SCIENCES SORBONNE UNIVERSITÉ

ISIR INSTITUT DES SYSTÈMES INTELLIGENTS ET DE ROBOTIQUE

Manuscript for

L'HABILITATION À DIRIGER DES RECHERCHES

of Sorbonne University - Paris

Specialty: COMPUTER SCIENCE

Defended by

Nicolas PERRIN-GILBERT

Ingredients for Motion Planning-powered Reinforcement Learning

Defended on October, 17th 2024

Aleksandra Faust	Research Director - Google Deepmind, USA	Reviewer
Matthieu Geist	Professor from University of Lorraine on leave at Cohere, France	Examiner
Nicolas Mansard	Research Director - LAAS-CNRS, France	Reviewer
Jochen J. Steil	Professor - Technical University of Braunschweig, Germany	Reviewer
Nicolas Thome	Professor - Sorbonne University, France	Examiner

Acknowledgements

First and foremost, I thank all the members of my jury: Aleksandra Faust, Nicolas Mansard, Jochen Steil, Matthieu Geist and Nicolas Thome. I am deeply honored that you accepted my invitation.

Contents

CHAPTER 1

Introduction

Let's be honest: at first sight, writing one's *Habilitation à diriger des recherches* (HDR) ranks high in the long list of useless bureaucratic tasks, somewhere between the *rapport à vague* of the CNRS and your typical collaborative project deliverable.

For the HDR, you have to write a relatively long document about things you've already published, and try to convince reviewers that this somehow makes you qualified for the job of grad student supervision you've already been doing for the last five to ten years, if not more. As much as I would have liked to postpone this task indefinitely, I fell in a trap and made a sort of academic promise that forces me to surrender now, because a student I have been supervising for over two years needs me to become a qualified supervisor shortly before he can graduate and no longer needs my services.

So here I am, about to perform the dreaded task, taking time off from actual research, and wondering: how can I make this as quick and painless as possible, both for me and my reviewers? I could ask around for advice, and, to be fair, I heard stories about people finding pleasure in the exercice, but, just like the *Big Foot* or the *Loch Ness Monster*, these are the sort of creatures you don't usually see with your own eyes. So, right now, I find myself a bit lonely in front of the bureaucratic monstrosity. Asking a large language model to handle this for me feels like a weak move. I intend to defend my human dignity by doing in dozens of hours what the machine would have done in seconds. At least, it will be done with my own sweat and tears, and when in a couple of years from now the HDR no longer exists, I will tell younger folks how lucky they are and how different things were back in my time.

Cutting to the chase, I will not pretend that I managed to give a sense of unity to the different research projects I've done in the past, which would probably be the ideal scenario for an HDR. I admit that I wandered in many directions and that there is no point trying to connect the dots backwards. What I will do instead is dig up just four ideas from my past works, ideas that are vaguely connected, that I find simple enough to explain briefly, that I am at least a little bit proud of, and perhaps ideas that would be worth working on again in the future.

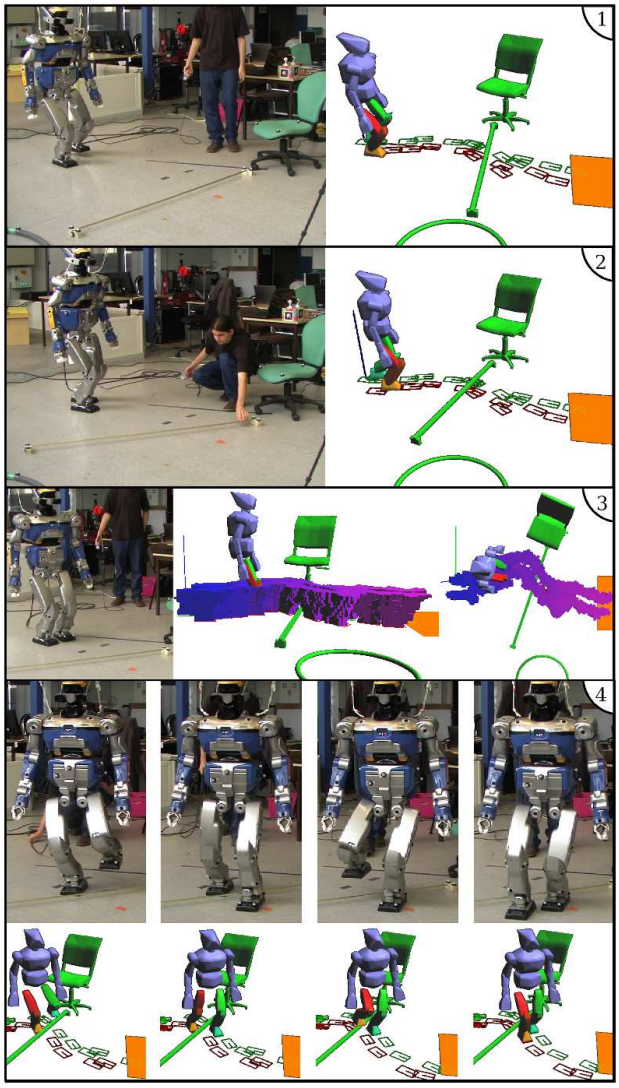


Figure 1.1: Experiment from [Perrin *et al.* 2012a]. (1): HRP-2 starts to execute the sequence initially found via footstep planning. (2): the bar is suddenly moved, and the current sequence of steps would lead to collisions. (3): while walking, a new sequence of steps toward the goal is computed in real-time. A concatenation of the swept volume approximations is used to ensure the avoidance of contacts with the bar. (4): the robot steps over the bar while optimizing the rest of the path toward the goal.

I will try to expose these ideas concisely in separate chapters—mainly by copying large chunks of the related papers, because why wouldn't I?—, and discuss some directions I would consider if I were to revisit them now. The ideas I selected are all related to an objective I've had for a long time: trying to boost exploration in Reinforcement Learning (RL) with sampling-based motion planning.

The need for efficient exploration in robotic policy learning is well known and has been tackled in various ways [Ecoffet *et al.* 2021, Xie *et al.* 2016, Nair *et al.* 2018, Stulp 2012, OpenAI *et al.* 2021], but there has been relatively few attempts to exploit sampling-based motion planning, which is arguably the most successful approach to efficiently explore robot configuration spaces.

One such attempt, RL-RRT [Chiang *et al.* 2019], lays down a foundational framework for motion planning-powered reinforcement learning. It suggests to train a control policy to move and avoid obstacles locally together with a reachability estimator that predicts the time needed to go from a configuration A to a configuration B in the presence of obstacles. A sampling-based motion planner is then used to grow a global exploration tree. To extend a path of the tree, it relies on a simulator to forward propagate the dynamics of the robot under its control policy and compute new states, i.e. new nodes in the exploration tree. The reachability estimator is used as a distance to determine how to extend the exploration tree.

RL-RRT has been successfully aplied to wheeled robots, but in other contexts, adapting the framework with new ingredients is required. In particular, the ability to simulate the effects of a controller in the state space cannot be taken for granted. This is especially true for legged robots, which are my main interest in terms of applications. Although some results presented in this manuscript are general, the primary motivation is to develop useful ingredients for the planning and generation of legged robot motions, a problem illustrated in Figure 1.1 showing a 2011 experiment with the HRP-2 robot.

Before moving on to the next chapter, I'd like to add a few words. It is common knowledge that nobody reads HDR theses, so if you are reading these lines, you are probably a member of my jury. I have already thanked you in the acknowledgements, but I would like to do it again. The one nice thing about the HDR is that I will get to spend some time with you! You are all researchers I admire, and I'll be looking forward to talking with you during the defense.

CHAPTER 2

Planning discrete motions like continuous ones

Contact-based locomotion, like humans or all legged animals do, is one of the most geometrically fascinating types of motion. The motion is continuous and happens in the 3D space, but it entirely relies on contacts, which are discrete events occuring on a 2D surface within this 3D space (equal to the union of object or ground surfaces on which contacts are possible). So, it is hybrid in two ways: it combines both 3D and 2D spaces, as well as continuous and discrete changes. To generate this type of motion, the usual approach is to plan the discrete sequence of contacts first, and then the 3D motion that achieves this sequence in a collision-free manner.

Unfortunately, planning contact sequences is a very particular and difficult problem. Contacts can be cast as variables of an optimization problem, which can be turned into mixed integer optimization [Ibanez *et al.* 2014] or sequential quadratic programming [Posa *et al.* 2014], but in both cases, the discrete aspects of contact-based locomotion makes the optimization problems very non-convex, thus hard to solve optimally, except in realtively simple situations. Using search trees to explore sequences of contacts is another option, but since precise contact locations are determined by continuous parameters, the branching factor of a search tree would by default be infinite, and heuristics are needed to make the number of potential choices finite without losing too much precision. Such heuristics can be efficient [Chestnutt *et al.* 2003], but they require expertise and heavily depend on the robot and type of motion task considered.

Sampling-based motion planning has also been adapted to discrete planning problems, for example by using optimal control to generate directed node extensions [Sleiman *et al.* 2023] and grow the exploration tree. These are interesting approaches, but they can be slower or lose part of the efficiency of original sampling-based motion planning algorithms. Such algorithms, like Rapidly-exploring Random Trees (RRT) [LaValle 1998], tend to maximize their efficiency when planning purely continuous motions, for instance to solve the famous piano mover's problem [Schwartz & Sharir 1983]. In the illustration of Figure 2.1, sampling-based methods would typically easily plan the motion of the sofa, but not the motion of the friendly neighbors carrying the sofa.

Figure 2.1: Ross, Chandler and Rachel getting stuck while moving a sofa up the stairs.
Friends Season 5 Episode 16.

With the co-authors of [Perrin *et al.* 2017], I argued that, instead of planning the contacts and then the continuous motion, following the opposite approach could be beneficial: first plan some continuous motion, and then the sequence of contacts enabling this continuous motion. It may seem strange at first, because the true continuous motion relies on the contacts, but when this kind of backward approach can be done, it permits to rely on the efficiency of sampling-based motion planning for the initial continuous motion plan.

2.1 The flea motion planning problem

To explain what I mean by "planning a continuous motion first", let's start with a simple example: the *flea motion planning problem* introduced in [Perrin *et al.* 2012b]. Consider a 2D environment $\mathcal{C} = \mathbb{R}^2$ (the configuration space) divided between the free space $\mathcal{F}$ (an open set) and the obstacles $\mathcal{O} = \mathcal{C} \setminus \mathcal{F}$. The

flea is represented by a point. It can make jumps in any direction and of any length strictly less than $l_{max} > 0$. The goal is to find a sequence of jumps from a configuration $(x_A, y_A) \in \mathcal{F}$ to a configuration $(x_B, y_B) \in \mathcal{F}$ such that every intermediate configuration is in $\mathcal{F}$. The discontinuous nature of the jumps makes the flea motion planning problem comparable to the planning of contact sequences for legged robots. Yet, by changing the notion of collision, it can be transformed into a purely continuous motion planning problem.

First, we prove an equivalence between discontinuous jumping motions of the flea and continuous motions of an open disk, using a new notion of collision-freeness. Let's assume that a sequence of jumps has been found, corresponding to the sequence of configurations $p_1 = (x_1, y_1)$, p_2, p_3, ..., $p_n = (x_n, y_n)$, with $(x_1, y_1) = (x_A, y_A)$ and $(x_n, y_n) = (x_B, y_B)$. We consider the continuous motion of an open disk of diameter l_{max} such that the trajectory $(x(t), y(t))_{t\in[0,1]}$ of its center follows the sequence of line segments between each (x_i, y_i) and (x_{i+1}, y_{i+1}), for $i = 1, 2, \ldots, n-1$, as depicted in Figure 2.2 (on the left).

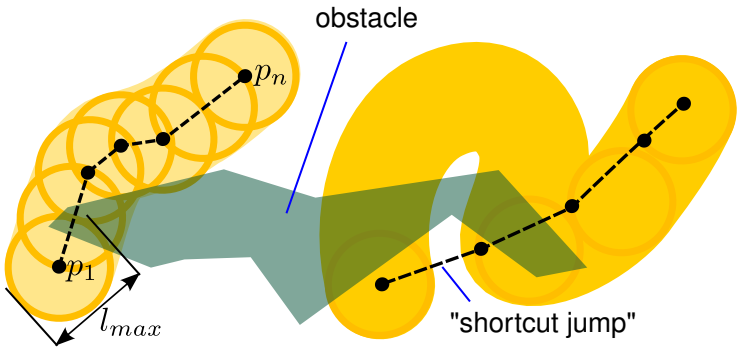


Figure 2.2: The "flea motion planning problem". On the left: from a collision-free sequence of flea jumps to a continuous "weakly collision-free" path of the disk. On the right: converting a continuous weakly collision-free path of the disk into a sequence of flea jumps, using a greedy algorithm.

The following property is a direct consequence of the upper bound l_{max} on the length of jumps:

Property 2.1. *For all $t \in [0, 1]$, the open disk of center $(x(t), y(t))$ and diameter l_{max} contains at least one of the flea configurations $p_1, p_2, \ldots, p_n$.*

This property suggests the definition of a new notion of collision-freeness:

Definition 2.1. We denote by $D_{(x,y)}$ the open disk of center (x, y) and diameter l_{max}. We say that the disk $D_{(x,y)}$ is collision-free if there exists at least one flea configuration (i.e. point) inside the disk which is collision-free.

We call this new notion of collision-freeness the "weak collision-freeness", and say that the disk is "weakly collision-free".
Conversely, if all flea configurations inside the disk are in collision (i.e. the disk does not intersect the free space), we say that the disk is in "strong collision". We say that a continuous path $(D_{(x(t),y(t))})_{t\in[0,1]}$ is weakly collision-free if for every $t \in [0, 1]$, $D_{(x(t),y(t))}$ is weakly collision-free. Figure 2.3 illustrates this definition.
A direct consequence of Property 2.1 is the following theorem:

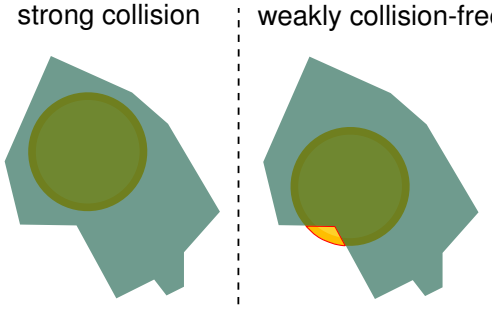


Figure 2.3: Weak collision-freeness.

Theorem 2.1. *If there exists a finite sequence of collision-free jumps from (x_A, y_A) to (x_B, y_B), then there also exists a weakly collision-free continuous path $(D_{(x(t),y(t))})_{t\in[0,1]}$ such that $(x(0), y(0)) = (x_A, y_A)$ and $(x(1), y(1)) = (x_B, y_B)$.*

Proof. Let $(D_{(x(t),y(t))})_{t\in[0,1]}$ be a path such that the center of the disk follows the line segments between the consecutive collision-free points in a sequence of jumps from (x_A, y_A) to (x_B, y_B). As mentioned above, such a path verifies Property 2.1, which means that for every $t \in [0, 1]$, $D_{(x(t),y(t))}$ contains at least one collision-free configuration and is therefore weakly collision-free. As a consequence the entire path is by definition weakly collision-free. □

We prove that the converse of Theorem 2.1 is also true:

Theorem 2.2. *If there exists a weakly collision-free continuous path* $(D_{(x(t),y(t))})_{t\in[0,1]}$ *from* (x_A, y_A) *to* (x_B, y_B)*, with* $(x(0), y(0)) = (x_A, y_A)$ *and* $(x(1), y(1)) = (x_B, y_B)$*, then there exists a finite sequence of collision-free jumps from* (x_A, y_A) *to* (x_B, y_B)*.*

Intuitively, the reason for this theorem to hold is that since the path is weakly collision-free, the free space intersects every disk along that path. And because the flea can make jumps as large as the diameter of these disks (l_{max}), it can always move forward while staying inside the free space. Figure 2.4 illustrates this property.

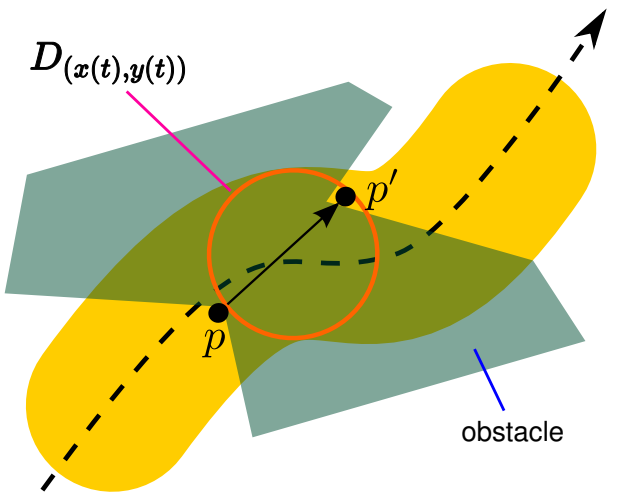


Figure 2.4: Progression inside a weakly collision-free path. The current collision-free position of the flea is p. A bit further along the path, we can choose a disk $D_{(x(t),y(t))}$ such that p is outside of it but arbitrarily close to its boundary. This disk is weakly collision-free, so its intersection with the free space is non-empty. If the disk is close enough to p, it is possible to find a position p' in this intersection such that the flea can directly jump from p to p' ($(d_2(p, p') < l_{max})$). This is the reason why the flea can always move forward along weakly collision-free paths.

Proof. We denote by d_2 the Euclidean distance in $\mathbb{R}^2$. For a point $p \in C$, its distance to the obstacles is $d_{obs}(p) = \inf\{d_2(p, o)|o \in C \setminus F\}$. For a disk $D_{(x,y)}$ we define:

$$\delta_{obs}(D_{(x,y)}) = \sup\{d_{obs}(p)|p \in D_{(x,y)}\}.$$

A disk $D_{(x,y)}$ is weakly collision-free if and only if $\delta_{obs}(D_{(x,y)}) > 0$. Let us consider a weakly collision-free continuous path $(D_{s(t)})_{t\in[0,1]}$ from (x_A, y_A) to (x_B, y_B), with $s(t) = (x(t), y(t))$.

We define $d_{inf} = \frac{1}{2}\inf\{\delta_{obs}(D_{s(t)})|t \in [0, 1]\}$. By continuity of $t \mapsto \delta_{obs}(D_{s(t)})$, we have $d_{inf} > 0$. By uniform continuity of $t \mapsto s(t)$, there exists $0 < \varepsilon < 1$ such that $\forall t \in [0, 1-\varepsilon], \forall \varepsilon' \in [0, \varepsilon], d_2(s(t), s(t+\varepsilon')) < \min(d_{inf}, l_{max})$.

Let us now consider $t \in [0, 1 - \varepsilon]$ and a collision-free configuration p of the flea in $D_{s(t)}$. First, we know that there exists $p' \in D_{s(t)}$ such that $d_{obs}(p') > d_{inf}$. Besides, since $D_{s(t)}$ is of diameter l_{max}, we have $d_2(p, p') < l_{max}$, and thus the flea can jump from p to p'. Then, since we have $d_2(s(t), s(t + \frac{1}{M})) < \min(d_{inf}, l_{max})$ for some $M \in \mathbb{N}_{>0}$ such that $\frac{1}{M} < \varepsilon$, there exists $p'' \in D_{s(t+\frac{1}{M})}$ such that $d_2(p', p'') < \min(d_{inf}, l_{max})$. It follows that the flea can jump from p to p'', and, since $d_{obs}(p) > d_{inf}$, that p is collision-free. So we have proved that if p is a collision-free configuration of the flea in $D_{s(t)}$ with $t \in [0, 1 - \frac{1}{M}]$, it is always possible to reach a collision-free configuration in $D_{s(t+1/M)}$ with at most 2 jumps. By induction, we deduce that a collision-free configuration $p_\alpha \in D_{s(1)}$ can be reached after no more than $2M$ jumps. We have $d_2(p_\alpha, (x_B, y_B)) < l_{max}$, thus the flea can jump directly from p_α to (x_B, y_B). This concludes the proof, and an example of sequence of jumps obtained from a weakly collision-free continuous path of the disk can be seen on the right side of Figure 2.2. □

The above demonstrations can be found in [Perrin *et al.* 2017]. Together, Theorem 2.1 and Theorem 2.2 form an equivalence between weakly collision-free paths of the disk and collision-free sequences of jumps of the flea. This equivalence can be used to efficiently solve the flea motion planning problem. Indeed, instead of looking for a discontinuous sequence of jumps, we can first look for a continuous path of the disk, which can be done with any conventional motion planning algorithm, provided that we implement new collision checks using Definition 2.1 (approximate weak collision checks can be performed with a finite number of standard collision checks). To convert a continuous path into a finite sequence of jumps, we can then apply a greedy approach that consists in repeatedly trying to jump from the current disk $D_{s(t)}$ to a disk $D_{s(t')}$ with t' as large as possible and obtained by dichotomy. This can result in "shortcut jumps", as shown in Figure 2.2 (on the right).

2.2 Generalization: a research direction proposal

Surprisingly, the transformation from discrete sequences of jumps to continuous paths and weak collision checks can be adapted to contact planning problems that are more complex than the flea motion planning problem. In [Perrin *et al.* 2012b] and in [Perrin *et al.* 2017], this type of transformation is applied to footstep planning for a biped robot or for a hexapod, as illustrated in Figure 2.5. Figure 2.6 and Figure 2.7 show how it was used to perform reactive vision-based footstep planning on the DLR-Biped robot [Ott *et al.* 2010], and locomotion planning on rough terrain with a hexapod in simulation. In [Perrin 2012], an abstract result demonstrates that a general class of discrete planning problems can be converted into continuous motion planning problems with a similar approach. The main issue is that the notion of weak collision-freeness is robot-dependent, and for any complex robot, defining it properly is difficult and requires expert knowledge.

If I were to work on this type of approach again, I would try to replace the weak collision-freeness definition by learned models, and I would focus on meaningful continuous trajectories for the first phase, for example the trajectory of the head of the robot. I will keep the head as an example of reference (which can be motivated from human neuroscience, see [Sreenivasa *et al.* 2009]), but other reference frames could be used as well, for instance the pelvis.

To build a general motion planning framework, I would start by assuming the availability of a low-frequency multi-contact controller $\pi^{①}$ taking the following information $(\mathcal{E}, \sigma, \mathbf{g}, \mathbf{q})$ in input:

- A description $\mathcal{E}$ of the local environment relative to the position and orientation of the robot head (e.g. a point cloud, or a higher-level and more compact representation of the robot surroundings, possibly a belief state like in [Hoeller *et al.* 2024]).
- An attitude estimation σ, i.e. knowledge of the gravity vector in the local environment.
- A local goal position and orientation $\mathbf{g}$ that should be reached by the robot head in limited time.
- Proprioception: the relative configuration $\mathbf{q}$ of the robot.

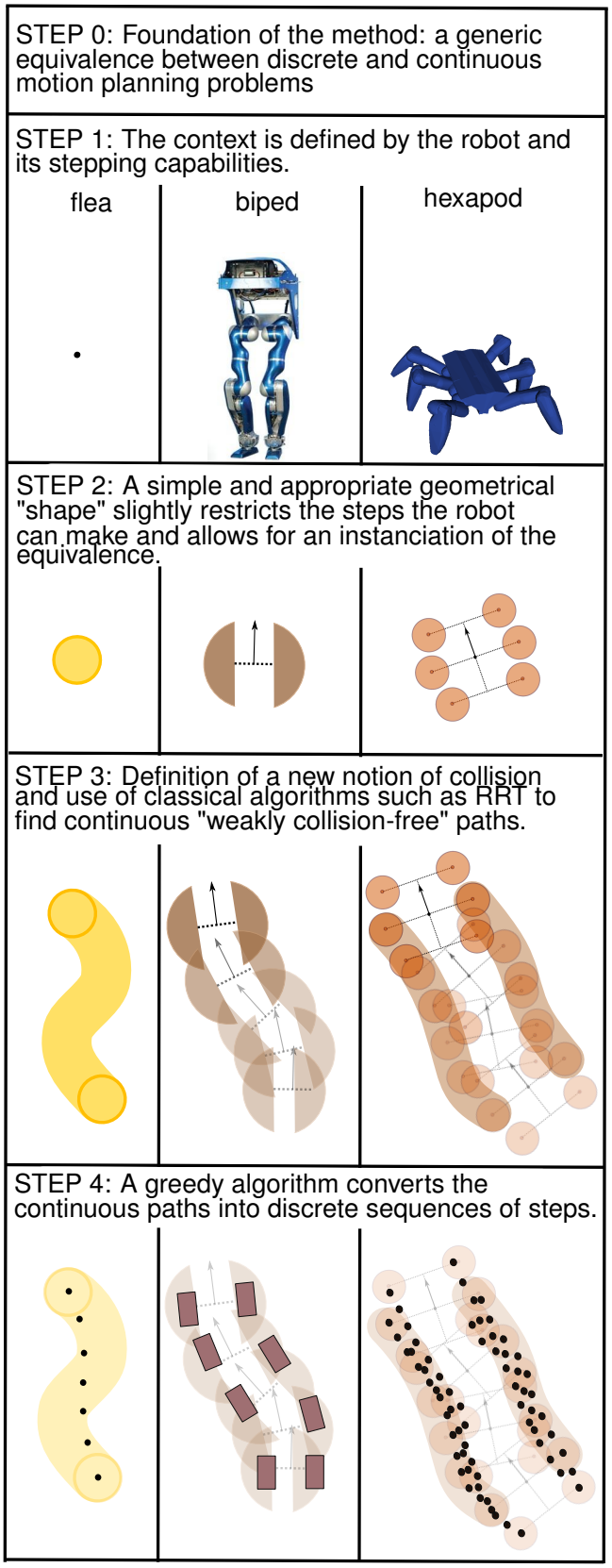


Figure 2.5: The flea motion planning method can be adapted to more complex contact planning problems.

Given knowledge of the global environment, we would like to use motion planning to define a trajectory

of the robot head, and send commands based on this trajectory to the multi-contact controller that would generate the full-body motion. We call "single motion" a motion generated by the multi-contact controller reaching the goal position and orientation of the control command.

To avoid simulating the controller during the planning phase, one could try to learn to predict, given the input information, the success of a control command. But this would not enable motion planning, which requires sequencing control commands, as the full configuration of the robot (including proprioception) at the end of a single motion cannot easily be known.
Adding a target configuration in the control commands would theoretically enable motion planning, but it would be a bad idea, for at least three reasons.
First, it would complicate the work of the multi-contact controller and would require a way higher level of precison.

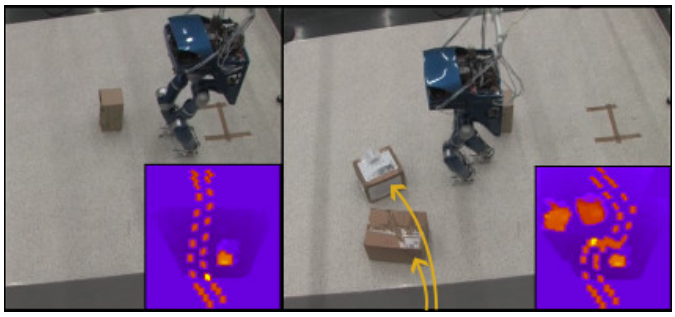


Figure 2.6: An experiment of vision-based footstep planning with the DLR-Biped (boxes are thrown in front of it while it walks, and the robot is able to reactively replan a path to the goal).

Second, it would make the dimensionality of the motion planning problem much larger, and finally it would also make the predictions of success much harder to learn.

A solution to the problem has been proposed in [Ichter & Pavone 2019] with L-SBMP (Latent Sampling-Based Motion Planning). The idea is to learn a plannable latent space, i.e. a representation of the low-dimensional manifold in which the system {robot + controller} remains. The full state of the robot is encoded into $\mathbf{z}$, a low-dimensional vector in this latent space, and the forward dynamics of the system in the latent space, given a control input $\mathbf{u}$, are learned. A collision checker in the latent space can be learned as well.

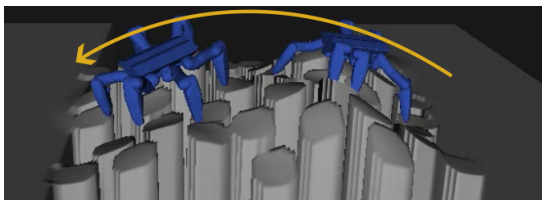


Figure 2.7: The motion accross this complex terrain was planned in 330 ms.

Learning the mapping from the full state to $\mathbf{z}$ and back can be done with a variational autoencoder [Kingma & Welling 2019] trained on data gathered from sequences of random rollouts in various environments. Sampling-based motion planning can be performed in the latent space, and when trying to extend a node $\mathbf{z}$, the idea is simply to sample a control input $\mathbf{u}$, apply $\mathbf{u}$ from state $\mathbf{z}$ for some amount of time, compute the estimated $\mathbf{z}'$, and add it to the exploration tree if the motion toward $\mathbf{z}'$ is predicted to be collision-free.

The extreme usefulness of latent spaces has been proved extensively in the recent literature, but there are still a couple of issues with L-SBMP. First, when performing the sampling-based motion planning, control inputs are sampled, and the subsequent states are estimated from learned forward dynamics, which can lead to the propagation and accumulation of errors. Second, by sampling the control input, the node extension will not necessarily be in the desired direction, which can reduce the efficiency of RRT (the underlying sampling-based motion planning algorithm). Instead, it would be nice to define a space in which we can sample $\mathbf{z}'$, and directly guess whether a transition of the form $\mathbf{z} \rightarrow \mathbf{z} + \alpha(\mathbf{z}' - \mathbf{z})$ is going to be feasible.

To do so, I propose to learn another type of latent space, which would be used as part of the control commands.

Given the local environment $\mathcal{E}$, the attitude estimation σ and the goal position and orientation $\mathbf{g}$, the idea would be to learn the manifold of feasible final configurations of the robot after a single motion. This manifold would be learned in a first phase of data gathering during which the original controller would be used randomly and extensively in various environments. This would produce a buffer of tuples $(\mathcal{E}, \sigma, \mathbf{g}, \mathbf{q}, \mathbf{q}')$ where $\mathbf{q}$ and $\mathbf{q}'$ are respectively the initial and final configurations of the robot during a successful single motion. Again, an autoencoder would be used for the latent space learning. The encoder would compute $\mathbf{z} = f_{enc}(\mathcal{E}, \sigma, \mathbf{g}, \mathbf{q}')$, and the decoder would compute $\mathbf{q}' = f_{dec}(\mathcal{E}, \sigma, \mathbf{g}, \mathbf{z})$. Notice that the encoder and decoder would never know anything about the initial configuration, but the final configurations are very

constrained by the environment, goal and controller, so the hope is that the latent space can be very low-dimensional and not too difficult to learn.

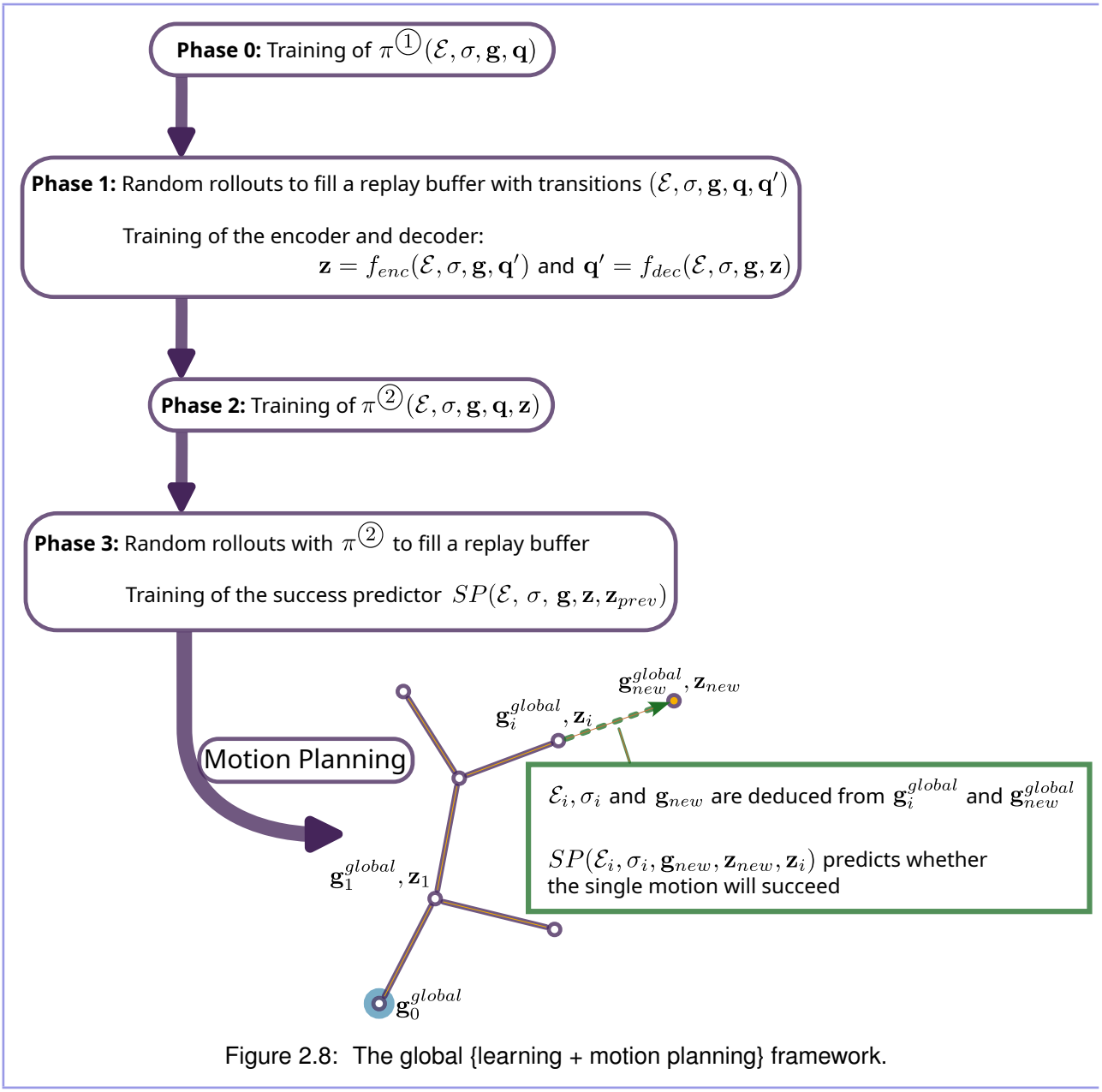


Figure 2.8: The global {learning + motion planning} framework.

In the second phase, a major modification is required: the controller would need to be modified to take as additional input a latent space goal $\mathbf{z}$ ($\pi^{①}$ is replaced by $\pi^{②}$). Ideally, the controller $\pi^{①}$ would be a neural net with a structure that already allows $\mathbf{z}$ in its input, so its current state could be used as initial state for the new training. In this new training, the controller$\pi^{②}$ would attempt not only to reach the position and orientation $\mathbf{g}$ in limited time, but also the latent target $\mathbf{z}$. This means that a single motion would be successful if, at the end of the motion, the position and orientation of the head is close to $\mathbf{g}$ and the configuration $\mathbf{q}_{final}$ is such that its encoded latent state $f_{enc}(\mathcal{E}, \sigma, \mathbf{g}, \mathbf{q}_{final})$ is close to $\mathbf{z}$.

Finally, once the second version $\pi^{②}$ of the controller is working well, a success predictor SP would be trained, taking in input $\mathcal{E}$, σ, $\mathbf{g}$, the target latent state $\mathbf{z}$ and the previous latent state $\mathbf{z}_{prev}$ (which gives information about the initial configuration), and returning true if the single motion is likely to succeed.

Having done all this would enable sampling-based motion planning in the Cartesian product between the head position and orientation goal space G (the $\mathbf{g}$ inputs), and the latent space Z (the $\mathbf{z}$ inputs). There should be a very limited accumulation of errors, because the final configurations after single motions would always be entirely constrained by the inputs $\mathbf{g}$ and $\mathbf{z}$. Furthermore, there is no need to sample control inputs, as the sampled states $(\mathbf{g}, \mathbf{z})$ *are* the inputs. This allows extending nodes exactly in the desired direction, which is crucial for algorithms such as RRT to operate at their maximum efficiency. This would

also be obtained by learning inverse dynamics in L-SBMP, which would enable the use of latent states $\mathbf{z}$ as actions. However, latent states in L-SBMP cover all the robot configurations the controller can lead to, so randomly sampling commands in this large space would often lead to unfeasible motions. In the proposed method, latent states are constrained by $\mathcal{E}$, σ and $\mathbf{g}$, so they represent a much smaller set and have a higher chance of leading to successful single motions, which should significantly simplify the motion planning. The two-phase learning also ensures that the target configuration (corresponding to the $\mathbf{z}$ input) are compatible with the controller.

Remarks:

- In the control commands, $\mathbf{g}$ inputs are *local* (they are expressed relatively to the initial position and orientation of the head), but in the sampling-based motion planning *global* values $\mathbf{g}^{global}$ would be considered. Similarly, knowledge of the global environment would be used to deduce the local values $\mathcal{E}$ and σ for each node of the exploration tree.
- Defining the distance between two pairs $(\mathbf{g}_1^{global}, \mathbf{z}_1)$ and $(\mathbf{g}_2^{global}, \mathbf{z}_2)$ will be tricky. A combination of distances seems like a decent option: $\alpha d_G(\mathbf{g}_1^{global}, \mathbf{g}_2^{global}) + \beta d_Z(\mathbf{z}_1, \mathbf{z}_2)$ (d_Z will probably be the Euclidean distance but the choice for d_G is not obvious). However, the ratio between α and β will likely need to be tuned carefully. α is expected to be much larger than β since our main interest is to explore and generate trajectories of the head.
- Other details such as how to sample $\mathbf{g}$ and $\mathbf{z}$ would need to be considered with attention.

Figure 2.8 illustrates the proposed framework.

CHAPTER 3

Using diffeomorphic matching to generalize motion plans

Motion planning can be performed online, in which case the planner should be queried at a relatively low frequency to regularly produce motion plans followed at a higher frequency by the main controller. This type of usage is very close to Model Predictive Control, which can be seen as a bridge between reactive control and planning [Benallegue *et al.* 2017], and has also strong ties with reinforcement learning [Bertsekas 2024].

If the global environment is mostly static, it may be better to perform motion planning offline. The idea is to generate one or several paths from initial configurations to a single or multiple target configurations. As in PRM [Kavraki *et al.* 1996], these paths are then used online as a roadmap to guide the robot motion without having to frequently replan.

By default, the motion planner produces a finite set of paths, but the roadmap should guide the robot even if its configuration does not precisely belong to one of these paths, so the problem of generalization naturally comes into play.

In this chapter, which presents a method and results published in Systems & Control Letters in 2016 [Perrin & Schlehuber-Caissier 2016], we consider a particular problem of generalization and show how diffeomorphic matching can solve it. To simplify a bit, we start by assuming that there is only one path leading to a single target configuration. The concrete objective is to turn this path, the *demonstration*, into a globally asymptotically stable vector field. The constraint of global asymptotic stability ensures that, no matter the initial configuration, the extrapolated path will always converge to the target configuration. The proposed method to solve the problem, illustrated in Figure 3.1, deforms a reference vector field to make it match and generalize the demonstration. It relies on several tools introduced in the next section.

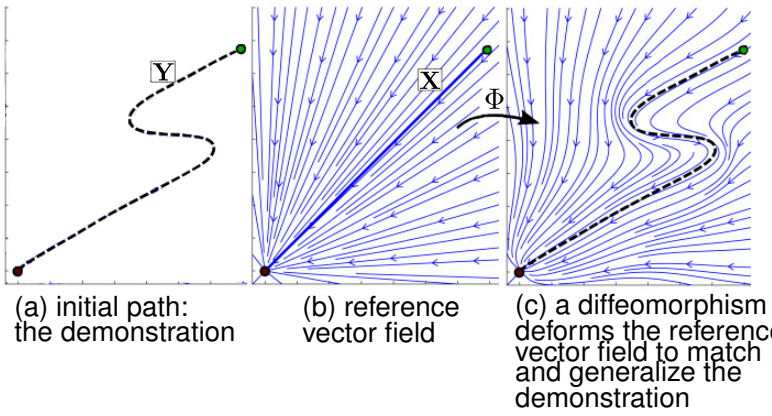


Figure 3.1: A diffeomorphism is computed that maps the straight trajectory $\mathbf{X}$ onto the demonstration $\mathbf{Y}$. $\mathbf{\Phi}$ transforms the whole reference vector field into a vector field with streamlines that match and generalize the demonstration while remaining globally asymptotically stable, i.e. ensuring convergence to the target configuration from any initial configuration.

3.1 Diffeomorphic matching

3.1.1 Diffeomorphic locally weighted translations

Given a smooth function $k_\rho(\mathbf{x}, \mathbf{y}) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, possibly depending on some parameter ρ, such that $\forall \mathbf{x}, k_\rho(\mathbf{x}, \mathbf{x}) = 1$ and $k_\rho(\mathbf{x}, \mathbf{y}) \to 0$ when $\|\mathbf{y} - \mathbf{x}\| \to \infty$, given a "direction" $\mathbf{v} \in \mathbb{R}^d$ and a "center" $\mathbf{c} \in \mathbb{R}^d$,

we consider the following *locally weighted translation*:

$$\psi_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{x}) = \mathbf{x} + k_\rho(\mathbf{x},\mathbf{c})\mathbf{v}.$$

Theorem 3.1. *If* $\forall(\mathbf{x},\mathbf{y}) \in \mathbb{R}^d \times \mathbb{R}^d$, $\frac{\partial k_\rho}{\partial \mathbf{x}}(\mathbf{x},\mathbf{y}) \cdot \mathbf{v} > -1$, *then* $\psi_{\rho,\mathbf{c},\mathbf{v}}$ *is a smooth* ($\mathcal{C}^\infty$) *diffeomorphism.*

Proof. For a given $\mathbf{x} \in \mathbb{R}^d$, let us try to find $\mathbf{y} \in \mathbb{R}^d$ such that $\psi_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{y}) = \mathbf{x}$. This can be rewritten $\mathbf{y} = \mathbf{x} - k_\rho(\mathbf{y},\mathbf{c})\mathbf{v}$, so we know that $\mathbf{y}$ must be of the form $\mathbf{x}+r\mathbf{v}$. The equation becomes $\psi_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{x}+r\mathbf{v}) = \mathbf{x}$, i.e.: $r\mathbf{v} + k_\rho(\mathbf{x}+r\mathbf{v},\mathbf{c})\mathbf{v} = \mathbf{0}$. If $\mathbf{v} = \mathbf{0}$, $\psi_{\rho,\mathbf{c},\mathbf{v}}$ is the identity (and a smooth diffeomorphism), and $\mathbf{y} = \mathbf{x}$. Otherwise, solving $\psi_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{y}) = \mathbf{x}$ amounts to solving $r + k_\rho(\mathbf{x}+r\mathbf{v},\mathbf{c}) = 0$.

Let us define:

$$h_\mathbf{x} : r \in \mathbb{R} \mapsto r + k_\rho(\mathbf{x}+r\mathbf{v},\mathbf{c}) \in \mathbb{R}.$$

If $\frac{\partial k_\rho}{\partial \mathbf{x}}(\mathbf{x},\mathbf{c}) \cdot \mathbf{v} > -1$, we get: $\forall r \in \mathbb{R}$, $\frac{dh_\mathbf{x}}{dr}(r) > 0$. Because of the absolute monotonicity of $h_\mathbf{x}$, and since $h_\mathbf{x}(r)$ tends to $-\infty$ when r tends to $-\infty$, and to $+\infty$ when r tends to $+\infty$, we deduce that there exists a unique $s_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{x}) \in \mathbb{R}$ such that $h_\mathbf{x}(s_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{x})) = 0$. It follows that the equation $\psi_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{y}) = \mathbf{x}$ has a unique solution: $\mathbf{y} = \mathbf{x} + s_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{x})\mathbf{v}$. We conclude that $\psi_{\rho,\mathbf{c},\mathbf{v}}$ is invertible, and:

$$\psi^{-1}_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{x}) = \mathbf{x} + s_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{x})\mathbf{v}.$$

The implicit function theorem can be applied to prove that $s_{\rho,\mathbf{c},\mathbf{v}}$ is smooth, and as a consequence $\psi_{\rho,\mathbf{c},\mathbf{v}}$ is a smooth diffeomorphism. □

Remark: there are strong similarities between locally weighted translations and the planar flows used in normalizing flows [Rezende & Mohamed 2015, Kobyzev *et al.* 2021] (although none of the two function classes is a superset of the other). Normalizing flows can be considered to solve diffeomorphic matching problems, but the method proposed in this chapter specifically exploits locally weighted translations with a heuristic that removes the need for any gradient-based learning.

Gaussian Radial Basis Function (RBF) kernel:

We now set k_ρ as the following symmetric positive definite kernel function (with $\rho \in \mathbb{R}_{>0}$):

$$k_\rho(\mathbf{x},\mathbf{y}) = \exp\left(-\rho^2\|\mathbf{x}-\mathbf{y}\|^2\right).$$

We have:

$$\frac{\partial k_\rho}{\partial \mathbf{x}}(\mathbf{x},\mathbf{y}) \cdot \mathbf{v} = -2\rho^2 \exp\left(-\rho^2\|\mathbf{x}-\mathbf{y}\|^2\right)(\mathbf{x}-\mathbf{y}) \cdot \mathbf{v},$$

with the lower bound:

$$\frac{\partial k_\rho}{\partial \mathbf{x}}(\mathbf{x},\mathbf{y}) \cdot \mathbf{v} \geq -2\rho^2 \exp\left(-\rho^2\|\mathbf{x}-\mathbf{y}\|^2\right)\|\mathbf{x}-\mathbf{y}\|.\|\mathbf{v}\|.$$

The expression on the right takes its minimum for $\|\mathbf{x}-\mathbf{y}\| = \frac{1}{\sqrt{2}\rho}$, which yields:

$$\frac{\partial k_\rho}{\partial \mathbf{x}}(\mathbf{x},\mathbf{y}) \cdot \mathbf{v} \geq -\sqrt{2}\|\mathbf{v}\|\rho \exp\left(-\frac{1}{2}\right).$$

We pose $\rho_{\max}(\mathbf{v}) = \frac{1}{\sqrt{2}\|\mathbf{v}\|}\exp\left(\frac{1}{2}\right)$. Applying Theorem 3.1, $\mathbf{v} = 0$ or $\rho < \rho_{\max}(\mathbf{v})$ implies that $\psi_{\rho,\mathbf{c},\mathbf{v}}$ is a smooth diffeomorphism. In that case, $s_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{x})$, and as a result $\psi^{-1}_{\rho,\mathbf{c},\mathbf{v}}(\mathbf{x})$, can be very efficiently computed with Newton's method.

3.1.2 The Greedy Diffeomorphic Matching (GDM) algorithm

In this section we are interested in addressing the following problem: given two sequences of distinct points $\mathbf{X} = (\mathbf{x}_i)_{i\in\{0,\dots,N\}}$ and $\mathbf{Y} = (\mathbf{y}_i)_{i\in\{0,\dots,N\}}$, compute a diffeomorphism Φ that maps each $\mathbf{x}_i$ onto $\mathbf{y}_i$, either exactly or approximately. More formally, defining $\mathrm{dist}(\mathbf{A},\mathbf{B}) = \frac{1}{N+1}\sum_i \|\mathbf{a}_i - \mathbf{b}_i\|^2$ for two sequences $\mathbf{A}$ and $\mathbf{B}$ of $N+1$ points, and denoting by $\Phi(\mathbf{X})$ the sequence of points $(\Phi(\mathbf{x}_i))_{i\in\{0,\dots,N\}}$, we want to find a diffeomorphism Φ that minimizes $\mathrm{dist}(\Phi(\mathbf{X}),\mathbf{Y})$.

State of the art

The sequences $\mathbf{X}$ and $\mathbf{Y}$ being potentially very different in shape, some of the state-of-the-art techniques to solve this problem are based on the Large Deformation Diffeomorphic Metric Mapping (LDDMM) framework introduced in the seminal article by Joshi and Miller [Joshi & Miller 2000]. Its core idea is to work with a time dependent vector field $v(\mathbf{x}, t) \in \mathbb{R}^d$ ($t \in [0, 1]$), and define a flow $\varphi(\mathbf{x}, t)$ via the transport equation:

$$\frac{d\varphi(\mathbf{x}, t)}{dt} = v(\varphi(\mathbf{x}, t), t),$$

with $\varphi(\mathbf{x}, 0) = \mathbf{x}$. With a few regularity conditions on v (see [Dupuis & Grenander 1998] for specific requirements), $\mathbf{x} \mapsto \varphi(\mathbf{x}, t)$ is a diffeomorphism. The resulting diffeomorphism $\Phi(\mathbf{x}) = \varphi(\mathbf{x}, 1)$ is given by:

$$\Phi(\mathbf{x}) = \mathbf{x} + \int_0^1 v(\varphi(\mathbf{x}, t), t)dt.$$

Using an appropriate Hilbert space, the vector fields $\mathbf{x} \mapsto v(\mathbf{x}, t)$ can be associated to an infinitesimal cost whose integration is interpreted as a deformation energy.

Various gradient descent algorithms have been proposed to optimize v with respect to a cost that depends both on the deformation energy and on the accuracy of the mapping, whether the objective is to map curves [Glaunès *et al.* 2008], surfaces [Vaillant & Glaunès 2005], or, as in our case, points [Guo *et al.* 2006].

A diffeomorphic matching heuritic based on locally weighted translations

The LDDMM framework has several advantages. For example, it tries to minimize the deformation, and allows the computation of similarity measures between diffeomorphic geometrical objects. However, Φ is not in closed-form, so once obtained, evaluating it requires an integration that can be slightly time-costly. In our context, it can be necessary to use Φ inside a control law, so its evaluation (and that of Φ^{-1}) must be very fast.

The proposed approach is based on the diffeomorphic locally weighted translations presented in the previous section, which are functions that can be evaluated extremely quickly.

We fix a number of iterations K, and two parameters $0 < \mu < 1$ and $0 < \beta \leq 1$. K is defined empirically, as the number of iterations required for a good approximation depends on the intrinsic difficulty of the problem. μ is a kind of "safety margin": strictly less than 1, it ensures that the results cannot be arbitrarily close to non-invertible functions. β is similar to a learning rate: a small value allows only small modifications at every iteration, resulting in a slower but usually more stable convergence. In the examples of Figures 3.2, 3.4 and 3.5, we use $K = 150$, $\mu \approx 0.9$ and $\beta \approx 0.5$.

Initially, we define $\mathbf{Z} := \mathbf{X}$. Every iteration updates $\mathbf{Z}$. The j-th iteration can be briefly described by the following steps:

1. We select the point $\mathbf{p}_j$ in $\mathbf{Z}$ that is the furthest from its corresponding target $\mathbf{q}$ in $\mathbf{Y}$ (see lines 5 to 7 in the pseudo-code below);

2. We consider the locally weighted translation $\psi_{\rho_j, \mathbf{p}_j, \mathbf{v}_j}$ of direction $\mathbf{v}_j = \beta(\mathbf{q} - \mathbf{p}_j)$, center $\mathbf{p}_j$, and Gaussian RBF kernel k_{ρ_j}, optimizing $\rho_j \in [0, \mu\rho_{\max}(\mathbf{v}_j)]$ to minimize the error between $\psi_{\rho_j, \mathbf{p}_j, \mathbf{v}_j}(\mathbf{Z})$ and $\mathbf{Y}$;

3. We perform the update: $\mathbf{Z} := \psi_{\rho_j, \mathbf{p}_j, \mathbf{v}_j}(\mathbf{Z})$.

The resulting (smooth) diffeomorphism is the composition of all the locally weighted translations:

$$\Phi = \psi_{\rho_K, \mathbf{p}_K, \mathbf{v}_K} \circ \cdots \circ \psi_{\rho_2, \mathbf{p}_2, \mathbf{v}_2} \circ \psi_{\rho_1, \mathbf{p}_1, \mathbf{v}_1}$$

Algorithm 1 describes the proposed algorithm, which we call GDM (for Greedy Diffeomorphic Matching), in pseudo-code.

Algorithm 1 GDM

1: **Input:**
$\mathbf{X} = (\mathbf{x}_i)_{i\in\{0,\dots,N\}}$ and $\mathbf{Y} = (\mathbf{y}_i)_{i\in\{0,\dots,N\}}$
2: **Parameters:**
$K \in \mathbb{N}_{>0}$, $0 < \mu < 1$, $0 < \beta \leq 1$
3: $\mathbf{Z}(= (\mathbf{z}_i)_{i\in\{0,\dots,N\}}) := \mathbf{X}$
4: **for** $j = 1$ to K **do**
5: $\quad m := \underset{i\in\{0,\dots,N\}}{\arg\max} \left(\|\mathbf{z}_i - \mathbf{y}_i\|\right)$
6: $\quad \mathbf{p}_j := \mathbf{z}_m$
7: $\quad \mathbf{q} := \mathbf{y}_m$
8: $\quad \mathbf{v}_j := \beta(\mathbf{q} - \mathbf{p}_j)$
9: $\quad \rho_j := \underset{\rho\in[0,\mu\rho_{\max}(\mathbf{v}_j)]}{\arg\min} \left(\text{dist}(\psi_{\rho,\mathbf{p}_j,\mathbf{v}_j}(\mathbf{Z}), \mathbf{Y})\right)$
10: $\quad \mathbf{Z} := \psi_{\rho_j,\mathbf{p}_j,\mathbf{v}_j}(\mathbf{Z})$
11: **end for**
12: **Return:**
$(\rho_j)_{j\in\{1,\dots,K\}}$, $(\mathbf{p}_j)_{j\in\{1,\dots,K\}}$, $(\mathbf{v}_j)_{j\in\{1,\dots,K\}}$

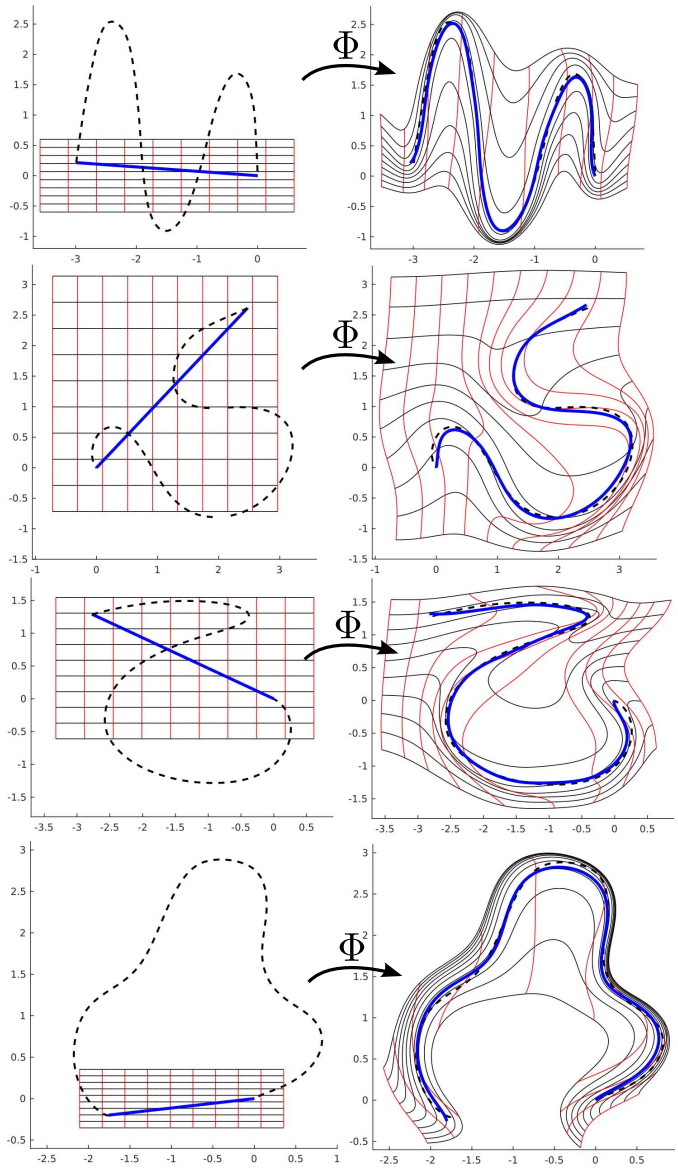


Figure 3.2: On the left, the dashed curve is a trajectory represented by a sequence of points $\mathbf{Y} = (\mathbf{y}_i)_{i\in\{0,\dots,N\}}$. The solid line is $\mathbf{X} = \left(\mathbf{y}_0 + \frac{i}{N}(\mathbf{y}_N - \mathbf{y}_0)\right)_{i\in\{0,\dots,N\}}$. The right side shows the result of the application of the diffeomorphism Φ constructed by the GDM algorithm to map $\mathbf{X}$ onto $\mathbf{Y}$.

Five remarks:

- Here the parameter β is constant, but we can also make it vary iteration after iteration, for example by increasing it toward 1.
- The line 9 of the algorithm performs a nonlinear optimization, but it depends only on one bounded real variable, so a minimum can be found very quickly and precisely.
- We can add a fixed upper bound $\rho_M > 0$ for all ρ_j, and a regularization term in the cost of the optimization problem of line 9, to prevent the diffeomorphism from overly deforming the space to get a perfect matching. Simply using inputs with a dense representation (large value of N) has a similar effect, and it barely slows the algorithm down if the implementation relies on vectorization.
- Again in line 9, dist can be replaced by any distance, e.g. the largest singular value norm of $(\mathbf{X} - \mathbf{Y})$ (with $\mathbf{X}$ and $\mathbf{Y}$ written as $(N+1)$-by-d matrices).
- The algorithm can get stuck in local minima, so a general proof of convergence cannot be found. However, as shown in the next sections, experimental results give empirical evidence that the algorithm is efficient and converges quickly in practice, even on difficult matching problems.

Experimental evaluation

We compare GDM to an implementation of diffeomorphic matching in the LDDMM framework developed by J. Glaunès (the "`Matchine`" software [Glaunès 2006]). Given a sequence of points $\mathbf{Y} = (\mathbf{y}_i)_{i\in\{0,\dots,N\}}$

representing a trajectory, we set $\mathbf{X} = (\mathbf{x}_i)_{i\in\{0,\dots,N\}} = \left(\mathbf{y}_0 + \frac{i}{N}(\mathbf{y}_N - \mathbf{y}_0)\right)_{i\in\{0,\dots,N\}}$ and apply our algorithm or the LDDMM algorithm to construct a diffeomorphism Φ such that $\Phi(\mathbf{X})$ and $\mathbf{Y}$ match. Figure 3.2 displays the result of GDM on four 2D trajectories, and Table 3.1 shows a comparison of the results obtained on these trajectories with GDM and the LDDMM algorithm. For each trajectory, we apply the algorithms with representations as sequences of 21, 51 and 101 points (i.e. $N = 20$, $N = 50$, $N = 100$). For both algorithms, the same parameters are kept across all the trials. In all cases, GDM provides a substantial speedup. For example, with $N = 50$, Φ is learned in average 58 times faster and evaluated 240 times faster, while the error $\text{dist}(\Phi(\mathbf{X}), \mathbf{Y})$ is 2.67 times smaller. The tests were made on an Intel(R) Core(TM) i7-4700MQ @ 2.4 GHz with 4GB of RAM.

	N	GDM (the proposed algorithm)	LDDMM
Average duration of the construction of Φ	20	0.25 s	2.78 s
	50	0.25 s	14.5 s
	100	0.26 s	53.3 s
Forward evaluation: average duration of the computation of $\Phi(\mathbf{X})$	20	3.05 ms	157 ms
	50	3.35 ms	804 ms
	100	3.72 ms	3130 ms
Backward evaluation: average duration of the computation of $\Phi^{-1}(\mathbf{Y})$	20	29.8 ms	145 ms
	50	35 ms	798 ms
	100	38.5 ms	3110 ms
Accuracy of the mapping: average value of $\text{dist}(\Phi(\mathbf{X}), \mathbf{Y})$	20	3.49×10^{-3}	18.2×10^{-3}
	50	8.32×10^{-3}	22.2×10^{-3}
	100	9.51×10^{-3}	22.0×10^{-3}
Generalization: average value of $\text{dist}(\Phi(\mathbf{X}_{1000}), \mathbf{Y}_{1000})$	20	19.8×10^{-3}	20.3×10^{-3}
	50	9.51×10^{-3}	21.6×10^{-3}
	100	11.5×10^{-3}	22.3×10^{-3}

Table 3.1: Comparison of experimental results for the 4 examples of Figure 3.2. Remark: standard deviations are negligible for our algorithm: it is deterministic, and the computation times depend almost entirely on the input size N and on the fixed number of iterations K. $\mathbf{Y}$ is obtained by subsampling from an initial recording of 1000 points: $\mathbf{Y}_{1000}$. $\mathbf{X}_{1000}$ is the linear progression from $\mathbf{y}_0$ to $\mathbf{y}_{999}$. To get a sense of how precisely the mapping generalizes around the set of training points, we compute $\text{dist}(\Phi(\mathbf{X}_{1000}), \mathbf{Y}_{1000})$. We observe that for $N = 50$ and $N = 100$, our results are about twice as accurate as the ones obtained with the algorithm based on LDDMM.

3.2 Computing globally asymptotically stable nonlinear dynamical systems

In this section, we show how GDM or any diffeomorphic matching algorithm can be used to compute globally asymptotically stable dynamical systems (DS) that reproduce demonstration trajectories.

3.2.1 Definitions and theorems

Remark: we only consider dynamical systems $\dot{\mathbf{x}} = f(\mathbf{x})$ such that $f(\mathbf{x})$ is locally Lipschitz.

Definition 3.1. A Lyapunov candidate L is a continuously differentiable function from $\mathbb{R}^d$ to $\mathbb{R}_{\geq 0}$ taking the value 0 at a "target point" $\mathbf{x}^*$, with no other local extremum, and radially unbounded ($\|\mathbf{x}\| \to \infty \Rightarrow L(\mathbf{x}) \to \infty$).

Definition 3.2. A Lyapunov candidate L with target point $\mathbf{x}^*$ is said to be compatible with the DS $\dot{\mathbf{x}} = f(\mathbf{x})$ if:

$$\forall \mathbf{x} \in \mathbb{R}^d,\ \mathbf{x} \neq \mathbf{x}^* \Rightarrow f(\mathbf{x}) \cdot \nabla L(\mathbf{x}) < 0.$$

The following is a classical theorem in Lyapunov stability theory (see for example [Khalil 2015]):

Theorem 3.2. *If a DS $\dot{\mathbf{x}} = f(\mathbf{x})$ is compatible with some Lyapunov candidate L, then it is globally asymptotically stable.*

Note that Definition 3.1 is stronger than the usual definition of Lyapunov candidates in that they must have no other local extremum than $\mathbf{x}^*$. This is very important when the objective is to construct globally asymptotically stable DS, as the standard approach is to first compute a good Lyapunov candidate, and then a DS that is compatible with it. But if the gradient of the Lyapunov candidate vanishes at several points (which can be difficult to check), then no DS can be compatible with it. Therefore, it is crucial to ensure *by construction* that the Lyapunov candidate has a single extremum.

Definition 3.3. We say that L is a Lyapunov function for the DS $\dot{\mathbf{x}} = f(\mathbf{x})$ if it is a Lyapunov candidate compatible with $\dot{\mathbf{x}} = f(\mathbf{x})$.

Definition 3.4. Two DS $\dot{\mathbf{x}} = f(\mathbf{x})$ and $\dot{\mathbf{x}} = g(\mathbf{x})$ are said to be diffeomorphic, or smoothly equivalent, if there exists a diffeomorphism $\Phi : \mathbb{R}^d \to \mathbb{R}^d$ such that:

$$\forall \mathbf{x} \in \mathbb{R}^d,\ g(\Phi(\mathbf{x})) = J_\Phi(\mathbf{x}) f(\mathbf{x}),$$

where $J_\Phi(x)$ is the Jacobian matrix: $J_\Phi(\mathbf{x}) = \frac{\partial \Phi}{\partial \mathbf{x}}(\mathbf{x})$. If Φ is a $\mathcal{C}^k$-diffeomorphism, then the DS are said to be $\mathcal{C}^k$-diffeomorphic.

Theorem 3.3. *If two DS $\dot{\mathbf{x}} = f(\mathbf{x})$ and $\dot{\mathbf{x}} = g(\mathbf{x})$ are diffeomorphic, then if one is globally asymptotically stable, both are.*

Proof. Without ambiguity we can call these DS f and g. Let Φ be a diffeomorphism such that $\forall \mathbf{x} \in \mathbb{R}^d,\ g(\Phi(\mathbf{x})) = J_\Phi(\mathbf{x}) f(\mathbf{x})$. For any forward orbit of f, i.e. any trajectory $(\mathbf{x}(t))_{t\geq 0}$ such that $\dot{\mathbf{x}} = f(\mathbf{x})$, let us consider the trajectory $\big(\Phi(\mathbf{x}(t))\big)_{t\geq 0}$. We have:

$$\frac{d}{dt}\big(\Phi(\mathbf{x}(t))\big) = J_\Phi(\mathbf{x}(t))\dot{\mathbf{x}}(t) = J_\Phi(\mathbf{x}(t)) f(\mathbf{x}(t)) = g\big(\Phi(\mathbf{x}(t))\big).$$

This implies that $\big(\Phi(\mathbf{x}(t))\big)_{t\geq 0}$ is a forward orbit of g. More generally, any orbit $(\mathbf{y}(t))_{t\geq 0}$ of g can be written $\big(\Phi(\mathbf{x}(t))\big)_{t\geq 0}$, with $\mathbf{x}(0) = \Phi^{-1}\big(\mathbf{y}(0)\big)$, and $(\mathbf{x}(t))_{t\geq 0}$ orbit of f. If f is globally asymptotically stable, then all orbits $(\mathbf{x}(t))_{t\geq 0}$ converge towards some target point $\mathbf{x}^*$, and thus all orbits $(\mathbf{y}(t))_{t\geq 0}$ of g converge towards $\Phi(\mathbf{x}^*)$, which proves that g is globally asymptotically stable. A similar demonstration proves the converse implication. □

Theorem 3.4. *Let $\dot{\mathbf{x}} = f(\mathbf{x})$ and $\dot{\mathbf{x}} = g(\mathbf{x})$ be two $\mathcal{C}^1$-diffeomorphic DS, and let Φ be a $\mathcal{C}^1$-diffeomorphism such that $\forall \mathbf{x} \in \mathbb{R}^d,\ g(\Phi(\mathbf{x})) = J_\Phi(\mathbf{x}) f(\mathbf{x})$. If L is a Lyapunov function for $\dot{\mathbf{x}} = f(\mathbf{x})$, then $L \circ \Phi^{-1}$ is a Lyapunov function for $\dot{\mathbf{x}} = g(\mathbf{x})$.*

Proof. Again, we call these DS f and g. We also pose $M = L \circ \Phi^{-1}$. Using Theorem 3.2, we know that f is globally asymptotically stable, and by Theorem 3.3, g is globally asymptotically stable as well. Let $\mathbf{x}^*$ be the target point of f. $\Phi(\mathbf{x}^*)$ is the target point of g. Let us consider a forward orbit $(\mathbf{y}(t))_{t\geq 0}$ of g. It can be written $\big(\Phi(\mathbf{x}(t))\big)_{t\geq 0}$, with $(\mathbf{x}(t))_{t\geq 0}$ forward orbit of f (cf. proof of Theorem 3.3). It follows that $M(\mathbf{y}(t)) = L(\mathbf{x}(t))$, and if $\mathbf{y}(t) \neq \Phi(\mathbf{x}^*)$, i.e. $\mathbf{x}(t) \neq \mathbf{x}^*$, then $\frac{d}{dt}\big(M(\mathbf{y}(t))\big) = g(\mathbf{y}(t)) \cdot \nabla M(\mathbf{y}(t)) = \frac{d}{dt}\big(L(\mathbf{x}(t))\big) < 0$. Besides, it can be verified that M is a Lyapunov candidate (with target point $\Phi(\mathbf{x}^*)$), so M is a Lyapunov function for g. □

3.2.2 Overview of the method

The objective of the approach is to learn a smooth diffeomorphism Φ that maps a forward orbits of the DS $\dot{\mathbf{x}} = -\mathbf{x}$ (i.e. line segments) onto the target trajectory. The mapping obtained is at first purely geometrical, but the initial DS can be transformed into $\dot{\mathbf{x}} = -\gamma(\mathbf{x})\mathbf{x}$, with $\gamma : \mathbb{R}^d \to \mathbb{R}_{>0}$, to adjust velocities without modifying forward orbits. If the matching is accurate, Φ deforms the whole DS $\dot{\mathbf{x}} = -\gamma(\mathbf{x})\mathbf{x}$ into the globally asymptotically stable DS $\dot{\mathbf{x}} = -\gamma(\Phi^{-1}(\mathbf{x}))J_\Phi(\Phi^{-1}(\mathbf{x}))\Phi^{-1}(\mathbf{x})$ that reproduces well the target trajectory

(the demonstration). Additionally, since $\mathbf{x} \mapsto \|\mathbf{x}\|$ is a Lyapunov function for $\dot{\mathbf{x}} = -\gamma(\mathbf{x})\mathbf{x}$, $\mathbf{x} \mapsto \|\Phi^{-1}(\mathbf{x})\|$ is a Lyapunov function for $\dot{\mathbf{x}} = -\gamma(\Phi^{-1}(\mathbf{x}))J_\Phi(\Phi^{-1}(\mathbf{x}))\Phi^{-1}(\mathbf{x})$ (cf. Theorem 3.4). This transformation of a globally asymptotically stable DS into another one via diffeomorphism has strong similarities with the construction of navigation functions proposed in [Rimon & Koditschek 1991], which is based on the fact that navigation properties are invariant under $\mathcal{C}^k$-diffeomorphisms for $k \geq 1$.

Trajectories being represented as sequences of points, the problem of forward orbits mapping can be cast as diffeomorphic matching. In the case of a unique demonstration $\mathbf{Y} = (\mathbf{y}(t_i))_{i\in\{0,\dots,N\}}$, with $t_i = i\Delta t$, we want to find a diffeomorphism that maps $\mathbf{X} = \big(\mathbf{y}(0) + \frac{i}{N}\big(\mathbf{0} - \mathbf{y}(0)\big)\big)_{i\in\{0,\dots,N\}}$ onto $\mathbf{Y}$ (the trajectory is assumed to arrive at the target: $\mathbf{y}(t_N) = \mathbf{0}$). To do so, we simply use the algorithm presented in Section 3.1.2. The diffeomorphism Φ_K obtained after K iterations can be such that $\Phi_K(\mathbf{0}) \neq \mathbf{0}$, so we add an extra iteration that picks $\mathbf{p}_{K+1} = \Phi_K(\mathbf{0})$ and $\mathbf{v}_{K+1} = \mathbf{0} - \Phi_K(\mathbf{0})$. This ensures that the final diffeomorphism Φ verifies $\Phi_K(\mathbf{0}) = \mathbf{0}$. Remark: the structure of Φ makes it easy to efficiently compute $J_\Phi(\mathbf{x})$ at any given point.

3.2.3 Results

The top row of Figure 3.3 shows the result of mapping the straight trajectory $\mathbf{X}$ (on the left) onto the trajectory $\mathbf{Y}$ (on the right). The diffeomorphism Φ that realizes this matching also transforms the entire dynamical system $\dot{\mathbf{x}} = -\mathbf{x}$ into a nonlinear globally asymptotically stable DS that reproduces the trajectory $\mathbf{Y}$ (as the forward orbit of $\mathbf{y}(0)$). Modifying the initial DS without changing the forward orbit of $\mathbf{x}(0)$ leads, by application of Φ, to another DS that still reproduces $\mathbf{Y}$. On the bottom row of Figure 3.3, we use a linear system that keeps $\mathbf{x}(0)$ as an eigenvector associated with eigenvalue -1 (ensuring that its forward orbit is not modified), but has a negative eigenvalue of absolute value greater than 1 in the orthogonal direction (unlike the DS $\dot{\mathbf{x}} = -\mathbf{x}$). This results in a transformed DS that "tracks" more agressively the trajectory $\mathbf{Y}$, bringing robustness in the sense that, after a perturbation, the systems goes back quickly to the reference trajectory $\mathbf{Y}$. The DS of the top row corresponds to another notion of robustness, in which the reproduction of the pattern has more importance.

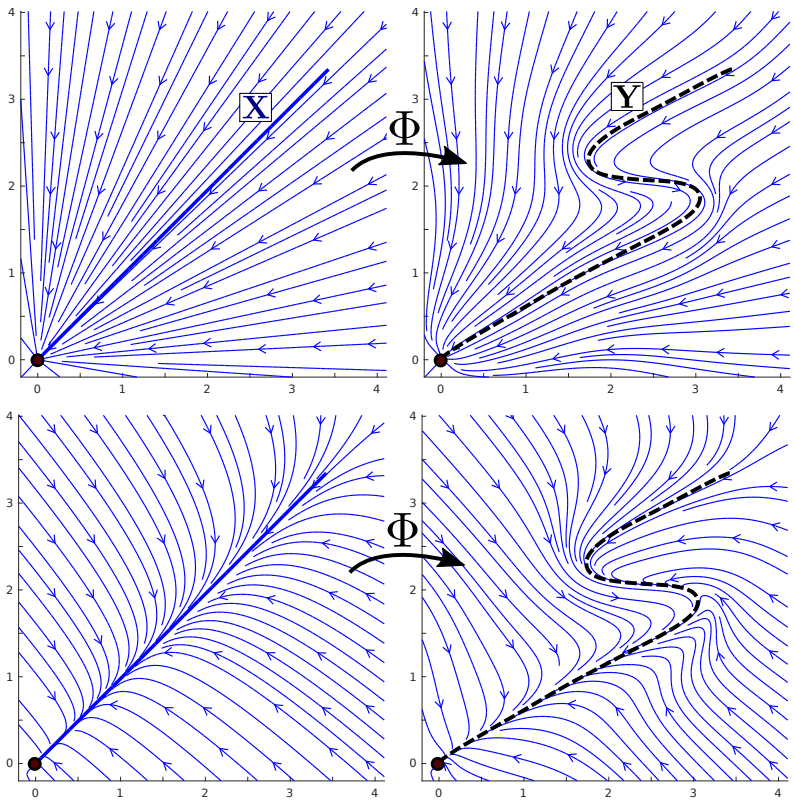


Figure 3.3: The diffeomorphism Φ, that maps the trajectory $\mathbf{X}$ onto $\mathbf{Y}$, transforms a globally asymptotically stable DS with $\mathbf{X}$ as a forward orbit into a globally asymptotically stable DS with $\mathbf{Y}$ as a forward orbit (*top row*). Using this property, we can modulate the initial vector field while keeping $\mathbf{X}$ unchanged to obtain systems with different behaviors that all reproduce the demonstration $\mathbf{Y}$.

We evaluate our approach on the LASA Handwriting Dataset [Khansari-Zadeh & Billard 2011], similarly to [Khansari-Zadeh & Billard 2011, Khansari-Zadeh & Billard 2014, Neumann & Steil 2015]. On all cases shown in Figure 3.4, seven trajectories ending at the same point demonstrate a single pattern of handwriting motion. For each of these patterns, we create an average timed sequence of points $\mathbf{Y} = (\mathbf{y}(i\Delta t))_{i\in\{0,\dots,N\}}$ based on the 7 trajectories, and apply our matching algorithm to construct a diffeomorphism Φ that maps $\mathbf{X} = \big(\frac{N-i}{N}\mathbf{y}(0)\big)_{i\in\{0,\dots,N\}}$ onto $\mathbf{Y}$. This gives a Lyapunov candidate $\mathbf{x} \mapsto \|\Phi^{-1}(\mathbf{x})\|$.

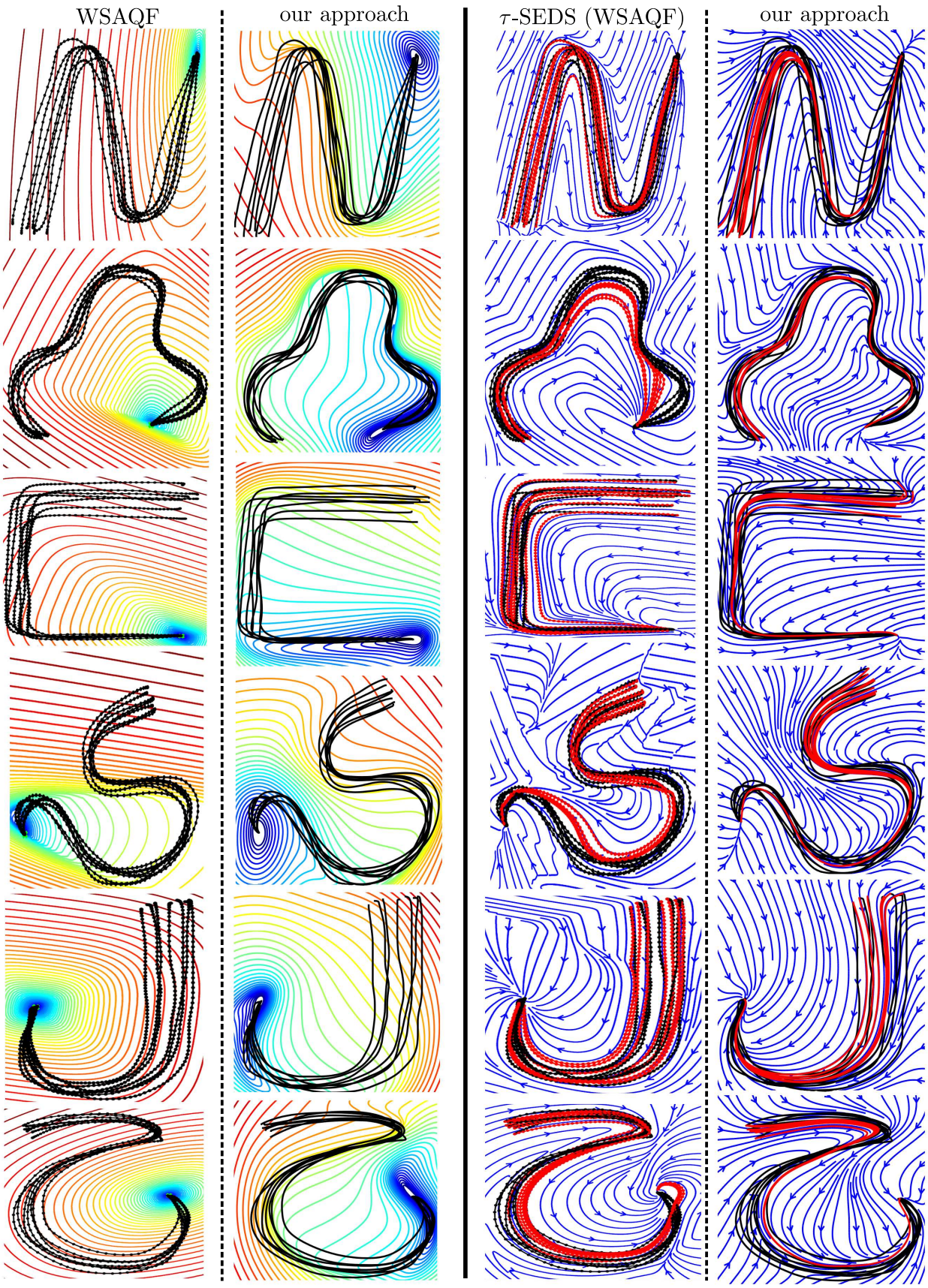


Figure 3.4: Demonstrations are in black, and reproduced trajectories in red. On the left: level sets of the Lyapunov candidates obtained with WSAQF (*1st column*) and with our approach (*2nd column*). On the right: streamlines of the DS produced by τ-SEDS (WSAQF) [Neumann & Steil 2015] (*3rd column*) and with our approach (*4th column*).

We compare our Lyapunov candidates to the WSAQF Lyapunov candidates obtained with the method of Khansari-Zadeh and Billard [Khansari-Zadeh & Billard 2014] also used in [Neumann & Steil 2015]. On the *1st column* of Figure 3.4 are displayed level sets of the WSAQF Lyapunov candidates, and on the *2nd column* level sets of our Lyapunov candidates. We can observe that the level sets of the Lyapunov candidates produced by our method have a richer geometry and exhibit variations that are more suitably adapted to the training data.

The CLF-DM method of Khansari-Zadeh and Billard [Khansari-Zadeh & Billard 2014] could be used with these Lyapunov candidates to correct any learned DS and ensure global asymptotic stability. But as mentioned above, the diffeomorphism also provides a way to directly get a globally asymptotically stable DS that reproduces the motion pattern. We define a function $\gamma : \mathbb{R}^d \to \mathbb{R}_{>0}$ such that, starting at $\mathbf{x}(0) = \mathbf{y}(0)$ with $t = 0$, the DS $\dot{\mathbf{x}} = -\gamma(\mathbf{x})\mathbf{x}$ produces a trajectory that passes by the points $\frac{N-i}{N}\mathbf{y}(0)$ at times $i\Delta t$, for $i \in \{1, \dots, N-1\}$, and converges asymptotically towards $\mathbf{0}$ for $t > (N-1)\Delta t$. A simple choice for γ is $\gamma(\mathbf{x}) = \frac{\|\mathbf{y}(0)\|}{N\Delta t\|\mathbf{x}\|}$ for $\|x\| \geq \frac{\|\mathbf{y}(0)\|}{N}$ and $\gamma(\mathbf{x}) = \frac{\|\mathbf{y}(0)\|}{N}$ otherwise (but it is easy to design a smoother function with the same desired properties).

Φ transforms the DS $\dot{\mathbf{x}} = -\gamma(\mathbf{x})\mathbf{x}$ into one that reproduces the demonstrations and their velocity profiles, as shown in Figure 3.5. The eigenvalue in the direction orthogonal to $\mathbf{y}(0)$ can be adjusted according to the variability of the 7 demonstrations, or to get a better rate of convergence towards the demonstrations (cf. Figure 3.3).

The vector fields obtained with our method are shown on the *4th column* of Figure 3.4, and the vector fields obtained with the τ-SEDS method of Neumann and Steil [Neumann & Steil 2015] based on WSAQF are shown on the *3rd column*.

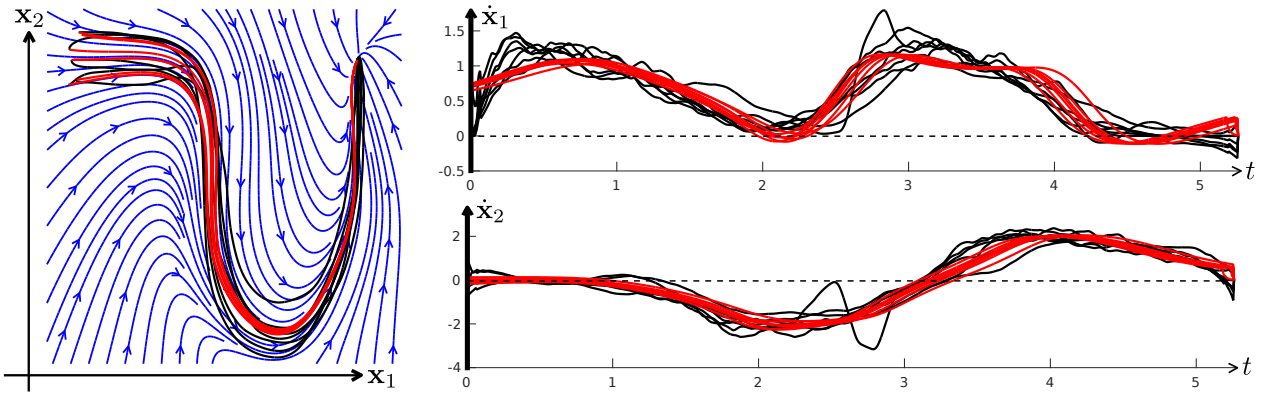


Figure 3.5: On the left: a smooth autonomous systems, learned with our method, that reproduces a motion pattern (demonstrations are in black, reproduced trajectories in red). The trajectories on the right show that the velocity profiles are quite accurately reproduced as well (again, demonstrations in black and reproductions in red).

3.2.4 Comparison with previous approaches

The existing approaches follow a two-step process:

1. Compute a Lyapunov candidate L, highly compatible with the demonstrations.
2. Compute a DS compatible with L that reproduces the demonstrations.

Neumann and Steil [Neumann & Steil 2015] add a diffeomorphic deformation between step 1 and step 2 to simplify the construction of the DS, and Khansari-Zadeh and Billard [Khansari-Zadeh & Billard 2014] separate step 1 and step 2 completely, noticing that *any* DS reproducing the demonstrations can be corrected into a globally asymptotically stable DS once the Lyapunov candidate L is known. In both approaches, step 1 is crucial because the Lyapunov candidate restricts the possibilities of the DS of step 2. But a major difficulty is that the set of Lyapunov candidates (Definition 3.1) is rather ill-behaved in the sense that it is non-convex and not closed under addition or multiplication: the sum or product of two Lyapunov candidates is not necessarily a Lyapunov candidate, as local extrema might appear. To circumvent this difficulty, a

solution is to restrict the search to a convex subset of the set of Lyapunov candidates, as in WSAQF [Khansari-Zadeh & Billard 2014, Neumann & Steil 2015]. As mentioned by Neumann and Steil (Lemma 1 in [Neumann & Steil 2015]), any WSAQF Lyapunov candidate L (with target point $\mathbf{0}$) is compatible with the DS $\dot{\mathbf{x}} = -\mathbf{x}$:

$$\forall \mathbf{x} \in \mathbb{R}^d, \mathbf{x} \neq \mathbf{0} \Rightarrow -\mathbf{x} \cdot \nabla L(\mathbf{x}) < 0.$$

Interestingly, the set of Lyapunov candidates that are compatible with a fixed globally asymptotically stable DS (in this case $\dot{\mathbf{x}} = -\mathbf{x}$) is a convex cone. Thanks to this property, searching for a WSAQF Lyapunov candidate can be done efficiently. But the compatibility to $\dot{\mathbf{x}} = -\mathbf{x}$ is a serious restriction that our method does not have. For example, in Figure 4, *2nd column*, the Lyapunov candidates of the *1st*, *2nd*, *5th* and *6th rows* are not compatible with $\dot{\mathbf{x}} = -\mathbf{x}$. Moreover, WSAQF Lyapunov candidates are constructed as sums of convex functions, and as such their level sets define convex regions: for any WSAQF Lyapunov candidate L, for any $\lambda > 0$, the set $\{\mathbf{x} \in \mathbb{R}^d \mid L(\mathbf{x}) \leq \lambda\}$ is convex. This is an even stronger restriction. In Figure 4, *2nd column*, all the Lyapunov candidates found with our method have level sets that define non-convex regions.

The proposed method can potentially learn more complex Lyapunov candidates because it finds them *indirectly* (via diffeomorphisms), and, instead of being based on good properties of a subset of the set of Lyapunov candidates, it is based on the stability under composition of diffeomorphisms. It should be noted, however, that we can only produce vector fields that are diffeomorphic to the DS $\dot{\mathbf{x}} = -\mathbf{x}$, which is not true for all globally asymptotically stable smooth autonomous systems.

Another strength of the proposed algorithm is its simplicity and efficiency. It also scales well in dimensionality, because the steps and parameters of the GDM are dimension-independent, which is not true for the optimization problems used in previous approaches (e.g. in [Khansari-Zadeh & Billard 2014] and [Neumann & Steil 2015]). It potentially leads to a speed-up that can be critical when the ability to compute the vector field quickly is important.

The algorithm can also be adapted to extrapolate around limit cycles, if for the reference DS, instead of $\dot{\mathbf{x}} = -\mathbf{x}$, we use a simple limit cycle oscillator.

3.3 Extension to multiple paths

So far, we have only considered single demonstrations. As shown in Figure 3.6, a problem with multiple demonstrations is that they can be conflicting, and a naive extension of the method could lead to vector fields that extrapolate poorly.

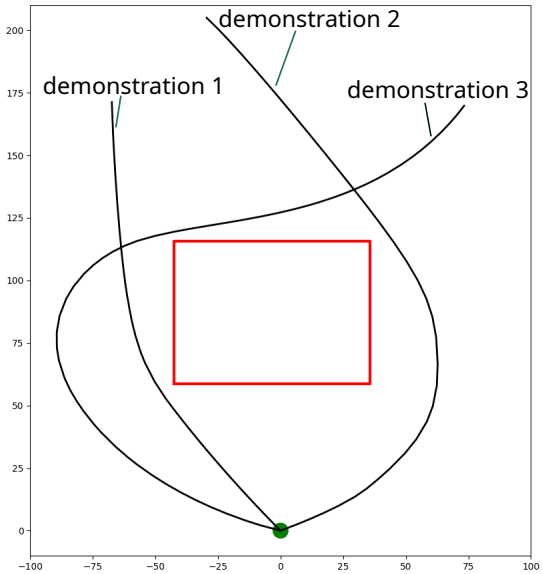
Instead of directly computing a single vector field for all the demonstrations, we can compute one vector field (i.e. one diffeomorphism) per demonstration, and in a second phase combine the vector fields to generate a new one.

To do so, we first define the concept of ε-*absorption*.

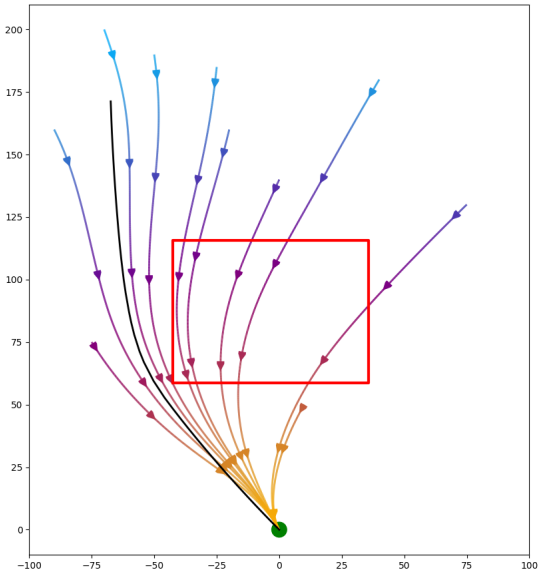
Although demonstrations are discrete sequences of states, we easily turn them into continuous paths and denote the i-th demonstration by $(\mathbf{x}_i(t))_{t\geq 0}$. All demonstrations should lead to the same target, so we assume that $\mathbf{x}_i(t) \to \mathbf{0}$ as $t \to \infty$. Let us also assume that we have computed n globally asymptotically stable vector fields $\dot{\mathbf{x}} = f_i(\mathbf{x})$, one for each demonstration. For a state $\mathbf{y}$, we denote by $(\mathbf{x}_{\mathbf{y},f_i}(t))_{t\geq 0}$ the trajectory generated from $\mathbf{y}$ with the i-th vector field. It can easily be approximated using Euler or more advanced integration schemes. We say that a state $\mathbf{x}_{\mathbf{y},f_i}(t)$ along this trajectory is ε-*absorbed* if the distance from $\mathbf{x}_{\mathbf{y},f_i}(t')$ to the target trajectory $(\mathbf{x}_i(t))_{t\geq 0}$ is less than ε for any $t' \geq t$. The distance from a point to a trajectory is usually defined as the minimum Euclidean distance betwen this point and any other point of the trajectory, but the Euclidean distance can be replaced by any distance in $\mathbb{R}^d$.

Definition 3.5. We define the time to ε-absorption α_i^ε as follows: for any state $\mathbf{y}$, $\alpha_i^\varepsilon(\mathbf{y})$ is the infinimum of the set of times t such that $\mathbf{x}_{\mathbf{y},f_i}(t)$ is ε-*absorbed*. In other words, $\alpha_i^\varepsilon(\mathbf{y})$ can be understood as the time it takes for the state to become and remain ε-close to the target trajectory.

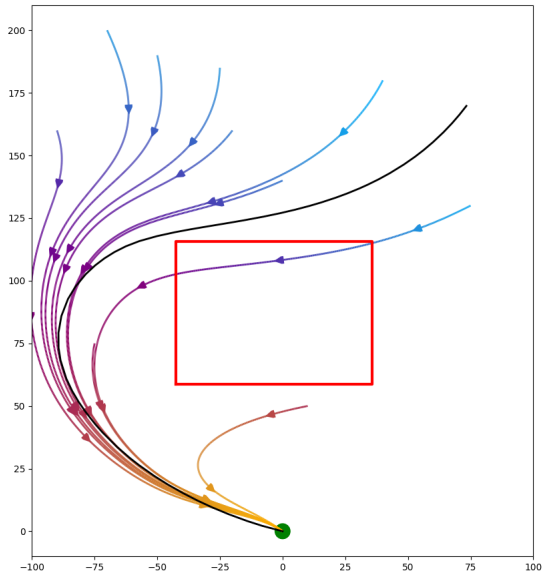
Definition 3.6. For any state $\mathbf{y}$, we also define the time to convergence κ_i as the time it takes for the state to enter and remain in the vicinity of $\mathbf{0}$ (which should be defined *a priori*). More formally, $\kappa_i(\mathbf{y})$ is the infinimum of the set of times t such that all $\mathbf{x}_{\mathbf{y},f_i}(t')$ with $t' \geq t$ are in the vicinity of $\mathbf{0}$.



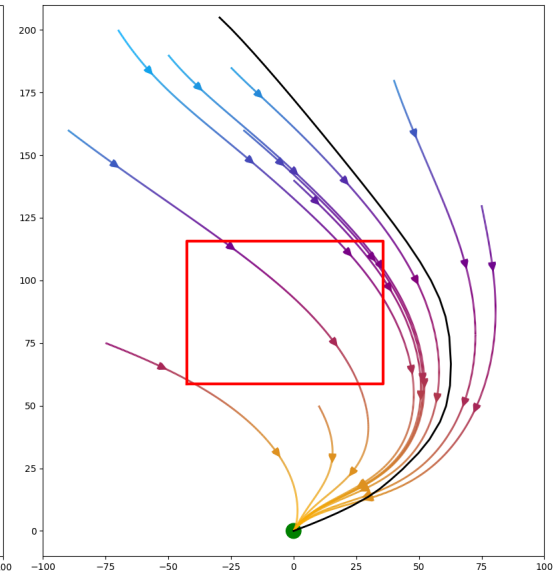
(a) 3 conflicting demonstrations



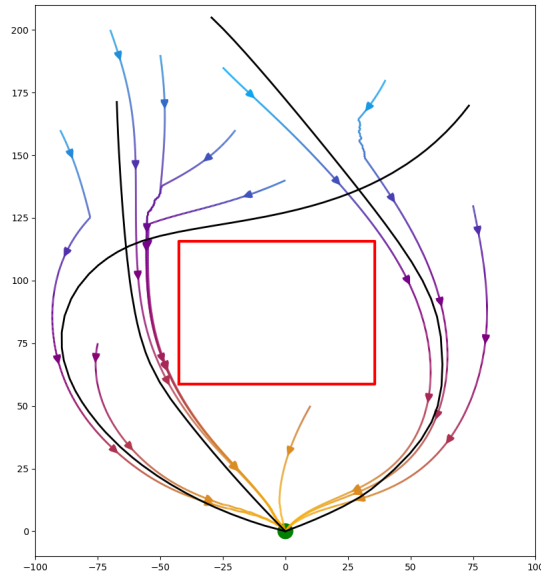
(b) $\dot{\mathbf{x}} = f_1(\mathbf{x})$



(c) $\dot{\mathbf{x}} = f_2(\mathbf{x})$



(d) $\dot{\mathbf{x}} = f_3(\mathbf{x})$



(e) $\dot{\mathbf{x}} = f^{\varepsilon}_{combined}(\mathbf{x})$

Figure 3.6: Single vector fields combined into $\dot{\mathbf{x}} = f^{\varepsilon}_{combined}(\mathbf{x})$, with $\varepsilon = 10$.

Both $\alpha_i^\varepsilon(\mathbf{y})$ and $\kappa_i(\mathbf{y})$ are straightforward to estimate once an approximation of $(\mathbf{x}_{\mathbf{y},f_i}(t))_{t\geq 0}$ is known. Next, we define the combined vector field (or DS) $\dot{\mathbf{x}} = f^\varepsilon_{combined}(\mathbf{x})$.

For a state $\mathbf{y}$, we compute the pairs $(\alpha_i^\varepsilon(\mathbf{y}), \kappa_i(\mathbf{y}))$ for $i \in \{1,\dots,n\}$. First, we find the vector fields leading to the shortest times to ε-absorption, i.e. $\underset{i\in\{1,\dots,n\}}{\arg\min}\, \alpha_i^\varepsilon(\mathbf{y})$. The idea is that staying close to a demonstration ensures reliability because it minimizes the need for extrapolation. Therefore, shorter times to ε-absorption increase the likelihood of success for the generated trajectory.

- If there is no "tie", i.e. if $\underset{i\in\{1,\dots,n\}}{\arg\min}\, \alpha_i^\varepsilon(\mathbf{y})$ contains a unique element j, then we set $f^\varepsilon_{combined}(\mathbf{x}) = f_j(\mathbf{x})$.
- If, on the contrary, there are several vector fields leading to the same minimum time to ε-absorption (usually 0, when $\mathbf{y}$ is ε-absorbed by more than one DS), then among them we choose the one ($\dot{\mathbf{x}} = f_{j'}(\mathbf{x})$) that leads to the shortest time to convergence, and we set $f^\varepsilon_{combined}(\mathbf{x}) = f_{j'}(\mathbf{x})$. If there is still a tie (usually because $\mathbf{y}$ is in the vicinity of $\mathbf{0}$), an arbitrary order on the vector fields is used to make a choice.

We cannot formally prove that the combined DS $\dot{\mathbf{x}} = f^\varepsilon_{combined}(\mathbf{x})$ is globally asymptotically stable, as with the above definition it is in general not continuous, which can pose challenges for the existence and uniqueness of the solutions. However, when solutions exist, they all converge to $\mathbf{0}$.

Theorem 3.5. *Any solution* $(\mathbf{x}_{\mathbf{y},f^\varepsilon_{combined}}(t))_{t\geq 0}$ *converges toward* $\mathbf{0}$*.*

Proof. From any state $\mathbf{y}$, each vector field $\dot{\mathbf{x}} = f_i(\mathbf{x})$ leads to times to ε-absorption that are finite and decreasing at a constant rate until reaching 0. In a trajectory of $\dot{\mathbf{x}} = f^\varepsilon_{combined}(\mathbf{x})$, we switch between vector fields in a way that can only reduce the time to ε-absorption. Therefore, the trajectory eventually reaches a state that is ε-absorbed for one of the $\dot{\mathbf{x}} = f_i(\mathbf{x})$ DS. At this point, all future switches to other DS will preserve the ε-absorption and the time to ε-absorption will remain equal to 0. Therefore, future switches can only decrease the time to convergence, which is for every system finite and decreasing at a constant rate until reaching 0. We conclude that the trajectory $(\mathbf{x}_{\mathbf{y},f^\varepsilon_{combined}}(t))_{t\geq 0}$ will also reach and remain in the vicinity of $\mathbf{0}$, where there can only be zero or one final switch to one of the $\dot{\mathbf{x}} = f_i(\mathbf{x})$ DS. Since these systems are all globally asymptotically stable, the trajectory will converge to $\mathbf{0}$. □

Figure 3.6 illustrates trajectories generated in practice with $\dot{\mathbf{x}} = f^\varepsilon_{combined}(\mathbf{x})$.

3.4 Limitations

An obvious weakness of the diffeomorphic matching-based approach is that it does not take dynamics into account, so in some sense the extrapolated vector fields assume omnidirectional controlability of the system. As a consequence, the set of applications in which the method can be applied is restricted. When it is crucial to take dynamics into account, extrapolating away from demonstrations is difficult. In the architecture proposed in Chapter 2, we learn to predict the success of a control command, so in order to get closer to a target trajectory (e.g. of the robot head), one could randomly sample control inputs with the objective of finding ones that are likely to be successful and would drive the robot closer to the known trajectory. This solution might work well in practice for a wide class of systems, but the notion of distance to the trajectory should ideally also take dynamics into account, and the efficiency of randomly sampling control inputs steeply declines in complex spaces. If a model of the dynamics is known, transforming the problem of getting closer to the known trajectory into an optimization problem is another possibility. If only partial information about the dynamics of the system is known, such as some of its basic physical properties, this knowledge can be used to learn a physics-informed dynamics model [Asri *et al.* 2024], which can then be leveraged to attempt to converge smoothly toward known trajectories.

CHAPTER 4

Sequencing motions

At the end of Chapter 2, the proposed motion planner constructs paths made of sequences of intermediate targets (for the robot head) and associated control commands that have a high likelihood of success. The sequence of commands might not be optimal, so to execute one of these paths, it would not be ideal to wait for the success of each command before executing the next one. The resulting motion is likely to be smoother if we perform *early switches*, i.e. if we try to switch to the next control command before actually completing the current one.

Definition 4.1. If there are two consecutive objectives or goals $\mathbf{g}_1$ and $\mathbf{g}_2$, we call it an *early switch* on $\mathbf{g}_1$ when we start to pursue $\mathbf{g}_2$ before actually achieving $\mathbf{g}_1$.

A similar principle can be applied if we just have the sequence of intermediate targets or goals and need to train from scratch a policy that should make the robot achieve the last goal.

In this context, we want to train a goal-conditioned policy π, i.e. a policy that takes in input both the state (or observation) $\mathbf{s}$ and a desired goal $\mathbf{g}$. A sparse reward of 1 is obtained when the policy achieves the desired goal. The environment is assumed to be a Markov Decision Process, and the training of the policy is done by repeating episodes (also called rollouts) starting from the neighborhood of a single initial configuration. Episodes are recorded as sequences of transitions of the form $(\mathbf{s}, \mathbf{g}, \mathbf{a}, \mathbf{r}, \mathbf{s}')$, where $\mathbf{s}$ is a state, $\mathbf{g}$ the current desired goal, $\mathbf{a}$ an action, $\mathbf{r}$ the reward obtained, and $\mathbf{s}'$ the next state. These transitions fill a *replay buffer* from which batches are randomly sampled to train the policy with an *off-policy* reinforcement algorithm like SAC [Haarnoja *et al.* 2018] or TD3 [Fujimoto *et al.* 2018]. Furthermore, to make the training more efficient, we rely on Hindsight Experience Replay [Andrychowicz *et al.* 2017] which consists in relabelling sequences of consecutive transitions by assuming that the achieved goal of the last transition was also the desired goal throughout the sequence, and by computing the virtual rewards that would have been obtained in that case.

The sequence of goals is $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_n$. At the beginning of each episode, the first desired goal is $\mathbf{g}_1$. Then, before the episode ends (we assume that episodes are truncated if they exceed a maximum length), we can decide at any moment to change the current desired goal to any of the goals from the list. Our objective is to manage switches in the best possible way so that the learning goes well and the policy eventually manages to follow the whole sequence of goals one after the other until the last one. Achieving the last goal is the true objective, while the intermediate goals are simply supposed to help guiding the robot toward it.

The goal-conditioned policy is used in a local way as it does not have advanced exploration capabilities. If during episodes we always quickly switch to $\mathbf{g}_n$, the problem might be too difficult and the policy will never learn how to reach it. Conversely, never switching early would result in trying to precisely reach each of the goals consecutively, which might be too difficult as well.

So, the question is: *when to switch?*

4.1 Dynamic Value Threshold

Off-policy actor-critic algorithms like SAC and TD3 train the policy but also a Q-value function that estimates the expected sum of future discounted rewards after making an action $\mathbf{a}$ in state $\mathbf{s}$. In the case of a sparse reward obtained when the desired goal is achieved, the the Q-value can be interpreted as the level of confidence in the ability to quickly reach the desired goal.

A high Q-value corresponds to a high probability of reaching it soon, so we could put a threshold on the Q-value to determine when it is time to switch to the next goal. The idea is that, if I'm confident that I could achieve the current goal quickly, I can forget about it and already start aiming at the next one. But how to set this threshold? In deterministic environments, the theoretical value of being k steps away from the goal can be calculated, but defining a fixed threshold in advance is not the best idea.

In fact, value-based RL algorithms tend to solve problems long before the convergence of the Q-value function. The reason is that, to make correct action decisions, a global bias in the Q-values is not harmful (what is only needed is that better actions lead to higher Q-values).

So, even when the trained policy quickly becomes optimal, the convergence of the Q-values can be slow. This is illustrated in Figure 5.11 by applying SAC to the *InvertedDoublePendulum* MuJoCo environment [Todorov *et al.* 2012, Towers *et al.* 2023]. The policy trained by SAC reaches a close-to-optimal behavior in less than 7,000 gradient steps, but even after 30,000 gradient steps Q-values continue to slowly increase. This means that, with a fixed value threshold, either there will be no switches for a long time, or the switches will end up occuring too much.

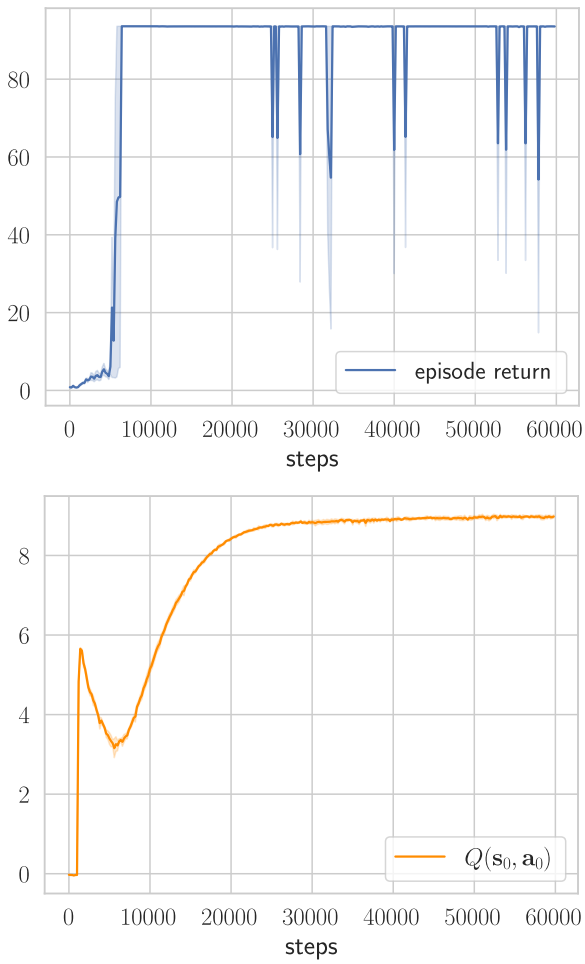


Figure 4.1: Training of Soft Actor-Critic (SAC) on InvertedDoublePendulum-v4. Results show the interquartile mean (see [Agarwal *et al.* 2021]) from experiments on 10 different random seeds. The orange curve (bottom) displays the Q-value of the first state and action of evaluation episodes. In fewer than 7,000 steps, the trained policy is near-optimal, yet after 30,000 steps, the Q-values are still slowly increasing.

The idea of the Dynamic Value Threshold is to adjust it continuously by observing what happens during the training. Whenever a goal is achieved, the corresponding value threshold is updated to a value that was observed a few steps before reaching the goal. More precisely, the proposed simple heuristic is to choose in advance an integer $k \geq 0$, and define the updates of $\tau^{\mathbf{g}}_{DVT}$, the Dynamic Value Threshold for a goal $\mathbf{g}$, as follows:

Definition 4.2. *Dynamic Value Threshold:* Assume that, during an episode, a new desired goal $\mathbf{g}$ is set in a state $\mathbf{s}_j$, and after a sequence of transitions, $\mathbf{g}$ is reached (with the desired goal unchanged during the sequence). Let $(\mathbf{s}_j, \mathbf{g}, \mathbf{a}_j, \mathbf{r}_j, \mathbf{s}_{j+1})$, $(\mathbf{s}_{j+1}, \mathbf{g}, \mathbf{a}_{j+1}, \mathbf{r}_{j+1}, \mathbf{s}_{j+2})$, …, $(\mathbf{s}_i, \mathbf{g}, \mathbf{a}_i, \mathbf{r}_i, \mathbf{s}_{i+1})$ be the sequence of transitions (the goal is reached at the last transition).

The value of $\tau^{\mathbf{g}}_{DVT}$ is updated to $Q(\mathbf{s}_q, \mathbf{g}, \mathbf{a}_q)$, where Q is the Q-function trained along with the policy, and $q = \max(j, i-k)$:

$$\tau^{\mathbf{g}}_{DVT} := Q(\mathbf{s}_q, \mathbf{g}, \mathbf{a}_q).$$

Remarks:

- We are using a goal-conditioned policy, so the Q-function also takes in input the desired goal, not only the state and action (but the desired goal can be thought of as a part of the state).
- In stochastic environments, the Q-value does not only depend on the expected number of steps to reach a goal, it also takes into account transition uncertainties. In this case, it might be preferable to update $\tau^{\mathbf{g}}_{DVT}$ with a moving average of the previously encountered $Q(\mathbf{s}_q, \mathbf{g}, \mathbf{a}_q)$ values.

- When there are good reasons to believe in a uniformity of $Q(\mathbf{s}, \mathbf{a})$, a unique τ_{DVT} (independent from the goals) could potentially be used.

4.2 Budget-based Switching

Now, we know how to trigger early switches, but we still need to decide whether to trigger them or not.

One naive solution would be to first learn how to reach a goal $\mathbf{g}_i$ by repeatedly reaching it, and then, once we are confident in our ability to achieve it, always perform early switches toward the next goal. This is problematic in two ways: first τ_{DVT} would stop being adjusted, so the Q-values drift might make early switches inadequate, and second, by lack of training we might eventually lose the capability to reach $\mathbf{g}_i$, thus if reaching it becomes required again, for instance to readjust the threshold, this could negatively impact the learning. The best idea is to find a good balance between trying to actually reach a goal and performing early switches toward the next one.
A fixed ratio between reaching and switching does not work, because as the sequence of goals gets longer, the probability of performing early switches on the whole sequence would drop exponentially.

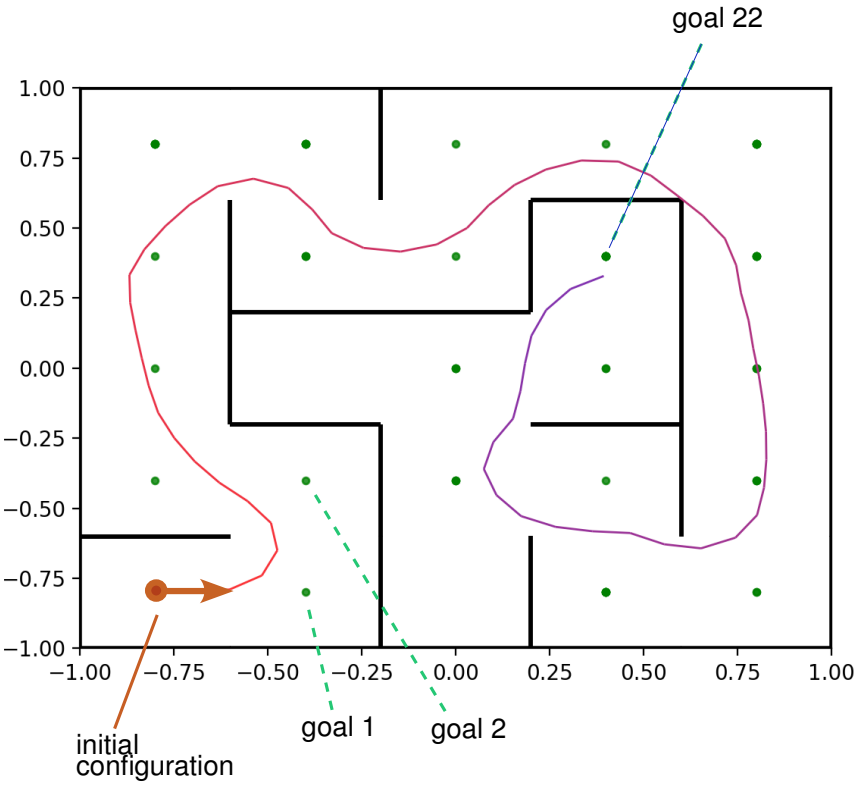


Figure 4.2: A Dubins car is trained to follow a sequence of 22 goals with Budget-based Switching (Algorithm 2). The off-policy RL with the goal-conditioned policy is relies on Soft Actor-Critic SAC and Hindsight Experience Replay (HER). After 109k gradient and environment steps, the policy is able to reach the last goal.

Instead, we consider the following scenario: if currently the furthest goal reached during the training is $\mathbf{g}_m$, ideally we would like to cycle through these phases repeatedly:

- Reach $\mathbf{g}_1$.
- Reach $\mathbf{g}_2$ (after an early switch on $\mathbf{g}_1$).

...

- Reach $\mathbf{g}_i$ (after early switches on $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_{i-1}$).

...

- Reach $\mathbf{g}_m$ (after early switches on $\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_{m-1}$).
- Early switches throughout the episode.

...

- Early switches throughout the episode.

} m times in total.

} Repeat.

In this scenario, each goal is reached once, and there are $2m-1$ early switches on $\mathbf{g}_1$, $2m-2$ early switches on $\mathbf{g}_2$, ..., m early switches on $\mathbf{g}_m$. This motivates the constant updating of a "budget" $\mathcal{B}_{\mathbf{g}_i}$ for each goal $\mathbf{g}_i$, which works as follows: when aiming at $\mathbf{g}_i$, if $\mathcal{B}_{\mathbf{g}_i} > 0$, then an early switch is allowed, and if it is indeed triggered then the budget is decreased by one. In parallel, every time $\mathbf{g}_i$ is reached, the budget $\mathcal{B}_{\mathbf{g}_i}$ is increased by $2m - i$.

Combining this budget-based switching with the Dynamic Value Threshold, we obtain the following learning algorithm:

Algorithm 2 Budget-based Switching

Input:
A sequence of goals $\mathbf{g}_1, \dots, \mathbf{g}_n$.
Parameters:
An integer $k \geq 0$.

Initialize a goal-conditioned policy π_θ, a Q-function Q_ψ, and an empty replay buffer $\mathfrak{R}_b$.
Initialize budgets $\mathcal{B}_{\mathbf{g}_i} = 0$ and thresholds $\tau_{DVT}^{\mathbf{g}_i} = 0$, $i \in \{1, \dots, n\}$.
Initialize $m_{max} = 0$.
for each episode **do**
  Reset state $\mathbf{s}$, set $b_{block} := \text{false}$, $i_{curr} := 1$ and $\mathcal{H} := [\,]$ ($\mathcal{H}$ is the "history").
  repeat
    Sample action $\mathbf{a} \sim \pi_\theta(\cdot|\mathbf{s}, \mathbf{g}_{i_{curr}})$.
    Perform environment step: $\mathbf{s}' = step(\mathbf{s}, \mathbf{a})$, get reward $r = 1$ if the step achieves $\mathbf{g}_{i_{curr}}$, 0 otherwise.
    Insert $(\mathbf{s}, \mathbf{g}_{i_{curr}}, \mathbf{a}, r, \mathbf{s}')$ in $\mathfrak{R}_b$ and append $Q_\psi(\mathbf{s}, \mathbf{g}_{i_{curr}}, \mathbf{a})$ to $\mathcal{H}$.
    If $r = 1$ and $i_{curr} > m_{max}$:
      Set $m_{max} := i_{curr}$.
    If $r = 1$ and $\mathcal{B}_{\mathbf{g}_{i_{curr}}} = 0$:
      Set $b_{block} := \text{true}$.
    If $r = 1$:
      Set $\mathcal{B}_{\mathbf{g}_{i_{curr}}} := \mathcal{B}_{\mathbf{g}_{i_{curr}}} + 2m_{max} - i_{curr}$ and $\tau_{DVT}^{\mathbf{g}_{i_{curr}}} := \mathcal{H}[\max(0, len(\mathcal{H}) - k)]$.
    If b_{block} is false and $Q_\psi(\mathbf{s}, \mathbf{g}_{i_{curr}}, \mathbf{a}) > \tau_{DVT}^{\mathbf{g}_{i_{curr}}}$ and $\mathcal{B}_{\mathbf{g}_{i_{curr}}} > 0$:
      Set $i_{curr} := \min(n, i_{curr} + 1)$, $\mathcal{B}_{\mathbf{g}_{i_{curr}}} := \mathcal{B}_{\mathbf{g}_{i_{curr}}} - 1$ and $\mathcal{H} = [\,]$.
    Draw a random batch of transitions from $\mathfrak{R}_b$.
    Perform a gradient descent step of training for π_θ and Q_ψ (using an off-policy RL algorithm).
    Set $s := s'$
  until episode termination or truncation.
end for

Remark: During an episode, once a goal is intentionally reached, no subsequent early switch is allowed. The variable b_{block} is used to block these disallowed early switches. This helps the learning stay closer to the "ideal scenario" discussed above.

Figure 4.2 shows a result obtained with this algorithm. The environment, generated with the python library gym-gmazes [Perrin-Gilbert 2022], consists of a Dubins car (a simple kinematic car model with constant speed and bounded turning rate [Dubins 1957]) in a 2D maze. At every reset, the car has the same initial position and orientation. A sequence of 22 goals guides the car toward its final target. The goals are only related to positions, not orientation (so the reward of 1 is given when the position x, y of the car enters a small disk centered at the goal location, no matter what the orientation is). A goal-conditioned policy is trained with SAC and HER. When aiming at a desired goal, the goal-conditioned policy does not have information on the next one, so without early switches it may reach the desired goal with an orientation that makes reaching the next goal difficult. This makes reaching distant targets very complex. On the other hand, with early switches, the car is not constrained to achieve all goals, and it has more time to adapt its motion properly when switching from a desired goal to the next.

As shown in Figure 4.2, after 109k steps (both 109k steps in the environment and 109k gradient steps on the policy and value networks) with Algorithm 2, the car manages to follow the whole path and reach the last goal.

4.3 Going further

The general problem we address in this chapter is learning skills more efficiently by breaking them down into sequences of simpler skills.

Although early switching can provide simple and effective solutions in some cases, in others it is is not enough, and each skill needs to actively prepare the next motion in advance. Let's consider a simple example: you want to jump over a large hole in the ground, and you decompose the motion into two skills: first getting close to the edge of the hole, and then jumping to the other side. Obviously, if you just walk toward the edge of the hole, and at some point switch to the jumping objective, you have only made the problem harder as you are closer to the edge but without momentum. The right way to break down the complexity of the problem is to understand that momentum is needed for the jump, and update the first objective to take that into account. The objective would incorporate a notion of compatibility with the next skill: we don't only solve the skill, we solve it in a way that is compatible with the next one. This goes further than early switches, and Alexandre Chenu, Olivier Serris, Olivier Sigaud and I wrote articles suggesting ways to do it properly [Chenu *et al.* 2022a, Chenu *et al.* 2022b, Chenu *et al.* 2024]. The DCIL-I algorithm (DCIL stands for Divide and Conquer Imitation Learning) is introduced in [Chenu *et al.* 2022a], and DCIL-II improves it in [Chenu *et al.* 2022b]. The key idea of DCIL-II is to define a Goal-Conditioned Markov Decision Process (GC-MDP) with an extended state space that adds information about futures goals, and modify the reward function to make it compatible with the HER relabelling mechanism. The information about future states relies solely on an integer index that is automatically updated when a goal is reached. Thanks the relabelling that densifies the reward signal, the value is efficiently propagated backward during learning, and the goal-conditioned policy quickly learns to reach any of the goals in a way that prepares for the next one.

Given a sequence of low-dimensional goals extracted from a single demonstration, DCIL-II can train a goal-conditioned policy to reproduce the demonstration with a sample-efficiency that significantly exceeds previous state-of-the-art results such as PWIL [Dadashi *et al.* 2021].

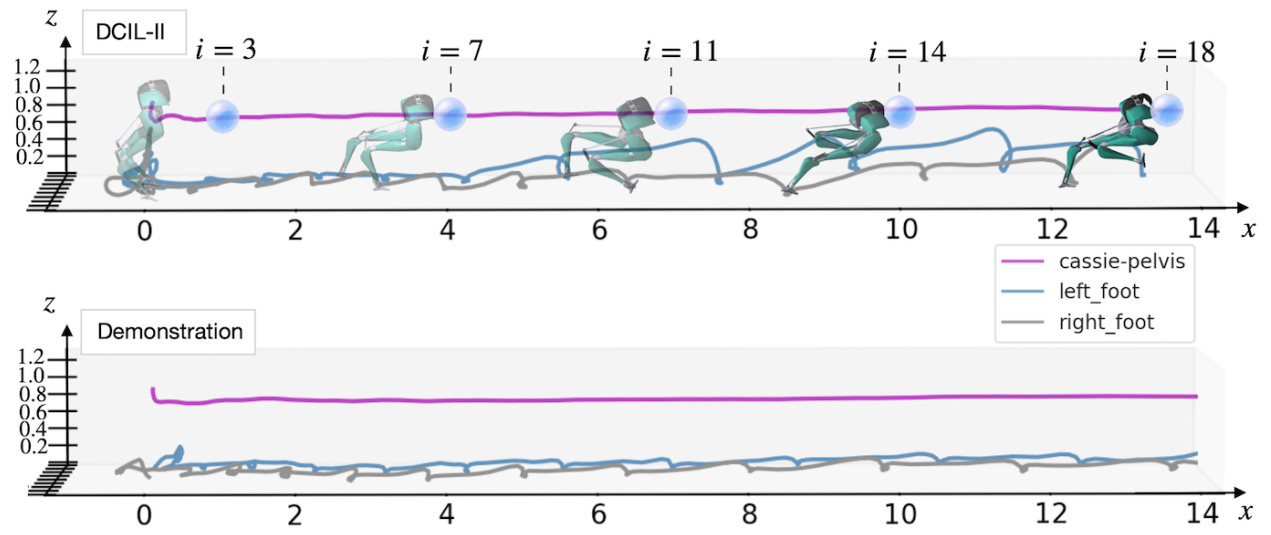


Figure 4.3: Visualization of sequential goal reaching with the Cassie biped robot in simulation. In this environment, a sequence of 19 goals corresponding to successive Cartesian positions of the pelvis is used by DCIL-II to learn the behavior using only 707k training steps on average. Only five goals are shown here. One can see that, though the straight pelvis trajectory is well reproduced, the obtained feet trajectories differ widely from the demon strated ones.

Figure 4.3 illustrates a result obtained with DCIL-II reproducing a walking trajectory with the biped robot Cassie in simulation. 19 milestone configurations and goals are extracted from the unique demonstration. One of the particularities of DCIL-II is that the goals are low-dimensional (in this example each goal is a target position for the pelvis), so we can see that in the reproduced walking motion, the trajectory of the pelvis matches that of thedemonstration, but the feet trajectories differ widely. This low-dimensionality of the goals reduces the complexity and helps focusing on what truly matters in a demonstration.

In DCIL-II, the system can be reset at milestone configurations, which correspond to the configurations in the demonstration for all the extracted goals. This prevents mechanisms like the Budget-based Switching of Algorithm 2 to be needed, and enables the learning to be much more efficient, as individual skills can be trained without having to first succeed in reaching them. Whenever these custom resets are possible,

I highly recommend to exploit them (e.g. in simulation), but there contexts in which a single reset is the only option.For this reason, we modified DCIL-II and proposed Single Reset-DCIL (SR-DCIL), which uses Dynamic Value Thresholds to govern early switches from one goal to the next.

CHAPTER 5

Off-policy RL in continuous action spaces

This is the chapter where I finally actually talk about reinforcement learning in detail.

It is somewhat orthogonal to the other chapters and is motivated by the fact that combining motion planning with RL generates a lot of *off-policy data*, specifically transitions created for exploration purposes that are not representative of standard rollouts with the current policy.

Yet, in continous state and action spaces, the most successful modern off-policy deep reinforcement learning algorithms tend to fail in batch settings, when a large part of the training data is uncorrelated to the distribution under the current policy [Fujimoto *et al.* 2019]. In this sense, they are not *truly* off-policy. On the other end, truly off-policy algorithms adapted from Q-learning [Watkins 1989], such as Implicit Q-Learning [Kostrikov *et al.* 2022], are adapted to offline RL but do not perform well in online RL. This motivates the quest for a truly off-policy algorithm that is well suited for online RL, which could be advantageous in the context of motion planning-powered RL.

The difficulty of adapting Q-learning to continuous and multi-dimensional action spaces arises from the need to compute the maximum of the Q-function over the action space. To circumvent this "max-Q problem", Q-learning can be combined with an actor-critic perspective [Silver *et al.* 2014], which involves coupling policy gradient with Q-learning updates to estimate the action-value function. Although this type of coupling primarily aims to evaluate the actor with a Q-learning critic, its byproduct is that the actor is trained to generate actions that maximize the Q-function, thereby solving the max-Q problem indirectly. This approach gave rise to DDPG [Lillicrap *et al.* 2015], a seminal actor-critic algorithm benefiting from the off-policy nature of Q-learning and consequently from a high sample-efficiency. Yet, in DDPG and its derivatives like TD3 [Fujimoto *et al.* 2018], the actor may become trapped in local optima [Matheron *et al.* 2020]. Other approaches have attempted to face the max-Q problem head-on, like CAQL [Ryu *et al.* 2020] or Implicit Q-Learning (IQL, [Kostrikov *et al.* 2022]), but the former does not scale well to high-dimensional action spaces, and requires adaptations such as constraining the action range for complex problems, whereas in the latter, the expectile loss becomes unbalanced when trying to produce estimates that are very close to the true maxima, which has so far restricted the application of IQL and similar methods to offline RL.

In this Chapter, I propose a novel way to solve the max-Q problem, using regression and conditional gradient scaling (see Section 5.3), resulting in a new algorithm that adapts Q-learning to continuous action spaces. The algorithm still has an actor to select actions and produce episodes, but unlike state-of-the-art model-free off-policy algorithms, most of which are derived from TD3 or SAC [Haarnoja *et al.* 2018], its critic updates are entirely independent from the actor. The novel algorithm is called AFU for “Actor-Free Updates”. In its first version, AFU-alpha (see Sections 5.4 and 5.5), a stochastic actor is trained like the actor in SAC. A simple failure mode of SAC (and AFU-alpha) is studied in Section 5.6, and it is shown that the value function trained by regression in AFU can help improve the actor update and make it less prone to local optima, resulting in a new version of the algorithm, AFU-beta (see Section 5.7), which does not fail in the same way. Experiments show that AFU-alpha and AFU-beta are competitive in sample-efficiency with TD3 and SAC without being more computationally expensive. To the best of my knowledge, AFU is the first model-free off-policy RL algorithm that is competitive with the state-of-the-art and truly departs from the actor-critic perspective.

5.1 Related Work

As previously mentioned, in continuous action spaces, direct approaches to solve the max-Q problem of Q-learning have faced limitations. Besides CAQL, which formulates the max-Q problem as a mixed-integer program, and IQL, which treats Q-functions as state-dependent random variables and relies on expectile regression to estimate their maxima, we can cite NAF [Gu *et al.* 2016] and ICNN [Amos *et al.* 2017], which impose action-convex Q-functions making the max-Q problem tractable, QT-Opt [Kalashnikov *et al.* 2018],

which uses a stochastic optimizer to tackle non-convex max-Q problems, or approaches based on a discretization of the action space, such as SMC-learning [Lazaric *et al.* 2007] and SDQN [Metz *et al.* 2017]. However, these methods often struggle with complex, high-dimensional continuous control tasks, either due to a lack of expressiveness or prohibitive computational costs. Close to IQL, $\mathcal{X}$-QL [Garg *et al.* 2023] is an offline RL algorithm relying on an objective directly estimating the optimal soft-value function in the maximum entropy RL setting without needing to sample from a policy. A variant of $\mathcal{X}$-QL [Garg *et al.* 2023] works in the online setting, but in this case critic updates depend on actions sampled by the actor for the Bellman backup. A unique off-policy algorithm, AWR [Peng *et al.* 2019], employs regression to train a value function and a policy but falls short of the sample efficiency achieved by state-of-the-art off-policy algorithms. Presently, the most successful approaches in model-free off-policy RL for continuous control are actor-critic algorithms with interwoven actor and critic updates. The first off-policy actor-critic algorithm was introduced in [Degris *et al.* 2012], and the most recent ones are typically based on TD3, an improvement of DDPG, or on SAC, which relies on an entropy maximization framework that led to various off-policy algorithms by creating connections between policy gradients and Q-learning updates (see [O'Donoghue *et al.* 2016]). Among the algorithms improving upon TD3 and SAC, we can mention TQC [Kuznetsov *et al.* 2020], a distributional approach to control the overestimation bias, REDQ [Chen *et al.* 2021] or AQE [Wu *et al.* 2021] which employ critic ensembles, DroQ [Hiraoka *et al.* 2021] which uses dropout and layer normalization in the critic networks, and BAC [Ji *et al.* 2023] which merges Q-function updates from SAC and IQL. While these ideas could be incorporated into the proposed algorithm AFU, this is left to future work, and the focus is kept on comparing AFU to SAC and TD3. In contrast to methods building upon SAC and TD3, AFU is structurally distinct because the critic updates remain unaffected by the actor. Notably, the critic is never trained with out-of-distribution (OOD) actions, yet AFU achieves a level of sample efficiency competitive with SAC and TD3. One might object that OOD actions can be beneficial in the online setting, because they favor exploration. As pointed out in [Garg *et al.* 2023], OOD actions in Bellman backups introduce over-optimism, but online learning allows agents to correct over-optimism by collecting additional data. Yet, achieving results comparable in sample-efficiency to TD3 and SAC without OOD actions shows that the benefit of Bellman backups with OOD actions in the online setting is in fact not so obvious. If OOD actions can introduce an over-optimism that then needs to be corrected, it may be preferable to design online learning methods that do not yield over-optimism in the first place, and use other strategies to favor exploration. Furthermore, unlike other direct adaptations of Q-learning to continuous control, AFU does not fail on the most complex tasks. On the contrary, the challenging MuJoCo task Humanoid is one of the environments in which AFU performs the best comparatively to SAC and TD3.

5.2 Preliminaries

We consider a discounted infinite horizon Markov Decision Problem (MDP) $< S, A, T, R, \gamma >$, where S is a state space, A a continuous action space, T a stochastic transition function, $R : S \times A \to \mathbb{R}$ a reward function, and $0 \leq \gamma < 1$ a discount factor. We denote by $\mathbf{s}'$ (resp. $\mathbf{s}_{t+1}$) a state obtained after performing an action $\mathbf{a}$ (resp. $\mathbf{a}_t$) in state $\mathbf{s}$ (resp. $\mathbf{s}_t$). Transitions are tuples $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ with $r = R(\mathbf{s}, \mathbf{a})$. The optimal Q-function Q^* is defined by: $Q^*(\mathbf{s}, \mathbf{a}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(\mathbf{s}_t, \mathbf{a}_t) \mid \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}, \pi^*\right]$, where the policy used from $t = 1$ onwards is π^*, which selects actions optimally in every state. The optimal value function V^* verifies $V^*(\mathbf{s}) = \max_{\mathbf{a} \in A}(Q^*(\mathbf{s}, \mathbf{a}))$. Let V_{φ_1} and V_{φ_2} denote two function approximators for the value function, and Q_ψ a function approximator for the Q-function (the critic). We use feed forward neural networks for all the function approximators. For the value function, we also consider target networks (see [Mnih *et al.* 2016]), i.e. parameter vectors $\varphi_1^{\text{target}}$ and $\varphi_2^{\text{target}}$ updated with the rule $\varphi_i^{\text{target}} \leftarrow \tau\varphi_i + (1-\tau)\varphi_i^{\text{target}}$ for some target smoothing coefficient $0 < \tau < 1$. We wish to train the critic on mini-batches B of transitions $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ taken from an experience replay buffer, with the following loss derived from the clipped Double Q-learning loss of TD3 [Fujimoto *et al.* 2018]:

$$L_Q(\psi) = \underset{(\mathbf{s},\mathbf{a},r,\mathbf{s}') \in B}{\text{Mean}} \left[\left(Q_\psi(\mathbf{s}, \mathbf{a}) - r - \gamma \min_{i \in \{1,2\}} V_{\varphi_i^{\text{target}}}(\mathbf{s}') \right)^2 \right] \tag{5.1}$$

The use of two function approximators V_{φ_1} and V_{φ_2} aims at avoiding the overestimation bias that can make Q-learning based approaches diverge (see [Hasselt 2010]). In practice, transitions can be terminal,

which requires a simple modification of the loss ignored here for the sake of clarity (γ is set to 0 for terminal transitions). Provided that $V_{\varphi_1}(\mathbf{s})$ and $V_{\varphi_2}(\mathbf{s})$ return good estimates of the maximum of $Q_\psi(\mathbf{s},\cdot)$, a maximization usually referred to as the max-Q problem (see [Ryu *et al.* 2020]), Equation (5.1) amounts to the mean squared Bellman error that drives Q_ψ toward Q^* [Baird 1999].

5.3 A new way to solve the max-Q problem

The main remaining problem is how to efficiently train V_{φ_1} and V_{φ_2}. For the learning to be successful, both V_{φ_1} and V_{φ_2} should converge to precise solutions of the max-Q problem quickly. If changes in Q_ψ are not tracked promptly, errors such as overestimation of Q-values could lead to failures.

5.3.1 Method

We introduce two new function approximators A_{ξ_1} and A_{ξ_2}, for the optimal advantage function defined by $A^*(\mathbf{s},\mathbf{a}) = Q^*(\mathbf{s},\mathbf{a}) - V^*(\mathbf{s})$. For any state-action pair $(\mathbf{s},\mathbf{a})$, $A^*(\mathbf{s},\mathbf{a}) \leq 0$. Preliminarily, we assume that outputs of A_{ξ_i} can only be non-positive. Assuming that Q_ψ is fixed, training V_{φ_i} and A_{ξ_i} can be done by minimizing the following regression loss on mini-batches B:

$$l_{V,A}(\varphi_i,\xi_i) = \underset{(\mathbf{s},\mathbf{a},\_,\_)\in B}{\text{Mean}} \left[\Big(V_{\varphi_i}(\mathbf{s}) + A_{\xi_i}(\mathbf{s},\mathbf{a}) - Q_\psi(\mathbf{s},\mathbf{a}) \Big)^2 \right]. \tag{5.2}$$

This loss causes the values $V_{\varphi_i}(\mathbf{s})$ to become upper bounds of $Q_\psi(\mathbf{s},\cdot)$, but not tight ones. A natural next step would be to add a regularization term penalizing large outputs of V_{φ_i}, which results in an approach very similar to methods based on regression with asymmetric losses such as IQL, SQL, EQL and $\mathcal{X}$-QL [Garg *et al.* 2023]. The issue is that the resulting convergence is either slow (for very small regularization coefficients) or significantly biased (for larger coefficients). For some problems, finding the right coefficient is possible, but in general standard regularization does not lead to satisfactory results in the context of online RL. We propose a different approach based on conditional gradient rescaling, noticing that when $V_{\varphi_i}(\mathbf{s}) + A_{\xi_i}(\mathbf{s},\mathbf{a})$ is greater than its target $Q_\psi(\mathbf{s},\mathbf{a})$, by gradient descent both $V_{\varphi_i}(\mathbf{s})$ and $A_{\xi_i}(\mathbf{s},\mathbf{a})$ would decrease by the same amount, and conversely when $V_{\varphi_i}(\mathbf{s}) + A_{\xi_i}(\mathbf{s},\mathbf{a})$ is smaller than the target, values would both increase by the same amount. Without regularization, all upper bounds of $Q_\psi(\mathbf{s},\cdot)$ are equally good values for $V_{\varphi_i}(\mathbf{s})$, but we can modulate the gradients to put a "downward pressure" on $V_{\varphi_i}(\mathbf{s})$ and make it progressively decrease as $A_{\xi_i}(\mathbf{s},\mathbf{a})$ progressively increases. To this end, we apply only a fraction of the gradient descent update on φ_i when $V_{\varphi_i}(\mathbf{s})$ would increase. It can be done by defining, for $0 < \varrho < 1$:

$$\Upsilon_i^{\mathbf{a}}(\mathbf{s}) = \left(1 - \varrho I_i^{\mathbf{s},\mathbf{a}}\right) V_{\varphi_i}(\mathbf{s}) + \varrho I_i^{\mathbf{s},\mathbf{a}} V_{\varphi_i^{\text{no\_grad}}}(\mathbf{s}),$$

where $\varphi_i^{\text{no\_grad}}$ is a copy of the parameters φ_i, and $I_i^{\mathbf{s},\mathbf{a}} = \begin{cases} 1, & \text{if } V_{\varphi_i}(\mathbf{s}) + A_{\xi_i}(\mathbf{s},\mathbf{a}) < Q_\psi(\mathbf{s},\mathbf{a}). \\ 0, & \text{otherwise.} \end{cases}$ Replacing $V_{\varphi_i}(\mathbf{s})$ by $\Upsilon_i^{\mathbf{a}}(\mathbf{s})$ in (5.2) yields a new version of the loss:

$$\Lambda_{V,A}(\varphi_i,\xi_i) = \underset{(\mathbf{s},\mathbf{a},\_,\_)\in B}{\text{Mean}} \left[\Big(\Upsilon_i^{\mathbf{a}}(\mathbf{s}) + A_{\xi_i}(\mathbf{s},\mathbf{a}) - Q_\psi(\mathbf{s},\mathbf{a}) \Big)^2 \right]. \tag{5.3}$$

In the next paragraph, we show that the proposed conditional gradient rescaling method is similar to an adaptive regularization scheme in which the weight of the regularization is proportional to the absolute value of the error.

Conditional gradient rescaling seen as adaptive regularization

Let us denote by $e(\mathbf{s},\mathbf{a})$ the error:

$$e(\mathbf{s},\mathbf{a}) = V_{\varphi_i}(\mathbf{s}) + A_{\xi_i}(\mathbf{s},\mathbf{a}) - Q_\psi(\mathbf{s},\mathbf{a}),$$

and let us assume that this error is negative (otherwise our method simply applies a standard gradient descent step). We denote by $e'(\mathbf{s},\mathbf{a})$ the following term:

$$e'(\mathbf{s},\mathbf{a}) = (1-\varrho) V_{\varphi_i}(\mathbf{s}) + \varrho V_{\varphi_i^{\text{no\_grad}}}(\mathbf{s}) + A_{\xi_i}(\mathbf{s},\mathbf{a}) - Q_\psi(\mathbf{s},\mathbf{a}),$$

which is equal to $e(\mathbf{s}, \mathbf{a})$ in value but has different gradients.

Our method applies a gradient descent step on $e'(\mathbf{s}, \mathbf{a})^2$ for both φ_i and ξ_i. For ξ_i, the gradient is the same as for $e(\mathbf{s}, \mathbf{a})^2$ and $e'(\mathbf{s}, \mathbf{a})^2$. For φ_i, the gradients are:

$$\nabla_{\varphi_i} e(\mathbf{s}, \mathbf{a})^2 = 2e(\mathbf{s}, \mathbf{a}) \nabla_{\varphi_i} V_{\varphi_i}(\mathbf{s}),$$

$$\nabla_{\varphi_i} e'(\mathbf{s}, \mathbf{a})^2 = (1 - \varrho) 2e(\mathbf{s}, \mathbf{a}) \nabla_{\varphi_i} V_{\varphi_i}(\mathbf{s}).$$

Let us define $e^{\text{no\_grad}}(\mathbf{s}, \mathbf{a})$: a "frozen" version of $e(\mathbf{s}, \mathbf{a})$ leading to no gradients at all (i.e. relying on copies of both φ_i and ξ_i). Our method is equivalent to the application of a gradient step (for both φ_i and ξ_i) to the following term:

$$e(\mathbf{s}, \mathbf{a})^2 - 2\varrho e^{\text{no\_grad}} V_{\varphi_i}(\mathbf{s}) = e(\mathbf{s}, \mathbf{a})^2 + 2\varrho |e^{\text{no\_grad}}(\mathbf{s}, \mathbf{a})| V_{\varphi_i}(\mathbf{s}),$$

where we see the squared error and a simple regularization term that penalizes large values of $V_{\varphi_i}(\mathbf{s})$ (thus putting a "downward pressure" on $V_{\varphi_i}(\mathbf{s})$). As a result, the proposed conditional gradient rescaling method can be understood as an adaptive regularization scheme in which the regularization weight is proportional to the absolute value of the error. It means that the convergence of $V_{\varphi_i}(\mathbf{s})$ toward $\max_{\mathbf{a} \in A}(Q_\psi(\mathbf{s}, \mathbf{a}))$ is not theoretically guaranteed: if the regression quickly converges to an exact solution, the gradients vanish and $V_{\varphi_i}(\mathbf{s})$ can remain strictly greater than the true maximum. However, if a non negligible error remains, the adaptive regularization is effective and $V_{\varphi_i}(\mathbf{s})$ progressively decreases toward an approximation of the true maximum, which is what we observe in practice.

Dealing with the sign of $A_{\xi_i}(\mathbf{s}, \mathbf{a})$

Up to now, we have assumed that $A_{\xi_i}(\mathbf{s}, \mathbf{a})$, which is going to be the output of a neural network, can only be non-positive. But imposing a strict constraint on the sign of $A_{\xi_i}(\mathbf{s}, \mathbf{a})$ could potentially lead to jittering gradients, so we instead restrict its sign in a soft way.

We let A_{ξ_i} possibly return positive outputs, but we modify the regression loss to have only non-positive *targets* for $A_{\xi_i}(\mathbf{s}, \mathbf{a})$. In Equation (5.3), we call $Q_\psi(\mathbf{s}, \mathbf{a}) - \Upsilon_i^{\mathbf{a}}(\mathbf{s})$ the *target* of $A_{\xi_i}(\mathbf{s}, \mathbf{a})$, as it is the value of $A_{\xi_i}(\mathbf{s}, \mathbf{a})$ minimizing $\left(\Upsilon_i^{\mathbf{a}}(\mathbf{s}) + A_{\xi_i}(\mathbf{s}, \mathbf{a}) - Q_\psi(\mathbf{s}, \mathbf{a})\right)^2$. If $Q_\psi(\mathbf{s}, \mathbf{a}) - \Upsilon_i^{\mathbf{a}}(\mathbf{s}) > 0$, the best non-positive *target* for $A_{\xi_i}(\mathbf{s}, \mathbf{a})$ is 0, in which case the *target* for $\Upsilon_i^{\mathbf{a}}(\mathbf{s}) - Q_\psi(\mathbf{s}, \mathbf{a})$ should also be 0. In this situation, we replace $\left(\Upsilon_i^{\mathbf{a}}(\mathbf{s}) + A_{\xi_i}(\mathbf{s}, \mathbf{a}) - Q_\psi(\mathbf{s}, \mathbf{a})\right)^2$ by $\left(\Upsilon_i^{\mathbf{a}}(\mathbf{s}) - Q_\psi(\mathbf{s}, \mathbf{a})\right)^2 + \left(A_{\xi_i}(\mathbf{s}, \mathbf{a})\right)^2$. To do so, we introduce Z:

$$Z(x, y) = \begin{cases} (x + y)^2, & \text{if } x \geq 0. \\ x^2 + y^2, & \text{otherwise.} \end{cases} \tag{5.4}$$

The loss of Equation 5.3 is updated as follows:

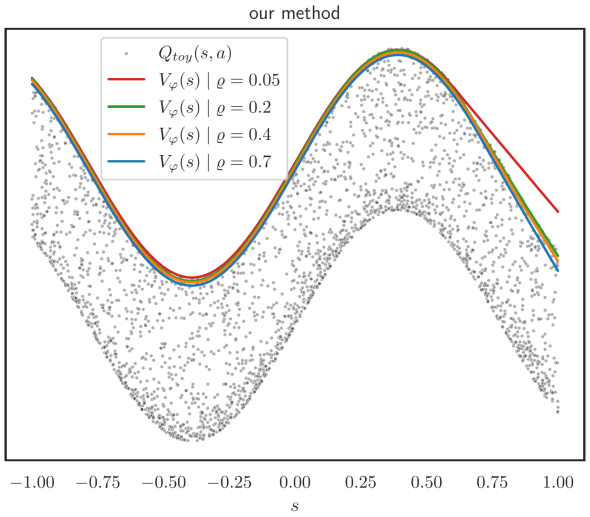
$$\Lambda'_{V,A}(\varphi_i, \xi_i) = \underset{(\mathbf{s}, \mathbf{a}, \_, \_) \in B}{\text{Mean}} \left[Z\left(\Upsilon_i^{\mathbf{a}}(\mathbf{s}) - Q_\psi(\mathbf{s}, \mathbf{a}), A_{\xi_i}(\mathbf{s}, \mathbf{a})\right) \right]. \tag{5.5}$$

5.3.2 Experiments

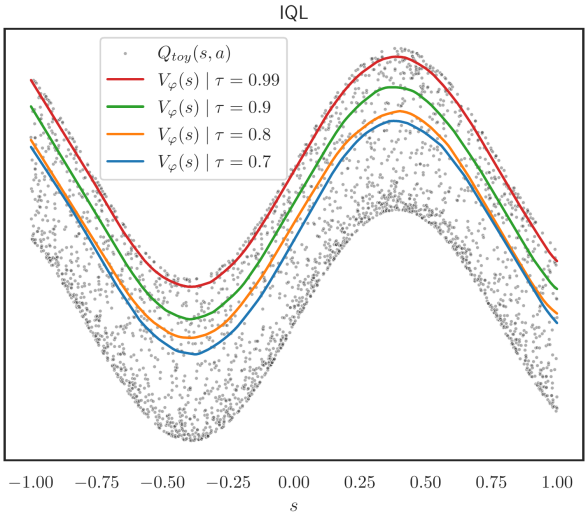
We empirically compare our method to 3 baselines (IQL, SQL and EQL) on a toy problem. We define the function $Q_{toy}(s, a) = \sin(4s) + 0.7\cos(4a)$ for $s \in [-1, 1]$ and $a \in [-1, 1]$. We use a single feedforward neural network for V (V_φ) and a single feedforward neural network for A (A_ξ). Both networks have two hidden layers of size 256 and ReLU activations in the hidden layers. Our method trains both V_φ and A_ξ, while the 3 baselines IQL, SQL and EQL directly train V_φ. All 3 baselines have been successfully applied to offline reinforcement learning.

IQL is Implicit Q-Learning [Kostrikov *et al.* 2022]. For a fixed s, IQL treats $Q_{toy}(s, a)$ as a random variable (the randomness being determined by the action) and uses an expectile regression loss to train $V_\varphi(s)$ to estimate a state conditional upper expectile of this random variable. The expectile is determined by the parameter $0 < \tau < 1$, and the closer τ is to 1, the closer $V_\varphi(s)$ gets to $\max_{a \in A}(Q_{toy}(s, a))$. However, if τ is very close to 1 (e.g. $\tau = 0.99$), the loss becomes unbalanced, with elements weighted hundreds of

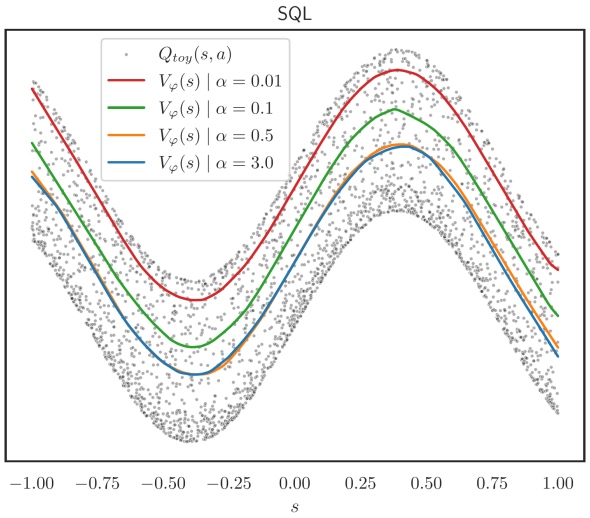
times more than others, which results in instabilities in the context of RL, so in practice the values used in [Kostrikov *et al.* 2022] are not greater than 0.9.



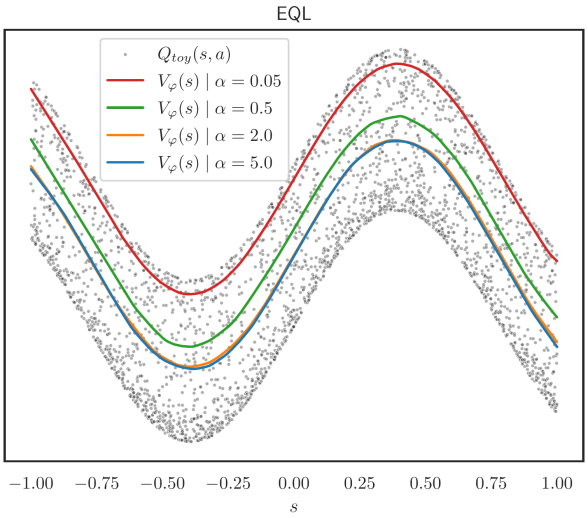
(a) $V_\varphi(s)$ is trained jointly with $A_\xi(s,a)$ by iterating gradient descent steps on the loss $\Lambda'_{V,A}(\varphi,\xi)$ described by Equation (5.5). $\varrho \in [0.2, 0.7]$ results in precise approximations of $s \mapsto \max_{a\in A}(Q_\psi(s,a))$.



(b) Results of the training with the loss from IQL [Kostrikov *et al.* 2022] for 4 different values of the hyperparameter τ. Values used in actual (offline) RL experiments are not greater than 0.9.



(c) Results of the training with the loss from SQL [Xu *et al.* 2023] for 4 different values of the hyperparameter α. Values used in actual (offline) RL experiments are not smaller than 0.1.



(d) Results of the training with the loss from EQL [Xu *et al.* 2023] for 4 different values of the hyperparameter α. Values used in actual (offline) RL experiments are not smaller than 0.5..

Figure 5.1: $Q_{toy}(s,a) = \sin(4s) + 0.7\cos(4a)$ for $s \in [-1,1]$ and $a \in [-1,1]$. We compare our method to IQL, SQL and EQL which all train $V_\varphi(s)$ to approximate the function $s \mapsto \max_{a\in A}(Q_{toy}(s,a))$, i.e. solve the max-Q problem. All trainings are done with 3000 gradient descent steps. At each step, a loss is computed on a batch composed of 256 uniformly randomly drawn values of s and a.

SQL and EQL are derived in [Xu *et al.* 2023] from a general method called Implicit Value Regularization. It relies on a behavior-regularized MDP with a term that penalizes policies diverging from the underlying behavior policy of the training dataset. Various f-divergences can be used to measure the difference between the policy and the behavior policy, resulting in distinct algorithms, including SQL and EQL which

are special cases. They have distinct losses for the training of $V_\varphi(s)$, both depending on a parameter α, and in both cases, for $\alpha \to 0$, $V_\varphi(s)$ is trained to approximate the maximum operator over in-support values, i.e. $\max_{a \in A}(Q_{toy}(s,a))$. However, similarly to IQL, very small values of α result in unbalanced losses, so in practice the values leading to the best results on the benchmarks tested in [Xu *et al.* 2023] are $\alpha = 0.1$, $\alpha = 0.5$, $\alpha = 1$ and $\alpha = 3$ for SQL, and $\alpha = 0.5$, $\alpha = 2.0$ and $\alpha = 5$ for EQL.

Figure 5.1 compares our method to IQL, SQL and EQL. For IQL, SQL and EQL, since unbalanced losses are not an issue on this simple toy problem, we include parameters leading to a better resolution of the max-Q problem, but that are not representative of the parameters working well in actual offline RL experiments. We observe that, with our method, although $\varrho = 0.05$ leads to overestimations, a wide range of parameter values (from $\varrho = 0.2$ to $\varrho = 0.7$) yield precise results, while with other methods precise results are only obtained with hyperparameter values that are inapplicable to RL (e.g. $\tau = 0.99$ for IQL). Besides, with our proposed approach, the different values of ϱ that perform well do not result in unbalanced losses.

5.4 Actor-free critic updates and actor training

Let us remove the dependency to Q_ψ in loss (5.5) by replacing $Q_\psi(\mathbf{s},\mathbf{a})$ by the targets used to train it in (5.1). We obtain the following loss (for $i \in \{1,2\}$):

$$L_{V,A}(\varphi_i,\xi_i) = \underset{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\in B}{\text{Mean}} \left[Z\Big(\Upsilon_i^{\mathbf{a}}(\mathbf{s}) - r - \gamma \min_{i\in\{1,2\}} V_{\varphi_i^{\text{target}}}(\mathbf{s}'), A_{\xi_i}(\mathbf{s},\mathbf{a}) \Big) \right]. \tag{5.6}$$

With the losses $L_Q(\psi)$ (5.1) and $L_{V,A}(\varphi_i,\xi_i)$ (5.6), we can train Q_ψ, V_{φ_i} and A_{ξ_i} without needing an actor. Compared to methods derived from DDPG (like TD3), solving directly the max-Q problem has an advantage over first using an actor to solve the argmax-Q problem, i.e. to approximate $\text{argmax}_{\mathbf{a}\in A}(Q_\psi(\mathbf{s},\mathbf{a}))$. The reason is that continuous changes in Q_ψ result in continuous changes of its state conditioned maxima, while it can result in discontinuous changes of its state conditioned argmax. So, in an off-policy setting, if the exploration policy discovers better results with very different actions, the maximum of $Q_\psi(\mathbf{s},\mathbf{a})$ can be tracked smoothly, while the tracking of the argmax can be much more difficult, with the potential arising of deceptive value landscapes in which the actor can get stuck (see [Matheron *et al.* 2020]). This theoretical advantage, as well as the actor-free Q_ψ, V_{φ_i} and A_{ξ_i} updates are all important aspects of our approach. However, since we are interested in online reinforcement learning, we still need an actor to select actions and produce episodes. To train this actor, if we would use the same gradient ascent over $a \mapsto Q_\psi(\mathbf{s},\mathbf{a})$ as in DDPG, our global method would be prone to the same failure modes as DDPG, and most of the advantages of the max-Q based training of V_{φ_i} would be lost. One thing we can notice is that, since we do not need the actor to return $\text{argmax}_{\mathbf{a}\in A}(Q_\psi(\mathbf{s},\mathbf{a}))$, we also do not need the actor to be deterministic. To benefit from a better exploration, we opt for a stochastic actor and follow the approach proposed in SAC [Haarnoja *et al.* 2018] with automatic tuning of the temperature parameter α.

Let π_θ denote the actor. We follow a common implementation in which its backbone is a feedforward neural network returning action distributions as state-dependent Gaussians with diagonal covariance matrices. Since actions are usually constrained between -1 and 1, we apply a $\tanh$ transformation to its outputs. Given a state $\mathbf{s}$, the resulting probability density function is $\pi_\theta(\cdot|\mathbf{s})$. The actor π_θ can transform input noise vectors sampled from a fixed distribution into action samples. Again, we train π_θ on mini-batches of transitions. We use the actor to resample an action $\mathbf{a}_s$ for each state $\mathbf{s}$ of a mini-batch B. The actor loss $L_\pi(\theta)$ is based on the average Kullback-Leibler divergence between the actor's output distributions and targeted Boltzmann policy distributions. It is defined as follows:

$$L_\pi(\theta) = \underset{\substack{(\mathbf{s},\_,\_,\_)\in B \\ \mathbf{a}_s\sim\pi_\theta(\cdot|\mathbf{s})}}{\text{Mean}} \Big[\alpha \log(\pi_\theta(\mathbf{a}_s|\mathbf{s})) - Q_\psi(\mathbf{s},\mathbf{a}_s) \Big]. \tag{5.7}$$

where α is a temperature parameter. As in SAC, we adjust this temperature via gradient descent on a loss aiming at keeping the average entropy of action distributions close to a target entropy $\bar{\mathcal{H}}$:

$$L_{\text{temp}}(\alpha) = \underset{\substack{(\mathbf{s},\_,\_,\_)\in B \\ \mathbf{a}_s\sim\pi_\theta(\cdot|\mathbf{s})}}{\text{Mean}} \Big[-\alpha \log(\pi_\theta(\mathbf{a}_s|\mathbf{s})) - \alpha\bar{\mathcal{H}} \Big]. \tag{5.8}$$

5.5 AFU-alpha

We combine the losses L_Q (5.1), $L_{V,A}$ (5.6), L_π (5.7), and L_{temp} (5.8) to devise a new off-policy reinforcement learning algorithm. It has a critic (Q_ψ) and an actor (π_θ), but the critic updates are derived from our novel adaptation of Q-learning to continuous action spaces (obtained with our new method to solve the max-Q problem), therefore they are independent from the actor. Hence the name AFU for the algorithm, for "Actor-Free Updates". We specifically call it AFU-alpha to contrast it with AFU-beta introduced in Section 5.7. AFU-alpha, described in Algorithm 3, alternates between environments steps that gather experience in a replay buffer and gradient steps that draw batches from the replay buffer to compute loss gradients and update all parameters of the function approximators. In our implementation, an iteration consists of a single environment step followed by a single gradient step.

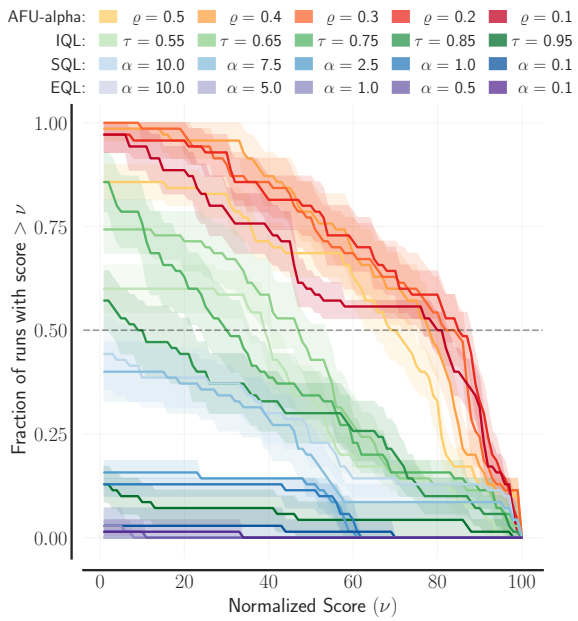
Algorithm 3 AFU-alpha and AFU-beta

Set $0 < \varrho < 1$, $0 < \tau < 1$, $\bar{\mathcal{H}}$, and learning rates η_Q, $\eta_{V,A}$, η_π, η_{temp}.
Initialize empty replay buffer $\mathfrak{R}_b$, and params ψ, $\varphi_1 = \varphi_1^{\text{target}}$, $\varphi_2 = \varphi_2^{\text{target}}$, ξ_1, ξ_2, α, θ.
for each iteration **do**
  for each environment step **do**
    Sample action $\mathbf{a} \sim \pi_\theta(\cdot|\mathbf{s})$.
    Perform environment step $\mathbf{s}, \mathbf{a} \to \mathbf{s}'$, compute $r = R(\mathbf{s}, \mathbf{a})$, and insert $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ in $\mathfrak{R}_b$.
  end for
  for each gradient step **do**
    Draw batch of transitions B from $\mathfrak{R}_b$ and compute loss gradients on that batch.
    $\psi \leftarrow \psi - \eta_Q \nabla_\psi L_Q(\psi)$
    $\varphi_{i\in\{1,2\}} \leftarrow \varphi_i - \eta_{V,A} \nabla_{\varphi_i} L_{V,A}(\varphi_i, \xi_i)$
    $\xi_{i\in\{1,2\}} \leftarrow \xi_i - \eta_{V,A} \nabla_{\xi_i} L_{V,A}(\varphi_i, \xi_i)$
    $\varphi_{i\in\{1,2\}}^{\text{target}} \leftarrow \tau\varphi_i + (1-\tau)\varphi_i^{\text{target}}$
    $\zeta \leftarrow \zeta - \eta_\pi \nabla_\zeta L_\mu(\zeta)$
    $\theta \leftarrow \theta - \eta_\pi \nabla_\theta L_\pi(\theta)$
    $\theta \leftarrow \theta - \eta_\pi \nabla_\theta^{\text{MODIF}} L_\pi(\theta)$
    $\alpha \leftarrow \alpha - \eta_{\text{temp}} \nabla_\alpha L_{\text{temp}}(\alpha)$
  end for
end for

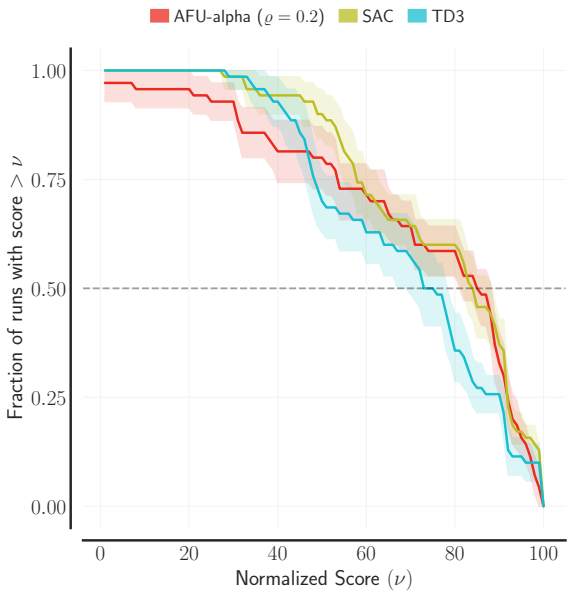
Experiments We test AFU-alpha on a classical benchmark of 7 MuJoCo [Todorov *et al.* 2012] tasks from the Gymnasium library [Towers *et al.* 2023]. We compare it to SAC and TD3, and to variants of AFU-alpha in which the loss $L_{V,A}$ aiming at solving the max-Q problem is replaced by the corresponding loss taken from IQL, SQL or EQL. The results are shown in Figure 5.2. For both SAC and AFU-alpha, we use the same heuristic for the definition of $\bar{\mathcal{H}}$: we set it to $-d$, where d is the dimension of the action space. Updates in AFU-alpha, SAC and TD3 use the same value of τ and same learning rates. For each algorithm, for each value of the hyperparameter (ϱ for AFU-alpha, τ for IQL, α for SQL and EQL), and for each of the 7 MuJoCo tasks, we perform 10 runs initialized with different random seeds, and evaluate the performance of the policy every 10,000 steps on 10 rollouts. The first 10,000 steps of each run use uniformly drawn random actions (and no gradient steps). Learning curves are smoothed with a moving average window of size 10. The raw score of a run is the last average return, i.e. the average return over the last 10 evaluations. For each task, we linearly rescale the scores based on two reference points: (1) the maximum evaluation seen across all algorithms and all runs corresponds to a score of 100, and (2) the mean episode return across all algorithms and runs corresponds to a score of 0. Following the recommendations of [Agarwal *et al.* 2021], we compute with the *rliable* library the performance profiles for each algorithm across the 7 tasks: Ant-v4, HalfCheetah-v4, Hopper-v4, Humanoid-v4, InvertedDoublePendulum-v4, Reacher-v4 and Walker2d-v4. The length of the runs is 1 million steps for InvertedDoublePendulum and Reacher, 3 million steps for Ant, Hopper, Humanoid and Walker2d, and 5 million steps for HalfCheetah.

In Figure 5.2 (a), we see that our proposed method for the max-Q problem yields significantly better results than the IQL, SQL and EQL baselines. The best results are obtained with $\varrho \in \{0.2, 0.3\}$. Figure 5.2 (b)

shows that AFU-alpha is competitive with SAC and TD3.



(a) AFU-alpha works best with $\varrho \in \{0.2, 0.3\}$. Using the IQL, SQL and EQL baselines to solve the max-Q problem results in a clear performance deterioration.



(b) AFU-alpha is competitive with SAC and TD3 on a benchmark of 7 diverse tasks, with action space dimensions ranging from 1 (InvertedDoublePendulum) to 17 (Humanoid), and observation space dimensions ranging from 11 to 376 (Humanoid).

Figure 5.2: Experimental evaluation of AFU-alpha on a benchmark of 7 MuJoCo tasks.

5.6 A simple failure mode of SAC

With a deterministic actor trained by stochastic gradient ascent over the Q-function landscape, DDPG, TD3 and similarly structured deterministic actor-critic algorithms can easily get stuck in local optima (see [Matheron *et al.* 2020]). With a stochastic actor and updates based on the Kullback-Leibler (KL) divergence between output distributions and target distributions of the form $\exp\big(\frac{1}{\alpha}Q(\mathbf{s},\cdot)\big)/z(\mathbf{s})$, algorithms like SAC are less prone to deadlocks. For instance, in areas where the gradient of the Q-function is close to zero, exploiting the KL loss results in an increase of the variance of the action distribution, which eventually helps find larger gradients and escape from the flat region. Yet, the policy networks used in practice mostly output unimodal action distributions[1], and with this restriction even the KL loss generates undesirable local optima. We illustrate this with a trivial environment which we call SFM (for "SAC Failure Mode"). It consists of a single state $\mathbf{s}_0$, and unidimensional actions in $[-1, 1]$. The reward of an action is given by the function:

$$R_{SFM}(\mathbf{s}_0, a) = \begin{cases} 5 - 100(a - 0.1)^2, & \text{if } a \geq -0.6, \\ 0, & \text{otherwise.} \end{cases} \tag{5.9}$$

All transitions are terminal, so all episodes stop after one step. The optimal policy selects $a = 0.1$ and yields a return of 5. We train SAC on SMF with the same hyperparameters as in our other experiments. We start by performing 1000 steps with random actions, which helps the critic Q_{SAC} quickly converge toward the optimal Q-function, Q^*, which is simply equal to R_{SFM}. Figure 5.3 shows Q_{SAC} after 20,000

[1]This is starting to change, thanks to the influence of recent methods such as diffusion policies (see [Chi *et al.* 2023, Hansen-Estruch *et al.* 2023]), but such expressive and multimodal stochastic policies are still more cumbersome than unimodal policies.

steps. Although Q_{SAC} converges toward a very precise approximation of Q^*, the actor policy converges toward a suboptimal solution, as shown in Figure 5.4 (a). If we just modify SAC by locking the mode of the policy distribution at 0, we can see in Figure 5.4 (b) that the actor loss becomes much smaller, even after convergence of the actor entropy, which indicates that the policy of the default SAC algorithm gets stuck in a local optimum. There are two phases in the failure mode: at the beginning, when the entropy is relatively large, the asymmetry of R_{SFM} makes the actor shift toward -1. As seen in Figure 5.3, Q_{SAC} approximates the discontinuity in R_{SFM} with a steep slope, and when the policy distribution becomes concentrated on the left of this slope, it acts as a barrier that traps the actor. Later in the training, when the entropy becomes smaller and converges to the target entropy (-1), it would be much preferable for the mode of the policy to converge back toward 0.1, but the steep slope results in a deceptive gradient in the KL loss that prevents it from happening, and SAC remains stuck in the local optimum.

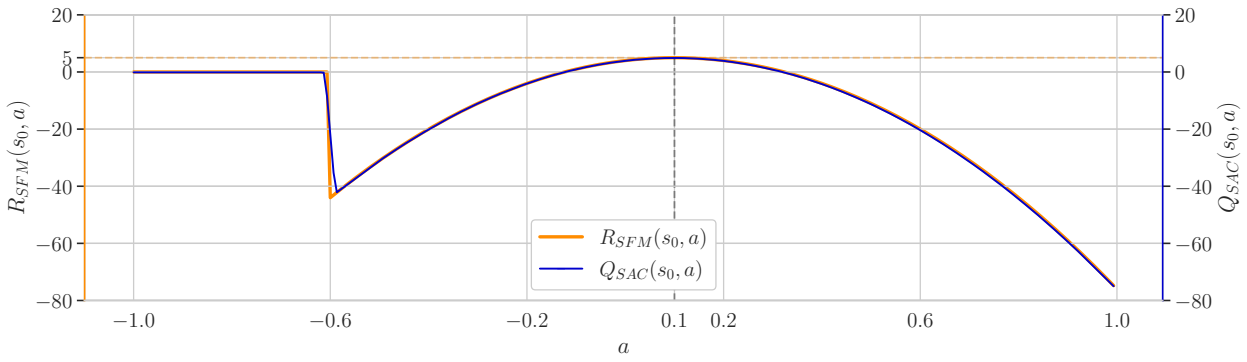
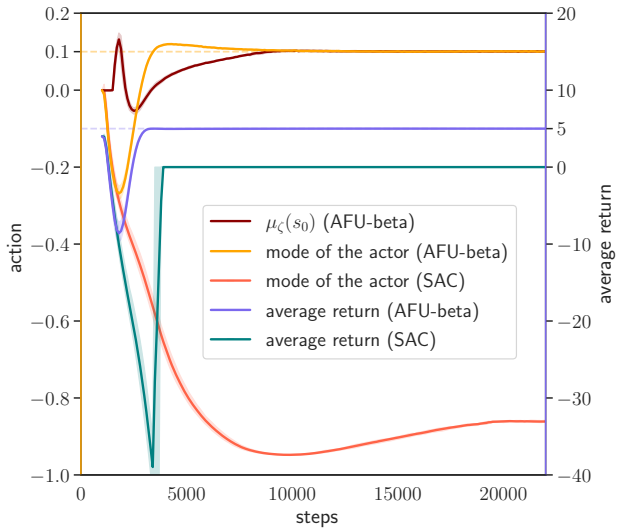
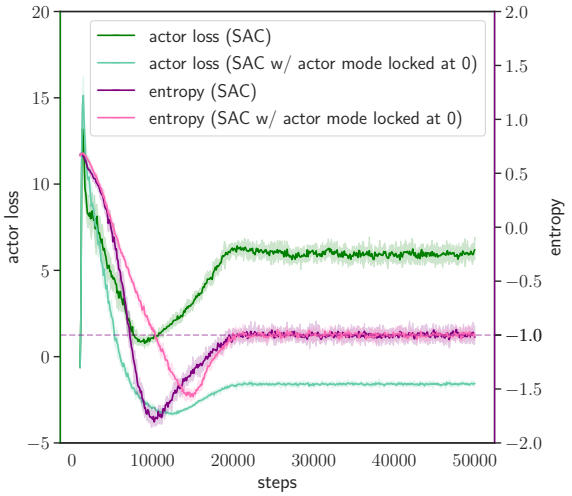


Figure 5.3: In orange: the reward function R_{SFM} of the SFM environment. Since all transitions are terminal, R_{SFM} coincides with the optimal Q-function. In blue: the critic (Q_{SAC}) obtained after a training of 20,000 steps with SAC [Haarnoja *et al.* 2018].



(a) AFU-beta converges to the optimal solution, while SAC converges to a suboptimal solution. How to read the figure: the y-axis on the left (action) applies to the first three curves ($\mu_\zeta(\mathrm{s})$ and the modes of the actor for AFU-beta and SAC), while the y-axis on the right (average return) applies for the two other curves (average returns for AFU-beta and SAC).



(b) Evolution of the actor loss and entropy for SAC and SAC with the mode of its actor locked at 0. The y-axis on the left (actor loss) applies to the first two curves (actor losses for both versions of SAC), and the y-axis on the right applies to the two other curves (entropies for both versions of SAC).

Figure 5.4: Trainings of SAC and AFU-beta in the SFM environment. Plots show results averaged over 10 runs with different random seeds, and shaded areas range from the 25th to the 75th percentile.

5.7 AFU-beta

With the same actor loss as SAC, AFU-alpha fails similarly on SFM. We propose to improve the actor loss to make it less likely to get stuck in local optima. The first idea is to train by regression an estimate of where the mode of the actor should be. If the learning progresses well, $Q_\psi(\mathbf{s},\mathbf{a}) > \min_{i\in\{1,2\}}\big(V_{\varphi_i}(\mathbf{s})\big)$ should only be possibly true in the vicinity of the argmax ($\text{argmax}_{\mathbf{a}\in A}\big(Q(\mathbf{s},\mathbf{a})\big)$), so we use actions $\mathbf{a}$ with a Q-value greater than $\min_{i\in\{1,2\}}\big(V_{\varphi_i}(\mathbf{s})\big)$ as targets. To find such actions we use both actions in the mini-batches and actions resampled with the actor on those mini-batches. Let us consider a mini-batch B of transitions $(\mathbf{s},\mathbf{a},r,\mathbf{s}')$, and actions $\mathbf{a}_s$ resampled with the actor. We denote by $\mathcal{M}(B)$ the set of state-action pairs $(\mathbf{s},\mathbf{a}_\bullet)$ such that $\mathbf{a}_\bullet = \mathbf{a}$ or $\mathbf{a}_\bullet = \mathbf{a}_s$ and $Q_\psi(\mathbf{s},\mathbf{a}_\bullet) > \min_{i\in\{1,2\}}\big(V_{\varphi_i}(\mathbf{s})\big)$. We introduce a new deterministic function approximator $\mu_\zeta : S \to A$ with parameters ζ and train it with the following loss:

$$L_\mu(\zeta) = \underset{(\mathbf{s},\mathbf{a}_\bullet)\in\mathcal{M}(B)}{\text{Mean}}\left[\Big(\mu_\zeta(\mathbf{s}) - \mathbf{a}_\bullet\Big)^2\right]. \tag{5.10}$$

In our implementation, most of the parameters between ζ and θ are shared: we simply modify the output dimension of π_θ to make it also return $\mu_\zeta(\mathbf{s})$. It does not change the approach in any way, but when computing the gradient of the loss (5.10), one must carefully ignore the influence of the parameters ζ on resampled actions $\mathbf{a}_s$.

Let us reconsider the actor loss from Equation (5.7). It balances two terms, the first one ($\alpha\log(\pi_\theta(\mathbf{a}_s|\mathbf{s}))$) that maximizes the entropy, and the second one ($-Q_\psi(\mathbf{s},\mathbf{a}_s)$) that encourages π_θ to output actions maximizing $Q_\psi(\mathbf{s},\cdot)$. In the gradient $\nabla_\theta L_\pi(\theta)$, which can be expressed by making explicit the relationship between sampled actions $\mathbf{a}_s$ and the input noise (see [Haarnoja *et al.* 2018]), the second term results in small modifications of θ that attempt to change the actions $\mathbf{a}_s$ in the direction of $\nabla_{\mathbf{a}}Q_\psi(\mathbf{s},\mathbf{a})$, where $\mathbf{a}$ is evaluated in $\mathbf{a}_s$, and which we write by abuse of notation $\nabla_{\mathbf{a}_s}Q_\psi(\mathbf{s},\mathbf{a}_s)$. If $\nabla_{\mathbf{a}_s}Q_\psi(\mathbf{s},\mathbf{a}_s)$ points away from the global optimum, it can contribute to the creation of a local minimum in the actor loss. We want to edit $\nabla_{\mathbf{a}_s}Q_\psi(\mathbf{s},\mathbf{a}_s)$ in order to avoid deceptive gradients. To do so, we compute the dot product between $\nabla_{\mathbf{a}_s}Q_\psi(\mathbf{s},\mathbf{a}_s)$ and $\mu_\zeta(\mathbf{s}) - \mathbf{a}_s$, which is an estimate of a direction toward $\text{argmax}_{\mathbf{a}\in A}\big(Q(\mathbf{s},\mathbf{a})\big)$. If the dot product is positive or zero, the gradient does not point away from $\mu_\zeta(\mathbf{s})$, so we can keep it unchanged. However, if $\nabla_{\mathbf{a}_s}Q_\psi(\mathbf{s},\mathbf{a}_s)\cdot(\mu_\zeta(\mathbf{s}) - \mathbf{a}_s) < 0$, then we project $\nabla_{\mathbf{a}_s}Q_\psi(\mathbf{s},\mathbf{a}_s)$ onto $(\mu_\zeta(\mathbf{s}) - \mathbf{a}_s)^\perp$ to anneal the dot product. We do it only if we estimate that $\mathbf{a}_s$ is not already in the vicinity of the argmax, i.e. if $Q_\psi(\mathbf{s},\mathbf{a}_s) < \min_{i\in\{1,2\}}\big(V_{\varphi_i}(\mathbf{s})\big)$. We introduce the following operator $\mathcal{G}^{\mathbf{s},\mathbf{a}_s} : A \to A$:

$$\mathcal{G}^{\mathbf{s},\mathbf{a}_s}(\mathbf{v}) = \begin{cases} \text{proj}_{(\mu_\zeta(\mathbf{s})-\mathbf{a}_s)^\perp}(\mathbf{v}), & \text{if } \mathbf{v}\cdot(\mu_\zeta(\mathbf{s}) - \mathbf{a}_s) < 0 \text{ and } Q_\psi(\mathbf{s},\mathbf{a}_s) < \min_{i\in\{1,2\}}\big(V_{\varphi_i}(\mathbf{s})\big). \\ \mathbf{v}, & \text{otherwise.} \end{cases} \tag{5.11}$$

When computing the gradient $\nabla_\theta L_\pi(\theta)$, we replace the terms $\nabla_{\mathbf{a}_s}Q_\psi(\mathbf{s},\mathbf{a}_s)$ (resulting from the chain rule) by $\mathcal{G}^{\mathbf{s},\mathbf{a}_s}\big(\nabla_{\mathbf{a}_s}Q_\psi(\mathbf{s},\mathbf{a}_s)\big)$. It leads to a modified gradient which we denote by $\nabla_\theta^{\text{MODIF}} L_\pi(\theta)$.

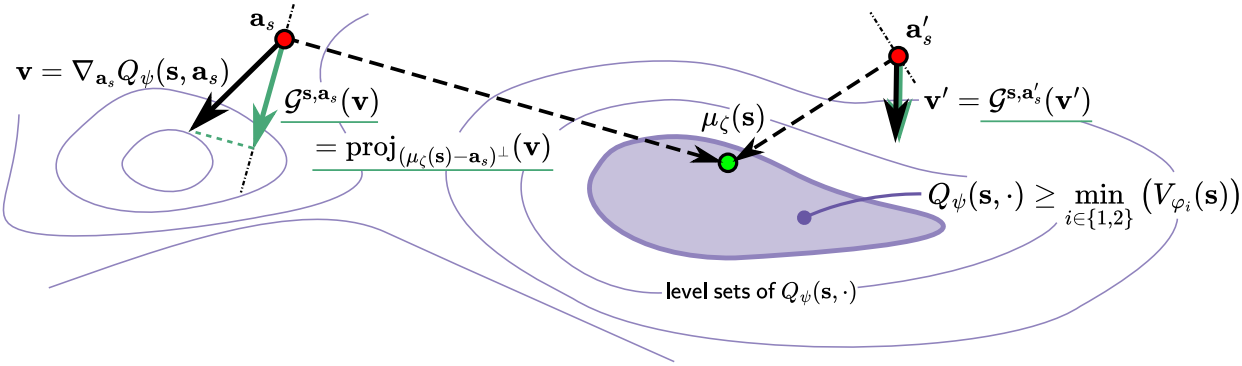


Figure 5.5: The gradient $\mathbf{v}$ at $\mathbf{a}_s$ (on the left) points away from $\mu_\zeta(\mathbf{s})$, which determines the direction toward the vicinity of the argmax of $Q_\psi(\mathbf{s},\cdot)$, so we modify $\mathbf{v}$ to get $\mathcal{G}^{\mathbf{s},\mathbf{a}_s}(\mathbf{v})$ by projecting it on the hyperplane orthogonal to $\mu_\zeta(\mathbf{s}) - \mathbf{a}_s$. The gradient $\mathbf{v}'$ at $\mathbf{a}'_s$ (on the right) points in the direction (half-space) of $\mu_\zeta(\mathbf{s})$, so we do not modify it, and $\mathcal{G}^{\mathbf{s},\mathbf{a}'_s}(\mathbf{v}') = \mathbf{v}'$.

This process is illustrated in Figure 5.5. It can be understood as an artificial modification of the landscape of $Q_\psi(\mathbf{s}, \cdot)$ so that, outside the region defined by $Q_\psi(\mathbf{s}, \cdot) \geq \min_{i \in \{1,2\}} \big(V_{\varphi_i}(\mathbf{s})\big)$, its gradient never points away from $\mu_\zeta(\mathbf{s})$. $\mu_\zeta(\mathbf{s})$ has the advantage of being trained by regression, and its training includes actions coming directly from the replay buffer, not only ones resampled by the actor. It means that, in a very off-policy setting, if a new peak of $Q_\psi(\mathbf{s}, \mathbf{a})$ appears far from the actions currently likely to be sampled, the update of $\mu_\zeta(\mathbf{s})$ can occur first and then guide the update of π_θ by removing all deceptive gradients that would need to be crossed to reach the new peak. More generally, the use of μ_ζ prevents the actor from being trapped in local optima, as long as the training of the critic is doing well. Since training the critic is independent from the actor, we believe that our proposed approach goes one step further in the development of sound foundations for a purely off-policy reinforcement learning algorithm performing well in continuous action spaces.

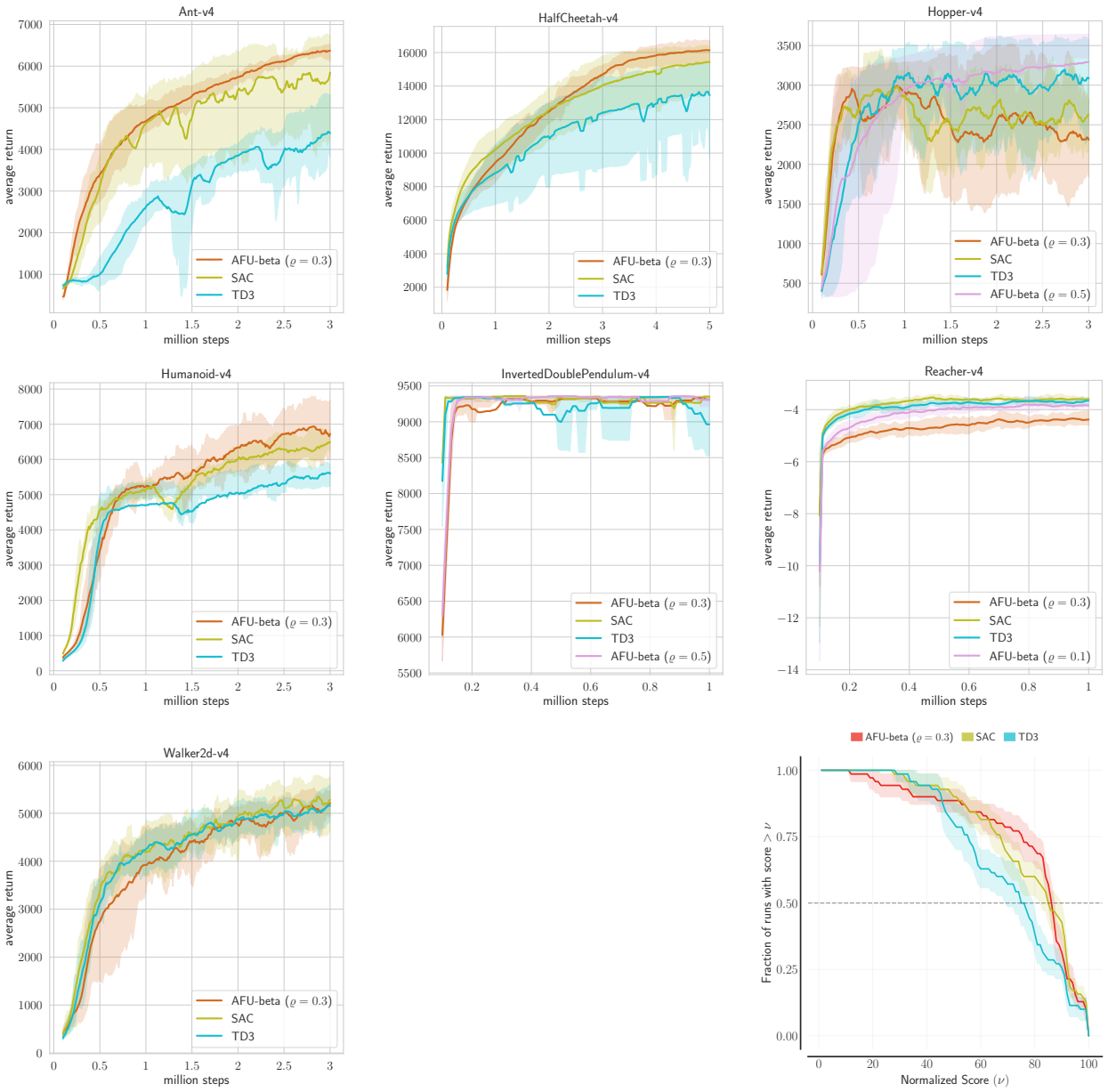


Figure 5.6: Experimental evaluation of AFU-beta for $\varrho = 0.3$ on 7 MuJoCo tasks. We show results with other values of ϱ (among $\{0.1, 0.2, 0.4, 0.5\}$) for tasks in which one of the other values performed significantly better than 0.3. Results are averaged over 10 runs with different random seeds, and the shaded areas range from the 25th to the 75th percentile. The performance profile plot at the bottom right summarizes results and shows that AFU-beta is competitive with SAC and TD3.

We call AFU-beta the updated algorithm. It works like AFU-alpha, with the additional training of μ_ζ and the replacement of $\nabla_\theta L_\pi(\theta)$ by $\nabla_\theta^{\text{MODIF}} L_\pi(\theta)$, as described in Algorithm 3. Figure 5.4 (a) shows that, in the SFM environment, unlike SAC, AFU-beta quickly converges to the optimal solution.

We evaluate AFU-beta on the MuJoCo benchmark in the same way as AFU-alpha and show an overview of the results in Figure 5.6. Again, AFU-beta is competitive with SAC and TD3. The differences between AFU-beta and AFU-alpha are not very significant on the MuJoCo benchmark, possibly because issues with local optima are rarely encountered in these environments. We leave for future work the search for meaningful and complex environments in which AFU-beta has a notable advantage over AFU-alpha.

5.8 Hyper parameters and learning curves

The following hyperparameters were used in all the experiments, and no reward scaling was done:

AFU, SAC & TD3 Hyperparameters

optimizer	Adam [Kingma & Ba 2014]
actor learning rate	$3 \cdot 10^{-4}$
critic learning rate	$3 \cdot 10^{-4}$
temperature learning rate **(only AFU & SAC)**	$3 \cdot 10^{-4}$
discount (γ)	0.99
replay buffer size	10^6
initial steps with random actions	10^4
number of hidden layers (all networks)	2
number of hidden units per layer	256
number of samples per mini-batch	256
nonlinearity	ReLU
target smoothing coefficient (τ)	0.01
target update interval	1
policy update interval **(only TD3)**	2
exploration noise standard deviation **(only TD3)**	0.2
noise clipping **(only TD3)**	0.5
target entropy **(only AFU & SAC)**	$-d$ ($d =$ action space dimension)
initial temperature **(only AFU & SAC)**	1
max. actor log std (before tanh) **(only AFU & SAC)**	2
min. actor log std (before tanh) **(only AFU & SAC)**	-10

The plots below show learning curves for AFU-alpha and AFU-beta on the 7 environments of the benchmark for all the values of ϱ in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$.

All learning curves are averaged over 10 runs with different random seeds, and the shaded areas range from the 25th to the 75th percentile. For each run, evaluations are done over 10 rollouts every 10,000 steps, and each run is smoothed with a moving average window of size 10. The first 10,000 steps are always done without gradient steps and with uniformly randomly drawn actions.

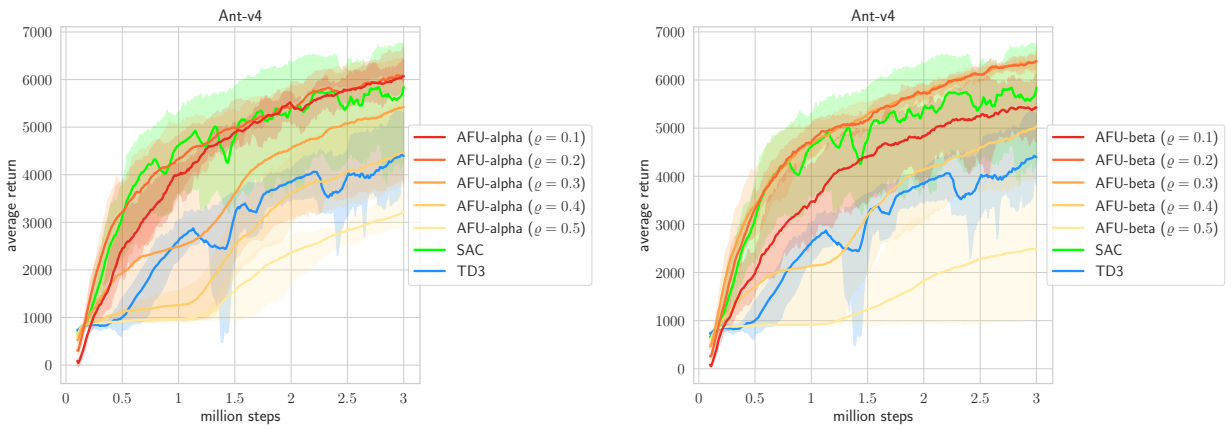


Figure 5.7: Ant-v4. *Left:* AFU-alpha. *Right:* AFU-beta.

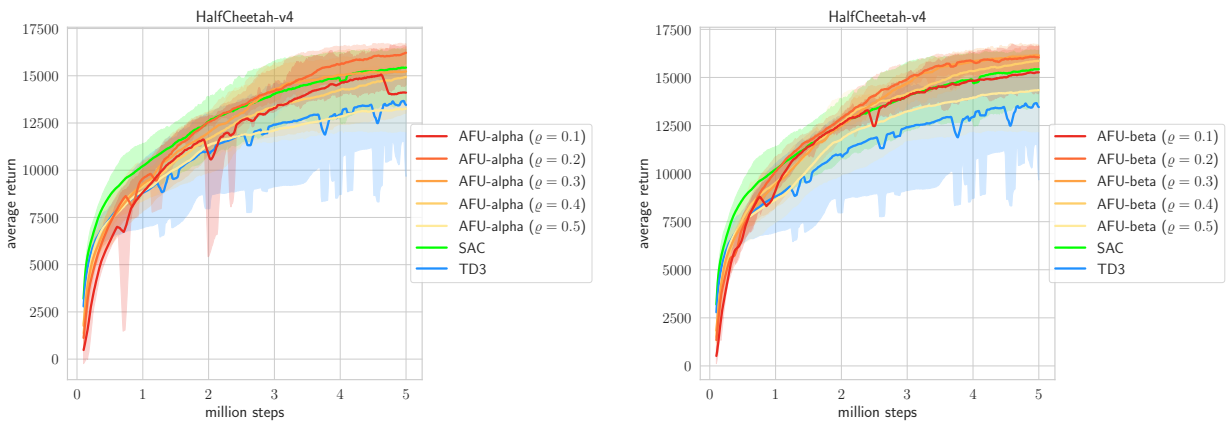


Figure 5.8: HalfCheetah-v4. *Left:* AFU-alpha. *Right:* AFU-beta.

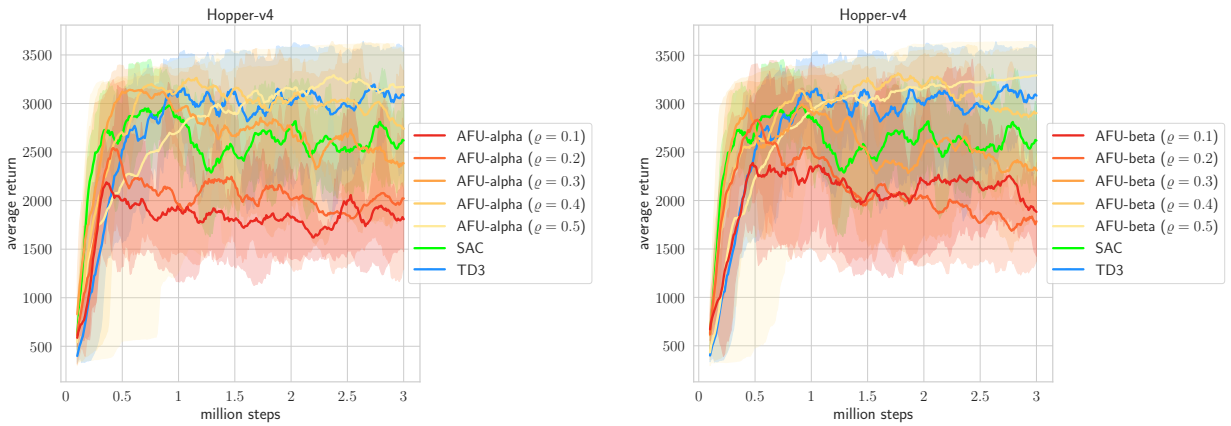


Figure 5.9: Hopper-v4. *Left:* AFU-alpha. *Right:* AFU-beta.

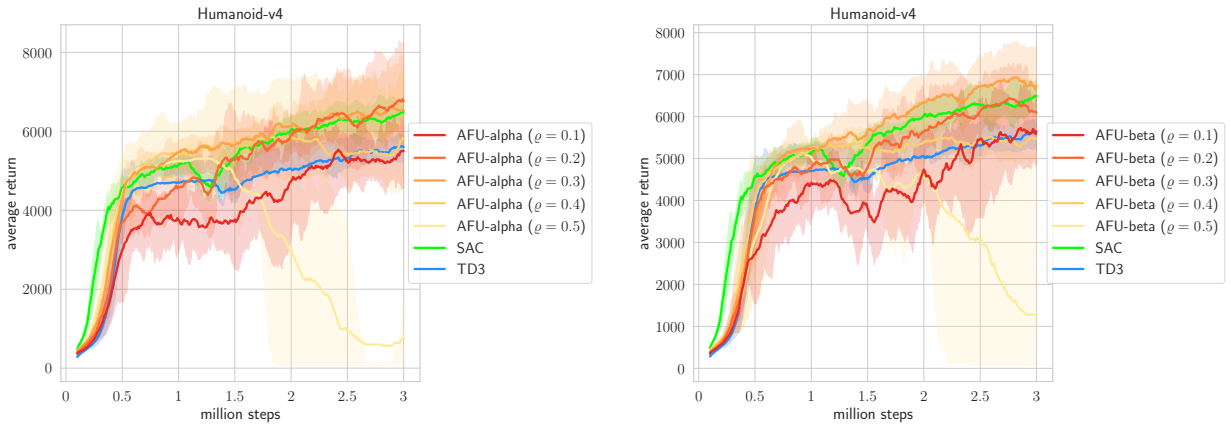


Figure 5.10: Humanoid-v4. *Left:* AFU-alpha. *Right:* AFU-beta.

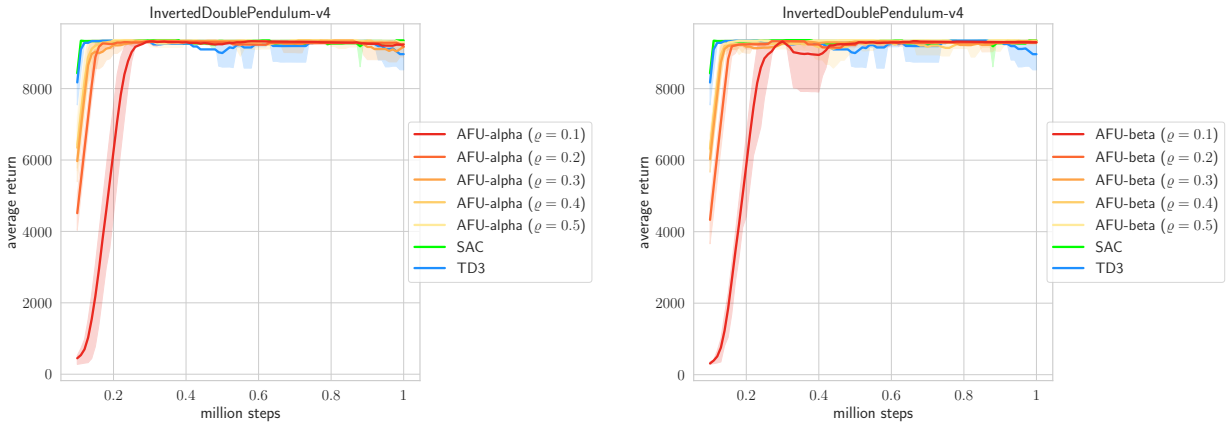


Figure 5.11: InvertedDoublePendulum-v4. *Left:* AFU-alpha. *Right:* AFU-beta.

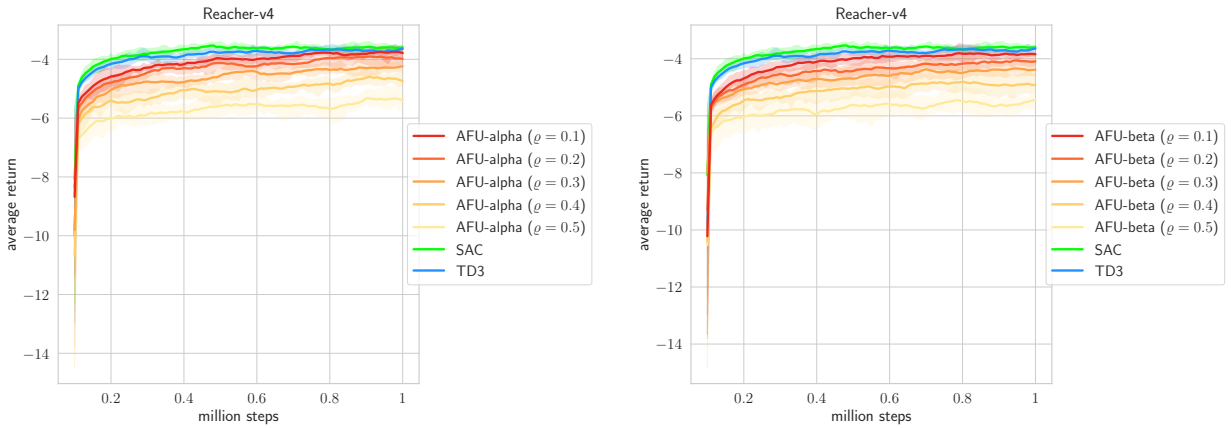


Figure 5.12: Reacher-v4. *Left:* AFU-alpha. *Right:* AFU-beta.

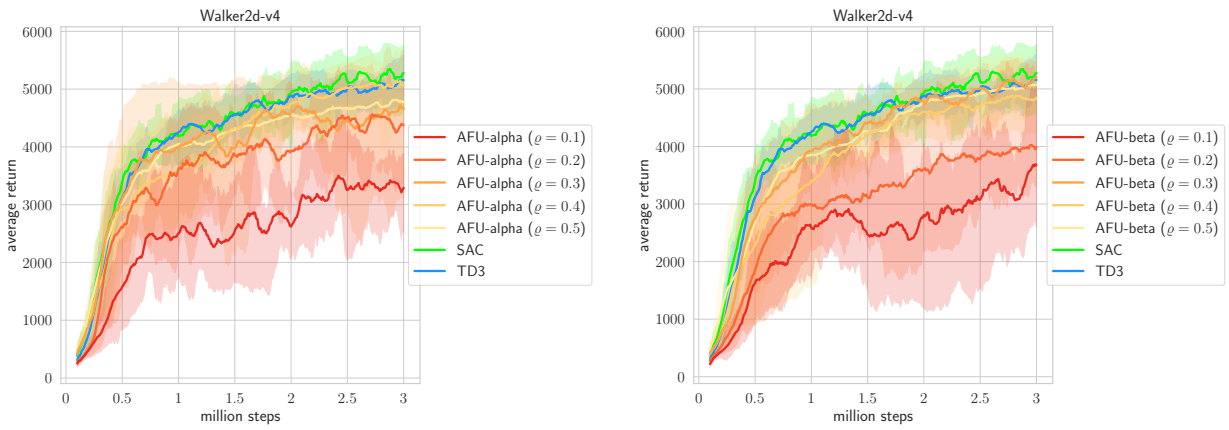


Figure 5.13: Walker2d-v4. *Left:* AFU-alpha. *Right:* AFU-beta.

CHAPTER 6

Conclusion

Hopefully, the different elements presented in this manuscript can be combined together and contribute to the design of an ambitious motion planning-powered reinforcement learning process for robot control.

In Chapter 2, I proposed a general framework combining RL and sampling-based motion planning. The obvious next step is to try to implement it. Things being usually easier written than done, will I actually do it? I don't know, but if I don't, it might actually be a good sign: maybe it will mean that I found a better way to address the problem, possibly thanks to the discussions during my HDR defense, who knows! Or maybe I will finally become convinced that just scaling end-to-end RL is the best way to move forward, and that ingenious motion planning techniques can be forgotten. In any case, I'm not too worried about my future research endeavors, as they will in all likelihood benefit from the confidence boost that might—or might not—come from beating the final boss of the French diploma game.

Let me now conclude with a drawing of my son, Evan, because Science should never ignore Art:

Bibliography

[Agarwal *et al.* 2021] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville and Marc Bellemare. *Deep reinforcement learning at the edge of the statistical precipice*. Advances in Neural Information Processing Systems, vol. 34, pages 29304–29320, 2021.

[Amos *et al.* 2017] Brandon Amos, Lei Xu and J Zico Kolter. *Input convex neural networks*. In International Conference on Machine Learning, pages 146–155. PMLR, 2017.

[Andrychowicz *et al.* 2017] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel and Wojciech Zaremba. *Hindsight Experience Replay*. In Advances in Neural Information Processing Systems, volume 30, pages 5048–5058, 2017.

[Asri *et al.* 2024] Zakariae El Asri, Olivier Sigaud and Nicolas Thome. *Physics-Informed Model and Hybrid Planning for Efficient Dyna-Style Reinforcement Learning*. arXiv preprint arXiv:2407.02217, 2024.

[Baird 1999] Leemon C. Baird. *Reinforcement learning through gradient descent*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, USA, 1999.

[Benallegue *et al.* 2017] Mehdi Benallegue, Jean-Paul Laumond and Nicolas Mansard. *Robot Motion Planning and Control: Is It More than a Technological Problem?* In Geometric and Numerical Foundations of Movements, volume 117 of *Springer Tracts in Advanced Robotics*, pages 1–10. 2017.

[Bertsekas 2024] Dimitri P. Bertsekas. *Model Predictive Control and Reinforcement Learning: A Unified Framework Based on Dynamic Programming*. CoRR, vol. abs/2406.00592, 2024.

[Chen *et al.* 2021] Xinyue Chen, Che Wang, Zijian Zhou and Keith Ross. *Randomized Ensembled Double Q-learning: Learning fast without a model*. arXiv preprint arXiv:2101.05982, 2021.

[Chenu *et al.* 2022a] Alexandre Chenu, Nicolas Perrin-Gilbert and Olivier Sigaud. *Divide & Conquer Imitation Learning*. In IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pages 8630–8637. IEEE, 2022.

[Chenu *et al.* 2022b] Alexandre Chenu, Olivier Serris, Olivier Sigaud and Nicolas Perrin-Gilbert. *Leveraging Sequentiality in Reinforcement Learning from a Single Demonstration*. CoRR, vol. abs/2211.04786, 2022.

[Chenu *et al.* 2024] Alexandre Chenu, Olivier Serris, Olivier Sigaud and Nicolas Perrin-Gilbert. *Single-Reset Divide & Conquer Imitation Learning*. CoRR, vol. abs/2402.09355, 2024.

[Chestnutt *et al.* 2003] Joel Chestnutt, James Kuffner, Koichi Nishiwaki and Satoshi Kagami. *Planning biped navigation strategies in complex environments*. In IEEE International Conference on Humanoid Robots, Humanoids, 2003.

[Chi *et al.* 2023] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel and Shuran Song. *Diffusion policy: Visuomotor policy learning via action diffusion*. arXiv preprint arXiv:2303.04137, 2023.

[Chiang *et al.* 2019] Hao-Tien Lewis Chiang, Jasmine Hsu, Marek Fiser, Lydia Tapia and Aleksandra Faust. *RL-RRT: Kinodynamic Motion Planning via Learning Reachability Estimators From RL Policies*. IEEE Robotics Autom. Lett., vol. 4, no. 4, pages 4298–4305, 2019.

[Dadashi *et al.* 2021] Robert Dadashi, Léonard Hussenot, Matthieu Geist and Olivier Pietquin. *Primal Wasserstein Imitation Learning*. In 9th International Conference on Learning Representations, ICLR, 2021.

[Degris *et al.* 2012] Thomas Degris, Martha White and Richard S Sutton. *Off-policy actor-critic*. arXiv preprint arXiv:1205.4839, 2012.

[Dubins 1957] Lester E Dubins. *On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents*. American Journal of mathematics, vol. 79, no. 3, pages 497–516, 1957.

[Dupuis & Grenander 1998] Paul Dupuis and Ulf Grenander. *Variational problems on flows of diffeomorphisms for image matching*. Q. Appl. Math., vol. LVI, no. 3, page 587–600, 1998.

[Ecoffet *et al.* 2021] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley and Jeff Clune. *First return, then explore*. Nat., vol. 590, no. 7847, pages 580–586, 2021.

[Fujimoto *et al.* 2018] Scott Fujimoto, Herke Hoof and David Meger. *Addressing function approximation error in actor-critic methods*. In International Conference on Machine Learning, pages 1587–1596. PMLR, 2018.

[Fujimoto *et al.* 2019] Scott Fujimoto, David Meger and Doina Precup. *Off-Policy Deep Reinforcement Learning without Exploration*. In Proceedings of the 36th International Conference on Machine Learning, ICML, volume 97, pages 2052–2062. PMLR, 2019.

[Garg *et al.* 2023] Divyansh Garg, Joey Hejna, Matthieu Geist and Stefano Ermon. *Extreme Q-Learning: MaxEnt RL without Entropy*. In The 11th International Conference on Learning Representations, ICLR, 2023.

[Glaunès *et al.* 2008] Joan Glaunès, Anqi Qiu, Michael I Miller and Laurent Younes. *Large Deformation Diffeomorphic Metric Curve Mapping.* Int J Comput Vis, vol. 80, no. 3, pages 317–336, 2008.

[Glaunès 2006] Joan Glaunès. *"Matchine" software, Copyright (C) Université Paris Descartes*. https://helios2.mi.parisdescartes.fr/ glaunes/matchine.zip, 2006.

[Gu *et al.* 2016] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever and Sergey Levine. *Continuous deep Q-learning with model-based acceleration*. In International Conference on Machine Learning, pages 2829–2838. PMLR, 2016.

[Guo *et al.* 2006] Hongyu Guo, Anand Rangarajan and S. Joshi. *Diffeomorphic point matching*. In Handbook of Mathematical Models in Computer Vision, pages 205–219. Springer, 2006.

[Haarnoja *et al.* 2018] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel and Sergey Levine. *Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*. In International Conference on Machine Learning, pages 1861–1870. PMLR, 2018.

[Hansen-Estruch *et al.* 2023] Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba and Sergey Levine. *IDQL: Implicit Q-Learning as an Actor-Critic Method with Diffusion Policies*. CoRR, vol. abs/2304.10573, 2023.

[Hasselt 2010] Hado Hasselt. *Double Q-learning*. Advances in Neural Information Processing Systems, vol. 23, 2010.

[Hiraoka *et al.* 2021] Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi and Yoshimasa Tsuruoka. *Dropout Q-functions for doubly efficient reinforcement learning*. arXiv preprint arXiv:2110.02034, 2021.

[Hoeller *et al.* 2024] David Hoeller, Nikita Rudin, Dhionis V. Sako and Marco Hutter. *ANYmal parkour: Learning agile navigation for quadrupedal robots*. Sci. Robotics, vol. 9, no. 88, 2024.

[Ibanez *et al.* 2014] Aurélien Ibanez, Philippe Bidaud and Vincent Padois. *Emergence of humanoid walking behaviors from mixed-integer model predictive control*. In 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pages 4014–4021. IEEE, 2014.

[Ichter & Pavone 2019] Brian Ichter and Marco Pavone. *Robot Motion Planning in Learned Latent Spaces*. IEEE Robotics Autom. Lett., vol. 4, no. 3, pages 2407–2414, 2019.

[Ji *et al.* 2023] Tianying Ji, Yu Luo, Fuchun Sun, Xianyuan Zhan, Jianwei Zhang and Huazhe Xu. *Seizing Serendipity: Exploiting the Value of Past Success in Off-Policy Actor-Critic*. arXiv preprint arXiv:2306.02865, 2023.

[Joshi & Miller 2000] Sarang C. Joshi and Michael Miller. *Landmark matching via large deformation diffeomorphisms*. IEEE Transactions on Image Processing, vol. 9, no. 8, pages 1357–1370, 2000.

[Kalashnikov *et al.* 2018] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke*et al. QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation*. arXiv preprint arXiv:1806.10293, 2018.

[Kavraki *et al.* 1996] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe and Mark H. Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. IEEE Trans. Robotics Autom., vol. 12, no. 4, pages 566–580, 1996.

[Khalil 2015] Hassan K. Khalil. *Lyapunov's Stability Theory*. In Encyclopedia of Systems and Control. Springer, 2015.

[Khansari-Zadeh & Billard 2011] Seyed Mohammad Khansari-Zadeh and Aude Billard. *Learning stable nonlinear dynamical systems with gaussian mixture models*. IEEE Transactions on Robotics, vol. 27, no. 5, pages 943–957, 2011.

[Khansari-Zadeh & Billard 2014] Seyed Mohammad Khansari-Zadeh and Aude Billard. *Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions*. Robotics and Autonomous Systems, vol. 62, no. 6, pages 752–765, 2014.

[Kingma & Ba 2014] Diederik P. Kingma and Jimmy Ba. *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.

[Kingma & Welling 2019] Diederik P. Kingma and Max Welling. *An Introduction to Variational Autoencoders*. Found. Trends Mach. Learn., vol. 12, no. 4, pages 307–392, 2019.

[Kobyzev *et al.* 2021] Ivan Kobyzev, Simon J. D. Prince and Marcus A. Brubaker. *Normalizing Flows: An Introduction and Review of Current Methods*. IEEE Trans. Pattern Anal. Mach. Intell., vol. 43, no. 11, pages 3964–3979, 2021.

[Kostrikov *et al.* 2022] Ilya Kostrikov, Ashvin Nair and Sergey Levine. *Offline Reinforcement Learning with Implicit Q-Learning*. In The 10th International Conference on Learning Representations (ICLR), 2022.

[Kuznetsov *et al.* 2020] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin and Dmitry Vetrov. *Controlling overestimation bias with truncated mixture of continuous distributional quantile critics*. In International Conference on Machine Learning, pages 5556–5566. PMLR, 2020.

[LaValle 1998] Steven LaValle. *Rapidly-exploring random trees: A new tool for path planning*. Research Report 9811, 1998.

[Lazaric *et al.* 2007] Alessandro Lazaric, Marcello Restelli and Andrea Bonarini. *Reinforcement learning in continuous action spaces through sequential monte carlo methods*. Advances in Neural Information Processing Systems, vol. 20, 2007.

[Lillicrap *et al.* 2015] 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*. arXiv preprint arXiv:1509.02971, 2015.

[Matheron *et al.* 2020] Guillaume Matheron, Nicolas Perrin and Olivier Sigaud. *Understanding failures of deterministic actor-critic with continuous action spaces and sparse rewards*. In International Conference on Artificial Neural Networks, pages 308–320. Springer, 2020.

[Metz *et al.* 2017] Luke Metz, Julian Ibarz, Navdeep Jaitly and James Davidson. *Discrete sequential prediction of continuous actions for deep RL*. arXiv preprint arXiv:1705.05035, 2017.

[Mnih *et al.* 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver and Koray Kavukcuoglu. *Asynchronous methods for deep reinforcement learning*. In International Conference on Machine Learning, pages 1928–1937. PMLR, 2016.

[Nair *et al.* 2018] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba and Pieter Abbeel. *Overcoming Exploration in Reinforcement Learning with Demonstrations*. In 2018 IEEE International Conference on Robotics and Automation, ICRA, pages 6292–6299. IEEE, 2018.

[Neumann & Steil 2015] Klaus Neumann and Jochen J. Steil. *Learning robot motions with stable dynamical systems under diffeomorphic transformations*. Robotics and Autonomous Systems, vol. 70, pages 1–15, 2015.

[O'Donoghue *et al.* 2016] Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu and Volodymyr Mnih. *Combining policy gradient and Q-learning*. arXiv preprint arXiv:1611.01626, 2016.

[OpenAI *et al.* 2021] OpenAI, Matthias Plappert, Raul Sampedro, Tao Xu, Ilge Akkaya, Vineet Kosaraju, Peter Welinder, Ruben D'Sa, Arthur Petron, Henrique Pondé de Oliveira Pinto, Alex Paino, Hyeonwoo Noh, Lilian Weng, Qiming Yuan, Casey Chu and Wojciech Zaremba. *Asymmetric self-play for automatic goal discovery in robotic manipulation*. CoRR, vol. abs/2101.04882, 2021.

[Ott *et al.* 2010] Christian Ott, Christoph Baumgärtner, Johannes Mayr, Matthias Fuchs, Robert Burger, Dongheui Lee, Oliver Eiberger, Alin Albu-Schäffer, Markus Grebenstein and Gerd Hirzinger. *Development of a biped robot with torque controlled joints*. In 10th IEEE-RAS International Conference on Humanoid Robots, Humanoids, pages 167–173. IEEE, 2010.

[Peng *et al.* 2019] Xue Bin Peng, Aviral Kumar, Grace Zhang and Sergey Levine. *Advantage-Weighted Regression: Simple and scalable off-policy reinforcement learning*. arXiv preprint arXiv:1910.00177, 2019.

[Perrin & Schlehuber-Caissier 2016] Nicolas Perrin and Philipp Schlehuber-Caissier. *Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems*. Syst. Control. Lett., vol. 96, pages 51–59, 2016.

[Perrin-Gilbert 2022] Nicolas Perrin-Gilbert. *gym-gmazes*. https://github.com/perrin-isir/gym-gmazes, 2022.

[Perrin *et al.* 2012a] Nicolas Perrin, Olivier Stasse, Leo Baudouin, Florent Lamiraux and Eiichi Yoshida. *Fast Humanoid Robot Collision-Free Footstep Planning Using Swept Volume Approximations*. IEEE Trans. Robotics, vol. 28, no. 2, pages 427–439, 2012.

[Perrin *et al.* 2012b] Nicolas Perrin, Olivier Stasse, Florent Lamiraux, Young J. Kim and Dinesh Manocha. *Real-time footstep planning for humanoid robots among 3D obstacles using a hybrid bounding box*. In IEEE International Conference on Robotics and Automation, ICRA, pages 977–982. IEEE, 2012.

[Perrin *et al.* 2017] Nicolas Perrin, Christian Ott, Johannes Englsberger, Olivier Stasse, Florent Lamiraux and Darwin G. Caldwell. *Continuous Legged Locomotion Planning*. IEEE Trans. Robotics, vol. 33, no. 1, pages 234–239, 2017.

[Perrin 2012] Nicolas Perrin. *From Discrete to Continuous Motion Planning*. In Algorithmic Foundations of Robotics X - Proceedings of the 10th Workshop on the Algorithmic Foundations of Robotics, WAFR, volume 86 of *Springer Tracts in Advanced Robotics*, pages 89–104. Springer, 2012.

[Posa *et al.* 2014] Michael Posa, Cecilia Cantu and Russ Tedrake. *A direct method for trajectory optimization of rigid bodies through contact*. Int. J. Robotics Res., vol. 33, no. 1, pages 69–81, 2014.

[Rezende & Mohamed 2015] Danilo Jimenez Rezende and Shakir Mohamed. *Variational Inference with Normalizing Flows*. In Proceedings of the 32nd International Conference on Machine Learning, ICML, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1530–1538. JMLR.org, 2015.

[Rimon & Koditschek 1991] Elon Rimon and Daniel E Koditschek. *The construction of analytic diffeomorphisms for exact robot navigation on star worlds*. Transactions of the American Mathematical Society, vol. 327, no. 1, pages 71–116, 1991.

[Ryu *et al.* 2020] Moonkyung Ryu, Yinlam Chow, Ross Anderson, Christian Tjandraatmadja and Craig Boutilier. *CAQL: Continuous Action Q-Learning*. In 8th International Conference on Learning Representations (ICLR), 2020.

[Schwartz & Sharir 1983] Jacob T. Schwartz and Micha Sharir. *On the "piano movers'" problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers*. Communications on pure and applied mathematics, vol. 36, no. 3, pages 345–398, 1983.

[Silver *et al.* 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra and Martin A. Riedmiller. *Deterministic Policy Gradient Algorithms*. In Proceedings of the 31th International Conference on Machine Learning, ICML, volume 32, pages 387–395. JMLR, 2014.

[Sleiman *et al.* 2023] Jean-Pierre Sleiman, Farbod Farshidian and Marco Hutter. *Versatile multicontact planning and control for legged loco-manipulation*. Science Robotics, vol. 8, no. 81, 2023.

[Sreenivasa *et al.* 2009] Manish N. Sreenivasa, Philippe Souères, Jean-Paul Laumond and Alain Berthoz. *Steering a humanoid robot by its head*. In 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pages 4451–4456. IEEE, 2009.

[Stulp 2012] Freek Stulp. *Adaptive exploration for continual reinforcement learning*. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, pages 1631–1636. IEEE, 2012.

[Todorov *et al.* 2012] 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, IROS, pages 5026–5033. IEEE, 2012.

[Towers *et al.* 2023] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen and Omar G. Younis. *Gymnasium*. https://zenodo.org/record/8127025, 2023.

[Vaillant & Glaunès 2005] Marc Vaillant and Joan Glaunès. *Surface matching via currents*. In Information Processing in Medical Imaging, pages 381–392, 2005.

[Watkins 1989] Christopher J.C.H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, United Kingdom, 1989.

[Wu *et al.* 2021] Yanqiu Wu, Xinyue Chen, Che Wang, Yiming Zhang and Keith W Ross. *Aggressive Q-learning with ensembles: Achieving both high sample efficiency and high asymptotic performance*. arXiv preprint arXiv:2111.09159, 2021.

[Xie *et al.* 2016] Chris Xie, Sachin Patil, Teodor Mihai Moldovan, Sergey Levine and Pieter Abbeel. *Model-based reinforcement learning with parametrized physical models and optimism-driven exploration*. In 2016 IEEE International Conference on Robotics and Automation, ICRA, pages 504–511. IEEE, 2016.

[Xu *et al.* 2023] Haoran Xu, Li Jiang, Jianxiong Li, Zhuoran Yang, Zhaoran Wang, Wai Kin Victor Chan and Xianyuan Zhan. *Offline RL with No OOD Actions: In-Sample Learning via Implicit Value Regularization*. In The 11th International Conference on Learning Representations (ICLR), 2023.