}(S_t, \boldsymbol{w}_t)\big]\nabla_{\boldsymbol{w}} \hat{v}(S_t, \boldsymbol{w}_t) \tag{3.33}$$

3.3. State-of-the-art

RL methods have been applied to flight control because of their inherent adaptive capabilities, ability to learn policies model-free, and flexibility. This section will give a short overview of the SOTA of RL methods for flight control.

A problem with using RL methods for aircraft is that initial suboptimal policies may lead towards states which cause complete loss of the vehicle. A way of solving this problem is by using samples of some expert controller to guide learning of the policy; guided policy search (GPS). In [89], Zhang et al. use GPS to safely learn optimal policies for UAV control using only on-board sensors. At training time the quadcopter is controlled by model predictive control (MPC) in conjunction with full state feedback provided by a testing rig. The trajectories which are followed by the behavior policy are chosen such that they simultaneously maximize a performance criterion and minimize the deviation with the current target policy, thus providing good guiding samples. The target policy is a ANN which is trained using supervised learning on the guiding samples, using only raw on-board measurements. The learned controller is able to effectively navigate hallways and avoid cylindrical obstacles using limited state feedback, while being computationally cheaper than MPC.

In [35], Hwangbo et al. control a quadcopter using an actor-critic RL scheme, where both actor and critic are parameterized by separate fully-connected ANNs with full-state input. The actor and critic are trained using MC-style GPI; each iteration consists of a rollout of the current policy until termination, optimizing the critic network using a gradient-based method on the observed on-policy samples, followed by updating the policy using a natural policy gradient method on all samples. Since the actor is deterministic, exploration is handled by interrupting the on-policy trajectories with off-policy ones, using randomized actions. To stabilize the vehicle during learning the control output of the actor is added to the control output of a simple attitude PD controller. As for the learning procedure; first, a simplified model of the MAV is used to train the agent in a simulation. Afterwards, without further training the controller deployed on a real drone and tested again using a testing rig which provides full-state feedback. The learning performance of the natural policy gradient method is compared to both deep deterministic policy gradients (DDPG)[53] and trust region policy optimization (TRPO)[70], and shown to outperform both. The learned policy was compared to a linear MPC controller and found to have much higher success rates when subjected to bad initial states. The move from simulation to reality shows that the dynamics of the real world affect performance of the policy negatively, since engine dynamics and the ground-effect have not been modeled. However, the agent is able to follow waypoints, albeit with worse tracking error than a conventional controller. Also, the agent is able to recover from bad initial states, like a downward velocity of 5 m/s upside down, even better than in simulation, which is attributed to unmodeled stabilizing drag forces.

A recursive least-squares (RLS) based RL method is used in [62] to transport a suspended load along a predetermined trajectory by quadcopter. First, the agent is trained in simulation using a GPI scheme. The learning procedure, called least-squares policy iteration (LSPI)[45], consists of updating the auto-covariance matrices each time step using RLS, re-estimating the action-value function once every K steps, followed by a greedy policy optimization. For exploration an ϵ-soft behavior policy is used, while the ϵ decays over time. The reward is a linear function of control-action magnitude, deviation from the reference path, and swing-angle of the suspended load. Training is performed in simulation, while later validated in an experiment without re-training. In both cases full-state feedback is available, using a testing rig with motion-capture capabilities. Results show the algorithm is able to keep the deviation from the reference path within a couple of centimeters, both on straight lines and more complex trajectories.

4

Learning from Demonstration

As explained in chapter 3, humans and animals are able to optimize their behavior based on their experiences, through a process of trial-and-error. This kind of learning inspired the creation of RL techniques for policy optimization. However, there exist other modes of learning new behavior in the animal kingdom, one of which is learning by imitation.

Instead of learning from one's own experiences, policies can be inferred from observed behavior of others, that animals and humans employ this strategy to learn from their peers is well understood[87]. Partly inspired by nature, and partly because of the need of transferring knowledge from man to machine in a intuitive way, researchers have employed learning by imitation in many shapes and forms to teach machines all kinds of tasks; from block stacking to flight control. Over the years many different terms have been used to convey the general idea of learning by imitation, some of them mentioned in section 4.1.1. For the rest of this research the term LfD will be used to refer to these methods in general.

This section presents an overview of these types of learning, which will be referred to as LfD. Starting with an explanation of the relevancy to this research, followed by a definition of what exactly is meant by the term. Thirdly, a short summary of how LfD techniques are generally applied, together with explanations of recurring themes and issues, is given. Finally, the history of several lines of research and the SOTA are investigated.

4.1. Definition and framework

Before investigating the different methods and applications from the literature, a general explanation of LfD methods will be presented. First, a proper definition of what is mean by LfD is given. This is followed by a framework through which LfD will be viewed and finally some recurring themes and issues encountered in the literature.

4.1.1. Definition

As it turns out there are many lines of research into the general idea of learning by imitation, using different taxonomies and applications. To allow a structured discussion of the field, an overview of definitions is given, followed by the definition used in this research.

The following definitions can be found in the relevant research:

- Argall et al. [3] define LfD as follows: "Within LfD, a policy is learned from examples, or demonstrations, provided by a teacher. We define examples as sequences of state-action pairs that are recorded during the teacher's demonstration of the desired robot behavior. LfD algorithms utilize this dataset of examples to derive a policy that reproduces the demonstrated behavior.".

- Hussein et al. define imitation learning as follows: "The process of imitation learning is one by which an agent uses instances of performed actions to learn a policy that solves a given task."[34].

- In their work on imitative learning[28], Hayes and Demiris first define imitation as "being a phenomenon which leads to some sort of similarity in behavior among two or more individuals". In this sense they presumably define imitative learning as learning behaviors by making use of imitation, without

explicitly stating so. They go on to describe a particular type of imitative learning, matched dependent behavior, as follows: "...rats negotiating a maze receive reinforcement faster if they match their behaviour to that of a leader rat, and then learn to associate the other stimuli in the environment at the time with the appropriate action.".

- In [7], Kang mentions: "The simplest manner an operator can program a robot is to demonstrate the task in front of the system and expect the system to replicate the task with little other human intervention. This is the method that we adopt in task programming; it is also known as the Assembly Plan from Observation (APO)". Later he defines programming by demonstration (PbD) as follows: "The APO approach of programming a robot belongs to a class of what we term as a programming-by-demonstration approach. This class of robot programming approach involves a user-friendly interface that allows the user to easily specify the task in a simple and intuitive manner without resorting primarily to hand-coding programs.".

- Ng and Russell define apprenticeship learning as follows: "The task of learning from an expert is called apprenticeship learning (also learning by watching, imitation learning, or learning from demonstration)"[2].

- In [21], Esmaili et al. define behavioral cloning (BC) to be "the process of reconstructing the skills from operators' behavioural traces by means of machine learning techniques.".

From the aforementioned definitions, it becomes apparent that in fact these authors are talking about roughly the same concept; LfD, imitation learning, imitative learning, PbD, apprenticeship learning, learning by watching, and BC are different names for the same idea. In this work these methods will be referred to as LfD, using the following definition:

LfD methods are methods which transfer task knowledge from one agent to another through experience tuples, called demonstrations.

4.1.2. Framework

To allow a structured discussion of LfD, this section will present a framework through which these methods are viewed. In fig. 3.1, the MDP framework was introduced graphically, an augmented version is used to aid the discussion here, shown in fig. 4.1.

The usual RL agent can be seen on the left side of fig. 4.1. In the usual framework the agent lives in the environment, which is everything outside its direct control. However, to facilitate the discussion concerning LfD, an explicit distinction will be made between the part of the environment which is close and unique to similar agents, and the world around them. This is intuitively understood as follows: humans are similar agents, however each has slightly different (1) physical attributes; limb length, muscle strength, etc, (2) sensors; differences in sensitivity of hearing, color-blindness, (3) state representation; differences in exact state representations formed in the brain are inevitable because of differences in experience and genetics. Despite these differences on a local level, all humans inhabit the same external world; the universe, which has the same dynamics and state for everyone.

The distinction is made explicit in the augmented framework by dividing up the environment into two pieces; the internal- and external part. The first is determined by the agent's embodiment; its sensors and actuators, and its interpreter, which determines the state representation and reward function. The external environment is the world outside of the agent, which has a world state, being much larger than the observed state and its representation, and state transition function.

In general LfD applications can be viewed as a transfer of knowledge from one MDP to another, each formed by the combination of an agent and environment as shown in fig. 4.1. There are three levels at which one can characterize LfD methods; MDP difference, MDP interface, use of transferred information.

- **MDP difference**
  In LfD problems, the differences between the two MDPs are chiefly due to a difference in the agents; the teacher and student. The fundamental difference being that the teacher has a (supposedly optimal) policy, which must be communicated to the student.

  Another difference lies in the embodiment of the agents; sensors, actuators and state representations are generally different for teacher and student. This has an important effect on the ease of imitation; a teacher which is too different from the student may not be able to provide useful information, or at least require some mapping to the student's coordinate frame.

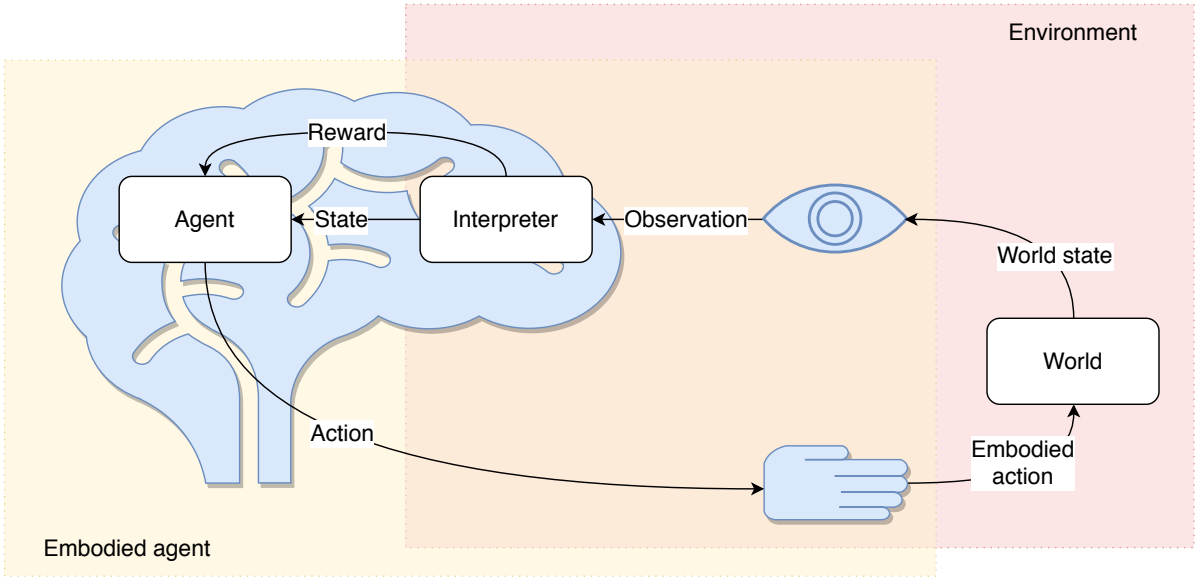


Figure 4.1: An augmented form of the basic MDP framework. The difference between the internal- and external environment is made explicit by the embodiment of the agent. The world state is viewed through an interpreter and actions are transformed into embodied actions by the agents actuators.

Also, the student does not have the same reward function as the teacher or might not even have a reward function at all. The teacher has no way of communicating its reward function directly and must rely on demonstrations to transfer that knowledge implicitly if needed.

In general the teacher and student inhabit the same external world, having the same state-set and transition probabilities.

- **MDP interface**
  LfD methods exclusively transfer knowledge using demonstrations, which are simply a kind of experience tuples; (s, a, s'). In this work it is assumed demonstrations are the result of a policy which, according to the teacher, optimizes behavior to complete the student's task. A setting where this requirement is not met ventures into the territory of transfer learning (TL) and is outside of the scope of this thesis.

  Using the framework from [3] by Argall et al, the interface between teacher and student is divided up into two-dimensions:

  – **Record mapping** is the mapping from the behavior of the teacher to the raw observations by the student.

  – **Embodiment mapping** is the mapping from the raw observations to the coordinate frame of the student.

  Subsequently, these mappings can be divided up into two cases; the case where the mapping is the identity function, and where it is not. The resulting subdivision of the interface is shown in fig. 4.2.

  The first of the four cases occurs when both record- and embodiment mapping are the identity function, called teleoperation. This setting applies when the teacher is able to demonstrate the behavior using the students own actuators and sensors, for instance when a pilot (teacher) is flying an aircraft to demonstrate correct behavior to an on-board learning system (student).

  When the sensors are placed on a teacher's body, the record mapping is the identity function, but the student has to deal with a non-identity embodiment mapping. Depending on the differences in embodiment between the two, this may require significant additional processing. An example of this case would be a setting where demonstrations are performed and logged on one aircraft, while the learning system is embodied by another, not necessarily the same one.

  Shadowing occurs when teacher and student have the same embodiment, while the behavior of the teacher is observed through some non-identity mapping. For instance, a student is observing demonstrations

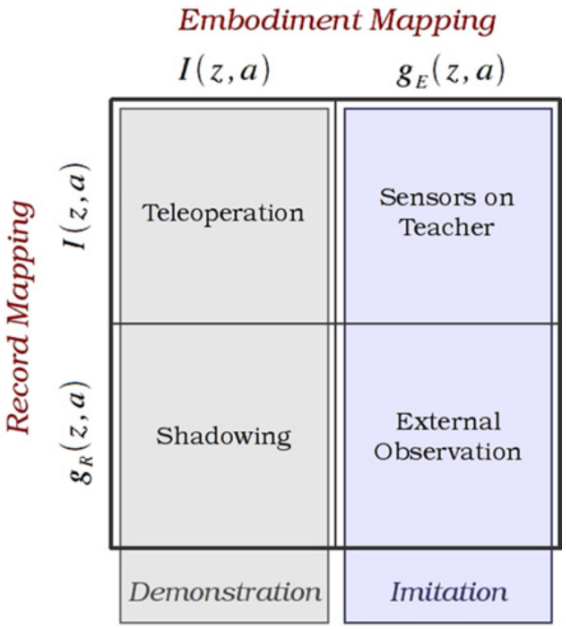


Figure 4.2: A subdivision of the MDP interface in LfD into four cases, based on the record- and embodiment mapping, taken from[3]. Note that Argall et al. separate demonstration from imitation, a distinction which does not fit with this work and can be ignored.

by an identically embodied teacher through a camera. The student will need to find a mapping from visual images to his own coordinate frame to be able to efficiently use the information.

The final, and arguably most difficult, case happens when both record- and embodiment mapping are non-identity, during which the student must find both a mapping from the observed data to states and actions of the teacher, and from the teachers coordinate frame to its own. An example of this setting would be when a civil aircraft observes demonstrations by a fighter jet using a camera feed.

Of course, this framework provides a simplified view, since the distinction between identity- and non-identity mapping is usually not clear-cut and there exists a spectrum.

- **Use of transferred information**

LfD algorithms make use of demonstrations in several ways, the most prominent one being to directly obtain a mapping of states to actions, assuming the demonstrations are the result of an optimal policy. In this case the student tries to directly imitate the teacher.

Secondly, demonstrations can be used to infer a reward function for the task at hand. This flips the RL problem upside down, which is useful in situations where the reward function corresponding to the task is too complex to state manually.

Also, demonstrations can simply be used as transition information to create a dynamic model of the world, which can be used later to plan ahead and learn policies.

Lastly, important task-specific features can be distilled by correlating states with actions. By employing these features within an RL algorithm, learning performance can be improved.

4.1.3. Important concepts

In this section several important concepts in LfD research will be explained, alongside several recurring issues and problems.

Embodiment and correspondence

In the classic MDP framework, the agent interacts with the environment, which is everything outside of its direct control. To facilitate the discussion about LfD, a distinction (presented in fig. 4.1) was made between the external- and internal environments, or; the world and the agent's embodiment. The latter is comprised

of the interpreter, sensors and actuators, it can be viewed as the agent's body. In the context of LfD the embodiment of the agent is important since it is often different for teacher and student, while the external environment is the same.

Differences in embodiment necessitate a mapping of the teacher's- to the student's coordinate frame to be able to utilize the information stored in the demonstrations. The problem of finding this mapping is called the correspondence problem. An example is a human teacher demonstrating behavior to a humanoid robot; although similar in embodiment, the robot has different limb-lengths, proportions, and joint ranges of motion from the teacher. As explained in section 4.1.2, the robot will first need to map the raw observations to the coordinate system of the teacher, followed by a mapping to its own coordinate frame. Due to the humanoid embodiment the correspondence problem is more easily solved, more serious problems arise when a six-legged robot would be used.

Motor primitive

Motor primitives, but also movement primitives or motion primitives, are elementary building blocks of behavior, of which the level of abstraction varies and is usually arbitrated by the researcher. An example of motion primitives in the context of fighting can be "punch", "kick". By viewing behavior as built up from these blocks, the task of recognizing and generating complex behavior is made more tractable, primitives can therefore be viewed as features, or abstractions, of behavior. In keeping with the fighting example, a demonstration of a fight might be represented by a sequence of punches and kicks, which makes it easier to understand what happened as compared to working simply with the raw movement sequences.

Active vs Passive

Based on the amount of interaction between teacher and student, LfD methods can be separated into two classes; active and passive. Passive, also called "batch", refers to the case when the student simply observes the teacher and has no way of influencing which demonstrations are performed. This requires the least amount of attention from the teacher, as demonstrations can be generated offline.

However, learning performance might be improved if the student could point out situations in which the demonstrations are lacking and it is unsure what to do (the "incomplete" area in fig. 4.3), i.e. the student has a way of influencing what demonstrations are provided. This mode of LfD is called (inter)active, and requires a feedback loop in which the student can query the teacher for demonstrations when and where needed.

The trade-off between active and passive demonstrations depends on the cost of teacher attention, which might be high in the case of a human, and the need for fast learning performance.

Quality of demonstrations

In LfD information about the teacher's policy is transferred to the student by executing it and allowing the student to observe, thus the quality of these demonstrations is an important factor in the success of these methods.

A fundamental problem is the fact that demonstrations cannot cover the entire state space, either because of the cost associated to provide them and/or the fact that it is a continuous domain. This requires the teacher to efficiently use demonstrations, which is influenced by which parts of the state-space are visited and how accurate this information is conveyed by the recordings/observations.

Maximizing the overlap of demonstrations with the parts of the state-space likely to be visited by the student increases the usefulness, graphically explained by fig. 4.3.

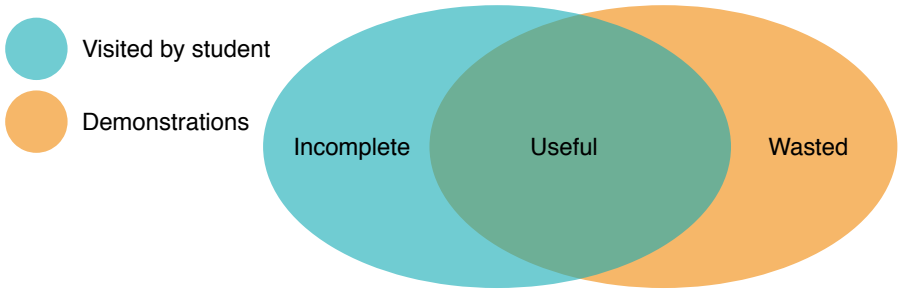


Figure 4.3: Simplified representation of the state-space and the parts visited by the student and the teacher during demonstrations, maximizing the size of the intersection improves demonstration usefulness.

When there are no demonstrations available for certain parts of the state-space visited by the student the demonstrations are incomplete, leaving the student unable to infer the policy of the teacher.

On the other hand, visiting parts of the state-space which will probably not be visited by the student wastes valuable resources. This is similar to the problem of ensuring demonstrations contain only actions part of the policy corresponding to the task at hand, avoiding any unnecessary actions.

Other factors which influence demonstration quality on the side of the teacher is avoiding ambiguous demonstrations, which might confuse the student. These occur when, in the same state, different actions are demonstrated.

The scarcity of demonstrations affects not only the teacher, but the student as well. Since, in practice, for much of the state-space there are no demonstrations available, the student is required to generalize to other states. The degree to which one can generalize across states and actions is determined by the amount of structure in these sets. Continuous domains are quite structured in the sense that similar states often require similar actions, while generalizing across discrete or abstract domains is much more difficult.

4.2. History and state-of-the-art

This section will investigate the history of LfD methods, followed by a short overview of the SOTA.

4.2.1. Programming by Demonstration

Research on using human demonstrations to learn optimal policies started around 1980, applied mainly to robotic manipulators, generally called PbD. Looking for ways of circumventing the arduous task of programming robots for more complex behaviors, researchers reasoned that task information is more conveniently transferred from human to machine by imitation. Based on AI research of the time, policies were usually learned using symbolic reasoning[69], and tasks were demonstrated using teleoperation by a human. An example is [18], where a policy for mating two parts via a robotic manipulator was learned from demonstrations. The applied method is two-phase; a training phase followed by an induction phase. During the first phase demonstrations are generated by teleoperation by a human operator, while logging sensor readings, obstacle- and goal positions. During the second phase primitive actions and goals are inferred from the demonstrations in the form of if-then rules and predefined motion state definitions. These are used to construct graph-based descriptions of each task, encoding the policy.

Using teleoperation for task demonstration is restricting, which lead to research into more convenient modes of demonstration, such as visual demonstrations [17, 36], sensor-rigged gloves called datagloves[79], and more recently kinesthetic guiding[43]. For instance in [44], visual recordings of human demonstrations of assembly tasks are used to form reusable task plans. First, a human is recorded by camera while performing the task by hand, the hand and objects are recognized by a computer vision algorithm. Secondly, the task is segmented into movement primitives, which are grouped into sub-procedures and finally a task plan by bottom-up plan inference. Finally, the task is reproduced by matching the initial state of objects with the relevant task plan and running it.

Gradually, symbolic methods have become more popular, both for segmenting tasks into motion primitives and to be used as policies. This lead to the use of fuzzy logic[17] for task segmenting, hidden Markov models (HMMs)[11, 33, 63, 78], and ANNs[6, 26, 54]. In [65], imitation learning is framed as an multi-class classification problem; actions are labels corresponding to states. State-action pairs found in demonstrations are assumed to be correct and a classifier, a ANN, is trained to assign them higher scores than other combinations. To generalize across unseen states the method relies on the generalization capabilities of the regression method and the classifier. To improve generalization a structured-margin loss function is used which exacerbates the loss differences between state-action combinations proportional to their score; similarity to demonstrated combinations. The method is applied to two problems; foot placement prediction of a quadruped robot, and optimal grasp orientation prediction of a three-fingered gripper grabbing a plethora of objects.

Inspired by advances in the field of neuroscience such as evidence for mirror neurons, bio-inspired methods gained in popularity. With an emphasis on biologically grounded methods of perception and computation, often explicitly modelling certain brain regions or parts of the nervous system of humans or primates[6, 26]. For instance, in [59] a method for learning biped locomotion is proposed. The method uses dynamical movement primitives as a central pattern generator (CPG), which is inspired by certain neurons found in humans. The movement primitives are fitted to demonstrations using locally weighted regression and a novel frequency adaptation algorithm. The method is applied to a bipedal robot succesfully, both in simulation and on a physical system.

More recent papers on PbD include [81], where generalized motion-trajectories for a robotic arm are inferred from human demonstrations. First the so-called "key points" are distilled from the data using a clustering algorithm, these are parts of the trajectory which are most important for generating the behavior. A feature vector is constructed from these clusters, which is used by a HMM

4.2.2. Behavioral cloning

Another line of research on using demonstrations to learn policies lead to methods dubbed BC, being focused more on policies for flight- and vehicle control. Early work on this was performed by Chambers and Michie, in [12] they used BC to obtain a pole-balancing policy from human examples. Years later, in [68] Sammut et al. use BC to obtain a flight controller using logged state-action pairs from human performance in a flight simulator. Esmaili et al. (including Sammut), have used two BC techniques to learn an obstacle avoidance policy for a small autonomous robot in [21]. First, data was gathered by letting a human operator perform obstacle avoidance for many runs. The data was cleaned and used by two different knowledge acquisition techniques to construct decision trees that best explained the dataset.

Bratko, Urbançiç, and Sammut publish their analysis of BC methods and their results and problems in [10]. According to their analysis, earlier BC applications have issues with brittleness and interpretability of learned controllers. Bain and Sammut attempt to address both of these issues in [5], where they conjecture that the fact BC methods are able to learn without the notion of the ultimate goal of the demonstrated policy, leads to the lack of robustness of its solutions. Therefore, their method learns, in addition to a state-action mapping, goals to assist in dealing with previously unseen states as well as interpretability. To increase interpretability even more, high level features are learned and the policy has the form of a decision tree with if-then rules. The method is again used for a simulated flight-control application. The goal of interpretable and robust BC methods is further pursued by Isaac and Sammut in [38], where they opt for combining PID controllers with their goal-directed hierarchy. Joining the robustness of traditional control algorithms with interpretability.

4.2.3. Imitation learning

In [66], Ross and Bagnell note that most imitation learning (IL) methods simply apply supervised learning methods to mimic the actions the teacher takes at each state, implicitly assuming i.i.d. states. They further note that this assumption is actually violated, as the state distribution over which the training takes place is affected by the learned policy, leading to the problem of compounding errors. To mitigate this problem, they propose the forward training algorithm. Each iteration it learns a different policy based on the previous one and a query of the teacher, over the state distribution caused by the previously learned policy. This iteratively changes the training distribution from one caused by the teacher, to one induced by the student.

The forward training algorithm learns a policy for each time step and is therefore non-stationary. The second algorithm presented in [66], stochastic mixing iterative learning (SMILe), learns a stationary policy to better be able to handle infinite time horizons. Instead of learning a different policy each time step, SMILe constructs a stochastic policy which mixes each policy in way that ensures the new one is chosen with a small probability. Again, the initial policy is simply querying the teacher, but as the number of iterations increases to infinity the probability of choosing the teacher's policy goes to zero. Also, at any time step the learned policy can be normalized so the reliance on the teacher is removed.

Instead of learning a stochastic policy, the dataset aggregation (DAgger) algorithm in [67], by Ross et al., obtains a deterministic one. Each iteration a distribution-generating policy is formed based on a mix of the previous one and the teacher. This new dataset is added to the preexisting one, which is used to train a new policy based on the actions of the teacher in those states. In this way each new policy is trained on the entire state distribution encountered over all previous iterations.

The fact that these IL algorithms do not require the notion of a reward function to be able to find policies is usually viewed as a feature. However, when a reward function is readily available, it would be unwise to not incorporate that information into the learning process; the aggregate values to imitate (AggeVaTe) algorithm does exactly that. Based on DAgger, this method does not try to minimize the classification loss of the policy w.r.t. actions taken by the teacher, but the cost-to-go when first following the learned policy while switching to the teacher at a random time step.

The aforementioned methods are so-called active LfD methods and require extensive interaction with the teacher, something which is not feasible in cases where the teacher is human[39]. To remedy this problem Judah et al. propose a method which tries to minimize the amount of teacher queries necessary for learning a policy; reduction-based active imitation learning (RAIL). This method learns a non-stationary policy each

iteration by querying the teacher at all states contained in the distribution under the previous policy. The practical version of this algorithm; RAIL-DW, where DW stands for density weighted, queries the teacher for only one state per iteration. This sample is taken from a state distribution which is the result of combining the the one under the new policy with a Bayesian prior formed from the distributions in previous iterations. The single queried state is determined using a density-weighted learning algorithms to identify queries which would provide the most useful information.

Different from the active methods mentioned before, Ho and Ermon apply generative adversarial learning to imitate demonstrations in [31]. Their method; generalized adversarial imitation learning (GAIL), pits two ANNs against each other; the generator learns a policy which mimics the state-action distribution of the demonstrations, while the discriminator learns to differentiate between actual demonstrations and the student. In this interaction "the discriminator network can be interpreted as a local cost function providing learning signal to the policy"[31]. Both networks are iteratively improved using policy gradient methods until the policy matches the state-action distribution from the demonstrations and the discriminator cannot differentiate between the two. Additionally, Ho and Ermon draw a connection between IL as a combination of inverse reinforcement learning (IRL) and RL, and GAIL. Experiments on several MuJoCo environments show that GAIL achieves the teacher's score on almost all tasks, performing well even with a small amount of demonstrations. While efficient in the amount of demonstrations, GAIL does require a large amount of environment interaction, leading to a training time equivalent to learning the task from reinforcement using TRPO.

4.2.4. Applications in reinforcement learning

The combination of RL and LfD has been studied mainly in the context of deriving a reward function from the behavior of a teacher; IRL. The roots of these methods lie in research by Kalman, in [41] he introduces the inverse optimal control problem; finding a performance criterion for which the control law is optimal. The problem is constrained to constant linear plants and control laws, with a quadratic cost function and a single control variable, as well as full-state feedback.

Early work by Ng and Russell attempts to solve a less-constrained version of this problem in the MDP framework. In [60] they first investigate the case where the teacher's policy is completely known and states and actions are discretized. The solution to the problem is not unique and degenerate, which is solved by adding certain heuristics. The authors posit some characteristics of a good reward solution; it maximizes the cost of a one-step deviation from the policy and generally has small reinforcement values. The latter is accomplished by adding a weight-decay term based on the ℓ_1-norm of the reinforcement values. In the case of continuous states and actions the reward function is assumed to be a linear function of a set of features, which are predefined by the authors. In the more realistic case of not knowing the exact teacher's policy, the assumption is made that the teacher is available to make roll-outs in the MDP from any state required, this is used to obtain feature expectations under the teacher's policy. Linear programming is used to find multiple reward functions which the teacher's policy optimizes, which are then used to train the student's policy. Experiments using the mountain car problem show the student is able to learn a policy which solves the initial task equally good as the teacher in most cases.

In [2], Abbeel and Ng use IRL explicitly to imitate a teacher. Again, the reward function is assumed to be a linear function of predefined features and the teacher is available to generate feature expectations via demonstrations. In this research the reward weight vector is constrained to a ℓ_2-norm of at most 1. The goal of the method is to find a policy with the same feature expectation as the teacher's. First, a method akin to support vector machine (SVM) is proposed, but a simplified algorithm which does not require a quadratic program solver is also presented, which uses a projection method. After each weight vector update a policy is learned based on the resulting reward function, the feature expectations of this new policy are used in the next iteration of the algorithm.

Another way of dealing with the non-uniqueness of reward functions which are optimized by a given policy is presented by Ziebart et al. in [90]. To resolve the ambiguity present in the problem they propose the use of a solution that maximizes entropy. Their algorithm finds a stochastic policy which matches the feature expectations of the teacher in a way that minimizes any further preferences over trajectories.

The aforementioned methods require intensive access to the MDP to obtain feature expectations, the methods in [2, 60] even find an optimal policy corresponding to an MDP each iteration. In contrast, the algorithm presented in [9] is more efficient w.r.t. the teacher's resources. The method finds a policy that minimizes the relative entropy between the distribution over demonstrated trajectories and its own using SGD. Each iteration the current learned policy is used to generate several trajectories, while importance

sampling is used to estimate the gradient of the learned- w.r.t. the teacher's policy.

Different from the methods mentioned up to now, the algorithm described in [51] by Levine et al. makes no assumption about the linearity of the reward w.r.t the features. Instead, the reward function is modeled as a Gaussian process which can express nonlinear functions of features.

In [84], Wulfmeier et al. apply ANNs to remove the need for predefined features, as used by most IRL algorithms. A fully-connected convolutional ANN is applied in combination with the maximum entropy IRL algorithm by Ziebart et al. [90]. The method uses end-to-end learning of a reward function from the raw input state, based on the gradient of the maximum entropy loss.

A method called approximate policy iteration with demonstration (APID), combines LfD and RL by using demonstrations to provide linear constrains for an approximate policy iteration (API) optimization method[42]. Utilizing both demonstrations and interaction, APID is able to find good policies even with very few demonstrations and is shown to outperform API, supervised learning, and the DAgger algorithm on several tasks.

In [32], Ho et al. combine TRPO with demonstrations in a method named IM-TRPO to obtain parameterized stochastic policies. The optimization objective requires a measure of the reward gained by the learned and teacher policy, however using importance sampling the need for extensive policy evaluation is avoided. Trajectories of the current policy are generated, but only used to approximate the KL-divergence between the old and new policy, which requires less samples.

An interesting family of methods named GPS, originally devised by Levine and Koltun, combines model-based trajectory optimization in a low-dimensional state-space with supervised learning to efficiently train high-dimensional policies in continuous-state and action RL tasks. These methods appear particularly promising and therefore they are explained in detail in the coming chapter.

5

Guided Policy Search

In section 4.2.4 a review of the literature concerning the improvement of sample-efficiency of RL methods for aerospace applications using LfD has been presented. From the many methods explored in this review, The family of methods which is seen as most promising for further investigation is GPS, which will be investigated in a more in-depth way in this chapter. For a history of the GPS family of algorithms, see section 2.A of the scientific paper in part I. In section 5.1, a short analysis that explores how well GPS fits flight control applications is presented. In section 5.2, a rough outline of GPS methods is provided, followed by a more thorough presentation on MD-GPS in section 5.3.

5.1. Motivation

The goal of this thesis is to find a way to improve the sample-efficiency of RL methods for aerospace applications, using LfD. An interesting application lies in the usage of high-dimensional policies, used to cope with a high-dimensional or complex state-space. An obvious use-case would be incorporating vision into a flight-controller, which requires a policy with many parameters and high expressive value; i.e. an ANN. The goal of this section is to explain why GPS methods are a feasible solution to improving the sample-efficiency of RL methods in these kinds of applications.

To go forward we must first go back; in the last decade, there has been a significant resurgence of interest towards ANNs. This arguably started with Hinton and Osindero's paper, showing how multi-layer ANNs could be trained efficiently using their pretraining approach[30]. The ability to train "deep" ANNs lead to the coining of the term "deep learning"; an umbrella term for any machine learning approach in which a ANN was used with more than three layers. Another important milestone was the success of deep learning based speech recognition, becoming the SOTA in the field[29]. The same year Krizhevsky et al. beat the SOTA on the Imagenet database[16] by a large margin, using a deep convolutional neural network (CNN). These advances lead to a host of deep learning based approaches mainly in computer vision, speech recognition, natural language processing, and... RL.

Following the advances in supervised deep learning, the RL research field has incorporated large ANNs as high-dimensional policies, to handle a large amount of state-dimensions (mainly, but not constrained to, images). These have been successful in several challenging simulated environments, as presented in section 3.3. However, the amount of samples needed to train these high-dimensional policies is very large when using RL methods, which makes them infeasible for real-world tasks. This is where GPS comes in; it's raison d'être is to efficiently train high-dimensional policies in optimal control tasks. GPS has been proven to work on many physical real-world and simulated tasks, a list of applications is presented in appendix A.

Not only should GPS be able to improve sample efficiency in general RL tasks, it should also be a good fit with the unique needs of a flight-control application. This introduces additional requirements, which GPS can meet as well:

- **Continuous state- and action-space**
  As most physical systems, aerospace systems live in continuous state- and action-spaces. Additionally, the dynamics of these systems are generally continuous as well.
  As for the first point, GPS methods naturally handle continuous state- and action-spaces. These methods stem from robotics, and are proven to work on multiple simulated RL environments and physical

robotics tasks as outlined in appendix A. The iLQG trajectory optimization method used in most GPS implementations is actually constrained to continuous states and actions. Additionally, since iLQG assumes local linear dynamics and a quadratic cost function, it lends itself to systems with smooth dynamics. As for the global policy, it can be chosen to handle continuous states and actions easily. Most GPS methods employ a conditional Gaussian as the global policy (usually a in the form of a ANN), which outputs a mean and covariance of a Gaussian from which an action is sampled.

- **Safety of the agent**
  Aerospace tasks share the unfortunate feature that poor policies can lead to unstable behavior and ultimately a complete loss of the agent. This poses a problem for most RL methods as they rely on trial-and-error to learn.
  The second feature concerns safety of the agent during training and thereafter, important in any application of a method on a real physical aerospace system. The GPS methods proposed up to MD-GPS (so: importance sampled GPS (IS-GPS), adaptive IS-GPS (AIS-GPS), variational GPS (V-GPS), constrained GPS (C-GPS), Bregmann alternating direction method of multipliers (BADMM)-GPS (BADMM-GPS)) do not use on-policy sampling during the training phase. This improves safety because it removes the need for sampling from a poor global policy on a real system and risking crashing the agent. Instead, TVLG controllers are used to sample from, which are less complex and easier to stabilize than a highly nonlinear high-dimensional global policy. However, as stated in [89], in the case of an inaccurate dynamic model or fitted dynamics (as in iLQG-FLM), the offline trajectory optimization performed by iLQG does not guarantee stable behavior. A proposed solution is to employ MPC online to follow the generated trajectories, even in the case of inaccurate models. As for more contemporary GPS methods, which are often based on MD-GPS, on-policy sampling is needed and in fact is shown to improve performance as it allows for stochastic initial states. However, in [13] the authors recognize the fact that on-policy sampling in the initial stages of training is unsafe, and off-policy sampling in the style of prior GPS methods is used instead. A different way GPS methods can improve safety is by using the generative motor reflexes (GMR)-based global policy as presented in [20]. The improvement is twofold; first, the global policy inherits parts of the stabilizing properties of linear controllers. Secondly, the policies are more robust to state disturbance and unseen states, improving generalization capacity.

- **Robustness to disturbance and unseen states**
  Optimized policies need to be able to handle disturbance and parts of the state-space outside of the (limited) training data, since it is not feasible to train over the entire state-space.
  The points about safety also tie into the third feature; robustness and generalization. Robustness and generalization of GPS methods is a function of the amount and variety of training trajectories, as well as the ability of the global policy parameterization (usually an ANN) to generalize to unseen states. As stated in [20], the policies found by these methods often perform well only near the set of training trajectories and fail to generalize. This means that a robust policy would require a large amount of training trajectories which span most of the state-space, which is undesirable and infeasible in high-dimensional state-spaces. Also, Ennen et al. state: "Usually, the MDGPS policy starts to oscillate even by small state disturbances.". Again, this can be solved by forcing the global policy to be a TVLG controller, leading to improved robustness and generalization when compared to a conditional Gaussian policy[20].

5.2. Rough outline

In this section the general structure of GPS algorithms is explained, followed by a more thorough investigation of the MD-GPS algorithm later.

As mentioned by Levine and Koltun in their seminal paper on GPS, direct policy search methods appear attractive to be applied in high-dimensional domains, but 1) require an inordinate amount of samples to train high-dimensional, flexible policies such as ANNs, and 2) are prone to bad local optima[48]. The authors argue this is due to the random exploration strategies utilized by these methods, which do not scale to high-dimensional state- and action- spaces, or expressive policies. Instead, GPS applies (model-based) trajectory-optimization in a lower dimensional state-space, which finds regions of high reward much more efficiently. These trajectories are subsequently used to train a complex and expressive global policy using supervised learning techniques. This policy is conditioned not on the low-dimensional state, but on an observation, which is generally of much higher dimension. Examples of observations are images, but in principle any

source of information can be used. In summary, GPS instead divides direct policy search up into two alternating GPS steps[56];

1. **Control-phase** (or C-step)
   During this phase trajectories are generated using a trajectory optimization algorithm, which optimizes not only the cost, but also forces the distribution of actions conditioned on the state $p(\boldsymbol{u}|\boldsymbol{x})$ (called local- or sub-policies) to be similar to the global policy $\pi_\theta(\boldsymbol{u}|\boldsymbol{o})$, which is being trained. This is achieved by either including a second cost term, or constrain the optimization using some measure of similarity. The trajectories are recorded as sets of state, observation, action triples, needed in the second step.

2. **Supervised-phase** (or S-step)
   During this phase the global policy $\pi_\theta(\boldsymbol{u}|\boldsymbol{o})$ is trained to match the action distributions of the sub-policies $p(\boldsymbol{u}|\boldsymbol{x})$, which have been recorded during the C-step, in a supervised way.

The fact that during the C-step trajectories are not optimized w.r.t. cost, but also similarity to the global policy is an important aspect of GPS. By adapting the trajectories to the global policy, limitations of it's parameterization and available information are taken into account. Compare this to an imitation learning method like DAgger[67], which assumes that for any teacher, the policy is able to achieve bounded error[56]. Obviously, this assumption does not hold in general, an example being the case of a piano teacher and his one-armed student. In this case the "parameterization" of the student prevents him to always be able to imitate his teacher with bounded error. Instead, the teacher should account for the difference in capability and show the students ways in which some piece of music can be played with one hand, or when that fails switch to pieces which are suitable to be played one-handed. The same principles apply when the student cannot differentiate some states from each other, a problem of observability. Both cases are accounted for by optimizing trajectories not only w.r.t. cost, but also similarity to the global policy.

As explained before, trajectory optimization takes place in a state-space which is generally of much lower dimension than the observation-space. This allows the sub-policies to be much simpler than the global policy, and it makes their optimization significantly more efficient than optimization in the observation-space directly. Additionally, the state-space can be chosen such that it is much more amenable to model-based optimization methods, which improves the efficiency of exploration substantially.

5.3. Mirror Descent Guided Policy Search

The specific GPS algorithm which will be used during this research is MD-GPS, using iLQG-FLM for trajectory optimization under unknown dynamics, while utilizing a nonlinear global policy, from [56]. In this section the algorithm will be explained thoroughly, starting by a summary of important symbols and notation in table 5.1, and an outline of the algorithm in algorithm 1. Although part I already contains an in-depth look at the MD-GPS algorithm, the explanation in this section of the thesis goes a bit deeper still.

Algorithm 1 Mirror descent guided policy search (MD-GPS): unknown dynamics, nonlinear global policy[56].

for $k = 1$ to K **do**
  Generate samples $\mathcal{D}_i = \{\boldsymbol{\tau}_{i,j}\}$ by running either $p_i(\boldsymbol{u}|\boldsymbol{x})$ or $\pi_{\theta,i}(\boldsymbol{u}|\boldsymbol{o})$
  Fit linear-Gaussian dynamics $p_i(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ using samples in $\mathcal{D}_i$
  Fit linearized global policy $\bar{\pi}_{\theta,i}(\boldsymbol{u}|\boldsymbol{o})$ using samples in $\mathcal{D}_i$
  C-step: $p_i \leftarrow \arg\min_{p_i} \mathrm{E}_{p_i(\tau)}\left[\ell(\boldsymbol{\tau})\right]$ such that $D_{KL}(p_i(\boldsymbol{\tau}) \,||\, \bar{\pi}_{\theta,i}(\boldsymbol{\tau})) \leq \epsilon$
  S-step: $\pi_\theta \leftarrow \arg\min_{\boldsymbol{\theta}} \sum_{t,i,j} D_{KL}(\pi_{\theta,i}(\boldsymbol{u}_t|\boldsymbol{x}_{t,i,j}) \,||\, p_i(\boldsymbol{u}_t|\boldsymbol{x}_{t,i,j}))$ via supervised learning
  Adapt ϵ
end for

Table 5.1: A summary of important symbols and notation used in the explanation of MD-GPS under unknown dynamics, utilizing a nonlinear global policy. Heavily inspired by [52].

Symbol	Explanation	Examples
$\boldsymbol{x}_t$	Markovian state vector at time $t \in [0, T-1]$	3-dimensional position, velocity, attitude, and angular rate vectors.
$\boldsymbol{o}_t$	Observation vector at time $t \in [0, T-1]$	Camera images, laser range measurements.
$\boldsymbol{u}_t$	Action vector at time $t \in [0, T-1]$	Motor torques, aileron/elevator deflection.
$\boldsymbol{\tau}$	Trajectory: $\boldsymbol{\tau} = (\{\boldsymbol{x}_t, \boldsymbol{o}_t, \boldsymbol{u}_t\}, \cdots, \boldsymbol{u}_{T-1}\})$, inclusion of observations depends on the context	Notational shorthand for a sequence of states, observations, and actions.
$\ell(\boldsymbol{\tau})$	The cost of a trajectory	Notational shorthand for cost functions over trajectories.
$p(\boldsymbol{x}_{t+1}\|\boldsymbol{x}_t, \boldsymbol{u}_t)$	Unknown system dynamics	System dynamics of a drone, or other aircraft.
$p_i(\boldsymbol{u}_t\|\boldsymbol{x}_t)$	Action distribution of the i^{th} trajectory distribution, also called the i^{th} sub-policy, which starts at $\boldsymbol{x}_0^i \sim p(\boldsymbol{x}_0)$	A TVLG controller, optimized by iLQG.
$p_i(\boldsymbol{\tau})$	The i^{th} trajectory distribution, induced by $p_i(\boldsymbol{u}_t\|\boldsymbol{x}_t)$ when starting at $\boldsymbol{x}_0^i$	Notational shorthand for a trajectory distribution induced by a sub-policy.
$\boldsymbol{\tau}_i^j$	The j^{th} realization of $p_i(\boldsymbol{\tau})$	Each sub-policy is used to generate multiple trajectories.
$p_i(\boldsymbol{x}_{t+1}\|\boldsymbol{x}_t, \boldsymbol{u}_t)$	Linearized system dynamics around $p_i(\boldsymbol{\tau})$, in the form of a linear-Gaussian model	Linearized aircraft dynamics around a nominal trajectory.
$\pi_\theta(\boldsymbol{u}_t\|\boldsymbol{o}_t)$	Global policy, action distribution conditioned on the observation, parameterized by weight vector $\boldsymbol{\theta}$	An ANN, which outputs a mean and covariance of a Gaussian distribution from which an action is sampled.
$\pi_{\theta,i}(\boldsymbol{\tau})$	The trajectory distribution induced by the global policy, while starting at $\boldsymbol{x}_0^i$	Notational shorthand for a trajectory distribution induced by the global policy.
$\bar{\pi}_{\theta,i}(\boldsymbol{\tau})$	The trajectory distribution induced by a policy which is the result of linearizing the global policy around $p_i(\boldsymbol{\tau})$, or $\pi_{\theta,i}(\boldsymbol{\tau})$, while starting at $\boldsymbol{x}_0^i$	Notational shorthand for a trajectory distribution induced by a linearized global policy.
$\mathscr{D}_i$	Set of sampled trajectories $\{\boldsymbol{\tau}_{i,j}\}$	A set of trajectories sampled on the quadcopter system
ϵ	KL-divergence step size	N/A

In fig. 5.1, a stylized representation of the MD-GPS is shown. The steps 1 to 5 are explained in further detail below:

1. Sample trajectories on the real system, using either the previous sub-policy or the global policy.
   A quadcopter is used to generate trajectories, controlled by a sub-policy (open-loop) or the global-policy (closed-loop). States (positions, velocities), inputs (rotor speed or voltage), and observations (camera images) will be logged at each time step. States can be measured on-line by motion-capture systems or on-board sensors, but are not required by the controller during flight and therefore allow off-line processing. However, when the global-policy is used to control the quadcopter, the observations are needed inside the control loop.

2. Fit TVLG model of the dynamics, around the sampled trajectories.
   The state-action-state tuples contained in the sampled quadcopter trajectories are used to fit a model of the quadcopter dynamics. This model will be used by the model-based trajectory optimization, instead of sampling the real system.

3. Fit TVLG model of the global policy, around the sampled trajectories.

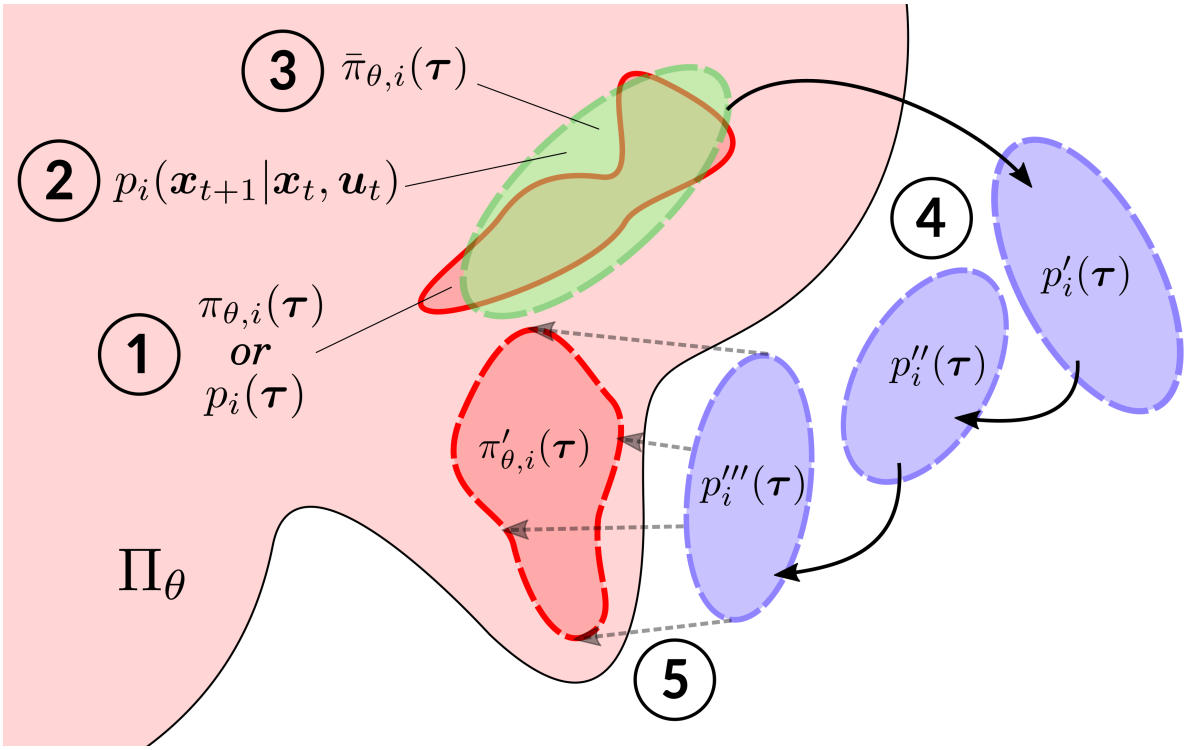


Figure 5.1: Stylized representation of the MD-GPS algorithm, corresponding to algorithm 1. Trajectories sampled on the real system have a solid outline, Gaussian approximations are dashed and are ellipse-shaped. The light red area represents the constraint manifold of the global policy parameterization Π_Θ. Distances in this space can be roughly interpreted as a KL-divergence, as it is a measure of difference between two probability distributions.

To enable efficient trajectory optimization with a KL-divergence constraint, a locally linearized version of the global-policy is needed.

4. Optimize sub-policies using dual gradient descent (DGD), minimize the surrogate cost using iLQG in combination with the dynamics model, and increment the dual variable to reduce KL-divergence, until convergence.
   Sub-policies are optimized using the iLQG algorithm, using locally modeled dynamics and a locally linearized global-policy. By using a model of the dynamics, trajectory optimziation is done off-line, without further sampling by the quadcopter.

5. Use supervised learning to match the global-policy to the sub-policies. This amounts to an approximate projection onto the constraint manifold Π_Θ.
   The final step is to train the global-policy by supervised learning, which is naturally done off-line using gradient-based optimization.

Step 1: Sample trajectories
The first step is to sample trajectories using the real dynamics of the system to control, obtained by either running the sub-policies or global policy, of the previous iteration. By using multiple different, but consistent, initial states $x_{0,i} \sim p(x_0)$, generalization is improved[56]. The i^{th} initial state will be the starting point of a trajectory distribution from which multiple trajectories are sampled: τ_i^j for $j = [1,2,3,\cdots]$, stored in set $\mathscr{D}_i$. Depending on whether trajectories are sampled by sub-policies or the global policy, the distribution is $p_i(\tau)$ or $\pi_{\theta,i}(\tau)$, respectively. According to Montgomery and Levine, on-policy sampling reduces learning speed as the global policy lags behind the sub-policies in performance, due to the approximate projection during the S-step of the algorithm[56]. As mentioned in [57], an advantage of on-policy sampling is that it can be utilized to allow stochastic initial states, which improve generalization and ease of application of these methods. However, this would require additional steps (as presented in [57]), and will not be considered in this work.

Step 2: Approximate dynamics
In this instantiation of MD-GPS, iLQG is used for trajectory optimization, under unknown nonlinear dynamics. This method requires a TVLG model of the dynamics, which is assumed to be unavailable beforehand, and so requires dynamics to be estimated online. Fortunately, iLQG requires only a local dynamics approximation, which allows the use of much easier to fit local dynamics models, provided trajectories do not stray too far from the nominal one. This caveat requires the iLQG objective to include a term that penalizes distance from the nominal trajectory, which is handled by the KL-divergence upper bound ϵ in eq. (5.3).

Dynamics are approximated around each set of trajectories $\mathscr{D}_i$ separately, which have been sampled in the previous step. The dynamics models take the form of TVLGs:

$$p_i(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,\boldsymbol{u}_t) = \mathscr{N}(\boldsymbol{f}_{xt}\boldsymbol{x}_t + \boldsymbol{f}_{ut}\boldsymbol{u}_t + \boldsymbol{f}_{ct}, \boldsymbol{F}_t) \tag{5.1}$$

Matrices $\boldsymbol{f}_{xt}$, $\boldsymbol{f}_{ut}$, $\boldsymbol{f}_{ct}$, and $\boldsymbol{F}_t$ will be estimated by fitting Gaussian distributions to the state transitions $(\boldsymbol{x}_t^{i,j}, \boldsymbol{u}_t^{i,j}, \boldsymbol{x}_{t+1}^{i,j})$ at every time step, using multiple samples of the trajectory distribution: $\boldsymbol{\tau}_i^j \in \mathscr{D}_i$, and then conditioning the distributions to obtain $p_i(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,\boldsymbol{u}_t)$.

This approach amounts to linear regression, the complexity of which grows linearly with the number of states, which causes the amount of sampled trajectories to grow quite large for more complex systems. To combat this issue, a global model is fitted alongside the local dynamics, which will act as a prior and reduce the number of samples needed.

Firstly, in the case of a multivariate Gaussian distribution, a normal-inverse-Wishart distribution can be used to combine prior- with new information:

$$\Sigma = \frac{\boldsymbol{\Phi} + N\hat{\boldsymbol{\Sigma}} + \frac{Nm}{N+m}(\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}_0)(\hat{\boldsymbol{\mu}} - \boldsymbol{\mu}_0)^T}{N + n_0} \quad \boldsymbol{\mu} = \frac{m\boldsymbol{\mu}_0 + n_0\hat{\boldsymbol{\mu}}}{m + n_0} \quad \boldsymbol{\Phi} = n_0\bar{\boldsymbol{\Sigma}} \quad \boldsymbol{\mu}_0 = \bar{\boldsymbol{\mu}} \tag{5.2}$$

Where, respectively, $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ are the sample mean and variance of the Gaussian distributions which have been fitted to the newly sampled vectors $\left[\boldsymbol{x}_t;\boldsymbol{u}_t;\boldsymbol{x}_{t+1}\right]_i$. Where, for the i^{th} trajectory distribution, there is one distribution per time step, which has been fitted using multiple trajectory samples. Additionally, n_0 is the amount of new data points over which the dynamics are being fitted, m is the number of points over which the prior mean $\boldsymbol{\mu}_0$ has been determined, and N is the number of points over which the prior covariance $\boldsymbol{\Phi}$ has been determined (in this case equal to m).

Secondly, a GMM is constructed over vectors $[\boldsymbol{x}_t;\boldsymbol{u}_t;\boldsymbol{x}_{t+1}]$, modeling the transition tuples as coming from a mixture model with hidden state $\boldsymbol{h}$. The elements of $\boldsymbol{h}$ encode the membership of a transition tuple to a particular conditional linear-Gaussian distribution (or mixture element), which corresponds to a stochastic linear dynamics model. By using the membership fractions in $\boldsymbol{h}$ a weighted average over the means and covariances of all mixture elements is used to determine $\bar{\boldsymbol{\Sigma}}$, and $\bar{\boldsymbol{\mu}}$, to finally calculate $\boldsymbol{\Sigma}$, and $\boldsymbol{\mu}$ using eq. (5.2). The Gaussian distribution with this mean and covariance is then conditioned on $[\boldsymbol{x}_t;\boldsymbol{u}_t]$ to obtain $p_i(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t\boldsymbol{u}_t)$.

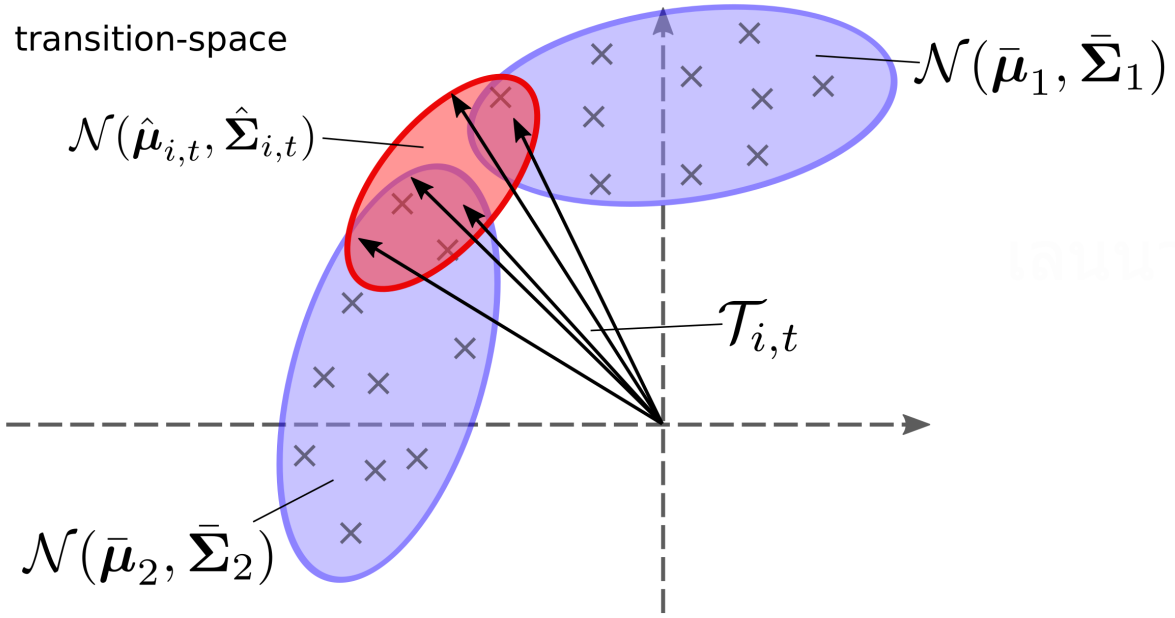
Step 3: Linearize global policy
The objective optimized during the C-step, see line 5 of algorithm 1, includes a KL-divergence constraint of the sub-policies w.r.t. the global policy. This leads to a surrogate cost function for the iLQG trajectory optimization in the next step, as presented in eq. (5.14). Needed for the surrogate cost is the logarithm of the action distribution, induced by the global policy $\log\pi_{\theta,i}(\boldsymbol{u}_t|\boldsymbol{x}_t)$, which will be approximated by its second order Taylor expansion (since iLQG approximates cost up to second order). In the case of highly nonlinear global policies, this approximation can be quite jagged during the beginning of the learning procedure[49], and prevent convergence[50]. To combat this, a TVLG approximation of the global policy, denoted $\bar{\pi}_{\theta,i}$, will be fitted for each sub-policy, in exactly the same way as for the dynamics.

Step 4: C-step
At this point the trajectories have been sampled, the dynamics fitted, and the global policy approximated, the next step is the optimization of the sub-policies; the C-step. The objective of this optimization is shown on line 5 of algorithm 1 and will be repeated here for convenience:

$$p_i \leftarrow \arg\min_{p_i} \mathrm{E}_{p_i(\tau)}\left[\ell(\boldsymbol{\tau})\right] \quad \text{s.t.} \quad D_{KL}(p_i(\boldsymbol{\tau}) \,||\, \bar{\pi}_{\theta,i}(\boldsymbol{\tau})) \leq \epsilon \tag{5.3}$$

In concordance with prior work, the sub-policies are chosen to be TVLG controllers, shown in eq. (5.16), which allow for efficient optimization using the iLQG algorithm[76].

Figure 5.2: A 2-D representation of an example transition-space, in which two GMM mixture elements reside. The red ellipse is the Gaussian distribution $\mathcal{N}(\hat{\boldsymbol{\mu}}_{i,t}, \hat{\boldsymbol{\Sigma}}_{i,t})$ fitted to the transitions of the trajectories in $\mathcal{D}_i$, at time step t; denoted by $\mathcal{T}_{i,t}$, represented by the black vectors. The two mixture elements are represented by the blue ellipses, and have been fitted using previous transition samples, represented by the grey crosses filling the space. The set $\mathcal{T}_{i,t}$ is assigned a hidden state $\boldsymbol{h}$, which contains membership fractions for both mixture elements, which are used as weightings to determine $\bar{\boldsymbol{\mu}}$ and $\bar{\boldsymbol{\Sigma}}$ for eq. (5.2). In this case the sampled vectors lie approximately between the two mixture elements, leading to the following hidden state: $\boldsymbol{h} \approx [\frac{1}{2}, \frac{1}{2}]^T$.



The general structure of the iLQG algorithm used in this work is presented in algorithm 2. It deviates slightly from normal iLQG, since there is no global model of the dynamics and sampling on the real system does not take place during the trajectory optimization. Only a local approximation of the dynamics is available, which is reused each iLQG iteration. To maintain model accuracy, trajectories are kept close to the nominal by inclusion of a KL-divergence term in the objective. Additionally, the forward pass does not take place on the system itself, but is instead performed using the local dynamics model.

Algorithm 2 Outline of the variant of iLQG used in this work.

Require: Initial nominal trajectory
1: Approximate dynamics around the nominal trajectory
2: **while** not converged **do**
3: Approximate cost around the nominal trajectory
4: Backward pass: determine optimal sub-policy around the nominal trajectory
5: Forward pass: use local dynamics model and the new sub-policy to obtain an open-loop trajectory
6: **end while**

The goal of iLQG is to minimize the expectation of some cost function over a distribution of finite trajectories induced by a sub-policy and corresponding initial state:

$$\min_{p_i} \mathrm{E}_{p_i(\tau)}\big[\ell(\boldsymbol{\tau})\big], \quad \text{where} \quad \ell(\boldsymbol{\tau}) = \sum_{t=0}^{T-1} \ell(\boldsymbol{x}_t, \boldsymbol{u}_t), \quad \text{and where} \quad \ell(\boldsymbol{x}_t, \boldsymbol{u}_t) = c(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{5.4}$$

Where, for now, $c(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is the usual cost/reward function used in optimal control and RL problems.

At each time step, using the system dynamics and cost function approximations, a linear quadratic Gaussian (LQG) controller is optimized. It is a method similar to differential dynamic programming (DDP), which approximates both up to second order, but only includes the linear term in the dynamics approximation instead.

In [27], iLQG is explained as follows: given a nominal trajectory:

$$\bar{\boldsymbol{\tau}} = \big(\{\bar{\boldsymbol{x}}_0, \bar{\boldsymbol{u}}_0\}, \{\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{u}}_1\}, \cdots, \{\bar{\boldsymbol{x}}_{T-1}, \bar{\boldsymbol{u}}_{T-1}\}\big) \tag{5.5}$$

define deviations as: $\hat{\boldsymbol{x}}_t = \boldsymbol{x}_t - \bar{\boldsymbol{x}}_t$, and $\hat{\boldsymbol{u}}_t = \boldsymbol{u}_t - \bar{\boldsymbol{u}}_t$. The deviation's dynamics and cost function are approximated as follows:

$$\begin{aligned} \tilde{\boldsymbol{x}}_{t+1} &= \boldsymbol{f}_{ct} + \begin{bmatrix} \boldsymbol{f}_{xt} \\ \boldsymbol{f}_{ut} \end{bmatrix}^T \begin{bmatrix} \hat{\boldsymbol{x}}_t \\ \hat{\boldsymbol{u}}_t \end{bmatrix} \\ \tilde{\ell}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) &= \bar{\ell} + \begin{bmatrix} \boldsymbol{\ell}_{xt} \\ \boldsymbol{\ell}_{ut} \end{bmatrix}^T \begin{bmatrix} \hat{\boldsymbol{x}}_t \\ \hat{\boldsymbol{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \boldsymbol{\ell}_{xxt} & \boldsymbol{\ell}_{xut} \\ \boldsymbol{\ell}_{uxt} & \boldsymbol{\ell}_{uut} \end{bmatrix} \end{aligned} \tag{5.6}$$

Where subscripts w.r.t. $\boldsymbol{x}$ and $\boldsymbol{u}$ denote Jacobians and Hessians, which can be obtained in multiple ways; finite differences, automatic differentiation, or analytical methods. It has already been explained how the dynamics are approximated in section 5.3. Using the method explained there, the terms $\boldsymbol{f}_{ct}$, $\boldsymbol{f}_{xt}$, and $\boldsymbol{f}_{ut}$ are obtained. As for the cost approximation, it is obtained using finite differences.

The state- and action-value functions are approximated up to second order and are defined :

$$V(\hat{\boldsymbol{x}}) = \bar{V} + \boldsymbol{V}_x^T \hat{\boldsymbol{x}} + \frac{1}{2} \hat{\boldsymbol{x}}^T \boldsymbol{V}_{xx} \hat{\boldsymbol{x}} \tag{5.7}$$

$$Q(\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}) = \bar{Q} + \begin{bmatrix} \boldsymbol{Q}_x \\ \boldsymbol{Q}_u \end{bmatrix}^T \begin{bmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{u}} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{u}} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{Q}_{xx} & \boldsymbol{Q}_{xu} \\ \boldsymbol{Q}_{ux} & \boldsymbol{Q}_{uu} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{u}} \end{bmatrix} \tag{5.8}$$

By starting at the final state and iterating backwards through time, each component of the state- and action-value functions is computed:

$$\begin{aligned} \boldsymbol{Q}_{xxt} &= \boldsymbol{\ell}_{xxt} + \boldsymbol{f}_{xt}^T \boldsymbol{V}_{xxt+1} \boldsymbol{f}_{xt} & \boldsymbol{Q}_{xt} &= \boldsymbol{\ell}_{xt} + \boldsymbol{f}_{xt}^T \boldsymbol{V}_{xt+1} \\ \boldsymbol{Q}_{uut} &= \boldsymbol{\ell}_{uut} + \boldsymbol{f}_{ut}^T \boldsymbol{V}_{xxt+1} \boldsymbol{f}_{ut} & \boldsymbol{Q}_{ut} &= \boldsymbol{\ell}_{ut} + \boldsymbol{f}_{ut}^T \boldsymbol{V}_{xt+1} \\ \boldsymbol{Q}_{uxt} &= \boldsymbol{\ell}_{uxt} + \boldsymbol{f}_{ut}^T \boldsymbol{V}_{xxt+1} \boldsymbol{f}_{xt} \end{aligned} \tag{5.9}$$

$$\begin{aligned} \boldsymbol{V}_{xt} &= \boldsymbol{Q}_{xt} - \boldsymbol{Q}_{uxt}^T \boldsymbol{Q}_{uut}^{-1} \boldsymbol{Q}_u \\ \boldsymbol{V}_{xxt} &= \boldsymbol{Q}_{xxt} - \boldsymbol{Q}_{uxt}^T \boldsymbol{Q}_{uut}^{-1} \boldsymbol{Q}_{ux} \end{aligned} \tag{5.10}$$

When optimizing the objective shown in eq. (5.4), the locally optimal sub-policy can then be described by the following time-varying linear controller:

$$g(\boldsymbol{x}_t) = \bar{\boldsymbol{u}}_t + \boldsymbol{K}_t (\boldsymbol{x}_t - \bar{\boldsymbol{x}}_t) + \boldsymbol{k}_t \tag{5.11}$$

Where:

$$\boldsymbol{K}_t = -\boldsymbol{Q}_{uut}^{-1} \boldsymbol{Q}_{uxt} \qquad \boldsymbol{k}_t = -\boldsymbol{Q}_{uut}^{-1} \boldsymbol{Q}_{ut} \tag{5.12}$$

However, the C-step optimizes a slightly different objective from eq. (5.4); it includes a KL-divergence constraint as in eq. (5.3). It turns out this objective can be optimized using iLQG as well. To this end, the constrained problem is transformed to its Lagrangian:

$$\begin{aligned} \mathscr{L}(p_i, \eta) &= \mathrm{E}_{p_i(\tau)} \big[\ell(\boldsymbol{\tau}) \big] + \eta (D_{KL}(p_i(\boldsymbol{\tau}) \,||\, \bar{\pi}_{\theta i}(\boldsymbol{\tau})) - \epsilon) \\ &= \sum_{t=0}^{T-1} \mathrm{E}_{p_i(x_t, u_t)} \big[\ell(\boldsymbol{x}_t, \boldsymbol{u}_t) - \eta \log \bar{\pi}_{\theta i}(\boldsymbol{u}_t | \boldsymbol{x}_t) \big] - \eta \mathscr{H}(p(\boldsymbol{u}_t | \boldsymbol{x}_t)) - \eta \epsilon \end{aligned} \tag{5.13}$$

Where the second line follows from the assumption of conditional Gaussian policies, identical TVLG dynamics, and identical initial states, as in [47, 56]. This Lagrangian can be optimized by changing the cost function to the following:

$$\ell(\boldsymbol{x}_t, \boldsymbol{u}_t) = \frac{1}{\eta} c(\boldsymbol{x}_t, \boldsymbol{u}_t) - \log \bar{\pi}_{\theta i}(\boldsymbol{u}_t | \boldsymbol{x}_t) \tag{5.14}$$

When the deterministic sub-policy in eq. (5.11) is adapted to be a TVLG, by using the same feedback and feedforward terms and by setting its covariance to:

$$\boldsymbol{C}_{t,i} = -\boldsymbol{Q}_{uut}^{-1} \tag{5.15}$$

The optimal sub-policy becomes a TVLG controller, which stabilizes the state around the nominal trajectory:

$$p_i(\boldsymbol{u}_t | \boldsymbol{x}_t) = \mathscr{N}(\bar{\boldsymbol{u}}_{t,i} + \boldsymbol{K}_{t,i} (\boldsymbol{x}_t - \bar{\boldsymbol{x}}_{t,i}) + \boldsymbol{k}_{t,i}, \boldsymbol{C}_{t,i}) \tag{5.16}$$

This way iLQG optimizes the following objective:

$$p_i = \arg\min_{p_i} \mathrm{E}_{p_i(x_t,u_t)}\big[\ell(\boldsymbol{x}_t,\boldsymbol{u}_t)\big] - \mathscr{H}(p_i(\boldsymbol{u}_t|\boldsymbol{x}_t)) \tag{5.17}$$

The Lagrangian is optimized using DGD, iteratively optimizing p_i using iLQG and incrementing the dual variable η. Since the dual function is convex and there is only one dual variable, a bracketed quadratic-fit line-search method can efficiently find the optimum[56]. To further speed up convergence, the DGD optimization is stopped when the KL-divergence is within 10% of the constraint, and the line-search is performed in log-space[47].

To update η, the constraint violation is estimated, which requires the KL-divergence over trajectories in eq. (5.3). In the case of identical initial states and dynamics, as well as conditional Gaussian policies, the divergence reduces to an operation over the policies directly[47, 56], and the KL-divergence can be calculated as follows:

$$\begin{aligned} D_{KL}(p_i(\boldsymbol{\tau}) \,||\, \pi_{\theta,i}(\boldsymbol{\tau})) = \sum_{t,i,j} D_{KL}(p_i(\boldsymbol{u}_t|\boldsymbol{x}_t) \,||\, \pi_\theta(\boldsymbol{u}_t|\boldsymbol{x}_t)) = \cdots \\ \sum_{t,i,j} \mathrm{E}_{p_i(\boldsymbol{x}_t,\boldsymbol{o}_t)}\Big[\mathrm{Tr}[\Sigma^\pi(\boldsymbol{o}_t)^{-1}\boldsymbol{C}_{ti}] - \log|\boldsymbol{C}_{ti}| + \cdots \\ \big(\mu^p_{ti}(\boldsymbol{x}_t) - \mu^\pi(\boldsymbol{o}_t)\big)^T \Sigma^\pi(\boldsymbol{o}_t)^{-1}\big(\mu^p_{ti}(\boldsymbol{x}_t) - \mu^\pi(\boldsymbol{o}_t)\big)\Big] \end{aligned} \tag{5.18}$$

Where $\mu^\pi(\boldsymbol{o}_t)$ and $\Sigma^\pi(\boldsymbol{o}_t)$ are the global policy mean and covariance, and $\mu^p_{ti}(\boldsymbol{x}_t)$ and $\boldsymbol{C}_{ti}$ are the sub-policy mean and covariance. The expectation over $p_i(\boldsymbol{x}_t,\boldsymbol{o}_t)$ can either be approximated by using a TVLG model of the global model[56], or by estimating the mean and covariance of the global policy by MC sampling[50].

The parameter ϵ in eq. (5.3) places a constraint on the maximum difference between the trajectory distributions of the sub-policy and the previous global policy (linearized around sampled trajectories). It serves two purposes; firstly, while sub-policies seek out low cost trajectories, it ensures their behavior is somewhat reproducible by the global policy. As mentioned in section 5.2, this is important in handling arbitrary global policy parameterizations and observability issues. The second purpose deals with the assumption that no global dynamics model is available and dynamics are estimated locally, around the sampled trajectories. Local models have the benefit of being much easier to fit, with the disadvantage that they are only valid in a small region around the samples. Large steps in trajectory space cause the dynamics model to fail, which is prevented by keeping ϵ sufficiently small.

To ensure fast convergence, the constraint is adapted online using one of two update rules, both based on differences in expected- and measured cost:

$$\ell^k_m = \sum_{t=0}^{T-1} \mathrm{E}_{p^k(x_t,u_t)}\big[\ell(\boldsymbol{x}_t,\boldsymbol{u}_t)\big] \qquad \ell^{k,\pi}_m = \sum_{t=0}^{T-1} \mathrm{E}_{\bar{\pi}^k_\theta(x_t,u_t)}\big[\ell(\boldsymbol{x}_t,\boldsymbol{u}_t)\big] \tag{5.19}$$

Where $p^k(\boldsymbol{x}_t,\boldsymbol{u}_t)$ and $\bar{\pi}^k_\theta(\boldsymbol{x}_t,\boldsymbol{u}_t)$ are the marginals of the local policy and linearized global policy at iteration k, respectively, when subject to dynamics fitted at iteration m. Since the sub-policy, global policy, and dynamics model are all linear Gaussians, the expectations in eq. (5.19) can be efficiently calculated by propagating the Gaussian distributions forward in time, using the procedure presented in appendix B.3 of [56].

The first update rule only takes into account differences due to inaccuracies in the dynamics model and is called the "classic" step size[56]:

$$\epsilon' = \frac{\epsilon}{2} \frac{\ell^k_{k-1} - \ell^{k-1,\pi}_{k-1}}{\ell^{k-1}_k - \ell^k_k} \tag{5.20}$$

While the second rule expands the first by including differences due to an inability of the global policy to reproduce behavior by the sub-policy exactly, called the "global" step size[56], which leads to a more conservative constraint update:

$$\epsilon' = \frac{\epsilon}{2} \frac{\ell^k_{k-1} - \ell^{k-1,\pi}_{k-1}}{\ell^{k-1}_k - \ell^{k,\pi}_k} \tag{5.21}$$

As shown in eq. (5.12) and eq. (5.15), the optimal sub-policy depends on the inverse of Q_{uut}, which is the Hessian of the action-value function w.r.t. the input. Care must therefore be taken to ensure this matrix is invertible, which is done by increasing the dual variable η until Q_{uut} becomes positive definite for all t.

To conclude the explanation of the C-step, it's pseudocode is presented in algorithm 3, explaining every step of this sub-routine.

Algorithm 3 C-step pseudocode for the i^{th} sub-policy.

Require:

- Initial sub-policy $p_i(\boldsymbol{u}|\boldsymbol{x})$
- A set of trajectories $\mathscr{D}_i$ sampled by either $p_i(\boldsymbol{u}|\boldsymbol{x})$ or $\pi_{\theta,i}(\boldsymbol{u}|\boldsymbol{o})$
- A TVLG model of system dynamics, approximated around $\mathscr{D}_i$; $p_i(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t,\boldsymbol{u}_t)$
- A TVLG model of the global policy, approximated around $\mathscr{D}_i$; $\bar{\pi}_{\theta,i}(\boldsymbol{u}|\boldsymbol{x})$
- A KL-divergence step size ϵ, obtained from eq. (5.20), or eq. (5.21)

Initialize $\eta \leftarrow \eta_0$
while not converged **do**
  Set $\ell(\boldsymbol{x},\boldsymbol{u})$ according to eq. (5.14)
  Approximate $\ell(\boldsymbol{x},\boldsymbol{u})$ as in eq. (5.6)
  while not converged **do**
    iLQG backward pass to determine V- and Q-function components as in eqs. (5.9) and (5.10), and local optimal sub-policy as in eq. (5.16)
    Use new sub-policy and the TVLG dynamics model to obtain new open-loop trajectory
  end while
  Update η using a bracketed quadratic-fit line-search
end while
return Sub-policy optimized as in eq. (5.3)

Step 5: S-step
The final step of a MD-GPS iteration is the S- or supervised step, its goal is to optimize the global policy by minimizing the KL-divergence between the trajectory distributions induced by itself and the sub-policies.

By again assuming identical TVLG dynamics, conditional Gaussian policies, and identical initial states, the KL-divergence over trajectories reduces to a divergence over the action distributions of the policies[47, 56]:

$$\pi_\theta \leftarrow \arg\min_{\boldsymbol{\theta}} \sum_{t,i,j} D_{KL}(\pi_{\theta,i}(\boldsymbol{u}_t|\boldsymbol{x}_{t,i,j}) \,||\, p_i(\boldsymbol{u}_t|\boldsymbol{x}_{t,i,j})) \tag{5.22}$$

This step amounts to an approximate projection of the sub-policies on the constraint manifold induced by the parameterization of the global policy. It is approximate because the dynamics used in the trajectory optimization are actually not equal to the dynamics of the real system; the iLQG algorithm uses only a locally fitted TVLG model. This makes MD-GPS an approximate instance of mirror-descent (MD)[56].

As explained previously in eq. (5.18), if the global policy is chosen to be a (nonlinear) conditional Gaussian in the form of:

$$\pi_\theta(\boldsymbol{u}_t|\boldsymbol{o}_t) = \mathscr{N}(\mu^\pi(\boldsymbol{o}_t), \Sigma^\pi(\boldsymbol{o}_t)) \tag{5.23}$$

where μ^π and Σ^π are arbitrary functions (e.g. ANNs) that determine the mean and covariance of the Gaussian distribution, then, in combination with the TVLG sub-policies, there is an elegant equation for the KL-divergence in eq. (5.22), in terms of the means and covariances of the distributions:

$$\begin{aligned}\sum_{t,i,j} D_{KL}(\pi_\theta(\boldsymbol{u}_t|\boldsymbol{x}_t) \,||\, p_i(\boldsymbol{u}_t|\boldsymbol{x}_t)) = \cdots \\ \sum_{t,i,j} \mathrm{E}_{p_i(\boldsymbol{x}_t,\boldsymbol{o}_t)}\Big[\overbrace{\mathrm{Tr}[\boldsymbol{C}_{ti}^{-1}\Sigma^\pi(\boldsymbol{o}_t)] - \log\big|\Sigma^\pi(\boldsymbol{o}_t)\big|}^{\text{covariance terms}} + \cdots \\ \underbrace{\big(\mu^\pi(\boldsymbol{o}_t) - \mu^p_{ti}(\boldsymbol{x}_t)\big)^T \boldsymbol{C}_{ti}^{-1}\big(\mu^\pi(\boldsymbol{o}_t) - \mu^p_{ti}(\boldsymbol{x}_t)\big)}_{\text{mean terms}}\Big]\end{aligned} \tag{5.24}$$

Where $\mu^p_{ti}(\boldsymbol{x}_t)$ and $\boldsymbol{C}_{ti}$ are the mean and covariance matrix of the i^{th} sub-policy, at time step t (see eq. (5.16)).

Looking at eq. (5.24), it is easy to see that the S-step objective consists of two terms; one term which is a weighted square difference in the means, and one which concerns the covariances. The weighting matrix on the difference between means is equal to the curvature of the action-value function $\boldsymbol{Q}_{uut}$ (see eq. (5.15)),

which means that errors are weighted based upon the amount they increase the approximated cost-to-go. The gradient of this quadratic cost term can easily be calculated using auto-differentiation or analytical means.

The covariance of the global policy is set to a constant value in many GPS applications, it can be optimized easily by a closed form solution that minimizes the objective as a function of a constant global policy covariance:

$$\boldsymbol{\Sigma}^{\pi} = \left[\frac{1}{NT} \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \boldsymbol{C}_{ti}^{-1} \right]^{-1} \tag{5.25}$$

If the covariance was to be a function of the observation $\Sigma^{\pi}(\boldsymbol{o}_t)$, then the optimization would again involve the calculation of derivatives and the application of some gradient descent method, as with the optimization of the mean.

6

Preliminary results

In this section, preliminary results are shown to investigate the feasibility of MD-GPS for a flight control application. To this end, two dynamical systems are investigated; a torque-controlled pendulum, and a force-controlled two-dimensional point mass. The goals of this section are to:

- Show a proof of concept
- Investigate the case where the global-policy suffers from unobservable states
- Investigate the ability of the global-policy to generalize outside its training trajectories
- Investigate the case where the trajectory optimization fails to find good solutions

First, the used methods and environments will be explained, followed by a presentation of the results, and finally an analysis.

6.1. Methods

In this work the GPS code implementation from [23], published by the authors of several GPS papers, is used. This repository includes implementations of many of the algorithms mentioned in the literature review presented in section 2.A of part I, including MD-GPS with iLQG trajectory optimization and fitted dynamics. To simulate environments, several agent interfaces are available; the MuJoCo simulator[77], ROS[64], and the Box2D physics simulator[1].

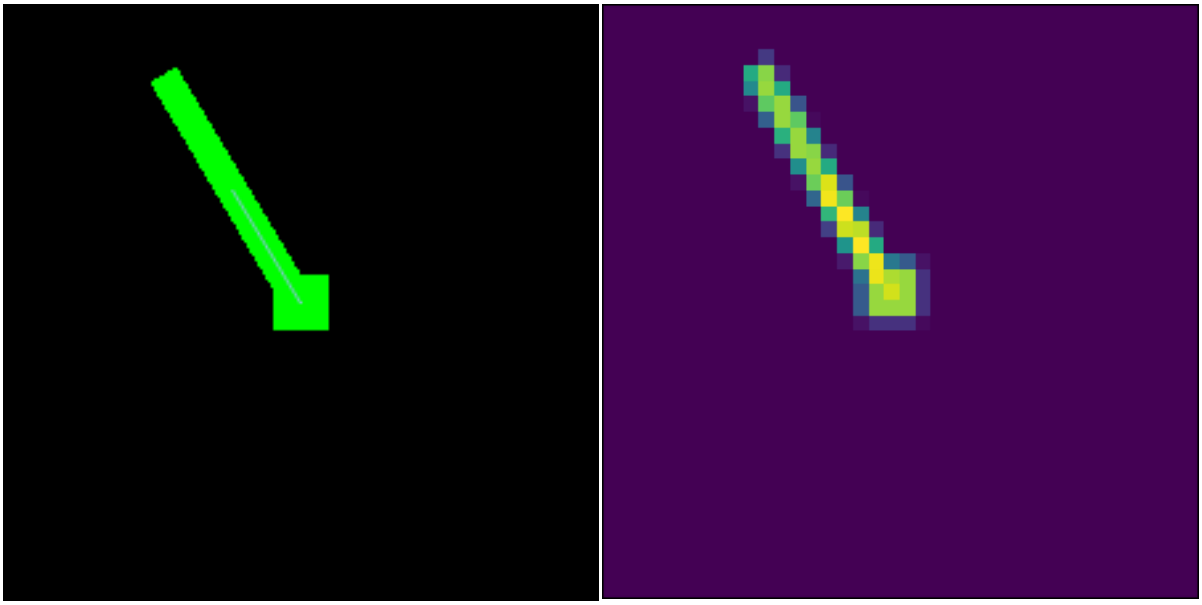
The tasks implemented in the preliminary analysis are all based on the Box2D simulator and the environments published in [23]. They are slightly reworked, to simplify, standardize dynamics to force and torque control, and implement rendered frames in the global policy. A description of the tasks is given below.

6.1.1. Pendulum

The pendulum environment consists of a frictionless torque-controlled pendulum subject to gravity, simulated using Box2D, and is based on the 2-link arm environment as published in [23]. The pendulum is a homogeneous rod of length 5.0 m, mass 0.4 kg·m^{-3}, with a gravitational acceleration of 10.0 m·s^{-2}. This results in a maximum gravitational torque of 10.0 N·m, which the agent may or may not be able to overcome, based on the maximum allowable torque input.

The state vector can be chosen in multiple ways, using either the angle of the pendulum w.r.t. the vertical; θ, or expanding the angle into a sine- and cosine component; $\sin\theta$ and $\cos\theta$. This has important consequences for the system identification step, since it assumes a TVLG model and approximates the dynamics up to the first order only. It also changes the definition of the cost function, which, during iLQG, is approximated as a second-order polynomial as a function of the state.

The observation consists of a single image of the pendulum at time t, and $\dot{\theta}$. To reduce the computational load, each frame has been downsampled from a $480 \times 640 \times 3$ RGB image, to a 38×38 grayscale image. Arguably, the velocity of the pendulum could also be encoded by providing multiple frames, but this requires substantially more weights in the ANN and in the interest of time this has not been implemented.



(a) Raw image.

(b) Image used as input to the global policy.

Figure 6.1: Comparison of the raw image and the downsampled image passed to the global policy as input, during the pendulum task. Downsampling is a simple average over a block of 6×6 pixels, resulting in a 38×38 grayscale image to be used as input to the global policy.

The goal of this task is to force it in an upright position and keep it there, an objective which is encoded by a quadratic cost function on the difference between state and the target-state $[0,1,0]^T$. The pendulum is initiated in the downward position, with zero velocity.

6.1.2. Point mass - full view

The point mass environment consists of a force-controlled point mass acting in a square box on a plane, its aim being to minimize its distance to some target position. It is a slightly altered version of the Box2D point mass environment as published in [23].

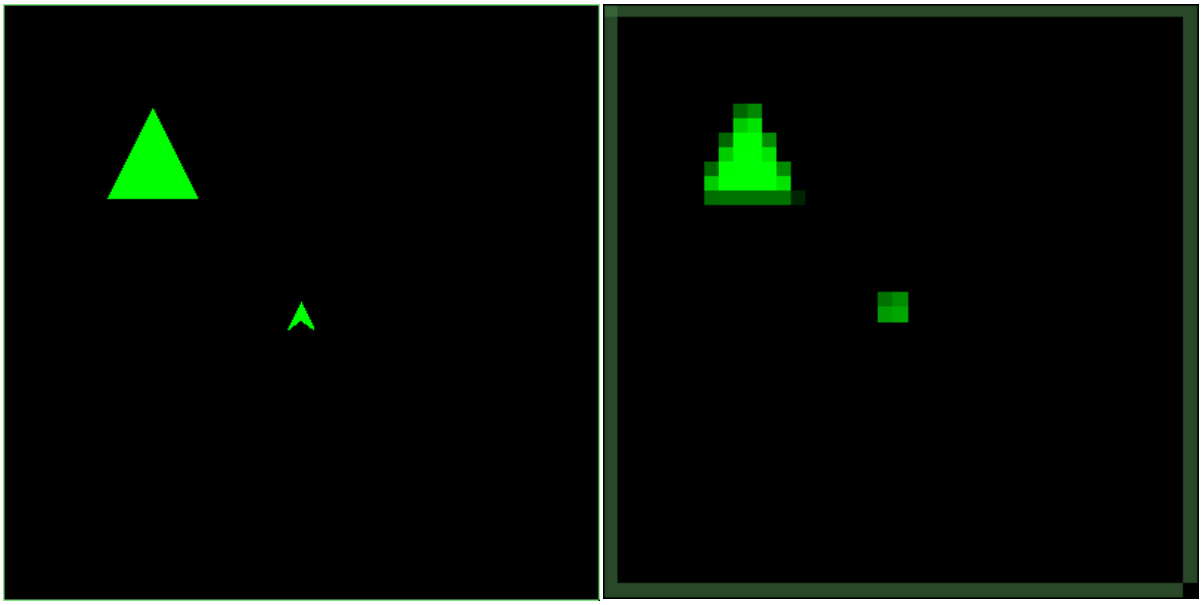
The state vector is as follows: $\boldsymbol{x} = [x, \dot{x}, y, \dot{y}]$, which is simply a vector containing the two-dimensional positions and their derivatives. The input is a two-dimensional vector with forces: $\boldsymbol{u} = [F_x, F_y]$.

The observation consists of a single image of the point mass world at time t, and the position derivatives $[\dot{x}, \dot{y}]$. The image is a top-down overview of the entire playing field, downsampled from a 401×401 RGB image, to a 41×41 RGB image. The agent's position is encoded by a green triangle in the overview.

The goal of this task is to force the point mass to a target position and keep it there, an objective which is encoded by a quadratic cost function on the difference between target- and actual position. The target position is also shown in the image as a smaller green triangle.

6.1.3. Point mass - partial view

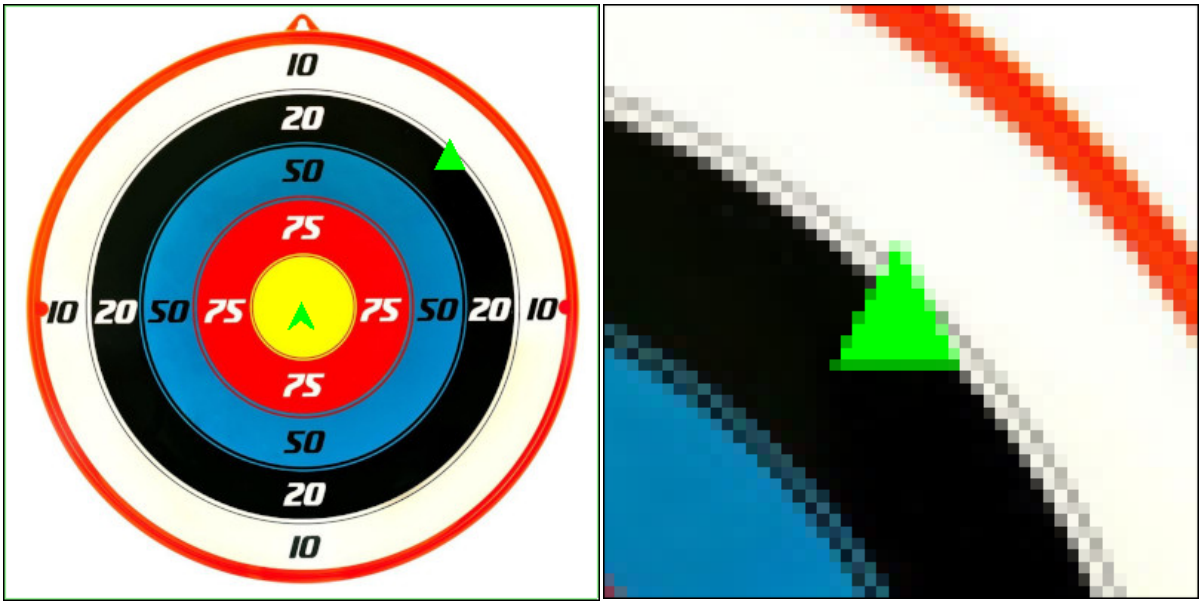
This experiment is similar to the one above, however, the input is changed to be a 100×100 RGB image centered on the agent, downsampled to 50×50 to reduce computational load. This change in input prevents the agent to infer the full state of the world as easily, a prolem which is solved by using a background image. During the training phase the agent will have to learn to match different parts of the background image to the actions as advised by the sub-policies. The structure of the image determines the observability of the problem in this case.



(a) Raw overview image.

(b) Image used as input to the global policy.

Figure 6.2: Comparison of the raw image and the downsampled image passed to the global policy as input, during the full view point mass task. Downsampling is a simple average over a block of 10×10 pixels, resulting in a 41×41 RGB image to be used as input to the global policy.



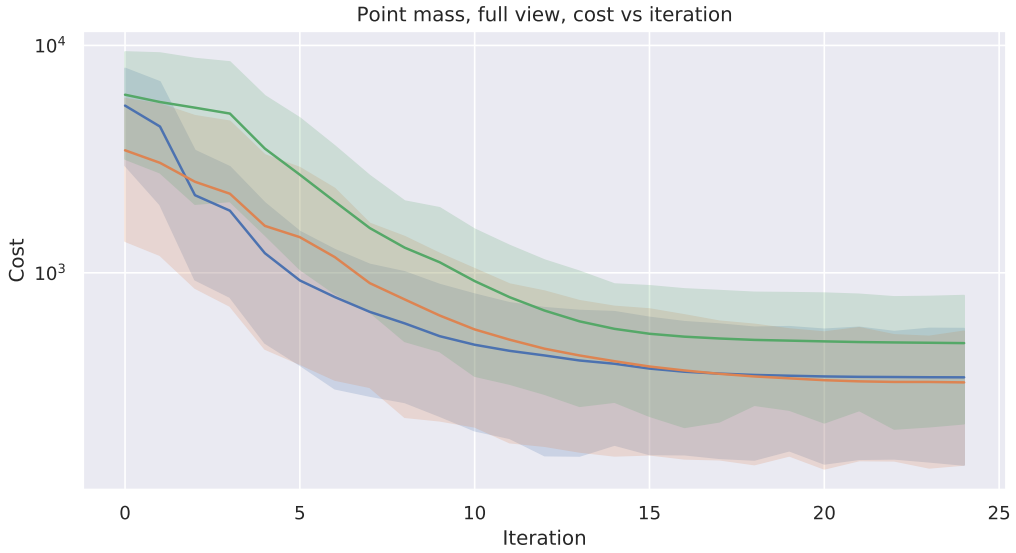
(a) Raw overview image, of partial-view point mass task using a background image.

(b) Image used as input to the global policy.

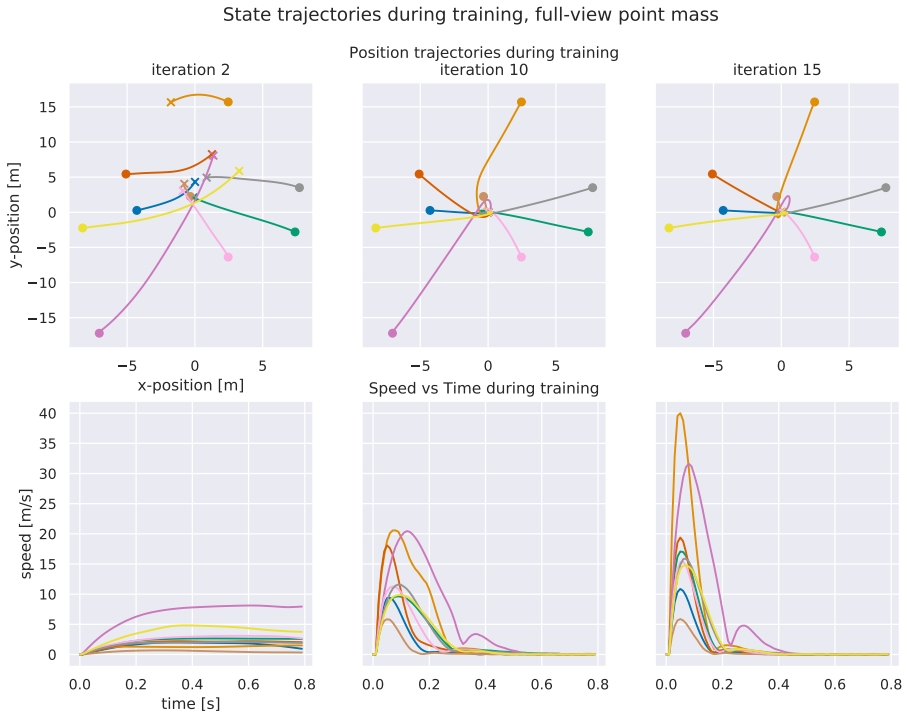
Figure 6.3: Comparison of the raw image and the downsampled image passed to the global policy as input, for the partial view point mass task. First, the image is zoomed into a 100×100 pixel square, centered around the agent. This is followed by downsampling; a simple average over a block of 2×2 pixels. The final product is an RGB image of 50×50 pixels, which will be passed to the global policy as input.

6.2. Results

In this section the results of the preliminary experiments are shown, starting with the results of the full-view point mass task. Secondly, the effect of different degrees of state observability is presented. Lastly, the results of an investigation into the degree of generalization to unseen states are presented.



(a) Cost versus training iteration, for the full-view point mass task without any background image, note the log-scale on the y-axis. Each line and corresponding shaded confidence intervals correspond to one experiment and 9 randomly chosen initial conditions. As can be seen the cost converges after around 15 iterations.



(b) The state trajectories at the second, tenth, and fifteenth training iteration of the full-view point mass experiment. The target is located at $[0, 0]$, starting- and end-points are denoted by dots and crosses, respectively.

Figure 6.4: Results of the full-view point mass experiment, using no background image.

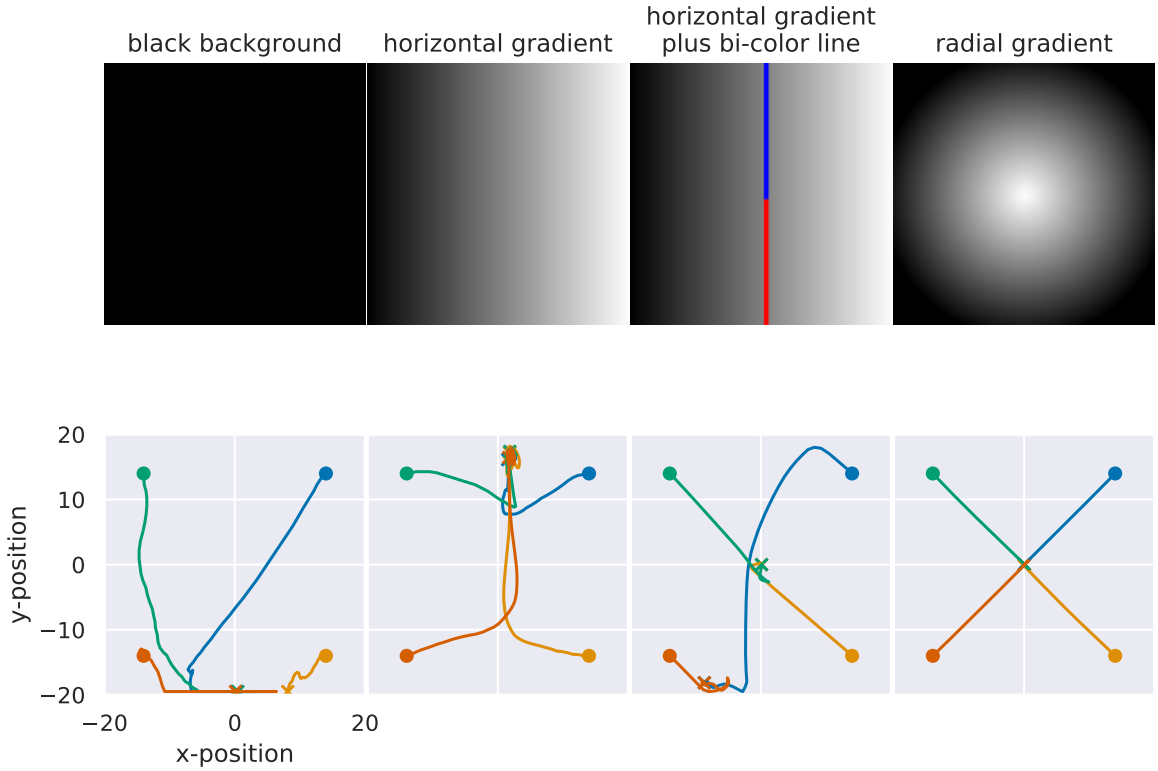
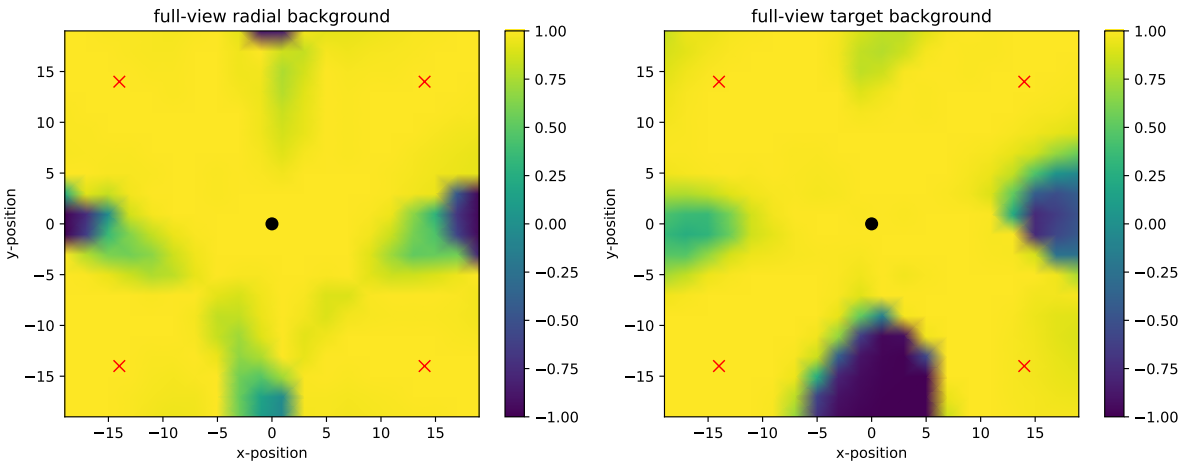
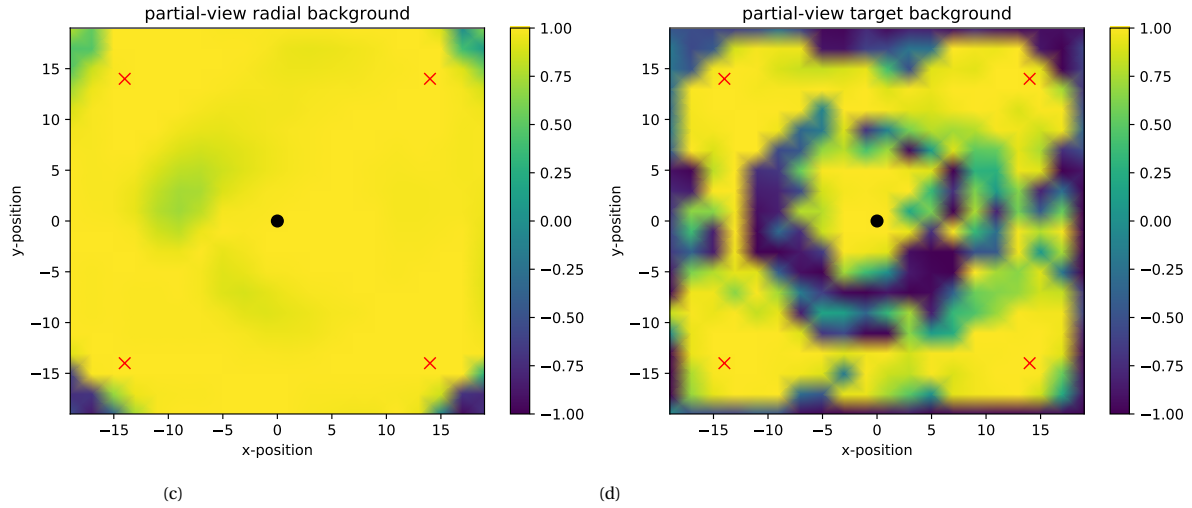


Figure 6.5: Comparison of different background on the behavior of the global-policy after training, from left to right improving observability of the state-space. Starting with no background, moving to a background offering only observability of the x-position, then adding low-quality vertical information with a bi-color line, finishing with a background that offers complete observability.



(a) (b)



(c) (d)

Figure 6.6: Comparison of the extent to which the global-policy is able to generalize to the entire state-space. The red crosses are the initial positions during training, the black dot the target position. The heatmap shows the cosine similarity of the action vector of the converged global-policy and the vector pointing to target, at zero velocity. At zero velocity the optimal action vector should at least point directly to the target; a cosine similarity of 1 denoted by a yellow color. From these plots the effect of background image and type of observation (full or partial) on generalization can be investigated, "radial background" refers to the radial gradient in fig. 6.5, "target background" refers to fig. 6.3a.

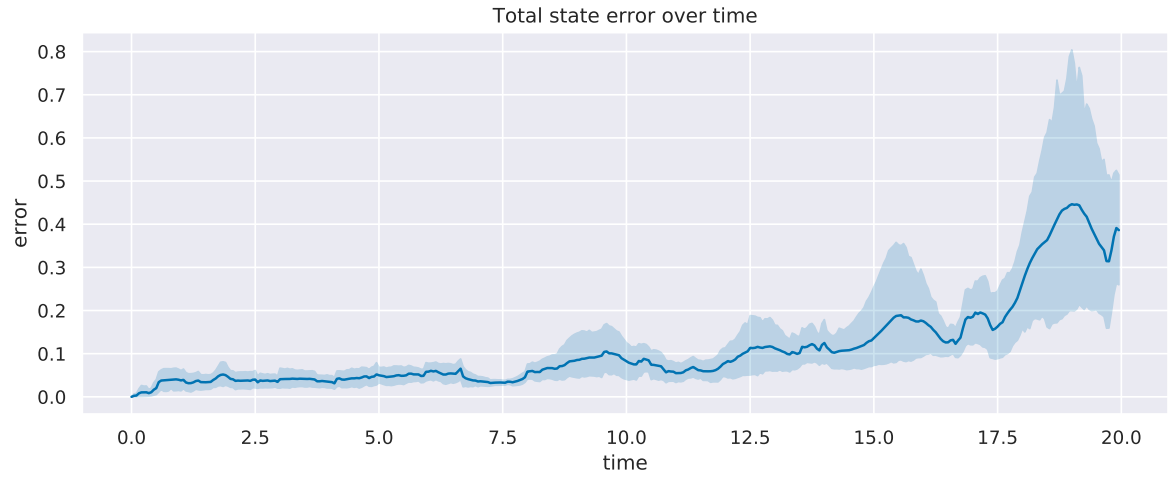


Figure 6.7: Accuracy of estimated state trajectories degrades over time, caused by compounding model error.

6.3. Analysis

This section discusses the results of the preliminary point mass and pendulum experiments presented in section 6.2.

6.3.1. Proof of concept

In fig. 6.4, the results of the point mass experiment, using a complete environment overview as observation and no background image, are shown. As can be seen in fig. 6.4a, the training of the global-policy converges around the 15^{th} iteration (note the log-scale). Each color line and shaded area represent the mean and 95% confidence intervals of the global-policy cost during the training of a single experiment, using 9 randomly chosen initial conditions and therefore trajectories. Obviously, the cost at convergence depends on the specific initial states of the experiment, however, the cost-iteration curves seem independent of this. This means the speed of convergence is the same for any initial state.

From the state-trajectories in fig. 6.4b it becomes clear that already by training iteration 2 the global-policy

is able to move roughly towards the target position (located at $[0,0]$). However, it does not slow down when near the goal as can be seen from the speed plots. At iteration 10, the global-policy is able to move towards the target and stay there, the main difference after another 5 iterations is the velocity increase. The end result is a global policy that is reliably able to move from the initial positions during training towards the target, and stay there.

These results show that the point mass task is easily solved when the global policy is offered a static top-down overview of the entire environment, in which it can see its own position at all times. An interesting question is whether the task can still be solved when the global-policy is offered a top-down overview of the agent's immediate surroundings, as explained in fig. 6.3.

6.3.2. Unobservable states

In fig. 6.5, the effect of different background images on the converged behavior is shown. As can be seen, a black background leaves the agent unable to distinguish between different states, leading to low-quality trajectories. The next background offers only the ability to distinguish the x-position, which leads to the agent moving towards the middle, but then drifting off. To provide a way for the global-policy to observe something about its y-position, two differently colored bars are placed in the middle. As can be seen, this leads the global policy to find the target for 2 of the 4 trajectories. It is interesting to see how the trajectories start out in the same direction per side (left or right). This leads two of the trajectories to move directly to the target, while the others move the wrong way. The trajectory starting in the upper right approaches the upper edge and then starts moving in the right direction, but overshoots the target. The lower left trajectory gets stuck near the boundary. The fact that the trajectories start out the same is a consequence of the initial observation being invariant w.r.t. the y-position, resulting in the same initial behavior. Theoretically the task should be solvable by simply moving towards the middle and then moving up or down when encountering a red or blue bar, respectively. However, the correct solution appears only sporadically during training, and the algorithm seems unable to settle. The final background image offers complete observability, since each position has a unique corresponding observation. As can be seen, the resulting behavior is simply to move directly towards the target and stay there.

As expected, the memory-less global-policy is unable to solve the task when suffering from unobservable states. Depending on the observable information about the state, the algorithm is able to guide the global-policy towards a solution that seems in line with the best solution available in that case. This is the result of the fact that the sub-policies are not only optimized w.r.t. cost, but also the KL-divergence w.r.t. the global-policy trajectory. Care should be taken to make sure the observations are such that the problem remains fully observable, since this leads to the fastest convergence. However, the algorithm is able to handle partial observability, as long as it leads the agent towards a region of observability as seen in the case of a horizontal gradient with bi-colored bars in the middle.

6.3.3. Generalization to unseen states

Another interesting question is how well the converged global-policy is able to generalize to states outside of its training trajectories. To this end, the plots in fig. 6.6 are made. What is shown is the extend to which the action vector at a particular position, and zero velocity, is pointing towards the target. Since the velocity is zero, the optimum action should always point directly towards the target, which makes it a measure of optimality of at least the action direction. The cosine similarity of the action vector and the vector pointing towards the target is presented as a heat map, yellow and blue signifying optimal and suboptimal behavior, respectively.

In fig. 6.6a and fig. 6.6b, the results of full-view point mass tasks are shown for two different backgrounds; the radial background from fig. 6.5 and the target background from fig. 6.3a, respectively. As seen in fig. 6.4b, the trajectories of a converged agent during the full-view task move directly towards the target and stay there. It seems the global policy is able to generalize quite well around these trajectories, as evidenced by the yellow areas around the imagined trajectories. The only blue occurs at the positions far away from these straight lines from the initial positions to the targets. In fig. 6.6c and fig. 6.6d, the results of the partial-view point mass task are shown for the same backgrounds as mentioned above. The radial background leads to much better generalization compared to the target background, which is probably a consequence of the fact that the radial background changes much more gradually as a function of position. Also, it stands out how the difference in generalization performance is much smaller for the full- than the partial-view task. This is probably a consequence of the fact that during the former, the position information is encoded by the location of the green triangle in the image, irrespective of the background. This is different from the partial-view task, where

the green triangle is ever-present at the same location in the image and thus provides no information, while the position is encoded by the background image.

6.3.4. Failure of trajectory optimization

The final preliminary results involve the C-step of MD-GPS, which is fundamentally the part that solves the task. It uses locally fitted TVLG models of the dynamics and a second-order model of the cost, combined with a model-based optimal control method called iLQG. Since inaccurate dynamics and cost models are used, long sequences potentially lead to inaccurate results. This is caused by compounding model error and the fact that the models are local, increasing inaccuracy when straying too far from the state-trajectories at which they have been fitted. In the case of linear dynamics and quadratic cost, like the point mass task, these problems are mitigated. This can be explained by the fact that the dynamics are linear, and as long as the sides of the environment are not touched, the locally fitted models are essentially global. However, when attempting tasks that include non-quadratic costs and nonlinear dynamics, these issues do crop op, as seen during the pendulum task.

Care must therefore be taken when choosing the exact state representation of the task. In the case of the pendulum task, several distinct representations are possible, each with it's own pros and cons. For instance, when choosing the pendulum state simply as $[\theta, \dot{\theta}]$, with θ being continuous, this offers a nice simple integrator description of the pendulum. However, since gravitational torque is proportional to the sine of the angle; $T_g \propto \sin\theta$, the effect of gravitational torque is a nonlinear (and actually periodic) function of the state. In addition, these effects must be identified for each period of the sine separately; dynamics fitted in the θ interval $[0, 2\pi)$ do not assist in the interval $[2\pi, 4\pi)$. As for the cost function, one could choose simply the sum of squared differences between the current and goal state. The goal state being a pendulum that sits still in an upright position, which can be encoded by a state of $[0, 0]$. However, this constrains the solution to a θ value of zero, while actually any integer multiple of 2π would satisfy the task. Another option is to set to cost to $(\cos\theta - 1)^2$, which would more accurately represent the task. However, the periodic cost function will then be modeled as a quadratic function of the state, which leaves it unable to capture the true character of the cost far from the states at which it is fitted.

Another option is to wrap θ to the interval $[0, 2\pi)$. This would solve the problem of having to model gravitational torque for every period separately, since the angle will never leave the one-period interval. However, it introduces a discontinuity at the upright position, which cannot be accurately modeled by a TVLG model.

The last option is to expand the state to 4 dimensions, using the Cartesian coordinates and velocities of the end point of the pendulum as state; $[x, y, \dot{x}, \dot{y}]$. This allows the gravitational torque to be proportional to the x coordinate and thus also solves the problem of having to fit these dynamics multiple times. Additionally, the cost function becomes a quadratic of the state. However, this state representation does not capture the constraints of the environment; the estimated pendulum end-points are not confined to a circle around the origin. This poses the problem of quickly deteriorating state estimation when treading too far from the fitted trajectories.

III

Additional results and Discussion

7

Additional results and discussion

From the research questions presented in chapter 2, the first has been answered in part II, while number 2, 3, and 4 have been handled in the scientific paper presented in part I. The results corresponding to the remaining research questions (5, 6, and 7) and their analysis are answered in this chapter sequentially. The experiments have been conducted using the methods explained in section 3 of part I, using the quadcopter model and controller presented in appendix D, and the default parameters outlined in section 3.G of part I and more thoroughly in appendix C.

7.1. Research question 5

This section shows and analyzes the results of experiments that are used to answer the fifth research. For convenience, the research question is repeated below:

How does pre-training the dynamics prior and TVLG models, using demonstrations, affect the sample efficiency of MD-GPS?

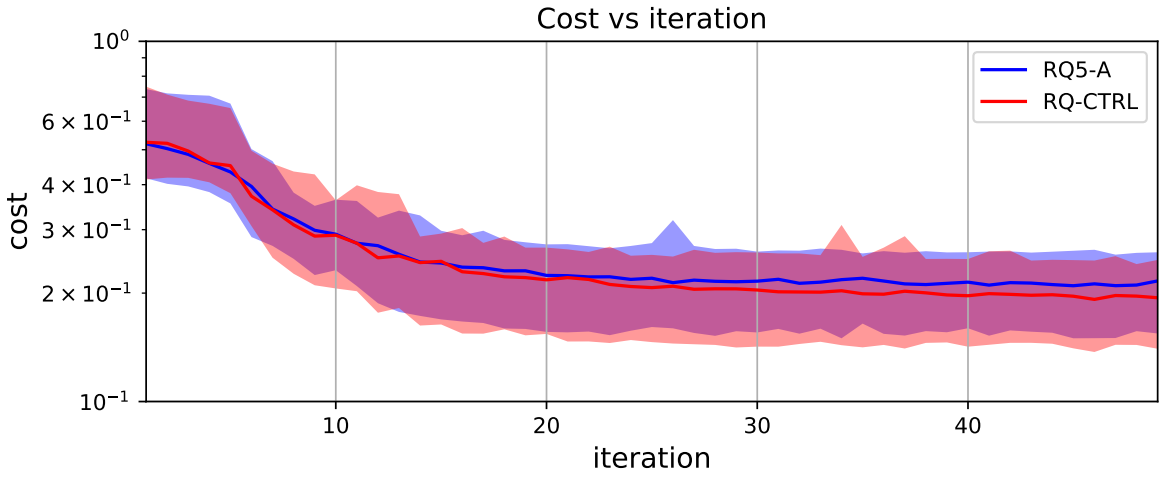


Figure 7.1: Cost-iteration curves, for experiment `RQ5-A` (without using dynamics-model pre-training), and `RQ-CTRL` (with using dynamics-model pre-training). The plot shows a thick line, which signifies the median, and a shaded area which signifies the 68.27% confidence interval. Take note of the log-scale on the y-axis.

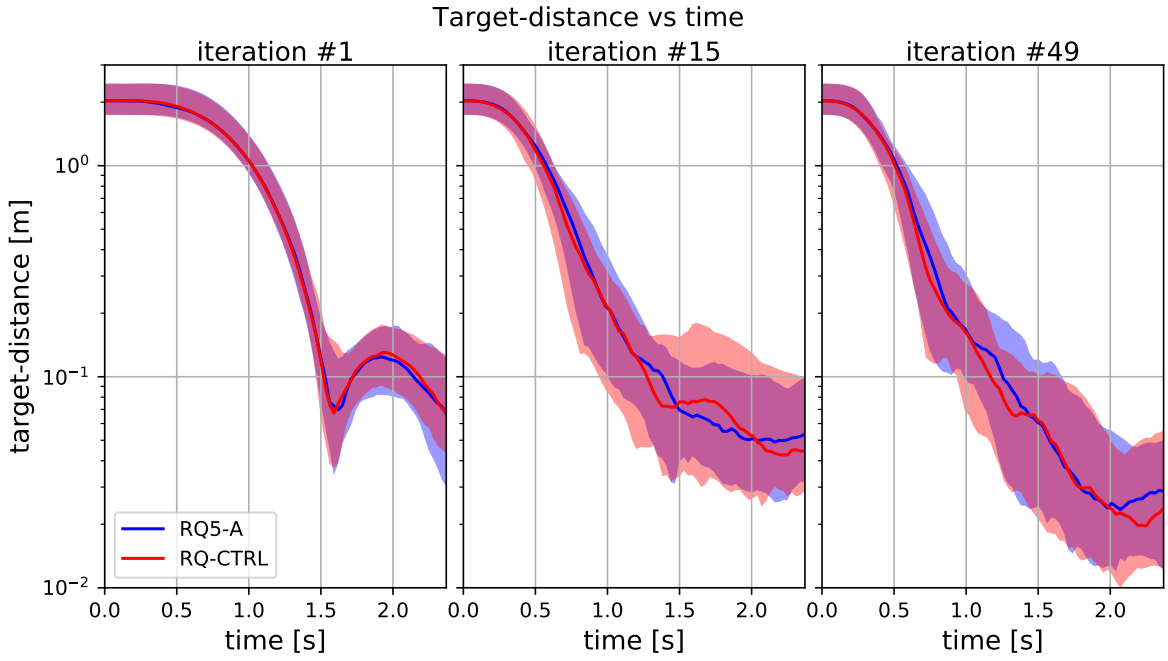


Figure 7.2: Three plots of target-distance versus time plots at different iterations, for experiment `RQ5-A` (without using dynamics-model pre-training), and `RQ-CTRL` (with using dynamics-model pre-training). The plots show a thick line, which signifies the median, and a shaded area which signifies the 68.27% confidence interval. Take note of the log-scale on the y-axis.

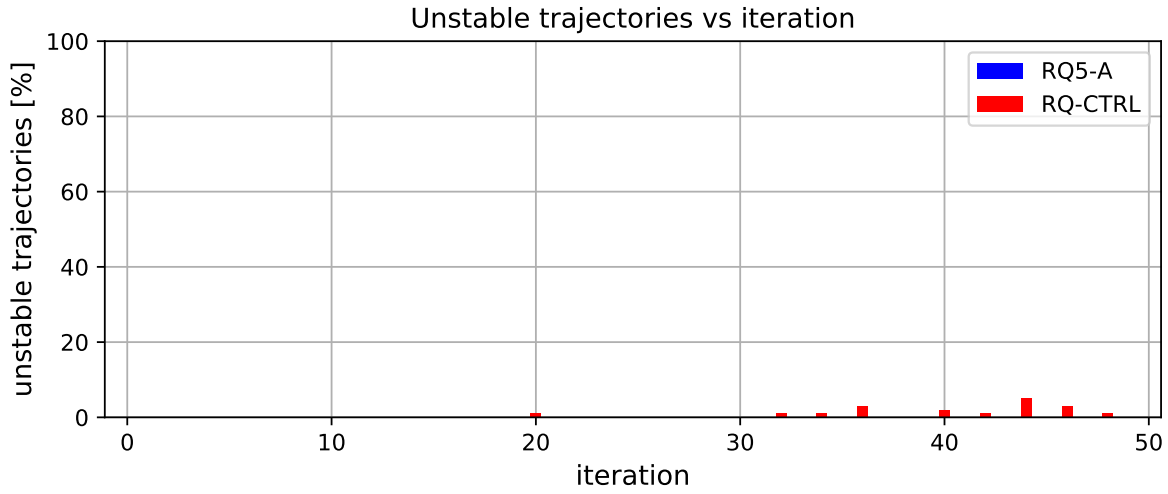


Figure 7.3: A bar-plot showing the percentage of sampled trajectories that is "unstable", for experiment `RQ5-A` (without using dynamics-model pre-training), and `RQ-CTRL` (with using dynamics-model pre-training). A trajectory is unstable if at any time step the distance to the target is larger than 6 m.

To investigate the effect of pre-training the dynamics models, the cost-iteration curves with- and without pre-training are shown in fig. 7.1. As can be seen, the curves are almost identical, the non-pre-trained cost even being a little lower. The same can be seen in fig. 7.2, where the target-distance curves are practically the same for both initialization methods. In fig. 7.3 the percentage of trajectories that are unstable are presented per training iteration. At the end of the training process (iteration 30 and up) the fraction of unstable trajectories seems to slightly increase when omitting pre-training the dynamics models. However, this increase is very minimal and stays below 6%.

Pre-training the dynamics models has no discernible effect on sample-efficiency, as the cost-iteration curves are practically identical with- and without it. The only discernible difference is the slight up-tick

in unstable trajectories near the end of training when not performing dynamics models pre-training. This contradicts the earlier hypothesis that stated the improved dynamics models would lead to better trajectory updates at the start of training. This could be explained by the fact that the step size at the start of training is deliberately set quite low, to prevent aggressive updates at a time when the dynamics models are still inaccurate. It could well be that when the initial KL-divergence step size is increased, a difference would be observable.

7.2. Research question 6

This section shows and analyzes the results of experiments that are used to answer the sixth research. For convenience, the research question is repeated below:

How does utilizing demonstrations to create a cost-function affect the sample-efficiency of MD-GPS?

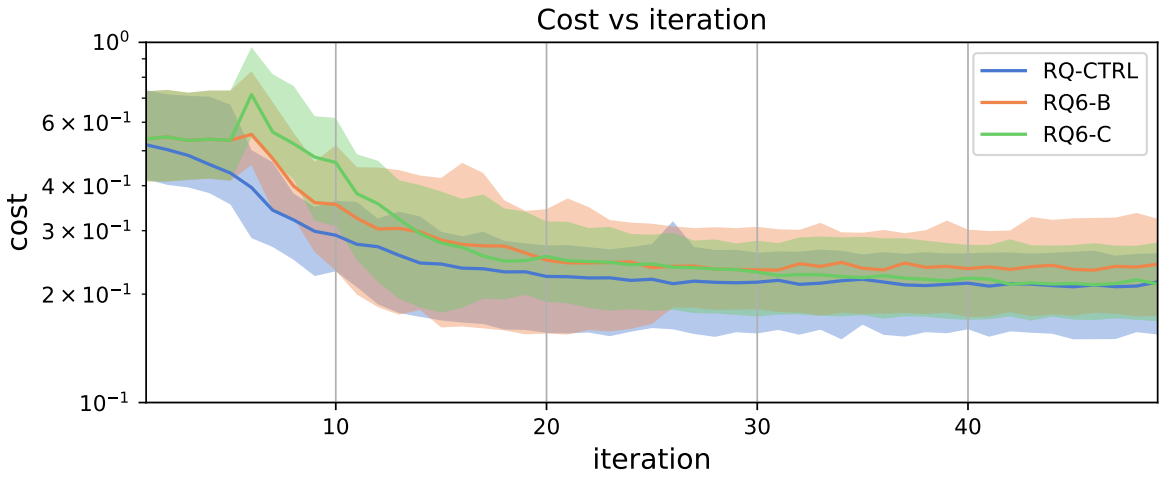


Figure 7.4: Cost-iteration curves, for experiment `RQ-CTRL` (using the normal cost-function), `RQ6-B` (using a demonstration-based cost-function up to iteration 5), and `RQ6-C` (using a demonstration-based cost-function up to iteration 10). The plot shows a thick line, which signifies the median, and a shaded area which signifies the 68.27% confidence interval. Take note of the log-scale on the y-axis.

To investigate the effect of demonstration-based cost on the sample-efficiency, the cost-iteration curves are plotted in fig. 7.4. As can be seen, the cost when using a demonstration-based cost for 5 iterations is higher than when using the normal cost, over the entire training process. As for using a demonstration-based cost for 10 iterations, it seems to ultimately converge to the same cost as using a normal cost-function, but the rate of convergence is much slower. This can also be observed in fig. 7.5, where the target-distance curves when using demonstration-based cost for 5 iterations converges to a higher value than the control at iteration 15 and 49, while using it for 10 iterations leads to a curve almost identical to the control at iteration 49. As for the occurrence of unstable trajectories, none of the experiments show any unstable behavior and therefore there is no plot shown.

These results contradict the earlier hypothesis that said the demonstration-based cost-function would improve sample-efficiency by providing more information than the standard position-based cost. A possible explanation is that while a demonstration-based cost-function provides more information, it ultimately rewards the wrong trajectories. Using the default settings, the cost decreases from iteration 1 onward, meaning already at the start it has enough information to determine effective trajectory updates. In that case, a demonstration-based cost-function will lead to sub-optimal trajectory updates.

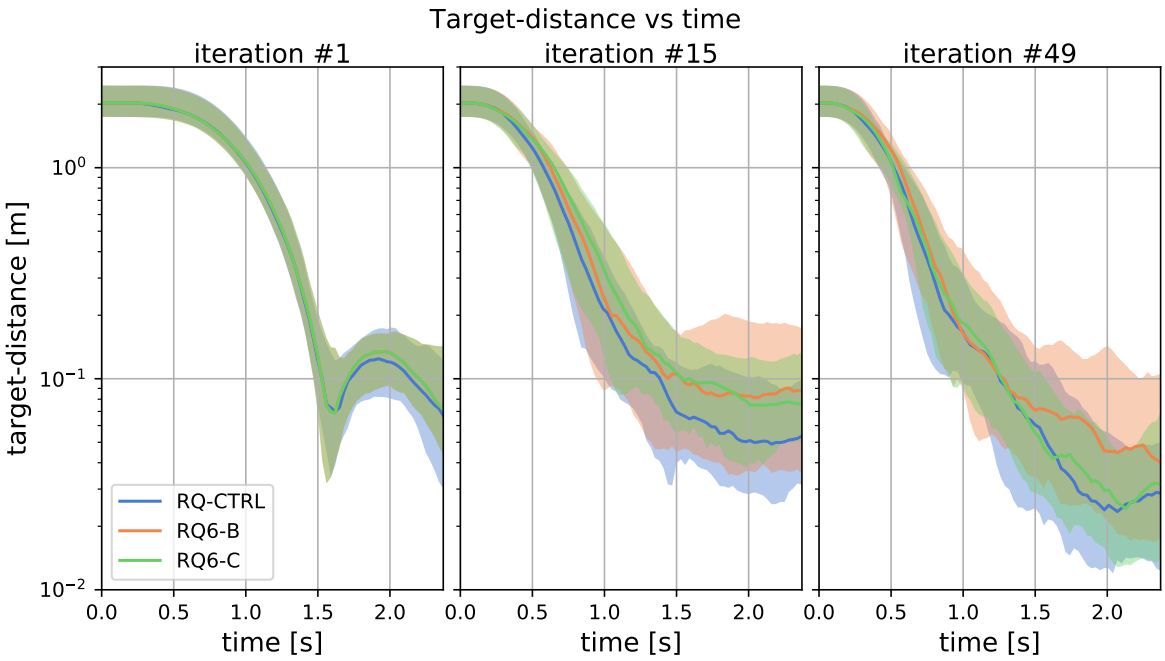


Figure 7.5: Three plots of target-distance versus time plots at different iterations, for experiment `RQ-CTRL` (using the normal cost-function), `RQ6-B` (using a demonstration-based cost-function up to iteration 5), and `RQ6-C` (using a demonstration-based cost-function up to iteration 10). The plots show a thick line, which signifies the median, and a shaded area which signifies the 68.27% confidence interval. Take note of the log-scale on the y-axis.

7.3. Research question 7

This section shows and analyzes the results of experiments that are used to answer the seventh research. For convenience, the research question is repeated below:

How robust is MD-GPS to different settings of the KL-divergence step size increase/decrease factor?

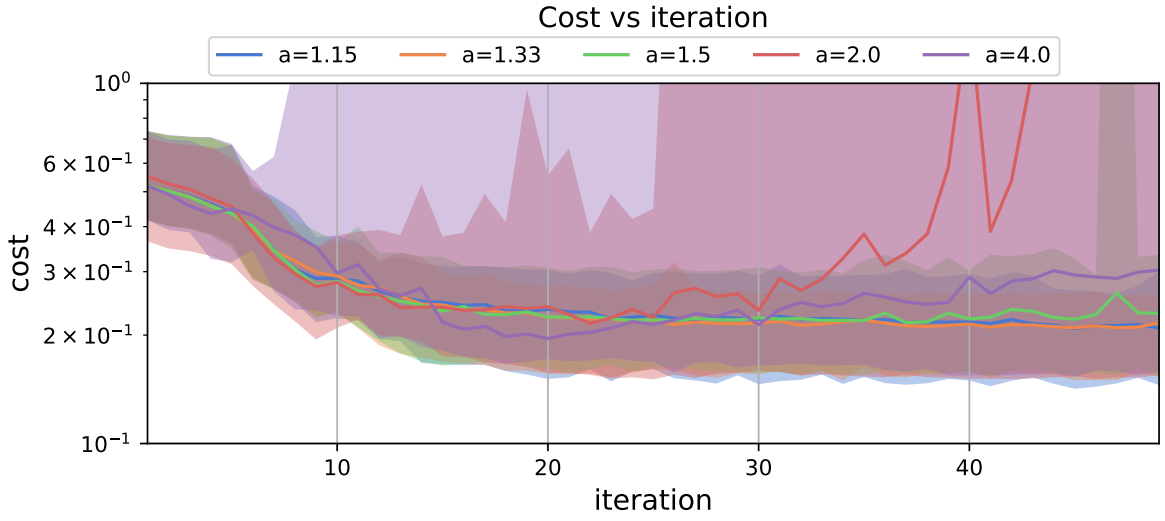


Figure 7.6: Cost-iteration curves, for experiment `RQ7-A` (using a learning rate adaptation factor of 1.15), `RQ7-B` (using a factor of 1.33), `RQ7-C` (using a factor of 1.5), `RQ7-D` (using a factor of 2.0), and `RQ7-E` (using a factor of 4.0). The plot shows a thick line, which signifies the median, and a shaded area which signifies the 68.27% confidence interval. Take note of the log-scale on the y-axis.

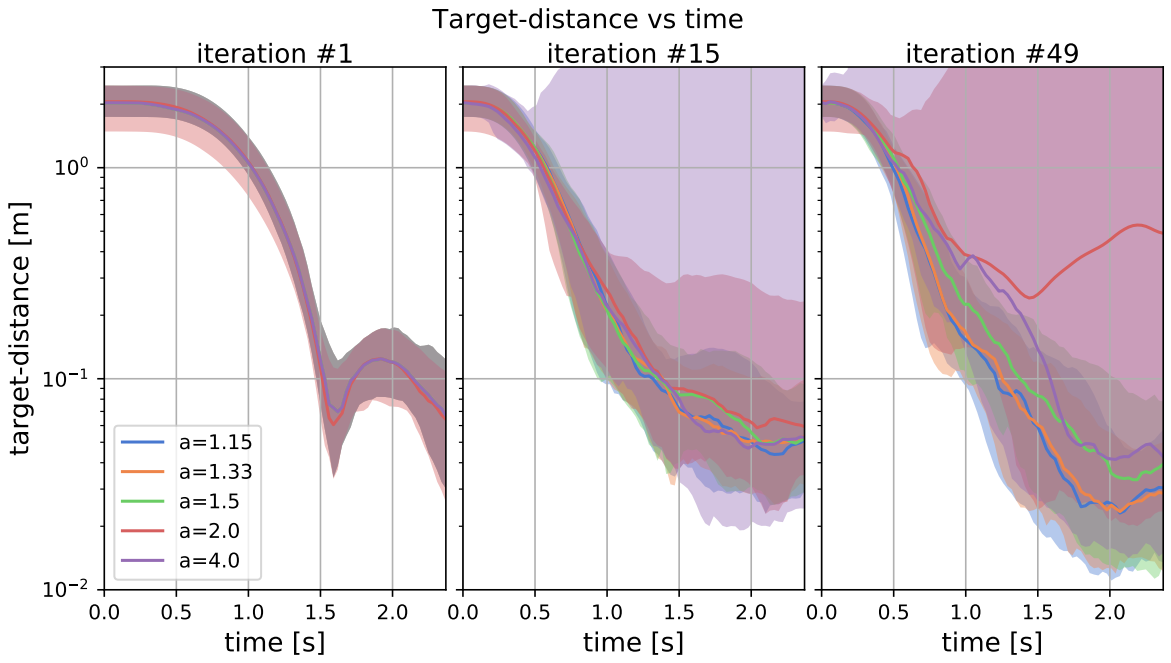


Figure 7.7: Three plots of target-distance versus time plots at different iterations, for experiment `RQ7-A` (using a learning rate adaptation factor of 1.15), `RQ-CTRL` (using a factor of 1.33), `RQ7-C` (using a factor of 1.5), `RQ7-D` (using a factor of 2.0), and `RQ7-E` (using a factor of 4.0). The plots show a thick line, which signifies the median, and a shaded area which signifies the 68.27% confidence interval. Take note of the log-scale on the y-axis.

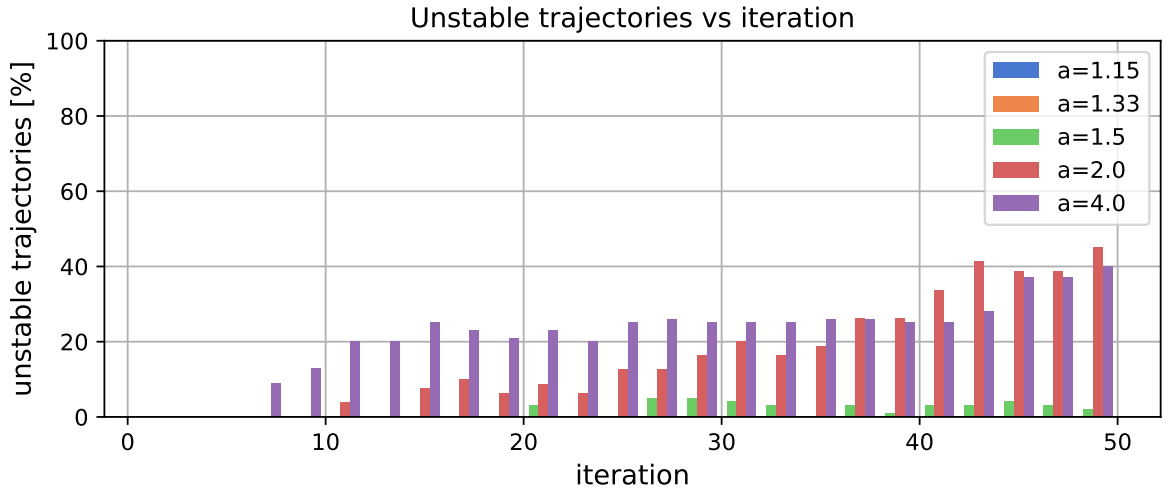


Figure 7.8: A bar-plot showing the percentage of sampled trajectories that is "unstable", for experiment `RQ7-A` (using a learning rate adaptation factor of 1.15), `RQ-CTRL` (using a factor of 1.33), `RQ7-C` (using a factor of 1.5), `RQ7-D` (using a factor of 2.0), and `RQ7-E` (using a factor of 4.0). A trajectory is unstable if at any time step the distance to the target is larger than 6 m.

To investigate the effect of the step size increase/decrease factor on the sample-efficiency, the cost-iteration curves for different values are plotted in fig. 7.6. As can be seen, the cost-iteration curves for the factors 1.15, 1.33, and 1.5 are almost identical, having similar median and confidence intervals. However, the curves for a value of 2.0 and 4.0 lead to a higher final cost value and a large increase in cost-variance. The rise in cost-variance occurs sooner for a factor of 4.0 then 2.0, but the latter has a larger final cost. Also, a value of 4.0 achieves a relatively low cost-value around iteration 20, but subsequently performance degrades to end up at a higher cost at iteration 49.

Looking at the target-distance curves in fig. 7.7, it is easy to see that the same trend occurs here; for factor values of 1.15, 1.33, and 1.5 the target-distance shows the same progression. However, the target-distance

confidence intervals for factors of 2.0 and 4.0 are significantly increased, with the former converging to a much larger distance than the other values.

In fig. 7.8, the percentage of unstable trajectories per iteration is shown for different factor values. As can be observed, a general rule seems to be the larger the increase/decrease factor, the larger the fraction of unstable trajectories. Interestingly, although the number of unstable trajectories for a factor of 2.0 starts rising more slowly than for a value of 4.0, both converge to the same percentage. Also, for both a value of 1.15 and 1.33 there are no unstable trajectories.

These results support the previously hypothesis that stated that a higher step size decrease/increase factor can lead to stability issues due to its aggressive trajectory updates. The results are as expected; large factor values lead to high percentage of unstable trajectories. It was also hypothesized that the upside of aggressive step size adaptation could be that it can increase the speed at which the algorithm adapts to new information, possibly leading to faster convergence. This however is not observed in the results, as the median cost-iteration curves are quite similar, with the lower factor values outperforming the higher. An explanation could be that for this system the valid region of the dynamics models is too small to allow for large step sizes and any changes in model validity are quite small, reducing the need for aggressive step size adaptations.

IV

Closure

8

Conclusion

This chapter concludes the thesis report, presenting the main results of the experiments, the outcome of the analysis, and the answers to the research questions. This will include the questions that have been handled in the scientific paper presented in part I. Firstly, the outcome of the literature study and preliminary results (presented in part II) is discussed, followed by the conclusion with respect to the research questions as specified in chapter 2.

The last decade there have been many exciting developments in machine learning; using expressive function approximators like ANNs, combined with large amounts of data and stochastic optimization methods, have lead to many interesting applications in computer vision and signal processing. Additionally, the field of RL has been making use of these new techniques to train high-dimensional policies that act directly on complex visual inputs, which has seen much use in creating agents that perform extremely well in a plethora of computer games. The promise of using high dimensional sensory information directly as control input has also lead to applications in the field of flight control, focusing mainly on UAVs. However, the amount of data needed to train these policies tends to be orders of magnitude larger than what is feasible for real-world systems. To solve this problem, the main goal of this thesis is to:

Main goal

Investigate how learning from demonstration can be used to improve the sample-efficiency of reinforcement learning applied to flight control.

To this end, the following initial research question is posed, which then leads to the additional ones:

Research question 1

What is an RL algorithm that fits well with a flight control application, is a good fit to be combined with LfD, and preferably has high baseline sample-efficiency?

As explained in chapter 1, this question is answered by a literature study into both RL and LfD, combined with several preliminary experiments, which are presented in part II. This investigation start by listing the main requirements a flight control RL algorithm needs to fulfill, in section 5.1. The first of which states that, since the flight controller needs to handle continuous state- and action-spaces, the algorithm should as well. This rules out any tabular RL methods, since these live in discrete state- and action-spaces. In general, model-free methods are infeasible because of their over-reliance on trial-and-error and low sample-efficiency. This naturally leads to the choice of model-based continuous RL methods.

Secondly, it is determined which type of LfD is the best fit when combined with the previously picked family of RL methods. In section 4.2, many different LfD methods are investigated. For a flight control application, the most practical use-case of LfD is when expert demonstrations are available in the form of state-action-state tuples, generated by a teacher with he same embodiment as the student. This removes the need for any mapping from observation to demonstration, or from demonstration to the student's own state-space. A desirable property of a good LfD method for flight control is one that allows the student to surpass

the master, a feature which many do not support. A family of methods which can effectively integrate LfD into continuous model-based RL, while allowing the student to improve upon the demonstrations, is GPS.

The type of GPS deemed feasible for further exploration in this thesis is MD-GPS, which has relatively few hyperparameters and shows good performance. This method efficiently trains a high-dimensional global-policy for a control task by combining supervised learning with local system-identification and model-based RL.

To investigate the feasibility and failure modes of MD-GPS, some preliminary experiments have been conducted in section 6.2. It is shown how MD-GPS is able to accurately control a two-dimensional point-mass using a vision-based control policy. Additionally, experiments show that when the global-policy suffers from unobservable states, the information that ís available is used efficiently to maximize performance. Thirdly, it is shown that the ability of MD-GPS to generalize to unseen states using vision-based control depends heavily on the type of camera-viewpoint, and the characteristics of the background. When using a third-person overview as the input, generalization is largely independent of background image characteristics, while using a first-person camera-view leads to generalization that is dependent heavily on the background image. Finally, it is shown how the total state-estimate error tends to increase over time, which is attributed to compounding modeling errors. The fact that the true dynamics are unknown and MD-GPS makes use of TVLGs impedes the use of long time horizons.

The answer to the initial research question leads to a more specified second set of questions that together attempt to answer how the sample-efficiency of MD-GPS can be further improved by making use of demonstrations by an expert; LfD. This main research question is divided up into six sub-questions, which have been researched by performing multiple experiments in simulation, using a nonlinear dynamics model of a quadcopter. The demonstrations take the form of state-action trajectories generated by a quaternion-based nonlinear controller, which generates high quality examples. Of these six research questions, the first three are handled in the scientific paper presented in part I, while the remaining ones are handled in part III. What follows now are the main conclusions of analysis corresponding to the remaining research questions, starting with a recap of each question.

Research question 2

How does initializing the sub-policies of MD-GPS with high-quality demonstrations affect the sample efficiency?

It has been shown that by initializing the trajectory-optimization with demonstration-based sub-policies, both the initial- and final cost are reduced significantly. As a matter of fact, with random sub-policy initialization the algorithm is not able to find a policy that converges to the target-position at all; demonstration-based sub-policy initialization appears necessary to find a controller that stabilizes the system.

Research question 3

How robust is MD-GPS to sub-policies initialized using sub-optimal demonstrations?

Although demonstrations are often examples of low-cost trajectories, in general they are not strictly optimal as judged by the cost-function of the agent. Therefore, the robustness of MD-GPS to sub-policy initialization with sub-optimal demonstrations is investigated. It has been shown that when the degree of sub-optimality is within certain bounds, the policy converges to a similar solution as when initialized by a high-quality demonstration. In fact, when the sub-optimality is a 0.25 m target-offset, the cost-value converges to the same solution as when using the high-quality demonstration. With a higher target-offset of 1.00 m, the solution converges to a solution that is worse, which indicates the degree of sub-optimality is too high and the initial trajectories lie outside the region of attraction of the optimal solution. In the case of a demonstration that is more noisy ("squiggly"), the cost reaches a slightly worse cost-value at the 49^{th} iteration, for both an input-noise variance of 7.5 and 15.0. This indicates that both variance levels are too large to allow convergence to the cost-value reached when using the high-quality demonstration.

Research question 4

How does the specific training structure affect the sample-efficiency of MD-GPS?

Demonstrations can also be used to train the global-policy before the MD-GPS training process starts; global-policy pre-training. Several experiments are conducted to compare different ways of structuring the training process, with and without pre-training. The MD-GPS algorithm uses the global-policy to sample trajectories in the real-world and constrain trajectory-optimization. However, results show that delaying the point at which the global-policy is used for the KL-divergence constraint to the fifth-, and global-policy sampling to the tenth iteration, sample-efficiency is greatly improved when compared to the unaltered algorithm. With alterations and pre-training, the amount of samples needed to achieve the cost-value that the unaltered algorithm achieves at convergence, is reduced by a factor of six. Without these delays a pre-training causes the solution to diverge.

Research question 5

How does pre-training the dynamics prior and TVLG models, using demonstrations, affect the sample efficiency of MD-GPS?

The effect of using demonstrations to pre-train the dynamics models is investigated, but in this work no effect on sample-efficiency was found. This can be explained by the fact that initial step size is set to a value that is too low to see any effect on cost. The first few iterations allow the dynamics models to become more accurate, after which step size increases.

Research question 6

How does utilizing demonstrations to create a cost-function affect the sample-efficiency of MD-GPS?

Another use of demonstrations is to define a custom cost-function, that forces the optimization to match the demonstration trajectories. Although it was hypothesized this could increase sample-efficiency by providing more information to the trajectory-optimization algorithm, the experiments show that it worsens the training performance instead. Leading to a slower convergence than when using the actual cost-function, but converging to a similar final cost.

Research question 7

How robust is MD-GPS to different settings of the KL-divergence step size increase/decrease factor?

Finally, the robustness of MD-GPS to step size adaptation factor has been investigated. Each iteration the KL-divergence step size is adapted based on some heuristics, depending on the agreement between the expected and actual cost-values. Results show that, for this particular quadcopter model, lower adaptation factors (1.5 and below) lead to more stable trajectories and improved cost.

It has been shown that by combining MD-GPS with demonstrations, sample-efficiency can be improved significantly, which is important for designing flight controllers. A frequent problem with synthesizing flight controllers is that an accurate model of the system is needed. While this can already be a problem with classical control, it is exacerbated when performing control in highly nonlinear regimes and using unorthodox sensory modes such as vision or sound, which are hard to model accurately. These are circumstances that occur more often in MAV control, which has been the focus of this thesis. Additionally, the complex nature of these unorthodox inputs requires expressive control policies, which generally require a large amount of samples when trained with RL methods. Combining MD-GPS and LfD opens up the way to improve sample-efficiency to an extent that highly expressive flight controllers can be synthesized using real-world interactions, which removes the need for accurate nonlinear models and circumvents the reality-gap problem. Also, the algorithm is information-source agnostic; the use of supervised learning and an expressive global-policy parameterization allows it to extract control-relevant information from essentially any continuous observation space. This flexibility opens up the way for creative new control strategies to augment classical control algorithms. Finally, this work has shown how demonstrations can be integrated into the MD-GPS algorithm in various ways, using only demonstration trajectories. No requirements are placed on the source of these trajectories, meaning both classical controllers and human experts can be used to obtain them. In conclusion, this work shows how LfD methods can be used to significantly improve the sample-efficiency of MD-GPS for

a flight-control application, opening up the way for training high-dimensional flight-control policies using real-world interactions.

9

Recommendations

The results from this thesis lead to the following recommendations for further research:

- **Utilizing visual input**
  In this work the global-policy uses full-state input, further research could investigate using camera images in combination with partial-state feedback, instead. This option has already been tested during the preliminary analysis presented in this thesis, but this was in combination with only a toy-model of a quadcopter. Visual input can be easily obtained by simulation, using of-the-shelf rendering tools.

- **Training a policy using a real-world quadcopter**
  The approach in this work has been to utilize full-state feedback in a quadcopter simulation. However, the amount of samples needed is such that it is definitely feasible to train a quadcopter controller using real-world experiments. By applying sophisticated motion-capture systems to estimate the state in real-time, the policy could the necessary state-feedback. Demonstrations could be generated by any pre-existing controller or even a human pilot.

- **Utilizing unorthodox sensory modes**
  MD-GPS makes no assumptions regarding the input to the global-policy, making it extremely flexible. In addition to the aforementioned visual input, one could imagine sound-based control input, or for instance LIDAR measurements.

- **Control based on partial-state feedback**
  The global-policy in this work has been parameterized by a purely feedforward, reactive ANN, which has no state. This kind of controller is able to achieve high performance because the full state was made available to it. Naturally, real-world applications not always allow for full-state feedback, which is often solved by employing explicit state-estimation techniques. It would be interesting to investigate using stateful global-policy parameterizations (recurrent neural networks (RNNs) for instance) to perform implicit state-estimation, trained end-to-end.

- **Combining MD-GPS with safe-RL methods**
  MD-GPS has to explore the state-space in order to accurately estimate the dynamics, it can run into problems regarding safety of the agent. This can perhaps be attributed to the fact that the origin of the method lies in robotics, controlling robot arms. In flight control the agent always is at risk of a crash, either into the ground or into a wall. An interesting avenue of research could be the integration of safe-RL methods into MD-GPS.

Bibliography

[1] Box2D | A 2D Physics Engine for Games. https://box2d.org/.

[2] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Twenty-First International Conference on Machine Learning - ICML '04*, page 1, 2004. ISBN 1581138285. doi: 10.1145/1015330.1015430.

[3] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009. ISSN 09218890. doi: 10.1016/j.robot.2008.10.024.

[4] J.A. Bagnell and J.G. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, 2:1615–1620, 2001. ISSN 1050-4729. doi: 10.1109/ROBOT.2001.932842.

[5] Michael Bain and Claude Sammut. A Framework for Behavioural Cloning. *Machine Intelligence*, 15 (June):103–129, 1996. doi: 10.1.1.25.1759.

[6] Aude Billard and Maja J. Matarić. Learning human arm movements by imitation: Evaluation of a biologically inspired connectionist architecture. *Robotics and Autonomous Systems*, 37(2-3):145–160, 2001. ISSN 09218890. doi: 10.1016/S0921-8890(01)00155-5.

[7] Sing Bing Kang. Robot Instruction by Human Demonstration. (7597), 1994.

[8] Boeing Commercial Airplanes. Statistical Summary of Commercial Jet Airplane Accidents. *Boeing Commercial Airplanes*, page 24, 2015.

[9] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[10] Ivan Bratko, Tanja Urbančič, and Claude Sammut. Behavioural Cloning: Phenomena, Results and Problems. *IFAC Proceedings Volumes*, 1995. ISSN 14746670. doi: 10.1016/S1474-6670(17)46716-4.

[11] Sylvain Calinon and Aude Billard. Learning of gestures by imitation in a humanoid robot. In *Imitation and Social Learning in Robots, Humans and Animals: Behavioural, Social and Communicative Dimensions*, pages 153–178. 2007. ISBN 978-0-511-48980-8. doi: 10.1017/CBO9780511489808.012.

[12] R A Chambers and Donald Michie. Man-Machine Co-operation on a Learning Task. In R D Parslow, R W Prowse, and R Elliot Green, editors, *Computer Graphics: Techniques and Applications*, pages 179–185. Springer US, Boston, MA, 1969. ISBN 978-1-4757-1320-6. doi: 10.1007/978-1-4757-1320-6_18.

[13] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path Integral Guided Policy Search. *arXiv:1610.00529 [cs]*, October 2016.

[14] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning. *arXiv:1703.03078 [cs]*, March 2017.

[15] Seungwon Choi, Suseong Kim, and H. Jin Kim. Inverse reinforcement learning control for trajectory tracking of a multirotor UAV. *International Journal of Control, Automation and Systems*, 15(4):1826–1834, August 2017. ISSN 20054092. doi: 10.1007/s12555-015-0483-3.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, Miami, FL, June 2009. IEEE. ISBN 978-1-4244-3992-8. doi: 10.1109/CVPR.2009.5206848.

[17] Rüdiger Dillmann, M Kaiser, and Ales Ude. Acquisition of elementary robot skills from human demonstration. *International Symposium on Intelligent Robotics Systems*, 1(7274):185–192, 1995. ISSN 19232926. doi: 10.1.1.45.7236.

[18] Bruno Dufay and Jean Claude Latombe. An Approach to Automatic Robot Programming Based on Inductive Learning. *The International Journal of Robotics Research*, 3(4):3–20, 1984. ISSN 17413176. doi: 10.1177/027836498400300401.

[19] Yaakov Engel, Peter Szabo, and Dmitry Volkinshtein. Learning to Control an Octopus Arm with Gaussian Process Temporal Difference Methods. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, NIPS'05, pages 347–354, Cambridge, MA, USA, 2005. MIT Press.

[20] Philipp Ennen, Pia Bresenitz, Rene Vossen, and Frank Hees. Learning Robust Manipulation Skills with Guided Policy Search via Generative Motor Reflexes. *arXiv:1809.05714 [cs]*, September 2018.

[21] Nasser Esmaili, Claude Sammut, and G. M. Shirazi. Behavioural cloning in control of a dynamic system. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 3:2904–2909, 1995. ISSN 08843627.

[22] Silvia Ferrari and Robert F. Stengel. Online Adaptive Critic Flight Control. *Journal of Guidance, Control, and Dynamics*, 27(5):777–786, 2004. ISSN 0731-5090. doi: 10.2514/1.12597.

[23] C. Finn, M. Zhang, J. Fu, W. Montgomery, X. Tan, Z. McCarthy, B. Stadie, E. Scharff, and S. Levine. Guided Policy Search Code Implementation. 2016. Software available from rll.berkeley.edu/gps.

[24] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep Spatial Autoencoders for Visuomotor Learning. *arXiv:1509.06113 [cs]*, September 2015.

[25] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[26] P. Gaussier, S. Moga, M. Quoy, and J. P. Banquet. From perception-action loops to imitation processes: A bottom-up approach of learning by imitation. *Applied Artificial Intelligence*, 12(7-8):701–727, 1998. ISSN 10876545. doi: 10.1080/088395198117596.

[27] Weiqiao Han, Sergey Levine, and Pieter Abbeel. Learning compound multi-step controllers under unknown dynamics. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6435–6442, Hamburg, Germany, September 2015. IEEE. ISBN 978-1-4799-9994-1. doi: 10.1109/IROS.2015.7354297.

[28] Gillian M. Hayes and John Demiris. *A Robot Controller Using Learning by Imitation*. University of Edinburgh, Department of Artificial Intelligence, 1994.

[29] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6):82–97, November 2012. ISSN 1053-5888. doi: 10.1109/MSP.2012.2205597.

[30] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, July 2006. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.2006.18.7.1527.

[31] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. (Nips), 2016.

[32] Jonathan Ho, Stefano Ermon, Ermon Cs, and Stanford Edu. Model-Free Imitation Learning with Policy Optimization. 48, 2016.

[33] Geir E. Hovland, Pavan Sikka, and Brenan J. McCarragher. Skill acquisition from human demonstration using a hidden Markov model. *Proceedings of IEEE International Conference on Robotics and Automation*, 3(April):2706–2711, 1996. ISSN 1050-4729. doi: 10.1109/ROBOT.1996.506571.

[34] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation Learning: A Survey of Learning Methods. *ACM Computing Surveys*, 50(2):1–35, April 2017. ISSN 03600300. doi: 10.1145/3054912.

[35] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a Quadrotor with Reinforcement Learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017. ISSN 2377-3766. doi: 10.1109/LRA.2017.2720851.

[36] K. Ikeuchi, M. Kawade, and T. Suehiro. Assembly task recognition with planar, curved and mechanical contacts. *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 688–694, 1993. ISSN 10504729. doi: 10.1109/ROBOT.1993.291879.

[37] Nursultan Imanberdiyev, Changhong Fu, Erdal Kayacan, and I-Ming Chen. Autonomous navigation of UAV by using real-time model-based reinforcement learning. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, volume 2016, pages 1–6, 2016. ISBN 978-1-5090-3549-6. doi: 10.1109/ICARCV.2016.7838739.

[38] Andrew Isaac and Claude Sammut. Goal-directed Learning to Fly. 2003.

[39] Kshitij Judah, Alan Fern, and Thomas G. Dietterich. Active Imitation Learning via Reduction to IID Active Learning. In *UAI*, pages 428–437, 2012.

[40] MW Kadous, Claude Sammut, and R. Sheh. Autonomous traversal of rough terrain using behavioural cloning. *the 3rd International Conference on Autonomous Robots and Agents*, (June), 2006.

[41] R. E. Kalman. When Is a Linear Control System Optimal? *Journal of Basic Engineering*, 1964. ISSN 00219223. doi: 10.1115/1.3653115.

[42] Beomjoon Kim, Amir massoud Farahmand, Joelle Pineau, and Doina Precup. Learning from limited demonstrations. In *NIPS*, pages 2859–2867, 2013. ISBN 0016-7037.

[43] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input. *Advanced Robotics*, 25(5):581–603, 2011. ISSN 01691864. doi: 10.1163/016918611X558261.

[44] Yasuo Kuniyoshi, Masayuki Inaba, and Hirochika Inoue. Learning by Watching: Extracting Reusable Task Knowledge from Visual Observation of Human Performance. *IEEE Transactions on Robotics and Automation*, 10(6):799–822, 1994. ISSN 1042296X. doi: 10.1109/70.338535.

[45] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4(6):1107–1149, 2004. ISSN 15324435. doi: 10.1162/1532443041827907.

[46] S. Levine, N. Wagener, and P. Abbeel. Learning contact-rich manipulation skills with guided policy search. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 156–163, May 2015. doi: 10.1109/ICRA.2015.7138994.

[47] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[48] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

[49] Sergey Levine and Vladlen Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems*, pages 207–215, 2013.

[50] Sergey Levine and Vladlen Koltun. Learning complex neural network policies with trajectory optimization. In *International Conference on Machine Learning*, pages 829–837, 2014.

[51] Sergey Levine, Zoran Popovic, Vladlen Koltun, Zoran Popovi$\backslash$'{c}, and Vladlen Koltun. Nonlinear Inverse Reinforcement Learning with Gaussian Processes. *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, pages 19–27, 2011. doi: 10.1177/1745691612459060.

[52] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-End Training of Deep Visuomotor Policies. *arXiv:1504.00702 [cs]*, April 2015.

[53] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2015. ISSN 1935-8237. doi: 10.1561/2200000006.

[54] S. Liu and H. Asada. Teaching and learning of deburring robots using neural networks. *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 339–345, 1993. ISSN 10504729. doi: 10.1109/ROBOT.1993.292197.

[55] Teppo Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics, Espoo*, 22, 2011.

[56] William Montgomery and Sergey Levine. Guided Policy Search as Approximate Mirror Descent. *arXiv:1607.04614 [cs]*, July 2016.

[57] William Montgomery, Anurag Ajay, Chelsea Finn, Pieter Abbeel, and Sergey Levine. Reset-Free Guided Policy Search: Efficient Deep Reinforcement Learning with Stochastic Initial States. *arXiv:1610.01112 [cs]*, October 2016.

[58] Mark Wilfried Mueller. Multicopter attitude control for recovery from large disturbances. *arXiv:1802.09143 [cs]*, February 2018.

[59] Jun Nakanishi, Jun Morimoto, Gen Endo, Gordon Cheng, Stefan Schaal, and Mitsuo Kawato. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47(2-3): 79–91, 2004. ISSN 09218890. doi: 10.1016/j.robot.2004.03.003.

[60] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[61] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement earning. *Springer Tracts in Advanced Robotics*, 21:363–372, 2006. ISSN 16107438. doi: 10.1007/11552246_35.

[62] Ivana Palunko, Aleksandra Faust, Patricio Cruz, Lydia Tapia, and Rafael Fierro. A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4896–4901, 2013. ISSN 10504729. doi: 10.1109/ICRA.2013.6631276.

[63] P.K. Pook and D.H. Ballard. Recognizing teleoperated manipulations. *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 578–585, 1993. ISSN 10504729. doi: 10.1109/ROBOT.1993.291896.

[64] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: An open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, volume 3, page 5, 2009.

[65] Nathan D Ratliff and J Andrew Bagnell. Imitation learning for locomotion and manipulation. *Humanoid Robots, 2007 \ldots*, 2007.

[66] Stéphane Ross and J Andrew Bagnell. Efficient Reductions for Imitation Learning. *Journal of Machine Learning Research - Proceedings Track*, 9:661–668, 2010. ISSN 15324435.

[67] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *Synlett*, 15(10):1664–1666, November 2010. ISSN 09365214. doi: 10.1055/s-0029-1217330.

[68] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In Derek Sleeman and Peter Edwards, editors, *In Proceedings of the Ninth International Conference on Machine Learning*, pages 385—-393. Morgan Kaufmann, San Francisco (CA), 1992. ISBN 1581138285. doi: 10.1016/B978-1-55860-247-2.50055-3.

[69] Stefan Schaal. *Is Imitation Learning the Route to Humanoid Robots?*, volume 3. 1999. ISBN 1364-6613. doi: 10.1016/S1364-6613(99)01327-3.

[70] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. 2015. ISSN 2158-3226. doi: 10.1063/1.4927398.

[71] Wolfram Schultz, Peter Dayan, and P Read Montague. A Neural Substrate of Prediction and Reward. page 8, March 1997.

[72] Benjamin Spector and Serge Belongie. Sample-Efficient Reinforcement Learning through Transfer and Architectural Priors. (3), 2018.

[73] Richard S. Sutton, Andrew G. Barto, and Ronald J. Williams. *Reinforcement Learning Is Direct Adaptive Optimal Control*, volume 12. 1992. ISBN 1066-033X. doi: 10.1109/37.126844.

[74] Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[75] R.S. Sutton and A.G. Barto. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998. ISSN 1045-9227. doi: 10.1109/TNN.1998.712192.

[76] E. Todorov and Weiwei Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306, Portland, OR, USA, 2005. IEEE. ISBN 978-0-7803-9098-0. doi: 10.1109/ACC.2005.1469949.

[77] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Vilamoura-Algarve, Portugal, October 2012. IEEE. ISBN 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1. doi: 10.1109/IROS.2012.6386109.

[78] S K Tso and K P Liu. Hidden Markov model for intelligent extraction of robot trajectory command from demonstrated trajectories. In *Industrial Technology, 1996. (ICIT '96), Proceedings of The IEEE International Conference On*, pages 294–298, 1996. ISBN 0-7803-3104-4. doi: 10.1109/ICIT.1996.601593.

[79] C.P. Tung and A.C. Kak. Automatic learning of assembly tasks using a DataGlove system. *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, 1:1–8, 1995. ISSN 1600-5368. doi: 10.1109/IROS.1995.525767.

[80] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. Adapting Deep Visuomotor Representations with Weak Pairwise Constraints. *arXiv:1511.07111 [cs]*, November 2015.

[81] Aleksandar Vakanski, Iraj Mantegh, Andrew Irish, and Farrokh Janabi-Sharifi. Trajectory learning for robot programming by demonstration using hidden markov model and dynamic time warping. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 42(4):1039–1052, 2012. ISSN 10834419. doi: 10.1109/TSMCB.2012.2185694.

[82] Michel Verhaegen, Stoyan Kanev, Redouane Hallouzi, Colin Jones, Jan MacIejowski, and Hafid Smail. *Fault Tolerant Flight Control - A Survey*, volume 399. 2010. ISBN 978-3-642-11689-6. doi: 10.1007/978-3-642-11690-2_2.

[83] Antony Waldock, Colin Greatwood, Francis Salama, and Thomas Richardson. *Learning to Perform a Perched Landing on the Ground Using Deep Reinforcement Learning*. 2017. doi: 10.1007/s10846-017-0696-1.

[84] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum Entropy Deep Inverse Reinforcement Learning. *arXiv:1507.04888 [cs]*, July 2015.

[85] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective Robot Reinforcement Learning with Distributed Asynchronous Guided Policy Search. *arXiv:1610.00673 [cs]*, October 2016.

[86] Yang Yu. Towards Sample Efficient Reinforcement Learning $_*$. *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5739–5743, 2018.

[87] Thomas R. Zentall. Imitation by animals: How do they do it? *Current Directions in Psychological Science*, 12(3):91–95, 2003. ISSN 09637214. doi: 10.1111/1467-8721.01237.

[88] Marvin Zhang, Xinyang Geng, Jonathan Bruce, Ken Caluwaerts, Massimo Vespignani, Vytas SunSpiral, Pieter Abbeel, and Sergey Levine. Deep Reinforcement Learning for Tensegrity Robot Locomotion. *arXiv:1609.09049 [cs]*, September 2016.

[89] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search. *arXiv:1509.06791 [cs]*, September 2015.

[90] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum Entropy Inverse Reinforcement Learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

V

Appendices

A

List of GPS applications

GPS has been used for a wide variety of simulated robotics tasks:

- Several MuJoCo[77] environments
  - Planar hopper [48]
  - Planar swimmer [47–50, 52]
  - Planar bipedal walker [48–50]
  - 2D navigation with obstacle avoidance [13, 25, 56]
  - 3-link reacher [25, 57]
  - 2D peg insertion [47, 52]
  - 3D peg insertion [25, 47, 52, 56, 57]
- Octopus arm (see: [19]) [47, 52]
- Control of a quadcopter [89]
- SUPERball [88]
- 7-DoF 2-finger robot arm
  - Reaching [85]
- OpenAI gym
  - 2-DoF Reacher [14]
- Door opening [14]
- 4-DoF gripper/pusher [14]

As well as physical robotics tasks:

- 7-DoF PR2 robot arm
  - Lego block stacking [24, 27, 46, 52]
  - Putting a ring on a peg [46, 52]
  - Toy airplane assembly [46, 52]
  - Putting shoe tree in shoe [46, 52]
  - Screwing top on pill bottle [46, 52]
  - Screwing top on water bottle [46, 52]
  - Applying a wrench to a bolt [27]

  - Wrench grasping [27]
  - Bag transfer [24]
  - Rice scoop [24]
  - Loop hook [24, 80]
  - Dish placement [25]
  - Cup pouring [25]
  - Hammer claw placement [57]
  - Block claw placement [57]
  - Hockey [14]
  - Power plug pulling [14]
- SUPERball [88]
- 7-DoF 2-finger robot arm
  - Door opening [13, 85]
  - Pick-and-place [13]

B

Preliminary experiments neural networks

In this chapter the structure of the ANNs used during the preliminary experiments are detailed.

B.1. Point-mass

Multi-modal input

1. 2-D convolutional layer:
    - 5×5 filters
    - 32 channels
    - Stride of 1
    - No padding
2. exponential linear unit (ELU) nonlinearity
3. 2×2 max-pooling
4. 2-D convolutional layer:
    - 3×3 filters
    - 32 channels
    - Stride of 1
    - No padding
5. ELU nonlinearity
6. 2×2 max-pooling

The output of this sub-network is then flattened, and concatenated with the position derivatives $[\dot{x}, \dot{y}]$, this forms the input for a fully-connected ANN with the following structure:

1. Hidden layer of size 200
2. ELU nonlinearity
3. Hidden layer of size 20
4. ELU nonlinearity
5. Output layer of size 2

C

Default parameter settings

This appendix describes the default settings of the MD-GPS algorithm used in this work, are used as a nominal point during the experiments.

C.1. General

- **Number of seeds:** 5
  To reduce the effect of randomness, each experiment is run for five times with different but consistent initial seeds.

- **Training iterations:** 50
  Based on some experimentation, fifty training iterations is enough to let the cost converge for the experiments that are performed in this thesis.

- **Number of initial-conditions:** 4
  This is the number of initial conditions and therefore sub-policies that are used to train the global-policy. For a real quadcopter application it would probably be better to train on a higher number of initial conditions, since this will improve generalization of the global-policy to different states. However, in the interest of time this number has been set to four.

- **Number of trajectory samples:** 5
  This is the number of trajectory samples obtained per initial-condition per training iteration. The value is based on available literature on GPS and prior experimentation shows it is a workable number.

- **Sampling time:** 3×10^{-2} s
  Prior experiments have shown that for the used quadcopter system, this sampling time works well.

- **Number of time steps:** 80
  With 80 time steps, each trajectory takes 2.4 s, which is enough time for the nonlinear controller to reach and keep the target-position. Also, prior experiments indicate that this amount of time steps still allows iLQG-FLM to optimize the trajectories.

- **Iteration at which to start supervised-learning:** 1
  Supervised training of the global-policy can start as soon as the sub-policies show stable trajectories. With the default settings this occurs at the first training iteration already.

- **Iterations at which to constrain w.r.t. sub-policies:** $1-4$
  These are the iterations at which the KL-divergence constraint in the trajectory optimization is calculated w.r.t. the previous sub-policies, instead of the global-policy linearization. The choice has been made to postpone constraining the optimization w.r.t. the global-policy linearization, since the global-policy might still be unstable. For the first 4 iterations the global-policy is in no way involved in either constraining or sampling.

- **Iterations at which to sample using sub-policies:** $1-9$
  From the 5^{th} iteration the KL-divergence used in the constraint is calculated w.r.t. the linearized global-policy. However, the global-policy is only being used to sample trajectories from the 10^{th} iteration onward. This will ensure the global-policy generates stable- and low-cost trajectories.

C.2. Sub-policies

- **Initial variance:** 1×10^{-2}
  Sub-policies are TVLGs, which are parameterized by $\boldsymbol{k}_t$, $\boldsymbol{K}_t$, $\boldsymbol{C}_t$, of which the first two are either generated from demonstration data (demonstration-based sub-policy initialization) or generated randomly (random sub-policy initialization). The covariance matrix at time t $\boldsymbol{C}_t$ must be set manually, and determines the amount of random variation in the trajectories, which can be thought of as exploration. Since sub-policies are only stable around their nominal trajectory, care should be taken in balancing exploration and stability.

  The initial covariance matrices are diagonal, with a variance of 1×10^{-2} for each input. Prior experiments have shown that this value gives good exploration while the quadcopter stays inside the stable region of the demonstration-based initial sub-policy.

- **Initial variance discount:** 1
  The initial variance discount is set to 1 which slightly equalizes variance along the nominal trajectory.

C.3. Trajectory optimization

- **Initial KL-divergence step size:** 1×10^{-1}
  When the dynamics- and cost-estimates are accurate, a large initial step size can speed-up training. However, a large value can also destabilize the training process when the update is in the wrong direction or it overshoots the optimum. Since the step size is adapted on-line and can quickly increase if performance is improving, the initial value is chosen conservatively based on prior experimentation. A value of 1×10^{-1} does not lead to destabilization and can quickly increase when allowed by the adaptation rule.

- **KL-divergence adaptation factor:** 1.33
  Each iteration the KL-divergence step size is adapted based on certain heuristics. The factor with which the step size is either in- or decreased determines the speed the algorithm adapts to dynamics- and cost-estimate accuracy. A large adaptation factor improves adaptation speed, but can also lead to destabilization of the algorithm. Based on prior experimentation a factor of 1.33 does not destabilize MD-GPS when applied to this quadcopter system, and still allows for moderately quick adaption.

C.4. Dynamics models

- **Pre-train dynamics?:** `TRUE`
  Both the TVLG dynamics models and the GMM dynamics prior will be trained before the first training iteration using demonstration trajectories. The algorithms used for training are the same as those during the normal training process.

- **Pre-training example gathering policy:** `High-quality demonstration`
  As mentioned before, dynamics pre-training examples are generated by adding input noise inside of the control-loop while using a generating policy as the controller. The default generating policy is the high-quality demonstration.

- **Number of dynamics pre-training samples:** 30
  This is the number of dynamics pre-training example trajectories that are sampled per initial-condition. Based on prior experimentation, it takes less than five iterations for the dynamics to be estimated with sufficient accuracy to make successful trajectory updates, with five trajectory samples per initial-condition per training iteration. To be conservative, the default number of dynamics pre-training trajectory samples per initial-condition has been set to 30, which is equals six training iterations.

- **Dynamics prior GMM maximum clusters:** 1000
  Each dynamics prior GMM has a maximum number of Gaussian clusters that it can define. The choice was made to set the maximum number of clusters to a value that ensures the dynamics prior is never

limited by it. Based on prior experimentation, a conservative value of 1000 clusters has been chosen, which ensures the limit is never reached with the default minimum samples per cluster and maximum trajectory storage settings.

- **Dynamics prior GMM minimum samples per cluster:** 20
  The minimum average amount of samples per GMM cluster. Based on prior experimentation a value of 20 seems to work well, balancing cluster resolution and accuracy.

- **Dynamics prior GMM maximum trajectory storage:** 10
  The maximum amount of trajectories a single GMM can remember. The choice was made to reduce this number down from 20 to reduce wall-clock time requirements. Prior experimentation has shown that the difference in training performance is quite small.

C.5. Global-policy

- **Pre-train global-policy?:** `TRUE`
  Before the first training iteration, the global-policy is pre-trained on demonstration trajectories. The training algorithm is the same as is being used during normal training iterations.

- **Pre-training example gathering policy:** `High-quality demonstration`
  As mentioned before, global-policy pre-training examples are generated by adding input noise inside of the control-loop while using a demonstration policy as the controller. The default demonstration policy is the high-quality demonstration.

- **Pre-training imitation-policy:** `High-quality demonstration`
  As mentioned before, dynamics pre-training examples are generated by adding input noise inside of the control-loop while using a demonstration policy as the controller. At each time step of the resulting trajectories, another (possibly different) demonstration policy is queried for its action based on the state. The default target-policy is the high-quality demonstration policy.

- **Global-policy pre-training samples:** 30
  This is the number of global-policy pre-training example trajectories that are sampled per initial-condition. Prior experimentation shows that 30 trajectories lead to a global-policy that imitates the demonstration behavior effectively enough to be useful to sample from.

- **Global-policy prior GMM maximum clusters:** 1000
  Each policy prior GMM has a maximum number of Gaussian clusters that it can define. The choice was made to set the maximum number of clusters to a value that ensures the dynamics prior is never limited by it. Based on prior experimentation, a conservative value of 1000 clusters has been chosen, which ensures the limit is never reached with the default minimum samples per cluster and maximum trajectory storage settings.

- **Global-policy GMM minimum samples per cluster:** 20
  The minimum average amount of samples per GMM cluster. Based on prior experimentation a value of 20 seems to work well, balancing cluster resolution and accuracy.

- **Global-policy GMM maximum trajectory storage:** 10
  The maximum amount of trajectories a single GMM can remember. The choice was made to reduce this number down from 20 to reduce wall-clock time requirements. Prior experimentation has shown that the difference in training performance is quite small.

D

Quadcopter model and controller

This chapter presents the non-linearl quadcopter model that is used during the main experiments of the thesis.

D.1. Dynamic model

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} & \dot{\boldsymbol{p}} & \boldsymbol{q} & \boldsymbol{\omega}_b & \dot{\boldsymbol{\omega}}_b & \boldsymbol{\alpha} \end{bmatrix}^T \tag{D.1}$$

Where $\boldsymbol{p}$ is the position vector in inertial coordinates, $\dot{\boldsymbol{p}}$ is the velocity vector w.r.t. the inertial frame in inertial coordinates, $\boldsymbol{q}$ is a unit-quaternion that rotates vectors from body- to inertial-frame, $\boldsymbol{\omega}_b$ is the angular-rate vector in body-coordinates, $\dot{\boldsymbol{\omega}}_b$ is the angular-acceleration vector in body-coordinates, and $\boldsymbol{\alpha}$ is the vector of propellor angular-rates.

The system of ordinary differential equations is as follows:

$$\dot{\boldsymbol{p}} = \dot{\boldsymbol{p}} \tag{D.2}$$

$$\ddot{\boldsymbol{p}} = \frac{1}{m}\boldsymbol{F} + \boldsymbol{G} \tag{D.3}$$

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \boldsymbol{\omega}_b^q \tag{D.4}$$

$$\dot{\boldsymbol{\omega}}_b = \boldsymbol{I}^{-1}\left(\boldsymbol{M}_b - \boldsymbol{\omega}_b \times \boldsymbol{I}\boldsymbol{\omega}_b\right) \tag{D.5}$$

$$\dot{\boldsymbol{\alpha}} = \frac{1}{I_p}\left(\boldsymbol{\tau} - \boldsymbol{\tau}_d\right) \tag{D.6}$$

Where m is the quadcopter mass, $\boldsymbol{F}$ is the vector of forces in inertial coordinates, $\boldsymbol{G}$ is the vector of gravitational acceleration in inertial coordinates, $\boldsymbol{\omega}_b^q$ is a quaternion with zero-valued scalar and complex part equal to the angular rate vector of the quadcopter in body-coordinates, $\otimes$ denotes the quaternion product, $\boldsymbol{I}$ is the mass-moment of inertia tensor, $\boldsymbol{M}_b$ is a vector of external moments in body-coordinates, I_p is the mass-moment of inertia of the propellers, $\boldsymbol{\tau}$ is the vector of motor-torques, and $\boldsymbol{\tau}_d$ is a vector of drag-moments per propeller.

$$\boldsymbol{F} = \boldsymbol{q} \otimes \boldsymbol{F}_b^q \otimes \bar{\boldsymbol{q}} \tag{D.7}$$

Where $\boldsymbol{F}_b^q$ is a quaternion with zero-valued scalar part, and a complex part equal to the vector of external forces in body-coordinates, and $\bar{\boldsymbol{q}}$ is the conjugate quaternion of $\boldsymbol{q}$.

$$\boldsymbol{F}_b = \boldsymbol{F}_{th,b} + \boldsymbol{F}_{d,b} \tag{D.8}$$

Where $\boldsymbol{F}_{th,b}$ is the force due to motor-thrust in body-coordinates, and $\boldsymbol{F}_{d,b}$ is the drag-force vector in body-coordinates.

The total thrust vector is the sum of the 4 thrust vectors per motor:

$$\boldsymbol{F}_{th,b} = \sum_{i=1}^{4} \boldsymbol{f}_{th,i,b} \tag{D.9}$$

The thrust per engine is proportional to the square of the propeller angular rate, and acts along the thrust unit vector, which is simply pointing along the body z-axis.

$$\boldsymbol{f}_{th,i,b} = k_l \cdot \alpha_i \cdot |\alpha_i| \cdot \hat{\boldsymbol{f}}_{i,b} = \begin{bmatrix} 0 & 0 & k_l \alpha_i |\alpha_i| \end{bmatrix}^T \quad for \quad i = 1,2,3,4 \tag{D.10}$$

Where k_l is the lift factor of the propellers, $\hat{\boldsymbol{f}}_{i,b}$ is the unit vector pointing in the direction of positive thrust for the i^{th} propeller.

Drag forces are simply modeled by assigning an independent drag factor for each direction of movement in the body frame, they are proportional to the square of the speed in each direction.

$$\boldsymbol{F}_{d,b} = -\boldsymbol{k}_d \odot \dot{\boldsymbol{p}}_b \odot \left|\dot{\boldsymbol{p}}_b\right| \tag{D.11}$$

Where $\boldsymbol{k}_d$ is a vector of three independent drag factors for the x-, y-, and z-velocity in the body frame, $\odot$ denotes the Hadamard product, $\dot{\boldsymbol{p}}_b$ is the velocity vector w.r.t. the inertial-frame in body-coordinates, and the absolute operator in $\left|\dot{\boldsymbol{p}}_b\right|$ is acting in element-wise fashion.

The gravitational acceleration vector in inertial frame coordinates is simply pointing along the z-axis:

$$\boldsymbol{G}_I = \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^T \tag{D.12}$$

Where g is the constant of gravitational acceleration.

The moments around the body-axes of the quadcopter are split up into two terms:

$$\boldsymbol{M}_b = \boldsymbol{M}_{th,b} + \boldsymbol{M}_{\tau,b} \tag{D.13}$$

Where $\boldsymbol{M}_{th,b}$ is the moment caused by the thrust of the i^{th} motor, and $\boldsymbol{M}_{\tau,b}$ is the moment caused by the torque of the i^{th} motor.

$$\boldsymbol{M}_{th,b} = \sum_{i=1}^{4} \boldsymbol{r}_{i,b} \times \boldsymbol{f}_{th,i,b} \tag{D.14}$$

Where $\boldsymbol{r}_{i,b}$ is the moment-arm of the i^{th} motor in body-coordinates.

$$\boldsymbol{M}_{\tau,b} = -\sum_{i=1}^{4} \tau_i \cdot \boldsymbol{u}_{m,i,b} \tag{D.15}$$

Where τ_i is the torque of the i^{th} motor, and $\boldsymbol{u}_{m,i,b}$ is the unit vector pointing in the direction of positive motor torque for the i^{th} motor in body-coordinates.

The drag moment around a motor is proportional to the square of the motor angular rate:

$$\boldsymbol{\tau}_d = k_{pd} \cdot \boldsymbol{\alpha} \odot |\boldsymbol{\alpha}| \tag{D.16}$$

Where k_{pd} is the drag-coefficient of the propellers, and the absolute operator in $|\boldsymbol{\alpha}|$ is acting in element-wise fashion.

D.2. Controller

This section presents the non-linear controller that is used to generate high-quality trajectories with the quadcopter. It is inspired by proportional tilt-prioritized control by Mueller in [58], prioritizing tilt-control (x- and y-axis) over control of rotation about the z-axis.

As stated in [58], quadcopters are characterized by an ability to generate much higher moments about their x- and y-axis, then their z-axis. Control of the position of a quadcopter can be seen as control of the direction and magnitude of the thrust vector. These two variables can be controlled separately, since moments can be generated completely independent of the total thrust (up to motor saturation)[58]. By assuming we can control the thrust vector much faster than the position dynamics, the position dynamics can be made approximately linear and of second order:

$$\ddot{\boldsymbol{p}}_{ref} = \boldsymbol{K}_p^T \left(\boldsymbol{p}_{ref} - \boldsymbol{p}\right) + \boldsymbol{K}_{\dot{p}}^T \left(\dot{\boldsymbol{p}}_{ref} - \dot{\boldsymbol{p}}\right) \tag{D.17}$$

Where $\ddot{\boldsymbol{p}}_{ref}$ is the desired acceleration, $\boldsymbol{K}_p^T$ and $\boldsymbol{K}_{\dot{p}}^T$ are gain matrices, $\dot{\boldsymbol{p}}_{ref}$ is the desired velocity, and $\boldsymbol{p}_{ref}$ is the desired position.

By correcting for gravitational acceleration and knowing the mass, the desired thrust vector is calculated:

$$\boldsymbol{F}_{ref} = m\left(\ddot{\boldsymbol{p}}_{ref} - \boldsymbol{G}\right) \tag{D.18}$$

Where $\boldsymbol{F}_{ref}$ is the desired thrust vector in inertial frame coordinates, m is the mass of the quadcopter, and $\boldsymbol{G}$ is the vector of gravitational acceleration in inertial frame coordinates.

Using the attitude quaternion $\boldsymbol{q}$, the desired thrust vector in body coordinates is determined:

$$\boldsymbol{F}_{ref,b} = \boldsymbol{q} \otimes \boldsymbol{F}_{ref} \otimes \bar{\boldsymbol{q}} \tag{D.19}$$

The goal is to rotate the thrust vector so it lies along the desired thrust vector, by tilting the quadcopter. To this end, the quaternion of smallest rotation is found which rotates $\hat{\boldsymbol{f}}_b$ to $\boldsymbol{F}_{ref,b}$, using the $FindQuaternion$ function:

Function 1 $FindQuaternion$ finds a unit-quaternion with the smallest rotation arc that rotates a vector parallel to $\boldsymbol{u}$ such that $\boldsymbol{u}' = \boldsymbol{q} \otimes \boldsymbol{u} \otimes \bar{\boldsymbol{q}}$ is parallel to $\boldsymbol{v}$

function FINDQUATERNION($\boldsymbol{u}$, $\boldsymbol{v}$)
  $\tilde{\boldsymbol{q}} = \left(\|\boldsymbol{u}\| \cdot \|\boldsymbol{v}\| + \boldsymbol{u} \cdot \boldsymbol{v} \quad , \quad \boldsymbol{u} \times \boldsymbol{v}\right)$
  $\boldsymbol{q} = \frac{\tilde{\boldsymbol{q}}}{\|\tilde{\boldsymbol{q}}\|}$
  return $\boldsymbol{q}$
end function

$$\boldsymbol{q}_{xy,ref} = FindQuaternion\left(\hat{\boldsymbol{f}}_b, \boldsymbol{F}_{ref,b}\right) \tag{D.20}$$

Where $\boldsymbol{q}_{xy,ref}$ is the unit quaternion that rotates $\boldsymbol{f}_b$ to lie along $\boldsymbol{F}_{ref,b}$, with smallest rotation magnitude possible.

After this tilting movement, the thrust vector (and therefore the body z-axis) will be aligned with the reference thrust vector. However, the rotation about the z-axis is still left unconstrained. At low tilt, this rotation is akin to the yaw-angle used in aeronautical navigation and it will be referred to as z-angle for the remainder of this section. Usually, a control system is provided with some desired yaw-angle, but in this work the z-angle will be constrained by using a reference body x-axis in inertial coordinates; a pointing direction. By constraining the z-angle in this way, the same technique can be used as for the thrust (body z-axis) pointing direction. The desired body x-axis in inertial coordinates will be chosen to be the following:

$$\boldsymbol{e}^b_{x,ref} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T \tag{D.21}$$

Which we then express in body-coordinates using $\boldsymbol{q}$:

$$\boldsymbol{e}^b_{x,ref,b} = \boldsymbol{q} \otimes \boldsymbol{e}^b_{x,ref} \otimes \bar{\boldsymbol{q}} \tag{D.22}$$

Aligning the thrust vector with its desired direction is prioritized over controlling the z-angle, this is facilitated by the fact that quadcopters can achieve much higher moments around the x- and y-axis than the z-axis. To further ensure the z-angle-control does not interfere with tilt-control, only rotation about the thrust-axis will be allowed. This is achieved by projecting $\boldsymbol{e}^b_{x,ref,b}$ onto the plane defined by normal-vector $\boldsymbol{F}_{ref,b}$, which is the same as projecting it onto the xy-plane of the body coordinate frame, as shown in eq. (D.23).

Function 2 $PlaneProjection$ finds the projection $\boldsymbol{w}$ of $\boldsymbol{u}$ onto a plane defined by normal-vector $\boldsymbol{v}$

function PLANEPROJECTION($\boldsymbol{u}$, $\boldsymbol{v}$)
  $\boldsymbol{w} = \boldsymbol{u} - \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\boldsymbol{v} \cdot \boldsymbol{v}} \boldsymbol{v}$
  return $\boldsymbol{w}$
end function

$$\tilde{\boldsymbol{e}}^b_{x,ref,b} = PlaneProjection\left(\boldsymbol{e}^b_{x,ref,b}, \boldsymbol{F}_{ref,b}\right) = \boldsymbol{e}^b_{x,ref,b} \odot \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^T \tag{D.23}$$

As for the tilt-control, the z-angle quaternion is the one that rotates the body x-axis to the projected desired pointing direction, while using the shortest rotation arc:

$$\boldsymbol{q}_{z,ref} = FindQuaternion\left(\boldsymbol{e}^b_{x,b}, \tilde{\boldsymbol{e}}^b_{x,ref,b}\right) \tag{D.24}$$

Where $\boldsymbol{q}_{z,ref}$ will always be a pure rotation around the body z-axis, since both $\boldsymbol{e}^b_{x,b}$ and $\tilde{\boldsymbol{e}}^b_{x,ref,b}$ lie in the xy-plane.

To utilize $\boldsymbol{q}_{xy,ref}$ and $\boldsymbol{q}_{z,ref}$ for control, they will be converted into their axis-angle representations, as shown in eqs. (D.26) and (D.27).

$$\ln \boldsymbol{q} = \begin{cases} \ln \|\boldsymbol{q}\| + \frac{\vec{\boldsymbol{q}}}{\|\vec{\boldsymbol{q}}\|} \arccos \frac{q_0}{\|\boldsymbol{q}\|}, & \|\vec{\boldsymbol{q}}\| \neq 0 \\ \ln \|\boldsymbol{q}\|, & \|\vec{\boldsymbol{q}}\| = 0 \end{cases} \tag{D.25}$$

Function 3 $AxisAngleRepresentation$ converts quaternion $\boldsymbol{q}$ to an axis-angle representation, where $\hat{\boldsymbol{\theta}}$ and ρ are a unit axis and the magnitude of rotation, respectively.

1: **function** AXISANGLEREPRESENTATION($\boldsymbol{q}$)
2: $\quad \boldsymbol{\theta} = 2 \ln \boldsymbol{q}$
3: $\quad \rho = \|\boldsymbol{\theta}\|$
4: $\quad \hat{\boldsymbol{\theta}} = \frac{\boldsymbol{\theta}}{\theta}$
5: $\quad$ **return** $\hat{\boldsymbol{\theta}}$, ρ
6: **end function**

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{xy,ref,b}, \tilde{\rho}_{xy,ref,b} &= AxisAngleRepresentation(\boldsymbol{q}_{xy,ref}) \\ \rho_{xy,ref,b} &= \frac{\tilde{\rho}_{xy,ref,b}}{\pi} \end{aligned} \tag{D.26}$$

Where $\hat{\boldsymbol{\theta}}_{xy,ref,b}$ and $\rho_{xy,ref,b}$ are the axis- and angle of rotation for tilt-control, respectively. The axis of rotation is in body-coordinates and can be directly used as the direction of a desired moment-vector, while the magnitude of rotation $\rho_{xy,ref,b}$ can be interpreted as a difference between desired and actual orientation, lying in the interval $[0,1]$ since it has been normalized.

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{z,ref,b}, \tilde{\rho}_{z,ref,b} &= AxisAngleRepresentation(\boldsymbol{q}_{z,ref}) \\ \rho_{z,ref,b} &= \frac{\tilde{\rho}_{z,ref,b}}{\pi} \end{aligned} \tag{D.27}$$

Where $\hat{\boldsymbol{\theta}}_{z,ref,b}$ and $\rho_{z,ref,b}$ are the axis- and angle of rotation for z-angle control, respectively.

In the case when $\boldsymbol{F}_{ref,b}$ is almost equal to $\boldsymbol{e}^b_{x,ref,b}$, the projection onto the xy-plane becomes very small and its direction becomes very sensitive to small changes in orientation. In this case it is quite useless to put much force into controlling the z-angle. To adapt the control effort to this problem, the rotation magnitude is multiplied by the norm of $\tilde{\boldsymbol{e}}^b_{x,ref,b}$. This value lies in the interval $[0,1]$ and goes to zero as $\boldsymbol{F}_{ref,b}$ becomes equal to $\boldsymbol{e}^b_{x,ref,b}$:

$$\tilde{\rho}_{z,ref,b} = \rho_{z,ref,b} \left\| \tilde{\boldsymbol{e}}^b_{x,ref,b} \right\| \tag{D.28}$$

Where $\tilde{\rho}_{z,ref,b}$ is the adapted rotation magnitude for the z-angle control.

Now, the reference value for the angular acceleration can be constructed:

$$\dot{\boldsymbol{\omega}}_{ref,b} = \boldsymbol{K}_\omega(\boldsymbol{\omega}_{ref} - \boldsymbol{\omega}) - \boldsymbol{K}_{xy}\rho_{xy,ref,b}\hat{\boldsymbol{\theta}}_{xy,ref,b} - \boldsymbol{K}_z\tilde{\rho}_{z,ref,b}\hat{\boldsymbol{\theta}}_{z,ref,b} \tag{D.29}$$

Where $\boldsymbol{K}_\omega$, $\boldsymbol{K}_{xy}$, and $\boldsymbol{K}_z$ are gain matrices.

Using the dynamics equation for angular acceleration from eq. (D.5) the desired moment vector can be calculated:

$$\boldsymbol{M}_{ref,b} = \boldsymbol{I}\left(\boldsymbol{I}\dot{\boldsymbol{\omega}}_{ref,b} + \boldsymbol{\omega}_b \times \boldsymbol{I}\boldsymbol{\omega}_b\right) \tag{D.30}$$

Since the thrust vector coincides with the body z-axis, the moment around the z-axis is fully generated by the sum of motor torques:

$$\boldsymbol{\tau}_{z,ref} = \frac{M_z}{4}\begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}^T \tag{D.31}$$

Where M_z is the moment around the body z-axis, and the vector of plus and minus ones is there to account for the configuration of positive motor torque.

The x- and y-moment are generated by a thrust differential over the motors. The differential thrust vector can be calculated as follows:

$$\Delta \boldsymbol{f} = \frac{M_x}{4r_x}\begin{bmatrix}-1 & -1 & 1 & 1\end{bmatrix}^T + \frac{M_y}{4r_y}\begin{bmatrix}1 & -1 & -1 & 1\end{bmatrix}^T \tag{D.32}$$

Where M_x and M_y are the moments around the x- and y-axis, respectively, r_x and r_y are the motor offsets from the center of gravity in x- and y-direction, and the vectors with plus and minus ones are there to account for the moment contribution of positive motor thrust.

The desired thrust per motor is the sum of the norm of the desired thrust vector $\boldsymbol{F}_{ref}$ and the differential thrust vector. Since the mean of the latter is zero by construction, the total thrust will be equal in magnitude to the norm of the desired thrust vector at all times. By using eq. (D.10), the desired motor angular rate can be calculated:

$$\begin{aligned}\boldsymbol{\beta} &= \frac{1}{k_f}\left(\frac{\left\|\boldsymbol{F}_{ref}\right\|}{4} + \Delta \boldsymbol{f}\right)\\ \boldsymbol{\alpha}_{ref} &= \operatorname{sign}\boldsymbol{\beta}\sqrt{|\boldsymbol{\beta}|}\end{aligned} \tag{D.33}$$

The input to the system is the motor torque vector $\boldsymbol{\tau}$, which is calculated using a proportional feedback to force the motor angular rates towards their desired values. Also, the desired motor torque is added directly for z-angle control. To force the motor dynamics to be invariant to $\boldsymbol{\alpha}$, the motor drag torque $\boldsymbol{\tau}_d$ is added.

$$\boldsymbol{\tau} = K_\alpha\left(\boldsymbol{\alpha}_{ref} - \boldsymbol{\alpha}\right) + \boldsymbol{\tau}_{z,ref} + \boldsymbol{\tau}_d \tag{D.34}$$

Where K_α is a gain.

The parameters of the quadcopter model and their descriptions are presented in appendix D.2, while the controller parameters are shown in appendix D.2.

Parameter	Description	Value
m	mass	4.68×10^{-1} kg
g	constant of gravitational acceleration	9.81 m·s^2
I_{xx}	principal mass-moment of inertia around body x-axis	4.856×10^{-3} kg·m^2
I_{yy}	principal mass-moment of inertia around body y-axis	4.856×10^{-3} kg·m^2
I_{zz}	principal mass-moment of inertia around body z-axis	8.801×10^{-3} kg·m^2
I_p	propeller mass-moment of inertia around rotation axis	3.357×10^{-5} kg·m^2
k_l	propeller lift factor	2.98×10^{-6} N·s^2
$k_{d,x}$	drag factor for movement along body x-axis	4.33×10^{-3} N·s^2·m^{-2}
$k_{d,y}$	drag factor for movement along body y-axis	4.33×10^{-3} N·s^2·m^{-2}
$k_{d,z}$	drag factor for movement along body z-axis	4.33×10^{-3} N·s^2·m^{-2}
r_x	motor offset from center of gravity in x-direction	2.25×10^{-1} m
r_y	motor offset from center of gravity in y-direction	2.25×10^{-1} m
r_z	motor offset from center of gravity in z-direction	0.0 m
k_{pd}	propeller drag factor	1.140×10^{-7} N·s^2
τ_{max}	maximum motor torque	8.0×10^{-1} N·m

Table D.1: Quadcopter model parameters, which have been copied from [55].

As can be seen in fig. D.1, the dynamics of the propellers are such that a very high sampling frequency is required to avoid aliasing. Some quick calculations show how the propeller dynamic are definitely operating

Parameter	Description	Value
$K_{p,x}$	position feedback gain in x-direction	4.0
$K_{p,y}$	position feedback gain in y-direction	4.0
$K_{p,z}$	position feedback gain in z-direction	4.0
$K_{\dot{p},x}$	velocity feedback gain in x-direction	4.0
$K_{\dot{p},y}$	velocity feedback gain in y-direction	4.0
$K_{\dot{p},z}$	velocity feedback gain in z-direction	4.0
$K_{\omega,x}$	angular rate feedback gain in x-direction	1.75×10^1
$K_{\omega,y}$	angular rate feedback gain in y-direction	1.75×10^1
$K_{\omega,z}$	angular rate feedback gain in z-direction	4.9×10^3
$K_{xy,x}$	tilt-control feedback gain in x-direction	5.0×10^2
$K_{xy,y}$	tilt-control feedback gain in y-direction	5.0×10^2
$K_{xy,z}$	tilt-control feedback gain in z-direction	5.0×10^2
$K_{z,x}$	yaw-angle control feedback gain in x-direction	0.0
$K_{z,y}$	yaw-angle control feedback gain in y-direction	0.0
$K_{z,z}$	yaw-angle control feedback gain in z-direction	7.0×10^3
K_{α}	motor angular rate feedback gain	1.0

Table D.2: Quadcopter nonlinear controller parameters.

at a different time scale than attitude- and position dynamics:

$$\begin{aligned} \dot{\alpha}_{max} &= \frac{\tau_{max}}{I_p} && \approx 2.38 \times 10^4 \,\text{rad}\cdot\text{s}^{-2} \\ \alpha_{max} &= \sqrt{\frac{\tau_{max}}{k_{pd}}} \\ \dot{\omega}_{max} &= \frac{4 r_x k_l \alpha_{max}^2}{I_{xx}} && \approx 3.88 \times 10^3 \,\text{rad}\cdot\text{s}^{-2} \\ \ddot{p}_{max} &= \frac{k_l \alpha_{max}^2}{m} && \approx 1.79 \times 10^2 \,\text{m}\cdot\text{s}^{-2} \end{aligned} \tag{D.35}$$

Where $\dot{\alpha}_{max}$ is the largest possible propeller rotational acceleration, α_{max} the largest possible propeller angular rate, $\dot{\omega}_{max}$ the largest possible angular acceleration, and $\ddot{p}_{max}$ the largest possible linear acceleration.

D.2.1. System Simplification

The problem with the current quadcopter model is that the propeller dynamics operate at a very different time scale from the rest of the dynamics. The sampling frequency required to prevent aliasing of the propeller dynamics is too high to allow for any interesting quadcopter dynamics to show in a number of samples that is still practical for use during trajectory-optimization. Therefore, the assumptions is made that the quadcopter can generate any combination of moments and total thrust[58]. This assumptions allows the propeller dynamics to be bypassed, using the desired moment vector $\boldsymbol{M}_{ref,b}$ and total thrust F_{ref} as inputs directly. This leaves us with the problem of defining maximum (and minimum) values for this new input, to prevent aliasing. Using the maximum motor torque and some other parameters, feasible input limits are calculated, as shown in eq. (D.36).

$$\begin{aligned} F_{max} &= 4 k_l \alpha_{max}^2 && \approx 8.36 \times 10^1 \,\text{N} \\ M_{x,max} &= 4 r_y k_l \alpha_{max}^2 && \approx 1.88 \times 10^1 \,\text{N}\cdot\text{m} \\ M_{y,max} &= 4 r_x k_l \alpha_{max}^2 && \approx 1.88 \times 10^1 \,\text{N}\cdot\text{m} \\ M_{z,max} &= 4 \tau_{max} && \approx 3.20 \,\text{N}\cdot\text{m} \end{aligned} \tag{D.36}$$

Summarizing, before the input is allowed to affect the system, its magnitude is clamped using the values above.

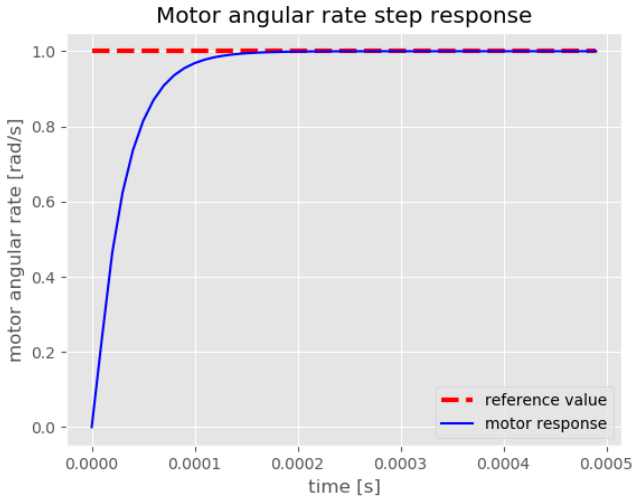


Figure D.1: Motor angular rate response to a step input. Note that it takes less than 2.0×10^{-4} s to converge to the reference value, requiring a very high sampling frequency to avoid aliasing (1.0×10^{-5} s in the run presented in this figure).

D.2.2. Simulation Results

This section shows the response of the quadcopter model when combined with the nonlinear controller. Results have been generated for two cases:

- **Attitude regulation**, where position and velocity are ignored and $\boldsymbol{F}_{ref}$ is set parallel to the inertial z-axis. The initial attitude quaternion is randomized. Furthermore:

$$\begin{aligned} \boldsymbol{F}_{ref} &= \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \\ \boldsymbol{\omega}_{0,b} &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \\ \boldsymbol{\omega}_{ref,b} &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \end{aligned} \tag{D.37}$$

- **Position regulation**, where the quadcopter is intialized with a position offset, which is either only in x-direction or completely randomized in three directions. Furthermore:

$$\begin{aligned} \dot{\boldsymbol{p}}_0 &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \\ \dot{\boldsymbol{p}}_{ref} &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \\ \boldsymbol{q}_0 &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T \\ \boldsymbol{\omega}_{0,b} &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \\ \boldsymbol{\omega}_{ref,b} &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \end{aligned} \tag{D.38}$$

For both cases the reference body x-axis pointing direction in inertial coordinates $\boldsymbol{e}^b_{x,ref}$ is set to $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$.

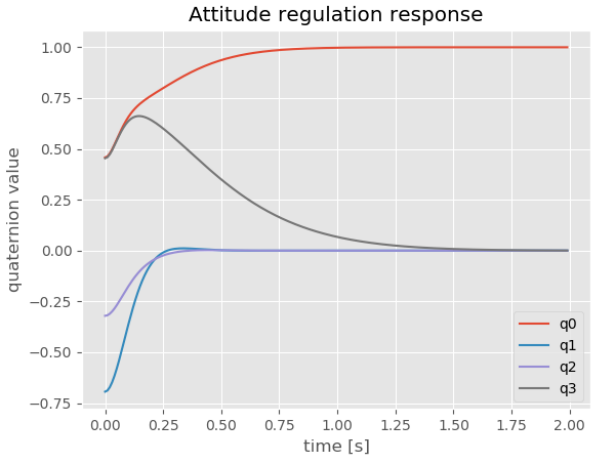


Figure D.2: Attitude quaternion response to attitude regulation case. Note how q_1 and q_2 converge at around 0.4 s, much quicker than q_3 at 1.5 s; the effect of tilt-prioritized control.

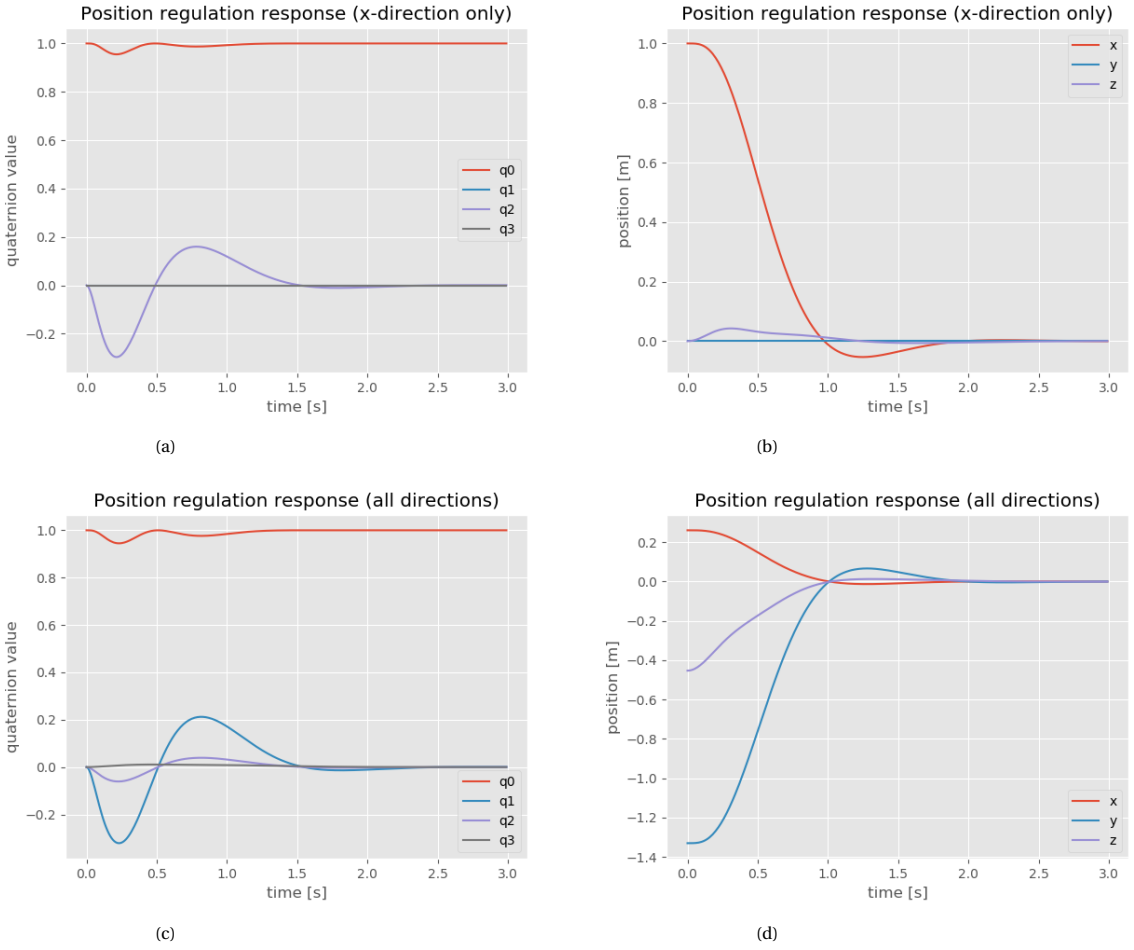


Figure D.3: Attitude and position response to the position regulation case, for both an offset in x-direction only (figs. D.3a and D.3b) and in all 3 directions (figs. D.3c and D.3d). Position converges around 2.0 s.