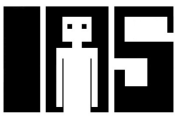
Task Space Exploration in Robot Reinforcement Learning

Arbeitsraum-Exploration im Roboter-Reinforcement-Learning
Master thesis in the field of study “Computational Engineering” by Johannes Heeg
Date of submission: September 16, 2023

1. Review: Prof. Jan Peters, Ph.D.
2. Review: Puze Liu, M.Sc.
3. Review: Dr. Davide Tateo
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT



Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Johannes Heeg, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 16. September 2023

Johannes Heeg

Abstract

Exploration is a key ingredient of reinforcement learning (RL). Most common RL algorithms for robot RL use Gaussian action noise to explore new behaviors, which does not make use of prior knowledge about the robot or the environment. This thesis studies how task space methods from classical robotics can improve exploration for robot RL. We approach the task space exploration problem from a local and a global perspective.

From the local perspective, we establish a maximum entropy formulation from which we derive an optimal sampling strategy. We define a new framework of Jacobian methods that generalizes the well-known Jacobian inverse and Jacobian transpose method, and we interpret the optimal sampling strategy with the aid of the new framework. Based on the theoretical findings, we develop task-space-aware action sampling (TAAS). Our empirical studies find that TAAS explores more efficiency and can increase the success rate of RL compared to common joint space exploration.

To improve global task space exploration directly, we couple the standard local action policy with an actively exploring global bias policy. By deriving an adapted surrogate loss function from the performance difference lemma, we integrate our approach, task space biasing (TSB), into the state-of-the-art RL algorithm proximal policy optimization (PPO). We compare TSB with pure PPO and residual reinforcement learning (RRL) on a complex reaching task with sparse rewards and find that it outperforms both baselines by a large margin. We argue that the presented analysis and developed methods enrich the possibilities of integrating prior knowledge into robot RL and, hence, have the potential of rendering more challenging robotic problems efficiently solvable in the future.

Zusammenfassung

Exploration ist ein Hauptbestandteil von Reinforcement Learning (RL). Für die Anwendung auf Roboter-Lernprobleme wird dazu üblicherweise Gauß'sches Rauschen über die Steuergröße gelegt. Dieses Verfahren nutzt keinerlei Vorwissen über den Roboter oder die ihn umgebende Umwelt. Die vorliegende Abschlussarbeit beschäftigt sich mit der Fragestellung, wie Arbeitsraummethoden der Robotik in das Roboter-Reinforcement-Learning integriert werden können, um die Exploration zu verbessern. Die Fragestellung wird sowohl aus einer lokalen als auch einer globalen Perspektive betrachtet.

Die lokale Perspektive ermöglicht eine analytische Problembeschreibung, aus welcher heraus eine optimale lokale Explorationsstrategie hergeleitet wird. Mithilfe einer neuen verallgemeinernden Definition von Jacobi-Methoden kann die hergeleitete Explorationsstrategie ebenfalls als Jacobi-Methode interpretiert werden. Auf Grundlage der theoretischen Untersuchung wird das Task Space Aware Action Sampling (TAAS) entwickelt, dessen Explorationseffizienz sich empirisch validieren lässt und im Vergleich zur gelenkweisen Exploration die Erfolgsrate von RL erhöhen kann.

Zur direkten Verbesserung der globalen Exploration wird die übliche lokale Policy durch eine aktiv explorierende globale Bias-Policy ergänzt. Die neue Methode, Task Space Biasing (TSB), kann durch die Herleitung einer angepassten Ersatz-Kostenfunktion in einen aktuellen RL-Algorithmus, Proximal Policy Optimization (PPO), integriert werden. Der Vergleich von TSB mit herkömmlichem RL und Residudal RL (RRL) anhand eines komplexen Erreichen-Problems mit spärlicher Belohnung zeigt ein deutlich verbessertes Lernverhalten. Es wird dargelegt, dass die vorliegende Analyse sowie die entwickelten Methoden die Möglichkeiten der Integration von Vorwissen in das Roboter-RL erweitern und somit das Potenzial besitzen, in Zukunft das effiziente Lösen von noch anspruchsvollerenden Robotikproblemen ermöglichen.

Contents

1 Introduction **1**

2 Related Work **3**
2.1 Exploration in RL 3
2.2 Task Space Learning 6
2.3 Residual Reinforcement Learning 7

3 Foundations **10**
3.1 Reinforcement Learning (RL) 10
3.2 Entropy 16
3.3 Task Space Control 17
3.4 Transformation of Random Variables 23
3.5 Extended Singular Value Decomposition (eSVD) 26

4 Local Task Space Exploration **29**
4.1 Local Maximum Entropy Task Space Exploration 30
4.2 Common Jacobian Framework 33
4.3 Interpretation and Comparison of Maximum Entropy Solution 35
4.4 Null Space Sampling 37
4.5 Task Space Aware Action Sampling (TAAS) 38
4.6 Empirical Evaluation 40
4.7 Discussion 49

5 Global Task Space Exploration **51**
5.1 Intuition 51
5.2 Task Space Biasing (TSB) 53
5.3 Empirical Evaluation 57
5.4 Discussion 61

6 Conclusion **63**

A Appendix **68**
A.1 Derivation of Jacobian Pseudo-Inverse Method 68
A.2 Extended Evaluation of Learning with TAAS 69
A.3 Experiment Specifications 72

1. Introduction

The principles of exploration and optimization are at the core of reinforcement learning (RL). An agent needs to experience that a certain action leads to higher rewards before this action can become an established part of the policy through optimization. This interplay of exploration and optimization makes good exploration a key mechanism of successful RL.

In robot RL, the common exploration strategy is based on Gaussian action noise [1, 2]. A stochastic policy is initialized isotropically, and at each time step, a random action is sampled from the policy and executed in the environment. This simple sampling strategy does neither harvest available knowledge about the task structure nor does it provide global exploration and can fail easily in sparse reward settings [3].

The limitations of Gaussian exploration have been addressed in various works. Regarding robot RL, the most prominent advanced exploration paradigm is correlated exploration [4, 5, 6, 7]. Instead of sampling an action independently at every time step, temporal or special correlation is used to further global exploration. While being conceptually promising, most methods cannot improve the learning process on a general level or are not applicable to on-policy learning. Moreover, they do not make use of prior task knowledge.

Task knowledge is available in every robot learning scenario. Long before the rise of deep RL and its application to robot learning, robots have been employed for many industrial applications, and, continuing to the current day, most robot controllers in use are based on first principles. One prevalent technique is task space or operational space control [8]. Based on the kinematics of the robot, it allows for planning trajectories in Euclidean space and transforming them into joint space for motor control.

Only few works have applied insights from task space control to RL. The most popular approach is using RL to improve on an existing hand-engineered base controller [9, 10]. This approach is known as residual reinforcement learning (RRL) because the RL agent learns the residual action, which is added to the action coming from the base controller. RRL improves learning convergence significantly. It is hypothesized that one reason for

its success is that the base controller acts as an exploration guide [10]. However, RRL is restricted to scenarios in which an engineered controller approximately solves the task and restricts exploration around the nominal trajectory given by the base controller.

The aim of this thesis is to investigate the use of task space methods for exploration in robot RL. We explore the potential benefits of the application of task space methods for exploration both analytically and experimentally. Based on our analysis, we develop methods that incorporate prior knowledge of the robot and its environment into the learning process and evaluate them on sparse reward RL problems.

This thesis approaches the task space exploration problem from two distinct directions. The first approach sheds light on the local task space exploration problem for which we establish a maximum entropy formulation that enables the use of analytical tools. With this formulation, we can derive an optimal exploration strategy. We define a common Jacobian framework unifying the well-known Jacobian inverse and Jacobian transpose methods. Using this framework, we link the newly derived optimal exploration strategy to task space control methods. Based on our theoretical findings, we develop task-space-aware action sampling (TAAS), an adaptation to Gaussian action sampling that can be easily integrated into existing RL algorithms and demonstrate its exploration efficiency.

The second approach focuses on global task space exploration. Based on concepts from RRL, we present task space biasing (TSB), that uses a learnable additive task space controller to drive exploration globally. Thereby, we enable fine-grained incorporation of prior knowledge into the learning process. We derive abyore general surrogate loss function from the performance difference lemma that lets us integrate TSB into the state-of-the-art RL algorithm proximal policy optimization (PPO). On a sparse reward complex reaching task, we show how TSB improves performance compared with plain PPO and RRL by a large margin.

This thesis is structured as follows. Chapter 2 summarizes related work around exploration in RL, task space learning, and RRL. Chapter 3 introduces concepts and tools that are important to understand the main parts. The fundamental research of this thesis is outlined in chapter 4 (Local Task Space Exploration) and chapter 5 (Global Task Space Exploration).

2. Related Work

This chapter introduces related work from RL and robotics. The first part gives an overview of prior work on exploration in RL and summarizes core ideas. The second part sheds light on previous works combining RL and task space control, and in the third part, we present related work on residual reinforcement learning (RRL).

2.1. Exploration in RL

Exploration is a core concept in RL. It enables the agent to learn about the environment and try new actions to possibly improve its performance. Unsurprisingly, exploration has been studied frequently in the last decades, and many methods have arisen. Even though, the core exploration mechanisms stay largely unchanged and depend on the type of action space. If the action space is discrete, actions are sampled either from a learned soft-max distribution or according to an ε-greedy method [11]. The soft-max distribution assigns each action a probability directly. On the other hand, ε-greedy methods perform the assumable best action with probability $1 - \varepsilon$ or randomly choose from the other actions with probability ε. If instead, the action space is continuous, like in the robot learning scenario that we are interested in, exploration is based on perturbing a mean action by Gaussian noise.

Historically, the most studied example of exploration is the multi-armed bandit problem. In its context, the upper confidence bound (UCB) method was developed [12]. It introduces two ideas still influential for modern exploration methods used in deep RL. First, it treats value estimation probabilistically and quantifies the amount of trust in the estimate, and second, it gives a bonus to less explored but potentially rewarding actions. Both techniques will appear throughout the following presentation of exploration paradigms shown in Figure 2.1. To learn more about recent methods beyond this brief overview, the reader is referred to the survey by Yang et al. [13].

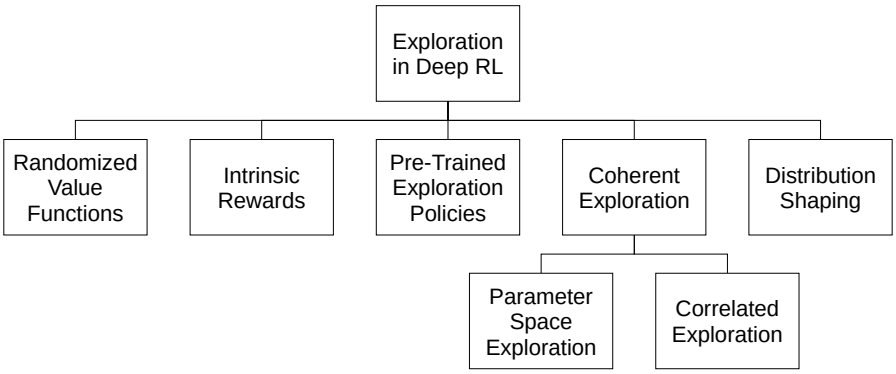


Figure 2.1.: Overview of exploration paradigms in deep RL.

A modern relative of probabilistic value estimation in the UCB method is **randomized value functions**. Instead of a single estimate of the state-action value function (Q-function), a distribution over Q-functions is updated as training progresses. In principle, this would allow for an optimistic approach similar to the UCB method. Recent works suggest Thompson Sampling as an alternative approach [14, 15, 16]. In each episode, a sample Q-function is drawn from the learned distribution, and actions are selected greedily with respect to the sample. While offering desirable theoretical properties for long-term exploration [17], its use is restricted to the Q-learning setting which only allows for discrete action spaces.

Another UCB-influenced paradigm that became popular in deep RL is **intrinsic rewards**. Similar to the UCB algorithm, a bonus for exploring actions or states that are less explored is added to the objective. This idea is implemented by changing the reward function in the following manner. The original reward function is interpreted as an extrinsic reward provided by the environment. Now, an additional term, the intrinsic reward provided by the agent itself, is added. Intrinsic rewards can be constructed in various ways. Pathak et al. [18] train a dynamics model of the environment on observations and define an interesting state as one that has unknown dynamics, i.e., the predictive error of the dynamics model is high. Alternatively, an intrinsic reward can be based on state visitation counts [19, 20]. Intuitively, this strategy rewards actions leading to states that are less often visited. Although being easy to integrate into existing algorithms, intrinsic rewards change the learning objective and do not affect the initial exploration characteristics.

While intrinsic rewards include the problem of learning to explore into the process of learning the task itself, **exploration pre-training** splits the two objectives clearly. First,

the policy is pre-trained to explore the environment in the absence of a task to solve. After that, the pre-trained policy is optimized to solve a specific task through regular RL. The pre-training objective is finding a policy that explores the state space as uniformly as possible, which is formulated as maximizing the entropy of the state space distribution [21]. The resulting pre-training problem is not an RL problem [22] and hence, needs different optimization techniques. Hazan et al. [21] and Lee et al.[22] propose solutions to this problem by iterating between density estimation and policy optimization and show that pre-trained policies can learn new tasks faster. However, Lee et al.[22] note that the pre-training problem is considerably more difficult than exploration for a specific task.

Instead of sampling actions at every time step independently, **coherent exploration** tries to act consistently over many time steps to reach more distant regions in the state space and explore more efficiently. This goal can be either achieved by perturbing the policy parameters or by correlating the action noise over time.

Parameter space exploration samples new policy parameters every episode leading to coherent trajectories during one trial. The strategy is known to work well for a small amount of policy parameters. In contrast, deep RL is based on neural network architectures with thousands or more parameters. Nonetheless, Plappert et al. [23] show that meaningful perturbations by adding Gaussian noise to the policy parameters at every episode are also possible with neural networks, and Fortunato et al. [24] learn a distribution over the parameters of the last policy layer. Both works demonstrate that deep RL can be combined with parameter space exploration but cannot improve on action-based exploration in continuous settings.

Lillicrap et al. [4] already suggested the use of an Ornstein-Uhlenbeck process as a substitute for white noise (Gaussian noise) to improve exploration in off-policy RL for continuous control problems. From there on, researchers investigated **correlated exploration**, which samples actions at every time step but introduces temporal correlation between them. Korenkevych et al. [5] develop a framework for using autoregressive processes as action noise and define hyperparameters to vary the degree of smoothness in the control input. They also discuss that action correlation requires the extension of the state space by an action history to ensure the Markovian property and allow for its application in on-policy RL methods. Instead of explicitly including a temporally correlated stochastic process, state-dependent exploration [25] samples a state-dependent noise perturbation function for each episode. Since this function is continuous with respect to changes in the state, the noise is also temporally correlated. Raffin et al. [6] bring this concept to deep RL and investigate the impact of the sampling interval on the learning performance. They show that episodic sampling makes learning inefficient and propose to use shorter

sampling intervals. This trade-off between long and short-term correlation has been further analyzed by Eberhard et al. [7] for using stochastic processes. In agreement with previous works, they propose pink noise, a noise pattern between uncorrelated white noise (Gaussian) and highly correlated red noise (similar to Ornstein-Uhlenbeck), as the new default exploration noise.

Only few works consider **distribution shaping** to use action distributions different from Gaussians with diagonal covariances. The popular soft actor-critic (SAC) algorithm [2] changes the distribution shape slightly by squashing the actions between the action limits using the hyperbolic tangent. In addition, Mazoure et al. [26] add a normalizing flow to the policy architecture to further extend the class of action distributions. A similar distribution-shaping approach is presented in this thesis. Different from their work, we consider shaping functions that are known in advance and relate to the kinematic structure of the robot.

This overview introduced a broad range of exploration strategies for deep RL. Notably, assuming knowledge about the kinematic structure of robots and exploiting it to guide exploration has not been studied so far.

2.2. Task Space Learning

Task space or operational space methods were developed to control robots more intuitively [8]. They allow for planning trajectories in world coordinates and transform task space set points to joint control commands that can be executed on the robot.

In recent works on robot RL, task space methods are often applied without further discussion. E.g., Nair et al. [27] and Zuo et al. [28] learn to control the position of the end-effector in Cartesian space together with the distance between the gripper fingers of a 7 degrees of freedom (DoF) manipulator. They use this setup to solve pick-and-place tasks. The remaining DoF are controlled by means of task space control methods and ensure that the gripper is always oriented in the same way, and the robot arm stays in an upright posture. These constraints assume regularity in the arrangement of the objects to manipulate.

Bellegarda and Byl [29] explicitly focus on the impact of task space control on the training speed of robot RL. They argue that the inverse kinematics have to be learned when applying RL to joint space control, and learning in task space would circumvent this necessity. Their experiments demonstrate faster convergence when learning the task space

control commands compared to joint space control commands. However, they also point out the dependence on an accurate inverse kinematics model and observe a significant performance drop in the case of model mismatch.

To mitigate the problems related to model mismatch, Kaspar et al. [30] perform a system identification step before learning a peg-in-hole task in simulation. They apply task space control to address various goals. First, they argue that a task space control policy generalizes better from simulation to the real robot and, hence, decreases the reality gap. Second, they integrate both joint and task space limits into the controller to protect the robot and its environment. And third, they make the policy easier to learn by reducing the number of controllable DoF.

A different application of task space methods can be found in the paper by Al-Hafez and Steil [31]. Instead of learning the task space control commands, they focus on redundancy resolution. They build upon the idea that if a goal is defined solely in task space, like reaching tasks, kinematically redundant actions lead to the same reward. Now, they integrate the concept of null space control from robotics where the redundant DoF can be used to target secondary objectives like keeping an upright posture or obstacle avoidance. The integration is realized by simply adding the analytically determined null space action as an action bias to the output of the RL agent. We will also make use of null space actions for integrating task space methods into RL and incorporate an action bias.

In conclusion, task space methods have primarily been used in RL to reduce the number of DoF for leaerning, which restricts the policy search space and seems to improve training speed. Task space methods also appear to improve generalization capabilities and disentangle primary and secondary objectives. Yet, an analysis of the effect of task space methods on exploration is missing.

2.3. Residual Reinforcement Learning

Residual reinforcement learning (RRL) was concurrently introduced by Johannink et al. [9] and Silver et al. [10] in 2019 for robot RL. The idea is quite simple and summarized in Figure 2.2. Instead of learning a full controller from scratch, RL is used to train a residual controller that optimizes the output of a given base controller. This technique can improve the learning speed of RL and the optimality of base controllers.

Johannink et al. [9] argue that many robotic problems can be approximately solved with conventional feedback control methods. However, these methods are very hard to tune or

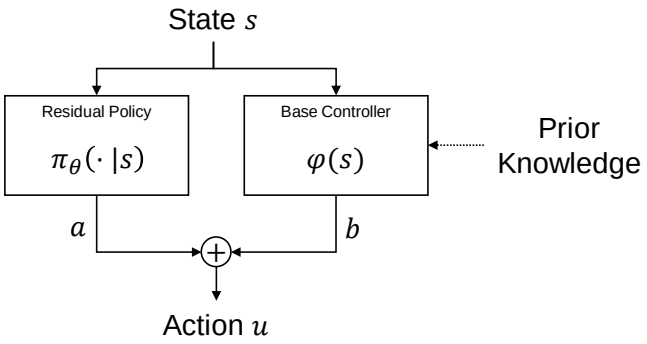


Figure 2.2.: In residual reinforcement learning (RRL), an action u is the superposition of an action a sampled from the residual policy π_θ and a base action b from the predefined base controller φ.

unable to provide optimal control when dealing with contacts and friction. On the other hand, RL provides a means of model-free controller optimization through environment interactions. The authors propose superposing both approaches leading to RRL where the RL agent needs only to identify the action deviation from the base controller. According to the authors, this partition corresponds to a typical reward structure in robotics tasks, where one part represents geometric relationships like distances to a goal while another part describes further general characteristics. The former is said to be easily optimizable a priori by employing a hand-engineered controller while the latter is optimized with RL. The authors successfully apply RRL on a real robot learning a block-assembly task.

Silver et al. [10] evaluate RRL on several manipulation tasks in simulation proving its usefulness in scenarios with partial observability, sensor noise, model misspecification, and controller miscalibration. Besides improving an existent base controller, the authors bring up the perspective of the base controller being a guide for exploration, which directly connects to our work. They describe three potential benefits of RRL compared to regular model-free RL. First, a good base controller offers a performance boost at the beginning of training. Second, the base controller improves exploration when it is unlikely to observe high rewards by chance. And third, the residual RL problem may be easier than the original one.

One adaptation to the framework, RRL from demonstrations by Alakuijala et al. [32], exchanges the hand-tuned controller with a neural base policy trained using behavioral cloning from visual inputs. Additionally, they include learned features from the base policy into the input of the residual policy. Another recent adaptation by Staessens et al. [33]

introduces a scaling factor into the superposition to ensure stability of the composite controller.

In all these works, the base policy is deterministic and fixed, ensuring the residual control problem is easily solvable by common RL algorithms. However, this limits the use of RRL to well-understood problems. Our work builds on ideas from RRL but focuses on guiding exploration rather than optimizing a base policy. Thereby, we try to make the base policy itself learnable, extending its usefulness to scenarios where only partial task knowledge is available a priori.

3. Foundations

The study of task space methods for robot RL requires knowledge of linear algebra, information theory, RL, and robotics. All of the techniques presented in this chapter will play an important role throughout the following analysis. Besides rephrasing well-known concepts, we introduce extended singular value decomposition (eSVD) which splits the scaling and projection operations into separate matrices.

3.1. Reinforcement Learning (RL)

This section introduces the framework of model-free RL. We introduce a specific type of model-free RL algorithms, namely policy gradient methods, and present a state-of-the-art policy gradient method, proximal policy optimization [1].

3.1.1. Introduction to Reinforcement Learning

Reinforcement learning (RL) formalizes the concept of learning by trial and error. As illustrated in Figure 3.1, to learn a new task, the agent interacts with the environment. Its policy maps the observed state s_t to an action distribution $\pi(a|s_t)$ from which an action a_t is sampled and applied in the environment responding with a reward r_t. The goal of an RL algorithm is to improve the agent's policy to maximize the expected cumulative reward during one trajectory τ of length T

$$\mathbb{E}[R(\tau)] = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

where the discount factor $\gamma \in (0, 1]$ controls how much to focus on immediate returns compared to future returns. In summary, RL is a machine learning paradigm in which

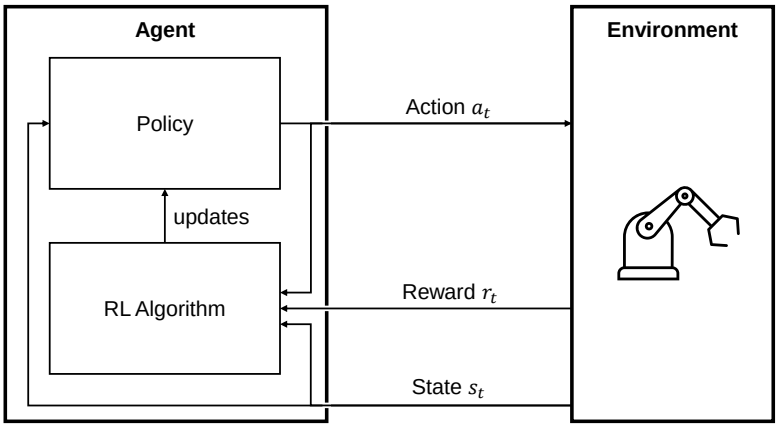


Figure 3.1.: In reinforcement learning (RL), the agent learns by interacting with the environment. The agent observes states, and its policy acts accordingly. The environment responses with a state transition and a reward signal. The RL algorithm updates the policy to maximize the cumulative reward.

an agent's policy is optimized to maximize cumulative rewards based on environment interactions [11].

The ideas of tasks and environments are formalized through Markov decision processs (MDPs) [11]. An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, r, P_0, \gamma)$ which represents all core objects that describe an RL problem. $\mathcal{S}$ is the state space, and an element $s \in \mathcal{S}$ is assumed to completely describe the current state of the environment and is fully observable by the agent. The action space $\mathcal{A}$ defines how an agent can interact with the environment. For robot learning, $\mathcal{S}$ could be the joint space and an action $a \in \mathcal{A}$ could be a joint velocity command.

The transition distribution P and in continuous scenarios its probability density function $p(s_{t+1}|s_t, a_t)$ model the dynamics of the environment and must satisfy the Markovian property. It states that the distribution over the next state s_{t+1} is independent of the history $s_0, s_1, ..., s_{t-1}$ given the current state s_t. The Markovian property implies that by observing the current state s_t, the agent has access to all knowledge necessary to act optimally. The task objective is encoded by the reward function $r(s_t, a_t, s_{t+1})$ and when a new trajectory τ starts, the initial state is sampled from the initial state distribution P_0 (or p_0). γ is the previously introduced discount factor.

3.1.2. Policy Gradient

Policy gradient methods enable the use of advanced gradient optimization schemes for RL by estimating the gradient of the RL objective function

$$\mathcal{J}(\theta) := \mathbb{E}_\tau[R(\tau)] = \int p_\theta(\tau) R(\tau) d\tau$$

with respect to the policy parameter θ. The likelihood of a trajectory $p_\theta(\tau)$ can be expressed as a function of the transition probabilities $p(s_{t+1}|s_t, a_t)$, the parameterized policy $\pi_\theta(a_t|s_t)$, and the initial state distribution $p_0(s_0)$:

$$p_\theta(\tau) = p_0(s_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t).$$

Note that when applying the logarithm to this function, the product turns into a sum, and it is possible to separate the known policy terms $\pi_\theta(a_t|s_t)$ from the unknown dynamics terms $p(s_{t+1}|s_t, a_t)$. With this setup, we derive the simplest form of the policy gradient [34] as follows

$$\begin{aligned}
\nabla_\theta \mathcal{J}(\theta) &= \nabla_\theta \int p_\theta(\tau) R(\tau) d\tau \\
&= \int \nabla_\theta p_\theta(\tau) R(\tau) d\tau \\
&= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) R(\tau) d\tau \\
&= \mathbb{E}_\tau[\nabla_\theta \log p_\theta(\tau) R(\tau)] \\
&= \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \right].
\end{aligned}$$

Finally, we can estimate the expectation over trajectories with Monte-Carlo sampling, providing an unbiased estimate of the policy gradient.

The policy gradient is a powerful and direct approach to solve RL problems. Its derivation highlights the possibility of optimizing an objective function that depends on the dynamics of the environment without knowing them but instead, by sampling transitions through interactions. However, the accuracy of the gradient is limited due to its high variance estimator, and hence, many interactions with the environment are needed. This problem motivates extensions to the simple policy gradient estimator.

Instead of considering the cumulative reward $R(\tau)$ of one complete trajectory, the variance is reduced by only taking into account the future reward. This strategy is possible because an action a_t only influences future time steps $t+1, ..., T$. Another approach uses the concept of variance-reducing baselines which led to the actor-critic framework in which an advantage function $A(s_t, a_t)$, also called the critic, estimates whether the performed action a_t is better or worse than expected.

3.1.3. Performance Difference Lemma and Surrogate Loss

The performance difference lemma was stated and proved by Kakade and Langford [35]. It is the mathematical base for modern policy gradient methods like proximal policy optimization (PPO) which we introduce in the next section.

The lemma states that if we have two policies π and q with performances $\mathcal{J}(\pi)$ and $\mathcal{J}(q)$ respectively, the performance difference can be expressed as

$$\mathcal{J}(\pi) - \mathcal{J}(q) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T-1} \gamma^t A^q(s_t, a_t) \right]$$

where $A^q(s_t, a_t)$ is the advantage function of the policy q and trajectories are sampled from the trajectory distribution $p_\pi(\tau)$ according to the policy π. Therefore, we can also rewrite the policy gradient as

$$\nabla_\theta \mathcal{J}(\theta) = \nabla \underbrace{\mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t A^q(s_t, a_t) \right]}_{\mathcal{E}}.$$

We further restructure the expectation $\mathcal{E}$ on the right-hand side to eliminate the sum over time steps:

$$\begin{aligned}
\mathcal{E} &= \int p_\pi(\tau) \sum_{t=0}^{T-1} \gamma^t A^q(s_t, a_t) d\tau \\
&= \int \sum_{t=0}^{T-1} \gamma^t p_\pi(\tau) A^q(s_t, a_t) d\tau \\
&= \int \sum_{t=0}^{T-1} \gamma^t \left(p_0(s_0) \prod_{i=0}^{T-1} p(s_{i+1}|s_i, a_i) \pi_\theta(a_i|s_i) \right) A^q(s_t, a_t) d\tau \\
&= \int \sum_{t=0}^{T-1} \gamma^t p_0(s_0) \left(\prod_{i=0}^{t-1} p(s_{i+1}|s_i, a_i) \pi_\theta(a_i|s_i) \right) \pi_\theta(a_t|s_t) A^q(s_t, a_t) d\tau.
\end{aligned}$$

Note that in the last step, we used the independence of future time steps, and $d\tau$ is a shorthand for the product of all ds_t and da_t. We now introduce the discounted state distribution of the policy π

$$d^{\pi_\theta}(s) := \frac{1-\gamma}{1-\gamma^T} \sum_{t=0}^{T-1} \gamma^t \int p_0(s_0) \left(\prod_{i=0}^{t-1} p(s_{i+1}|s_i, a_i) \pi_\theta(a_i|s_i) \right)_{s_t = s} d\tau_{t-1}$$

which is the γ-discounted likelihood of visiting state s under policy π at any time within the horizon T. Here, $d\tau_{t-1}$ is a shorthand for the product of all ds_t and da_t up to the time step $t-1$. Integrating d^{π_θ} into the former equation yields

$$\begin{aligned}
\mathcal{E} &= \frac{1-\gamma^T}{1-\gamma} \int d^{\pi_\theta}(s) \pi_\theta(a|s) A^q(s, a) ds da \\
&= \frac{1-\gamma^T}{1-\gamma} \mathbb{E}_{s \sim d^{\pi_\theta},\ a \sim \pi_\theta(\cdot|s)} \left[A^q(s, a) \right].
\end{aligned}$$

For estimating the gradient of the resulting expression, it remains the issue that d^{π_θ} depends on θ and is unknown. Hence, for a practical implementation, d^{π_θ} is replaced by the undiscounted state distribution u^q of the policy q leading to the surrogate loss function

$$\begin{aligned}
\mathcal{L}(\theta) &= \mathbb{E}_{s \sim u^q,\ a \sim \pi_\theta(\cdot|s)} \left[A^q(s, a) \right] \\
&= \mathbb{E}_{s \sim u^q,\ a \sim q(\cdot|s)} \left[\frac{\pi_\theta(a|s)}{q(a|s)} A^q(s, a) \right]
\end{aligned} \tag{3.1}$$

where importance sampling is applied to remove the dependence on θ from the sampling distribution. The surrogate loss and its gradient can be easily estimated by Monte Carlo sampling.

3.1.4. Proximal Policy Optimization

Algorithm 1: Proximal Policy Optimization (PPO)

Initialize policy π_θ
Initialize value function V_ϕ
for *iteration = 1 to number of iterations* **do**
  Initialize data buffer $\mathcal{D}$
  for *interaction = 1 to number of interactions* **do**
    Run π_θ and collect sample (s_t, r_t) in $\mathcal{D}$
  end
  Compute advantages A_t
  Update V_ϕ
  for *epoch = 1 to number of optimization epochs* **do**
    Compute surrogate loss over $\mathcal{D}$
    Update π_θ using the gradient of the surrogate loss
  end
end

Proximal policy optimization (PPO) is an on-policy, actor-critic RL algorithm by Schulman et al. [36]. It is a policy gradient method that is strongly influenced by trust region policy optimization [37] and uses an adaption of the surrogate loss function (3.1). Therein, the policy q from equation (3.1) is referred to as $\pi_{\theta_{\text{old}}}$ indicating that its parameters are fixed for the policy update.

To keep the update within a trust region, PPO clips the surrogate loss according to a threshold ϵ (usually $\epsilon = 0.2$). Thus, the full surrogate loss for the policy update becomes

$$\mathcal{L}(\theta) = \mathbb{E}\left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}A_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)A_t\right)\right].$$

Note that we dropped the subscript of the expectation operator to improve readability. The advantages A_t are estimated through generalized advantage estimation [38] based on a parameterized value function $V_\phi(s)$ trained on interaction samples to predict the expected

reward to come at a specific state. To reduce the greediness of the method, a common approach is to add an entropy bonus to the loss function. Algorithm 1 summarizes the method. To better understand the objective and how it relates to a KL penalty, the reader is referred to the original paper [1].

3.2. Entropy

To analyze the expected information gain of a message, Claude Shannon introduced the information-theoretic entropy of a random variable. For a random variable x which takes discrete values in $\mathcal{X}$ and follows the probability mass P, the entropy reads

$$H(P) = -\sum_{x \in \mathcal{X}} P(x) \log P(x) = \mathbb{E}[-\log P(x)].$$

Which base to choose for log can vary for different applications, but we will use the natural logarithm throughout the thesis. The entropy is lower bounded by 0 which corresponds to a random variable that always gives the same output, i.e., it is deterministic.

For continuous settings, the differential entropy is a commonly used concept to quantify the randomness of a random variable. It follows the same computational structure but is based on the density function $p(x)$ and is defined by

$$H(p) = -\int_{\mathcal{X}} \log p(x) dx = \mathbb{E}[-\log p(x)].$$

In contrast to the original entropy, the differential entropy is not lower bounded. Instead, as the random variable gets more deterministic, the differential entropy tends to $-\infty$. Note that the term entropy is used to refer to both definitions, and its concrete meaning depends on the setting (discrete or continuous).

For Gaussian random variables $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, the (differential) entropy can be expressed in closed form as

$$H(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})) = \frac{1}{2} \log \det(2\pi e \boldsymbol{\Sigma}) = \frac{n}{2} + \frac{n}{2} \log(2\pi) + \frac{1}{2} \log \det \boldsymbol{\Sigma}. \tag{3.2}$$

This equation includes constant terms scaled by the dimensionality of the random variable and a term depending on the log-determinant of the covariance matrix. The latter one captures the property that with increasing covariance, the entropy increases as well.

It is worth mentioning that we can decompose the entropy of the joint distribution $p_{xy}(x, y)$ of two independent random variables x and y as

$$\begin{aligned} H(p_{xy}) &= \mathbb{E}[-\log p_{xy}(x, y)] \\ &= \mathbb{E}[-\log(p_x(y)p_y(y))] \\ &= \mathbb{E}[-\log p_x(y)] + \mathbb{E}[-\log p_y(y)] \\ &= H(p_x) + H(p_y). \end{aligned}$$

We can exploit this fact to facilitate entropy calculations for new sampling methods.

3.3. Task Space Control

Task space control is a fundamental concept in robotics that focuses on controlling the position and orientation of a robot's end-effector in a desired task space. It enables robots to perform precise and accurate manipulation tasks, navigate complex environments, and interact with objects and humans. Unlike joint space control, task space control operates directly in the Cartesian space, allowing for more intuitive and versatile control strategies. By explicitly specifying the desired task space behavior, robots can exhibit dexterous and adaptive behavior, making task space control an essential technique for achieving sophisticated robotic applications.

Essential for task space methods is the theory of differential kinematics that relates the end-effector movement in task space to joint velocity commands. This relation is fundamentally described by the Jacobian matrix of the robot. We will introduce the Jacobian matrix and its geometric interpretation and describe two well-known Jacobian-based methods, namely, the Jacobian pseudo-inverse method and the Jacobian transpose method. To delve deeper into the concepts behind task space control, we refer the reader to the textbook by Siciliano et al. [8].

3.3.1. The Manipulator Jacobian

The knowledge about the structure of a robot is encoded in its forward kinematics

$$\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{q}), \tag{3.3}$$

specifying the map from joint configurations $\boldsymbol{q}$ in joint space to end-effector positions $\boldsymbol{x}$ in task space. When moving the robot, we are interested in the relation between joint and task space velocities. Since velocity is the time derivative of position, we find the task space velocity by deriving equation (3.3) and obtain the differential kinematics equation

$$\dot{\boldsymbol{x}} = \underbrace{\frac{\partial \boldsymbol{f}(\boldsymbol{q})}{\partial \boldsymbol{q}}}_{\boldsymbol{J}(\boldsymbol{q})} \dot{\boldsymbol{q}}. \tag{3.4}$$

As indicated, the derivative of the kinematics with respect to the joint angles $\boldsymbol{q}$ is called the (analytical) Jacobian $\boldsymbol{J}(\boldsymbol{q})$.

Figure 3.2 depicts a simple model of a 2 degrees of freedom (DoF) manipulator. The robot has two rotational joints colored blue and red, respectively. If the blue joint at the robot

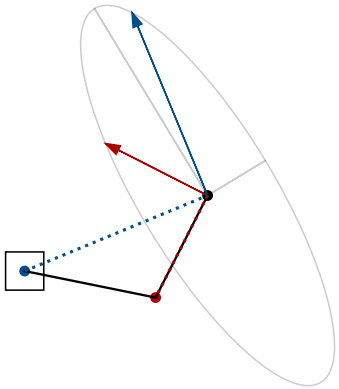


Figure 3.2.: The Jacobian matrix of a 2 DoF manipulator. The blue and red arrow illustrate how the blue and red joint can move the end-effector locally. They are the columns of the Jacobian matrix. The manipulability ellipsoid and its principle axes are depicted in grey.

base moves, the whole robot arm rotates around the robot base. The velocity and the direction are indicated by the blue arrow at the end-effector. Analogously, the red arrow symbolizes how the end-effector moves if the red joint turns. The dotted lines which are perpendicular to the arrows, show the leverage of the joints. Intuitively, with increased distance between the end-effector and the joint, the influence of a joint increases, and the final impact of a joint, illustrated by its color-coded arrow, increases as well.

The arrows in Figure 3.2 visualize exactly the columns of the Jacobian at the depicted configuration which provides us with a graphical interpretation of the Jacobian. The gray ellipse in the figure represents the manipulability ellipsoid. It shows where the end-effector can move if $\dot{\boldsymbol{q}}$ has unitary norm, i.e.,

$$\|\dot{\boldsymbol{q}}\| = 1 \Leftrightarrow \dot{\boldsymbol{q}}^\intercal \dot{\boldsymbol{q}} = 1.$$

The arrows indicating the columns of the Jacobian always lie on the ellipsoid. Since in our example, the columns are not orthogonal to each other, the arrows do not align with the principle axes of the ellipse shown in gray and the fact that one principle axis is shorter than the other means that movements along different task space directions might require different action effort $\dot{\boldsymbol{q}}^\intercal \dot{\boldsymbol{q}}$.

With the geometric meaning of the Jacobian in mind, we can interpret equation (3.4) as a linear combination of the velocity vectors of each joint

$$\dot{\boldsymbol{x}} = \sum_{i=0}^{n} \boldsymbol{J}_i(\boldsymbol{q}) \dot{\boldsymbol{q}}_i,$$

where $\boldsymbol{J}_i(\boldsymbol{q})$ is the i-th column of the Jacobian, $\dot{\boldsymbol{q}}_i$ is the joint velocity of the i-th joint, and n is the number of joints. Figure 3.3 visualizes this idea and highlights how each joint contributes to the resulting task space motion.

3.3.2. Differential Inverse Kinematics

If we are given a target trajectory in the task space coordinates of the end-effector, we need to find joint space commands that result in the desired motion. For simple structures, we can invert the kinematic equation (3.3), but in general, it is not possible due to non-linearities and redundancies [8]. Differential inverse kinematics avoids this limitation by exploiting the linear relationship between task space and joint velocities as seen in equation (3.4).

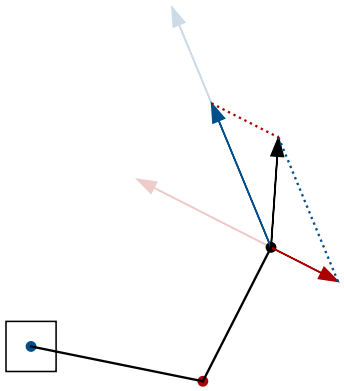


Figure 3.3.: Kinematics of a 2 DoF manipulator. The blue joint rotates with 0.6 rad/s and the red one with -0.5 rad/s. The scaled column vectors of the Jacobian superpose and result in an end-effector velocity illustrated with the black arrow.

The most straightforward approach is to impose a task space velocity $\boldsymbol{v}$ and solve for the individual joint velocities. For an equal number of task space and joint dimensions and a full rank Jacobian, we obtain the solution by inverting equation (3.4) giving

$$\dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})^{-1}\boldsymbol{v}.$$

We can interpret the equation in the same manner as before. Regarding Figure 3.3, $\dot{\boldsymbol{q}}$ is the vector of coefficients of the linear combination of the light red and light blue vector that results in the desired velocity vector $\boldsymbol{v}$ visualized by the black arrow.

For redundant robots, $\boldsymbol{J}(\boldsymbol{q})$ is not square and, hence, not invertible, which means that there is an infinite number of solutions $\dot{\boldsymbol{q}}$ that lead to the same task space velocity $\boldsymbol{v}$ and solve equation (3.4). To find a unique solution, a common approach is solving for the least squares solution, i.e., determining $\dot{\boldsymbol{q}}$ that minimizes $\dot{\boldsymbol{q}}^\intercal\dot{\boldsymbol{q}}$ while providing the desired $\boldsymbol{v}$.

This formulation yields the convex optimization problem

$$\min \dot{\boldsymbol{q}}^\intercal\dot{\boldsymbol{q}}, \quad \text{s.t. } \boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}},$$

which has the closed form solution

$$\begin{aligned}\dot{\boldsymbol{q}} &= \boldsymbol{J}(\boldsymbol{q})^\intercal \left(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^\intercal\right)^{-1} \boldsymbol{v} \\ &= \boldsymbol{J}(\boldsymbol{q})^\dagger \boldsymbol{v},\end{aligned}$$

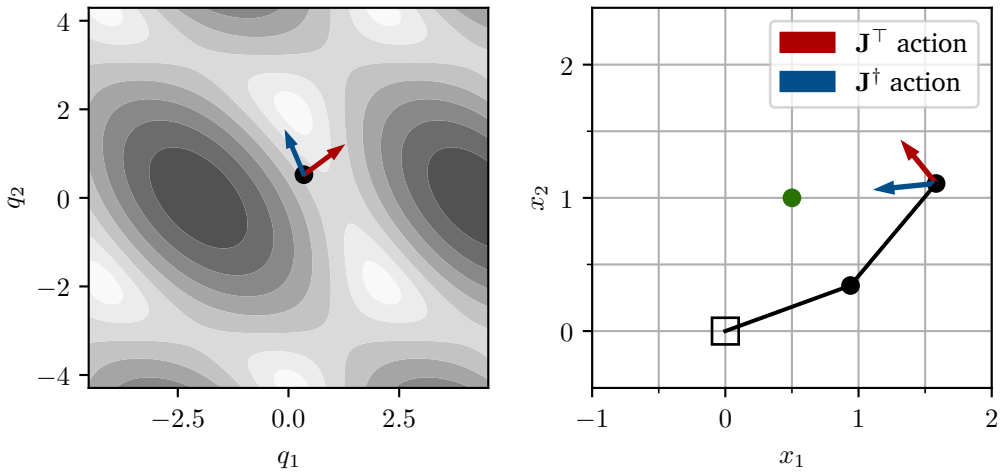


Figure 3.4.: Comparison of the Jacobian pseudo-inverse and the Jacobian transpose method in joint space (left) and task space (right). The Jacobian pseudo-inverse action in blue moves the end-effector along the direct task space path to the goal in green. The Jacobian transpose action corresponds to a gradient step minimizing the task space error in joint space.

where $\boldsymbol{J}(\boldsymbol{q})^\dagger$ denotes the pseudo-inverse. We present the full derivation of the **Jacobian pseudo-inverse** method in appendix A.1.

A different method can be derived from a goal-reaching perspective. Instead of enforcing a certain task space velocity, the **Jacobian transpose** method focuses on minimizing the task space error, i.e., the Euclidean distance between the current task space position $\boldsymbol{f}(\boldsymbol{q})$ and a desired task space position $\boldsymbol{x}_d$.

This approach gives rise to the optimization problem

$$\min \frac{1}{2}\|\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{q})\|^2 \tag{3.5}$$

and $\dot{\boldsymbol{q}}$ is the gradient descent step

$$\begin{aligned}\dot{\boldsymbol{q}} &= -\alpha \frac{\partial}{\partial \boldsymbol{q}} \left(\frac{1}{2}\|\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{q})\|^2 \right) \\ &= \alpha \boldsymbol{J}(\boldsymbol{q})^\intercal (\boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{q})) \\ &= \alpha \boldsymbol{J}(\boldsymbol{q})^\intercal \boldsymbol{e}\end{aligned}$$

with step size α and task space error $\boldsymbol{e} = \boldsymbol{x}_d - \boldsymbol{f}(\boldsymbol{q})$. Hence, $\dot{\boldsymbol{q}}$ minimizes the task space error as fast as possible. If we further introduce the vector $\boldsymbol{v} = \alpha \boldsymbol{e}$ and substitute, we obtain $\dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})^\intercal \boldsymbol{v}$, which highlights the similar structure of the Jacobian transpose and the Jacobian pseudo-inverse method.

Figure 3.4 compares the two methods and shows how the Jacobian pseudo-inverse method finds an action that moves the end-effector on a direct path in task space to the goal. The left plot depicts the contour lines of the task space error in joint space. This loss landscape illustrates the objective (3.5). Its gradient is proportional to the action in red proposed by the Jacobian transpose method.

3.3.3. Null Space Control

Differential inverse kinematics methods find one solution to the differential kinematics equation (3.3). However, as previously discussed, this solution is not unique for redundant robots. Mathematically, this phenomenon is described by the null space of $\boldsymbol{J}(\boldsymbol{q})$, which is the set of $\dot{\boldsymbol{q}}$ such that every element satisfies

$$\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = \boldsymbol{0} \tag{3.6}$$

where $\boldsymbol{0}$ is the zero vector. Figure 3.5 illustrates null space actions, i.e., solutions $\dot{\boldsymbol{q}}$ to equation (3.6). We observe that the end-effector of the robot remains in the same position

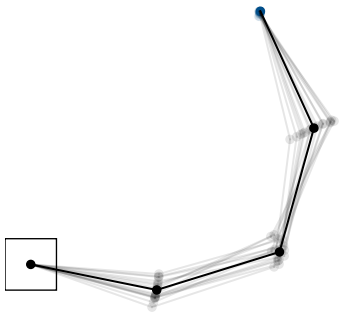


Figure 3.5.: Null space actions of a 4 DoF manipulator. While the joint configuration changes, the end-effector remains at the same task space position.

while the links and joints move. This variability can be exploited to perform secondary tasks like keeping an upright posture or obstacle avoidance.

Null space projection [39] offers a simple approach for performing secondary tasks without interfering with the task space objective. It takes advantage of

$$\boldsymbol{J}(\boldsymbol{q})\left(\boldsymbol{I}-\boldsymbol{J}(\boldsymbol{q})^{\dagger}\boldsymbol{J}(\boldsymbol{q})\right)\dot{\boldsymbol{q}}_0=\boldsymbol{0}.$$

I.e., we can find $\dot{\boldsymbol{q}}_0$ to perform a secondary task and then, project it onto the null space using $\left(\boldsymbol{I}-\boldsymbol{J}(\boldsymbol{q})^{\dagger}\boldsymbol{J}(\boldsymbol{q})\right)\dot{\boldsymbol{q}}_0$. With this setup, the final joint space velocity command reads

$$\dot{\boldsymbol{q}}=\boldsymbol{J}(\boldsymbol{q})^{\dagger}\boldsymbol{v}+\left(\boldsymbol{I}-\boldsymbol{J}(\boldsymbol{q})^{\dagger}\boldsymbol{J}(\boldsymbol{q})\right)\dot{\boldsymbol{q}}_0$$

where $\boldsymbol{J}(\boldsymbol{q})^{\dagger}\boldsymbol{v}$ could be substituted by $\boldsymbol{J}(\boldsymbol{q})^{\intercal}\boldsymbol{v}$.

3.4. Transformation of Random Variables

To analyze how the task space distribution changes with different action distributions, we must understand the transformation characteristics of random variables. These characteristics are captured by the change of variable formula that we will introduce in this section. After that, we will focus on an important case: affine transformations of Gaussian distributed random variables across different dimensions.

3.4.1. Change of Variable Formula

We consider two random variables $\boldsymbol{x},\boldsymbol{y}\in\mathbb{R}^n$ which are connected by the differentiable bijective transformation

$$\boldsymbol{y}=\boldsymbol{f}(\boldsymbol{x}),\quad \boldsymbol{x}=\boldsymbol{f}^{-1}(\boldsymbol{y}).$$

Assuming $\boldsymbol{x}$ is distributed according to a known density function $p_x(\boldsymbol{x})$ (notation: $\boldsymbol{x}\sim p_x$), the distribution p_y over $\boldsymbol{y}$ can be written as

$$p_y(\boldsymbol{y})=p_x\left(\boldsymbol{f}^{-1}(\boldsymbol{y})\right)\left|\det\left(\frac{\partial\boldsymbol{f}^{-1}(\boldsymbol{y})}{\partial\boldsymbol{y}}\right)\right|$$

where the last term is the determinant of the Jacobian of the inverse transformation. If we know the corresponding pair $(\boldsymbol{x}, \boldsymbol{y})$, the distribution can also be expressed as

$$p_y(\boldsymbol{y}) = \underbrace{p_x\left(\boldsymbol{x}\right)}_{\text{association}} \underbrace{\left|\det\left(\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right)\right|^{-1}}_{\text{morphing}}$$

which can be helpful in cases where the Jacobian of the forward path is already known.

As indicated, the change of variable formula has a distinct structure. The first part associates the unknown density with the known density at the point of interest. Intuitively, if the corresponding likelihood of $\boldsymbol{x}$) is low, the resulting density at $\boldsymbol{y}$ is also low. The latter part accounts for the local morphing of the space. E.g., if $\boldsymbol{f}(\boldsymbol{x})$ contracts the space, the resulting values for $\boldsymbol{y}$ are close, which should increase the density in the area of accumulation. Indeed, where $\boldsymbol{f}(\boldsymbol{x})$ is contracting, its determinant becomes closer to 0 leading to an increased morphing factor.

Even though the change of variable formula can handle complex transformations, it is not directly applicable to redundant robots. The issue arises from the redundancy: kinematics are not invertible. A whole set of joint movements can lead to identical end-effector positions. To alleviate this problem, we need to define a marginalization that associates redundant joint configurations with one exemplary solution to the inverse kinematics problem. By definition, the resulting mapping from task space to the set of exemplars is bijective. In general, this procedure is hard to perform but in the special case of Gaussian densities and affine transformations, the procedure yields a simple closed-form solution.

3.4.2. Affine Transformations of Gaussian Random Variables

A Gaussian distribution can be fully characterized by its mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, and its density function reads

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right).$$

In the following, we will use the shorthand $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to denote that $\boldsymbol{x}$ is a Gaussian distributed random variable with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ and refer to its density function by $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$.

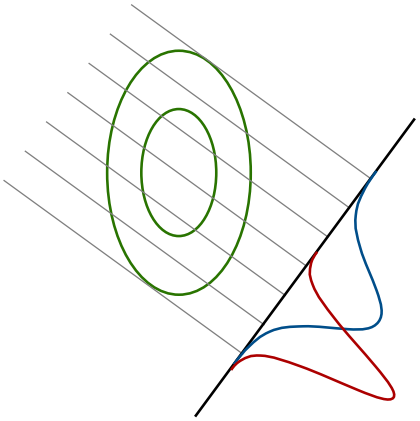


Figure 3.6.: An affine transformation of a Gaussian distribution (in green) can be decomposed into the marginalization over the null space of the transformation (blue) and a scaling and translation operation (red).

A useful property of Gaussian distributions is that affine transformations of Gaussian random variables yield Gaussian random variables themselves. More precisely, if $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ where $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{y}, \boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{A} \in \mathbb{R}^{m \times n}$,

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{b}, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\intercal). \tag{3.7}$$

To make the connection to the change of variable formula more evident, we can express the transformation rule if $m \leq n$ as

$$p_y(\boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{A}^\dagger(\boldsymbol{y} - \boldsymbol{b}); \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) |\underline{\det}\boldsymbol{A}|^{-1},$$

where $\boldsymbol{A}^\dagger = \boldsymbol{A}^\intercal(\boldsymbol{A}\boldsymbol{A}^\intercal)^{-1}$ is the (right) pseudo-inverse, and $\underline{\det}\boldsymbol{A} = \sqrt{\det(\boldsymbol{A}\boldsymbol{A}^\intercal)}$ is the pseudo-determinant. This simple expression implicitly solves the previously discussed problem of the general change of variable formula, and the marginalization becomes part of the matrix-vector multiplication.

Figure 3.6 illustrates an affine transformation of a 2-dimensional Gaussian distribution. First, we integrate over the null space dimension of the transformation and find the marginal distribution. This step is equivalent to a projection operation. Then, the distribution is translated by the bias $\boldsymbol{b}$ and scaled according to the morphing term $|\underline{\det}\boldsymbol{A}|^{-1}$.

3.5. Extended Singular Value Decomposition (eSVD)

singular value decomposition (SVD) is a well-known tool for analyzing linear maps $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ with vectors $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$ and a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. It generalizes the concept of eigenvalues and eigenvectors to non-square matrices. It allows for studying linear maps by only analyzing rotations, projections, and scaling operations. We suggest looking at a linear algebra textbook like the one by Strang [40], which provides a more thorough introduction to SVD and its properties.

SVD decomposes a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ into three matrices

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\intercal.$$

where $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices, and $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ is a diagonal matrix. The diagonal entries of $\boldsymbol{D}$ are non-negative and called singular values. They are ordered from the largest singular value s_1 to the smallest s_m or s_n depending on whether $m \leq n$ or $m > n$. The columns of $\boldsymbol{U}$ and $\boldsymbol{V}$ are referred to as left and right singular vectors, respectively. To simplify our analysis, we further decompose the diagonal matrix $\boldsymbol{D}$ into a square matrix $\boldsymbol{S}$ containing all singular values and a projection matrix $\boldsymbol{P}$ with ones on the diagonal and zeros elsewhere. We call this decomposition extended singular value decomposition (eSVD).

To better understand eSVD, we look at the case $m < n$. This case is of particular interest to us since it captures the characteristics of a redundant robot acting in a lower-dimensional task space. The regular SVD has the structure

$$\boldsymbol{A} = \boldsymbol{U} \cdot \underbrace{\left[\begin{array}{ccc|c} s_1 & & & \\ & \ddots & & \boldsymbol{0} \\ & & s_m & \end{array}\right]}_{\boldsymbol{D}} \cdot \boldsymbol{V}^\intercal.$$

Here, $\boldsymbol{0}$ denotes the zero matrix with suiting dimensionality. Now, eSVD splits $\boldsymbol{D}$ into its scaling part and a separate projection, resulting in the decomposition

$$\boldsymbol{A} = \boldsymbol{U} \cdot \underbrace{\begin{bmatrix} s_1 & & \\ & \ddots & \\ & & s_m \end{bmatrix}}_{\boldsymbol{S}} \cdot \underbrace{\left[\begin{array}{ccc|c} 1 & & & \\ & \ddots & & \boldsymbol{0} \\ & & 1 & \end{array}\right]}_{\boldsymbol{P}^\intercal} \cdot \boldsymbol{V}^\intercal.$$

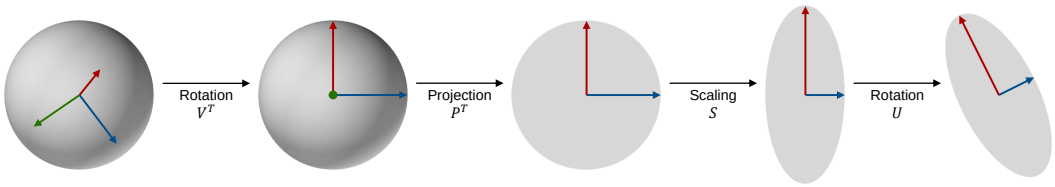


Figure 3.7.: Illustration of the extended singular value decomposition (eSVD) of a 2 by 3 matrix. The arrows correspond to the singular vectors.

The $m < n$ case works analogously, but the projection happens after the scaling, and we write $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{P}\boldsymbol{S}\boldsymbol{V}^\intercal$.

The appeal of decomposition methods for analyzing linear maps is that the resulting matrices provide insides into the characteristics and are nicely interpretable. Figure 3.7 illustrates the effect of the individual matrices resulting from applying eSVD to a 2 by 3 matrix. $\boldsymbol{V}^\intercal$ rotates the frame to be aligned with the right singular vectors. The projection $\boldsymbol{P}^\intercal$ eliminates the third dimension. Then, $\boldsymbol{S}$ scales the space according to the singular values, and finally, $\boldsymbol{U}$ rotates the frame to complete the mapping.

Besides the intuitive interpretation, the resulting matrices have practical properties which simplify derivations.

- The rotational matrices $\boldsymbol{U}, \boldsymbol{V}$ are orthogonal leading to

  $$\boldsymbol{U}\boldsymbol{U}^\intercal = \boldsymbol{U}^\intercal\boldsymbol{U} = \boldsymbol{I}_m, \quad \boldsymbol{V}\boldsymbol{V}^\intercal = \boldsymbol{V}^\intercal\boldsymbol{V} = \boldsymbol{I}_n$$

  where $\boldsymbol{I}_d$ is the the d-dimensional identity matrix.
- Applying any power to $\boldsymbol{S}$ applies the operation element-wise to its diagonal.
- The multiplication of $\boldsymbol{P}$ and its transpose either leads to the lower dimensional identity $\boldsymbol{P}^\intercal\boldsymbol{P} = \boldsymbol{I}_m$ or it results in

  $$\boldsymbol{P}\boldsymbol{P}^\intercal = \begin{bmatrix} \boldsymbol{I}_m & \\ & \boldsymbol{0}_{n-m} \end{bmatrix},$$

  an identity-like matrix where the last $(n-m)$ diagonal entries are zeros. For $m > n$, the indices m, n are swapped.

In summary, given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with $m < n$, performing eSVD leads to

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^{\intercal}\boldsymbol{V}^{\intercal}. \tag{3.8}$$

In the opposite case, where $m > n$, we write

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{P}\boldsymbol{S}\boldsymbol{V}^{\intercal}.$$

In this thesis, we will use eSVD to formulate a common framework for Jacobian methods, derive a new sampling strategy, and analyze its properties for exploration. Note that since it is only a small extension to the original SVD, we can base numerical calculations on standard implementations of SVD.

4. Local Task Space Exploration

The common approach in robot RL applies Gaussian action noise to the individual joints. For redundant robots, this approach can lead to inefficient exploration. E.g, if two joints are moving the end-effector in the same way, exciting them in sync leads to larger task space coverage. On the other hand, individual Gaussian noise might anneal the effect of both joints and hence, hinder exploration.

Figure 4.1 visualizes the general idea of local task space exploration. By focusing on joint actions that effectively move the end-effector, we increase task space exploration without increasing the action effort.

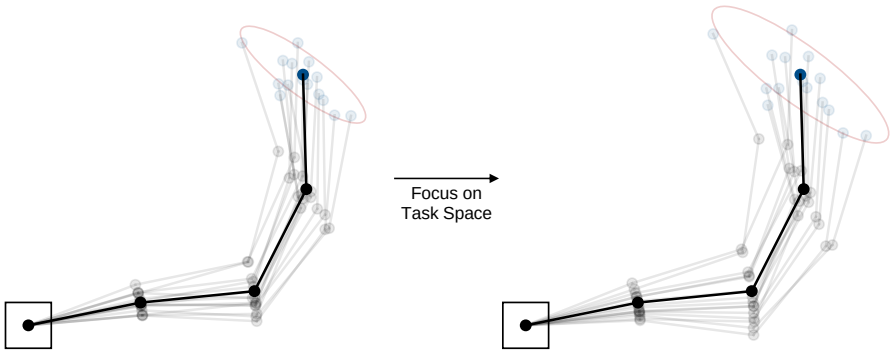


Figure 4.1.: Focusing on effective actions in task space increases task space entropy for single step exploration without increasing action effort. End-effector positions are shown in blue. The red ellipse corresponds to the standard deviation in task space.

4.1. Local Maximum Entropy Task Space Exploration

In this section, we find the optimal Gaussian action distribution for the single-step exploration problem. To this end, we derive the covariance matrix that maximizes task space entropy under an action penalty given the dynamics of the system.

We assume that the mapping from the action $\boldsymbol{u}_i$ at time step i to the task space coordinate $\boldsymbol{x}_{i+1}$ at the next time step $i+1$ is given by an affine transformation

$$\boldsymbol{x}_{i+1} = \boldsymbol{A}_i \boldsymbol{u}_i + \boldsymbol{b}_i, \tag{4.1}$$

where $\boldsymbol{x}_{i+1}, \boldsymbol{b}_i \in \mathbb{R}^m$, $\boldsymbol{A}_i \in \mathbb{R}^{m \times n}$ with full rank, $\boldsymbol{u}_i \in \mathbb{R}^n$, and $m < n$. It's readily apparent that all control affine systems, including the illustrative case of the manipulator, adhere to this structure up to linearization. Since we are only interested in the single-step problem, we omit the subscript and write $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{u} + \boldsymbol{b}$.

The Gaussian action distribution is denotes by $p_u(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}; \boldsymbol{\mu}_u, \boldsymbol{\Sigma}_u)$. For the purpose of this derivation, we set $\boldsymbol{\mu}_u = \boldsymbol{0}$. However, for every fixed $\boldsymbol{\mu}_u$, the derivation is identical. Using the fact that the tasks space transformation (4.1) is affine, we can apply the transformation rule for Gaussian random variables (3.7) and find the task space distribution

$$p_x(\boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{b}, \boldsymbol{A}\boldsymbol{\Sigma}_u\boldsymbol{A}^\intercal\right).$$

The entropy of this density is obtained by (3.2) and reads

$$H(p_x) = \frac{m}{2} + \frac{m}{2}\ln(2\pi) + \frac{1}{2}\ln\left(\det\left(\boldsymbol{A}\boldsymbol{\Sigma}_u\boldsymbol{A}^\intercal\right)\right).$$

Since the first two terms are independent of the parameters, we will omit them in the further derivation.

Easily, increasing the variance results in greater entropy. However, simply increasing the variance is undesirable because of action limits and the search for energy-efficient policies. Hence, we introduce the effort $E(p_u) = \mathbb{E}\left[\boldsymbol{u}^\intercal\boldsymbol{u}\right]$ as a penalty term in the optimization. We can rearrange the term with the aid of matrix identities

$$\mathbb{E}\left[\boldsymbol{u}^\intercal\boldsymbol{u}\right] = \mathbb{E}\left[\mathrm{Tr}\left[\boldsymbol{u}\boldsymbol{u}^\intercal\right]\right] = \mathrm{Tr}\left[\mathbb{E}\left[\boldsymbol{u}\boldsymbol{u}^\intercal\right]\right] = \mathrm{Tr}\left[\boldsymbol{\Sigma}_u\right],$$

which uncovers the direct dependency on the parameters.

Finally, we arrive at the optimization objective

$$\max \quad \underbrace{H(p_x)}_{\text{maximize task space entropy}} - \underbrace{\frac{\lambda}{2}E(p_u)}_{\text{minimize effort}}$$

where $\lambda > 0$ is the penalty coefficient. Plugging in the expressions for entropy and effort leads to the objective function

$$\mathcal{L}(\boldsymbol{\Sigma}_u) = \frac{1}{2}\ln\left(\det\left(\boldsymbol{A}\boldsymbol{\Sigma}_u\boldsymbol{A}^\intercal\right)\right) - \frac{\lambda}{2}\operatorname{Tr}\left[\boldsymbol{\Sigma}_u\right].$$

To make sure the optimizer $\boldsymbol{\Sigma}_u^*$ is a valid covariance matrix, we parameterize $\boldsymbol{\Sigma}_u$ as $\boldsymbol{\Sigma}_u = \boldsymbol{M}\boldsymbol{M}$ where $\boldsymbol{M}$ is the non-negative symmetric square root of $\boldsymbol{\Sigma}_u$. We will ensure symmetry and non-negativity later on. With the new parameterization, we rewrite $\mathcal{L}$ and get

$$\mathcal{L}(\boldsymbol{M}) = \frac{1}{2}\ln\left(\det\left(\boldsymbol{A}\boldsymbol{M}\boldsymbol{M}\boldsymbol{A}^\intercal\right)\right) - \frac{\lambda}{2}\operatorname{Tr}\left[\boldsymbol{M}\boldsymbol{M}\right].$$

To obtain a solution, we calculate the gradient and set it to zero, which gives the necessary condition

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{M}} = \frac{1}{2}\left(\boldsymbol{A}^\intercal\left(\boldsymbol{A}\boldsymbol{M}\boldsymbol{M}\boldsymbol{A}^\intercal\right)^{-1}\boldsymbol{A} - \lambda\boldsymbol{I}_n\right)\boldsymbol{M} = \boldsymbol{0}, \tag{4.2}$$

where $\boldsymbol{0} \in \mathbb{R}^{n\times n}$. We can substitute $\boldsymbol{A}$ with its eSVD (3.8) $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal$ yielding

$$\begin{aligned}
\boldsymbol{0} &= \frac{1}{2}\left(\boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal\left(\boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\boldsymbol{M}\boldsymbol{M}\boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal\right)^{-1}\boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal - \lambda\boldsymbol{I}_n\right)\boldsymbol{M} \\
&= \frac{1}{2}\left(\boldsymbol{V}\boldsymbol{P}\left(\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\boldsymbol{M}\boldsymbol{M}\boldsymbol{V}\boldsymbol{P}\right)^{-1}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal - \lambda\boldsymbol{I}_n\right)\boldsymbol{M} \\
&= \frac{1}{2}\boldsymbol{V}\left(\boldsymbol{P}\left(\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\boldsymbol{M}\boldsymbol{M}\boldsymbol{V}\boldsymbol{P}\right)^{-1}\boldsymbol{P}^\intercal - \lambda\boldsymbol{I}_n\right)\boldsymbol{V}^\intercal\boldsymbol{M}.
\end{aligned}$$

Now, both sides are multiplied by $\boldsymbol{V}^\intercal$ from the left and $\boldsymbol{V}$ from the right and we get

$$\boldsymbol{0} = \frac{1}{2}\left(\underbrace{\boldsymbol{P}\left(\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\boldsymbol{M}\boldsymbol{M}\boldsymbol{V}\boldsymbol{P}\right)^{-1}\boldsymbol{P}^\intercal)}_{\text{rank } m} - \lambda\boldsymbol{I}_n\right)\boldsymbol{V}^\intercal\boldsymbol{M}\boldsymbol{V}.$$

Written in this form, we observe that the indicated term is an n by n matrix of rank m. Hence, the expression in parentheses can only become zero along the first m dimensions.

We extract these first dimensions through multiplication by $\boldsymbol{P}^\intercal$ from the left and $\boldsymbol{P}$ from the right and solve the sub-problem

$$\begin{aligned} \boldsymbol{0} &= (\boldsymbol{P}^\intercal \boldsymbol{V}^\intercal \boldsymbol{M}\boldsymbol{M}\boldsymbol{V}\boldsymbol{P})^{-1} - \lambda \boldsymbol{I}_m \\ \Leftrightarrow \frac{1}{\lambda}\boldsymbol{I}_m &= \boldsymbol{P}^\intercal \boldsymbol{V}^\intercal \boldsymbol{M}\boldsymbol{M}\boldsymbol{V}\boldsymbol{P}. \end{aligned}$$

By means of pseudo-inversion of $\boldsymbol{P}$, we find

$$\boldsymbol{M}\boldsymbol{M} = \frac{1}{\lambda}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal \quad \Leftarrow \quad \boldsymbol{M} = \frac{1}{\sqrt{\lambda}}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal$$

being a solution to this equation and satisfying the symmetry and non-negativity constraints. Moreover, the last $(n-m)$ eigenvalues of $\boldsymbol{M}$ are 0 which suggests that this choice of $\boldsymbol{M}$ solves the necessary condition (4.2). We conclude our derivation by validating this claim:

$$\begin{aligned} &\frac{1}{2}\left(\boldsymbol{A}^\intercal\left(\boldsymbol{A}\boldsymbol{M}\boldsymbol{M}\boldsymbol{A}^\intercal\right)^{-1}\boldsymbol{A} - \lambda\boldsymbol{I}_n\right)\boldsymbol{M} \\ &= \frac{1}{2}\Bigg(\boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal\left(\boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\frac{1}{\sqrt{\lambda}}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\frac{1}{\sqrt{\lambda}}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal\right)^{-1}\cdot \\ &\qquad\qquad \cdot\boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal - \lambda\boldsymbol{I}_n\Bigg)\frac{1}{\sqrt{\lambda}}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal \\ &= \frac{1}{2}\left(\boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal\left(\frac{1}{\lambda}\boldsymbol{U}\boldsymbol{S}^2\boldsymbol{U}^\intercal\right)^{-1}\boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal - \lambda\boldsymbol{I}_n\right)\frac{1}{\sqrt{\lambda}}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal \\ &= \frac{1}{2}\left(\lambda\boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal\boldsymbol{U}\boldsymbol{S}^{-2}\boldsymbol{U}^\intercal\boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal - \lambda\boldsymbol{I}_n\right)\frac{1}{\sqrt{\lambda}}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal \\ &= \frac{1}{2}\left(\lambda\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal - \lambda\boldsymbol{I}_n\right)\frac{1}{\sqrt{\lambda}}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal \\ &= \frac{\sqrt{\lambda}}{2}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal - \frac{\sqrt{\lambda}}{2}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal \\ &= \frac{\sqrt{\lambda}}{2}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal - \frac{\sqrt{\lambda}}{2}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal \\ &= \boldsymbol{0}. \end{aligned}$$

Finally, we obtain the optimal covariance matrix

$$\boldsymbol{\Sigma}_u^* = \boldsymbol{M}\boldsymbol{M} = \left(\frac{1}{\sqrt{\lambda}}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\right)\left(\frac{1}{\sqrt{\lambda}}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\right) = \frac{1}{\lambda}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal.$$

After establishing a general framework for Jacobian methods in the next section, we will provide some intuition on this abstract result and analyze potential benefits.

4.2. Common Jacobian Framework

Joint actions that effectively move the end-effector are usually obtained by applying Jacobian methods (section 3.3). We will now generalize the Jacobian inverse and Jacobian transpose method and define a new class of Jacobian methods of power p. This generalization will allow us to compare the newly found maximum entropy solution to well-known task space methods.

As previously mentioned in section 3.3, the Jacobian transpose and the Jacobian inverse method share a similar structure. Expressing both methods using the eSVD of $\boldsymbol{J} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal$ gives

$$\boldsymbol{J}^\intercal\boldsymbol{v} = \boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal\boldsymbol{v}$$

for the Jacobian transpose method and

$$\begin{aligned}
\boldsymbol{J}^\dagger\boldsymbol{v} &= \boldsymbol{J}^\intercal\left(\boldsymbol{J}\boldsymbol{J}^\intercal\right)^{-1}\boldsymbol{v} \\
&= \boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal\left(\boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal\boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal\right)^{-1}\boldsymbol{v} \\
&= \boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal\left(\boldsymbol{U}\boldsymbol{S}^2\boldsymbol{U}^\intercal\right)^{-1}\boldsymbol{v} \\
&= \boldsymbol{V}\boldsymbol{P}\boldsymbol{S}\boldsymbol{U}^\intercal U\boldsymbol{S}^{-2}\boldsymbol{U}^\intercal\boldsymbol{v} \\
&= \boldsymbol{V}\boldsymbol{P}\boldsymbol{S}^{-1}\boldsymbol{U}^\intercal\boldsymbol{v}
\end{aligned}$$

for the Jacobian inverse method. Here, we only consider $m < n$, but $m \geq n$ can be analyzed analogously.

Looking at the equations, it becomes obvious that their only difference is the exponent of the singular value matrix $\boldsymbol{S}$. It is $\boldsymbol{S}^{+1}$ for the Jacobian transpose method and $\boldsymbol{S}^{-1}$ for the Jacobian inverse method. This observation motivates the following definition of a general class of Jacobian methods of power p.

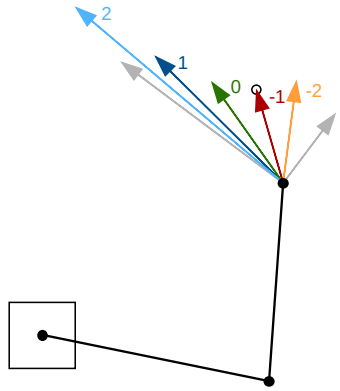


Figure 4.2.: Jacobian methods of power p transform a target velocity (black circle) differently. The resulting velocities are depicted by the arrows, and the power of the corresponding Jacobian method is indicated. The principle axes of the manipulability ellipsoid are shown in gray.

Definition 1. *Let $\boldsymbol{J}$ be a Jacobian matrix and $\boldsymbol{J} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\mathsf{T}\boldsymbol{V}^\mathsf{T}$ its eSVD. A mapping from a task space coordinate $\boldsymbol{v}$ to a joint space coordinate $\boldsymbol{u}$ of the form*

$$\boldsymbol{u} = \boldsymbol{V}\boldsymbol{P}\boldsymbol{S}^p\boldsymbol{U}^\mathsf{T}\boldsymbol{v}$$

is called Jacobian method of power p and we denote the corresponding (pseudo) Jacobian matrix as

$$\boldsymbol{J}_p = \boldsymbol{U}\boldsymbol{S}^{-p}\boldsymbol{P}^\mathsf{T}\boldsymbol{V}^\mathsf{T}.$$

Besides rephrasing the well-known Jacobian methods as Jacobian methods of power 1 and -1, this definition allows for other powers in between and outside. Figure 4.2 visualizes the effect of different powers on the end-effector movement. The target velocity vector $\boldsymbol{v}$ is transformed to joint space using different powers p, resulting in different end-effector velocities. We observe that only the $p = -1$ method (Jacobian inverse) enforces exactly matching the target velocity vector while the other methods deviate. This deviation leads to velocity vectors closer to the larger principle axis of the manipulability ellipsoid for $p > -1$ and velocity vectors closer to the smaller axis for $p < -1$.

We can explain this observation by analyzing the transformation from the target velocity $\boldsymbol{v}_\mathrm{d}$ to the resulting velocity $\boldsymbol{v}_\mathrm{r}$, which is given by

$$\boldsymbol{v}_\mathrm{r} = \boldsymbol{J}\boldsymbol{J}_p^\dagger\boldsymbol{v}_\mathrm{d}.$$

Expanding the expression with eSVD, we find

$$\begin{aligned}\boldsymbol{v}_\mathrm{r} &= \boldsymbol{J}\boldsymbol{J}_p^\dagger \boldsymbol{v}_\mathrm{d} \\ \boldsymbol{v}_\mathrm{r} &= \boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal \boldsymbol{V}^\intercal \boldsymbol{V}\boldsymbol{P}\boldsymbol{S}^p \boldsymbol{U}^\intercal \boldsymbol{v}_\mathrm{d} \\ \boldsymbol{v}_\mathrm{r} &= \boldsymbol{U}\boldsymbol{S}^{p+1}\boldsymbol{U}^\intercal \boldsymbol{v}_\mathrm{d}.\end{aligned}$$

For $p > -1$, singular values that are larger than 1 get enforced while smaller singular values get diminished. The opposite is true for $p < -1$.

4.3. Interpretation and Comparison of Maximum Entropy Solution

In section 4.1, we found the optimal Gaussian distribution for local task space exploration. For a system $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{u} + \boldsymbol{b}$ and $A = \boldsymbol{U}\boldsymbol{S}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal$, the resulting optimal covariance matrix reads

$$\boldsymbol{\Sigma}_u = \frac{1}{\lambda}\boldsymbol{V}\boldsymbol{P}\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal. \tag{4.3}$$

In robotics applications, this system matrix $\boldsymbol{A}$ is usually the Jacobian $\boldsymbol{J}$. This assumption even holds for acceleration-controlled robots. Hence, we can exchange $\boldsymbol{A}$ for $\boldsymbol{J}$. Moreover, since $\boldsymbol{S}^0 = \boldsymbol{I}_m$ and $\boldsymbol{U}^\intercal\boldsymbol{U} = \boldsymbol{I}_m$, we can expand expression (4.3), which gives

$$\begin{aligned}\boldsymbol{\Sigma}_u &= \frac{1}{\lambda}\boldsymbol{V}\boldsymbol{P}\boldsymbol{S}^0\boldsymbol{U}^\intercal U\boldsymbol{S}^0\boldsymbol{P}^\intercal\boldsymbol{V}^\intercal \\ \boldsymbol{\Sigma}_u &= \frac{1}{\lambda}\boldsymbol{J}_0^\intercal\boldsymbol{J}_0 \\ \boldsymbol{\Sigma}_u &= \boldsymbol{J}_0^\intercal\left(\frac{1}{\lambda}\boldsymbol{I}_m\right)\boldsymbol{J}_0.\end{aligned}$$

Finally, we can define a new diagonal covariance matrix $\boldsymbol{\Sigma}_v$ with variance σ_v^2 for every dimension as

$$\boldsymbol{\Sigma}_v = \sigma_v^2\boldsymbol{I}_m = \frac{1}{\lambda}\boldsymbol{I}_m.$$

Now, sampling the action $\boldsymbol{u}$ from $\mathcal{N}(\boldsymbol{\mu}_u, \boldsymbol{\Sigma}_u)$ is equivalent to sampling $\boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_v)$ and applying the transformation

$$\boldsymbol{u} = \boldsymbol{\mu}_u + \boldsymbol{J}_0^\intercal\boldsymbol{v} = \boldsymbol{\mu}_u + \boldsymbol{J}_0^\dagger\boldsymbol{v},$$

which means sampling a target velocity from an isotropic Gaussian distribution and transforming it to joint space using the Jacobian method of power 0.

Alternatively, we can interpret the sampling process as joint space sampling with null space filtering. Here, a candidate action $\bar{\boldsymbol{u}}$ is drawn from an isotropic Gaussian distribution $\mathcal{N}(\mathbf{0}, \sigma_v^2 \boldsymbol{I}_n)$. The final action $\boldsymbol{u}$ is then obtained by $\boldsymbol{u} = \boldsymbol{\mu}_u + \boldsymbol{J}^\dagger \boldsymbol{J} \bar{\boldsymbol{u}}$, which filters out the null space components of $\bar{\boldsymbol{u}}$. We can prove the equivalence by showing that the resulting covariance matrix is identical

$$\begin{aligned}
\boldsymbol{\Sigma}_u &= \boldsymbol{J}^\dagger \boldsymbol{J} (\sigma_v^2 \boldsymbol{I}_n) \boldsymbol{J}^\dagger \boldsymbol{J} \\
&= \sigma_v^2 \boldsymbol{J}^\dagger \boldsymbol{J} \boldsymbol{J}^\dagger \boldsymbol{J} \\
&= \sigma_v^2 \boldsymbol{J}^\dagger \boldsymbol{J} \\
&= \sigma_v^2 \boldsymbol{V} \boldsymbol{P} \boldsymbol{S}^{-1} \boldsymbol{U}^\intercal \boldsymbol{U} \boldsymbol{S} \boldsymbol{P}^\intercal \boldsymbol{V}^\intercal \\
&= \sigma_v^2 \boldsymbol{V} \boldsymbol{P} \boldsymbol{P}^\intercal \boldsymbol{V}^\intercal.
\end{aligned}$$

Note that the null space of $\boldsymbol{J}_p$ is invariant to changes of p.

While the first interpretation is useful for the implementation of a working algorithm, the latter one highlights two key properties of the maximum entropy solution. First, the manipulability ellipsoid does not change compared to joint space sampling. I.e., the resulting task space distribution is the same for joint space sampling and maximum entropy sampling. And second, the improved efficiency comes from reducing null space actions. Thus, by using the maximum entropy sampling, you can either reduce the action effort while keeping the same task space entropy or increase task space entropy without increasing the action effort.

The theoretical benefit can be quantified as follows. Assuming isotropic Gaussian joint space exploration with zero mean and standard deviation σ_{old} along each dimension, the effort is

$$\mathbb{E}[\boldsymbol{u}^\intercal \boldsymbol{u}] = \text{Tr}[\sigma_{\text{old}}^2 \boldsymbol{I}_n] = n \sigma_{\text{old}}^2.$$

where n is the number of action dimensions. For the maximum entropy exploration, we have

$$\begin{aligned}
\mathbb{E}[\boldsymbol{u}^\intercal \boldsymbol{u}] &= \text{Tr}[\sigma_v^2 \boldsymbol{V} \boldsymbol{P} \boldsymbol{P}^\intercal \boldsymbol{V}^\intercal] \\
&= \sigma_v^2 \, \text{Tr}[\boldsymbol{P} \boldsymbol{P}^\intercal \boldsymbol{V}^\intercal \boldsymbol{V}] \\
&= \sigma_v^2 \, \text{Tr}[\boldsymbol{P} \boldsymbol{P}^\intercal] \\
&= m \sigma_v^2
\end{aligned}$$

where m is the number of task space dimensions. Setting both expressions equal, we find

$$\sigma_v = \sqrt{\frac{n}{m}} \sigma_{\text{old}}. \tag{4.4}$$

Hence, reducing null space exploration enables $\sqrt{n/m}$ wider task space exploration without a larger expected action norm. Notably, this result also highlights that if the number of task space dimensions m and action dimensions n is equal, the maximum entropy solution coincides with common joint space exploration.

4.4. Null Space Sampling

Even though our previous analysis suggests reducing null space exploration completely, null space exploration might still be important for RL. The derived optimum only considers the single-step exploration problem from a fixed configuration. However, the configuration itself impacts the exploration and changes due to null space actions. E.g., the manipulability ellipsoid might have a different shape. Another important factor stems from the learning objective. Even though the goal of a task is fully described in low-dimensional task space coordinates, the objective can have null space dependencies. Such dependencies could be obstacle avoidance, where for a fixed end-effector position, the robot might collide in some joint configurations, or time objectives, rewarding faster completion of the task. Moreover, when learning to control all joints of the robot, the policy network can produce null space actions that need correction and therefore, null space exploration.

To add null space exploration to the task space exploration scheme, we use concepts from null space control (subsection 3.3.3). An action $\boldsymbol{u}_0$ is drawn from the null space distribution $\mathcal{N}(\mathbf{0}, \sigma_0 \boldsymbol{I}_n)$, transformed to the null space by

$$\left(\boldsymbol{I}_n - \boldsymbol{J}^\dagger \boldsymbol{J}\right) \boldsymbol{u}_0,$$

and added to the task space sample. Alternatively, it is possible to adapt the covariance matrix $\boldsymbol{\Sigma}_u$ using the theory of affine transformations of Gaussian random variables (subsection 3.4.2).

Adding null space noise to the exploration scheme changes the expected squared action norm, and hence, the expected theoretical benefit is lower. Incorporating a null space standard deviation $\sigma_0 \leq \sigma_{\text{old}}$, the updated equation (4.4) reads

$$\sigma_v = \sqrt{\frac{n}{m}\sigma_{\text{old}}^2 - \frac{n-m}{m}\sigma_0^2}. \tag{4.5}$$

This equation also shows that if $\sigma_{\text{old}} = \sigma_0 \Rightarrow \sigma_v = \sigma_{\text{old}}$, i.e., without null space reduction, there is no exploration benefit.

4.5. Task Space Aware Action Sampling (TAAS)

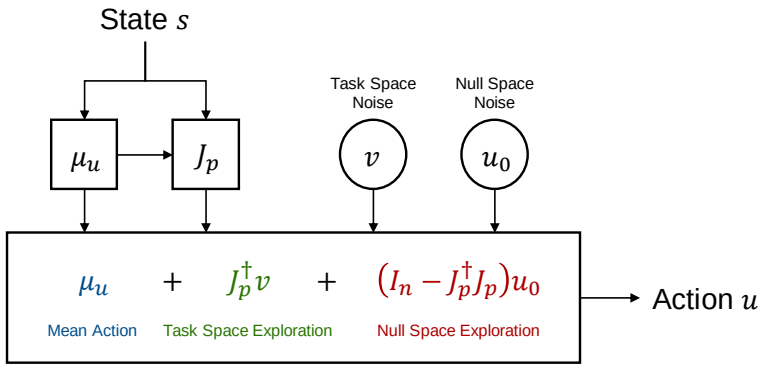


Figure 4.3.: Task-space-aware action sampling (TAAS) constructs a local kinematics model (the Jacobian) to split exploration noise into a task space part and a null space part.

As a consequence of the previous analysis, we compile maximum entropy and null space sampling into one RL algorithm, which we call task-space-aware action sampling (TAAS). We start by describing the sampling procedure as depicted in Figure 4.3, and then we present how action probabilities and entropy can be calculated in this setting. In the end, we will arrive at a simple RL algorithm incorporating kinematic knowledge to further local exploration.

Figure 4.3 illustrates how an action sample is drawn using kinematic information. The mean action $\boldsymbol{\mu}_u$ is the output of a mean policy network. The Jacobian of power p, $\boldsymbol{J}_p$, is calculated from the state. The noise sample $\boldsymbol{v}$ is drawn from a Gaussian in task space coordinates, transformed to joint space using a Jacobian method, and added to $\boldsymbol{\mu}_u$. Finally, another noise sample is added that is drawn from a joint space distribution and projected onto the null space. Note that we use $\boldsymbol{\mu}_u$ to correct the Jacobian to explore around the mean trajectory instead of the current state. E.g., for a velocity controlled manipulator, the Jacobian is calculated as $\boldsymbol{J}_p(s + \Delta t\ \boldsymbol{\mu}_u)$ (Δt denotes the time step) instead of $\boldsymbol{J}_p(s)$.

An alternative approach that yields the same action distribution is constructing a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_u, \boldsymbol{\Sigma}_u)$ using affine transformations of the task space distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_v)$ and the null space distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_0)$. The resulting covariance reads

$$\boldsymbol{\Sigma}_u = \boldsymbol{J}_p^\dagger \boldsymbol{\Sigma}_v \left(\boldsymbol{J}_p^\dagger\right)^\mathsf{T} + \left(\boldsymbol{I}_n - \boldsymbol{J}_p^\dagger \boldsymbol{J}_p\right) \boldsymbol{\Sigma}_0 \left(\boldsymbol{I}_n - \boldsymbol{J}_p^\dagger \boldsymbol{J}_p\right).$$

However, $\boldsymbol{\Sigma}_u$ is badly conditioned if null space exploration is strongly reduced, and in the limit, without null space exploration, the covariance matrix does not have full rank, which is incompatible with standard sampling methods.

To compute the probability of an action given a state $\boldsymbol{s}$, we split the final action $\boldsymbol{u}$ into the mean $\boldsymbol{\mu}_u$, the task space noise $\boldsymbol{v}$, and a null space component $\hat{\boldsymbol{u}}_0$

$$\begin{aligned}\boldsymbol{\mu}_u &= \boldsymbol{\mu}_u(\boldsymbol{s}) \\ \boldsymbol{v} &= \boldsymbol{J}_p(\boldsymbol{s}, \boldsymbol{\mu}_u)(\boldsymbol{u} - \boldsymbol{\mu}_u) \\ \hat{\boldsymbol{u}}_0 &= \boldsymbol{u} - \boldsymbol{\mu}_u - \boldsymbol{J}_p(\boldsymbol{s}, \boldsymbol{\mu}_u)^\dagger \boldsymbol{v}.\end{aligned}$$

Calculating the probability of $\boldsymbol{v}$ is straightforward. In contrast, almost surely, the null space component $\hat{\boldsymbol{u}}_0$ will not be equal to the original null space sample $\boldsymbol{u}_0$ due to the null space projection. As a remedy, assuming the null space covariance matrix $\boldsymbol{\Sigma}_0 = \sigma_0^2 \boldsymbol{I}_n$, we can still come up with the correct probability of the projected sample. For that, we express $\hat{\boldsymbol{u}}_0$ and the null space distribution using an $(n-m)$-dimensional null space basis. We define $\boldsymbol{R} \in \mathbb{R}^{n \times (n-m)}$ to be the counterpart to the matrix $\boldsymbol{P}$, which stems from the eSVD of $\boldsymbol{J}_p$, such that the concatenation $[\boldsymbol{P}|\boldsymbol{R}] = \boldsymbol{I}_n$. Now, $\boldsymbol{R}^\intercal \boldsymbol{V}^\intercal$ projects onto an orthogonal null space basis, and the null space probability is given by

$$\mathcal{N}\left(\boldsymbol{R}^\intercal \boldsymbol{V}^\intercal \hat{\boldsymbol{u}}_0; \boldsymbol{0}, \boldsymbol{R}^\intercal \boldsymbol{V}^\intercal \boldsymbol{\Sigma}_0 \boldsymbol{V} \boldsymbol{R}\right).$$

The probability of $\boldsymbol{u}$ given $\boldsymbol{s}$ $\pi(\boldsymbol{u}|\boldsymbol{s})$ finally reads

$$\pi(\boldsymbol{u}|\boldsymbol{s}) = \underbrace{\mathcal{N}\left(\boldsymbol{v}; \boldsymbol{0}, \boldsymbol{\Sigma}_v\right)}_{\text{task space}} \underbrace{\mathcal{N}\left(\boldsymbol{R}^\intercal \boldsymbol{V}^\intercal \hat{\boldsymbol{u}}_0; \boldsymbol{0}, \boldsymbol{R}^\intercal \boldsymbol{V}^\intercal \boldsymbol{\Sigma}_0 \boldsymbol{V} \boldsymbol{R}\right)}_{\text{null space}} \underbrace{|\underline{\det} \boldsymbol{J}_p|}_{\text{scaling}}.$$

This splitting is possible because the task space and null space distributions are independent, and the scaling factor $|\underline{\det}\boldsymbol{J}_p|$ is multiplicative. Without null space sampling, the corresponding term must be omitted, and for $p = 0$, the scaling factor $|\underline{\det}\boldsymbol{J}_0| = 1$.

The action entropy can be calculated analogously to the probability. Using the splitting property presented in section 3.2, we can write

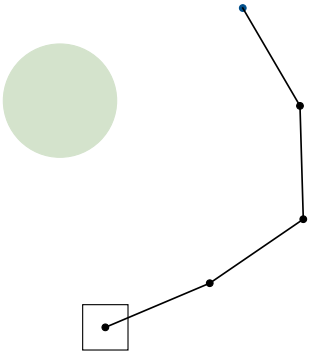
$$H(\pi(\cdot|\boldsymbol{s})) = H\left(\mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Sigma}_v\right)\right) + H\left(\mathcal{N}\left(\boldsymbol{0}, \boldsymbol{R}^\intercal \boldsymbol{V}^\intercal \boldsymbol{\Sigma}_0 \boldsymbol{V} \boldsymbol{R}\right)\right) - \log\left(|\underline{\det}\boldsymbol{J}_p|\right).$$

It is important to note that if no null space exploration is used, the action distribution is degenerate for $m < n$, and the entropy is always $-\infty$, which is impractical and not informative. Therefore, without null space exploration, it is necessary to exclude the null space term.

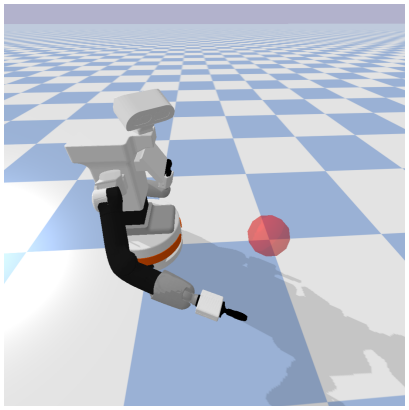
4.6. Empirical Evaluation

After establishing the theory for local exploration and deriving a maximum entropy approach, we now aim to validate the theoretical findings. Therefore, we ask whether there is empirical evidence for the local entropy being maximized with a Jacobian method with $p = 0$, and whether reducing null space exploration actually improves local exploration. Furthermore, we extend the investigation to a global level, asking whether the benefits we derived locally are also present for global task space exploration. Then, we consider the learning problem, where we focus on the impact of null space reduction using TAAS.

Figure 4.4 shows the simulation environments for the experiments in this chapter. The 4 DoF manipulator in Figure 4.4a has links of unit length and can move the joints continuously without joint limits. The manipulator only acts in the x-y plane and is velocity-controlled at 10 Hz. We use this scenario to validate the theoretical findings and learn a simple reaching task. Within 300 time steps, the robot shall position and keep its end-effector inside the green area. For every time step inside the green area, it receives a reward of $+1$, otherwise, it is neither rewarded nor penalized. We use a discount factor of $\gamma = 0.99$.



(a) Planar reaching: A 4 DoF toy robot can move its end-effector freely on the x-y plane. Its task is reaching the goal area in green.



(b) TIAGo reaching: The 7 DoF arm of the TIAGo robot is trained to reach the red goal.

Figure 4.4.: Simulation Environments for local exploration experiments.

Figure 4.4b depicts the second, more complex scenario. The 7 DoF manipulator is velocity-controlled at 25 Hz and shall learn to reach the red goal with a 10 cm radius. The horizon length is 500, $\gamma = 0.99$, and the movement of the manipulator is constrained by joint and action limits and collisions. The reward structure is the same as before. In both scenarios, a goal position and an initial joint configuration are sampled randomly for the whole training, and the agents observe the joint angles. We provide more details about the environments and the hyperparameters in appendix A.3.

4.6.1. Choice of Sampling Method for Local Exploration

For the first experiment, we estimated the local exploration entropy of joint space exploration and Jacobian methods of power -1, 0, and 1 with varying standard deviations. For the entropy estimation, we constructed a Gaussian distribution from the end-effector positions resulting from 20 random actions and averaged over 100 random joint configurations. The results, including the 95 % confidence interval, are shown in Figure 4.5.

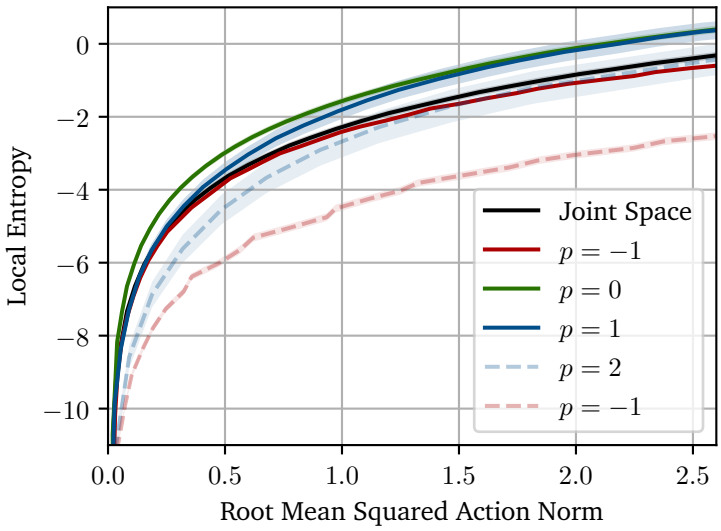


Figure 4.5.: Relationship between the mean action norm and the local task space entropy. Average and 95 % confidence interval from 100 random joint configurations.

We can observe that increasing the mean action norm increases local entropy, supporting the idea of fixing a mean action norm for comparison. As suggested by the theory, the green $p = 0$ line shows the highest local entropy for all reported action norms. Surprisingly, the Jacobian transpose method ($p = 1$) approaches the green $p = 0$ line as the action norm increases. We think this behavior is due to action clipping, which is not considered by our theoretical derivation. The Jacobian inverse method performs slightly worse than joint exploration, indicating that the efficiency gain due to null space reduction is completely countervailed. This phenomenon is caused by the Jacobian inverse method enforcing isotropic task space exploration even close to singularities, which is inefficient in terms of task space exploration. Jacobian methods outside the typical range $[-1, 1]$ perform significantly worse than all other methods. In particular, large negative powers seem to hurt exploration drastically.

For a fixed mean squared action norm,here $\mathbb{E}\left(\|\boldsymbol{u}\|^2\right) = 0.5$, Figure 4.6 visualizes the relation between the power parameter and local task space entropy. In accordance with the maximum entropy derivation, the peak is at $p = 0$, and the curve lowers towards higher and lower powers. The entropy decrease is much stronger for lower powers than higher ones,

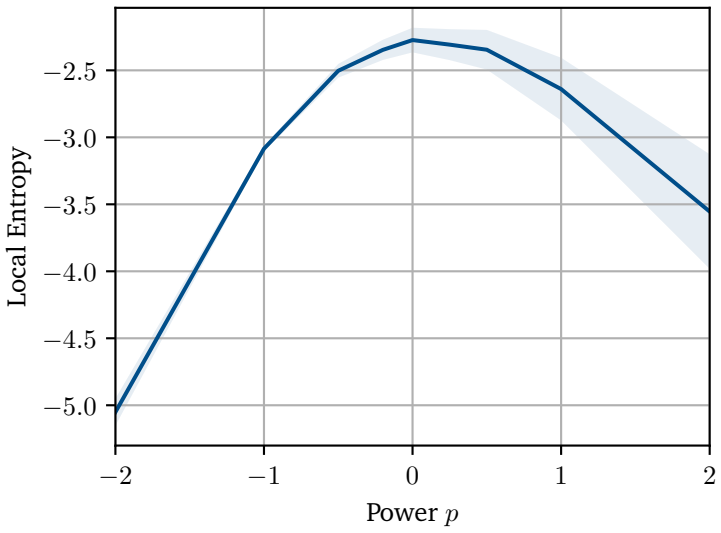


Figure 4.6.: Relation between the power of a Jacobian method and local task space entropy for a mean action norm of 0.25. Average and 95 % confidence interval from 100 random joint configurations.

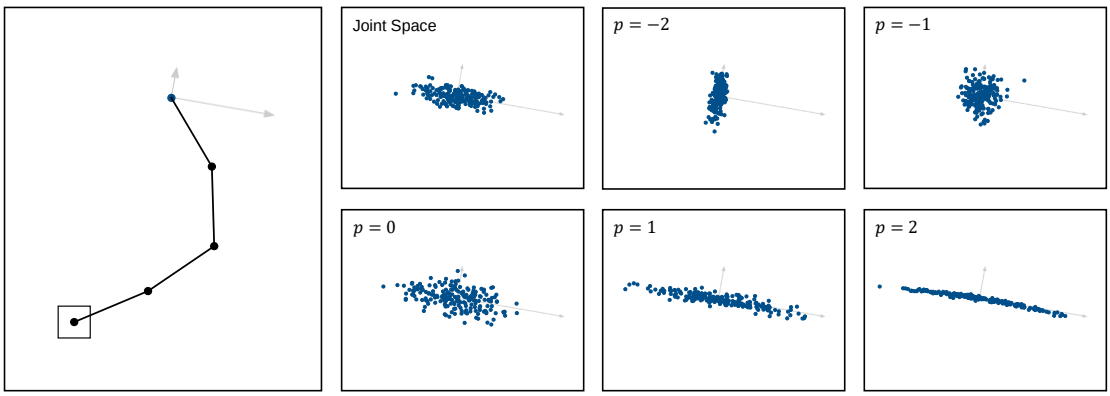


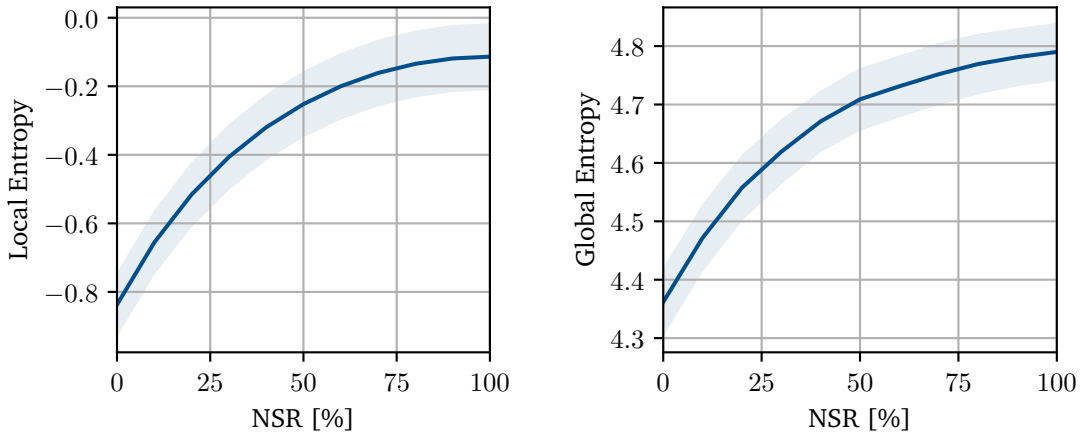
Figure 4.7.: Task space sample patterns for different Jacobian methods and joint space exploration with equal mean squared action norm (0.5) at the same joint configuration (depicted on the left). 200 action samples are drawn each. The principle axes of the manipulability ellipsoid are shown in gray.

which matches our observation from Figure 4.5. Unexpectedly, the confidence interval for larger powers widens, indicating that using a larger power method can potentially improve local task space entropy at certain joint configurations.

Figure 4.7 provides some intuition for the numerical results of the previous experiments. For the depicted joint configuration, the 6 panels on the right show the end-effector position for 200 samples. While for $p = -2$, the samples approach the weak singular axis of the manipulability ellipsoid, the samples for $p = 1$ and $p = 2$ arrange around the strong axis. The Jacobian inverse method ($p = -1$) forms a circular pattern normalizing the axes, and joint space sampling and $p = 0$ sampling follow the proportionality of the axes, confirming that $p = 0$ does not change the shape of the task space distribution. However, while all distributions share the same mean squared action norm of 0.5, the samples of $p = 0$ are spread wider, supporting its optimality.

4.6.2. Impact of Null Space Reduction (NSR) on Exploration

We have already discussed that the Jacobian method of power $p = 0$ is effectively joint space exploration with null space reduction (NSR). After our previous experiments confirmed the optimality of $p = 0$, we now investigate the importance of null space actions on local and also global exploration.



(a) Local exploration

(b) Global exploration

Figure 4.8.: Relationship between null space reduction (NSR) and task space entropy. Average and 95 % confidence interval are reported.

(a) 0 % NSR

(b) 90 % NSR

Figure 4.9.: End-effector traces starting from the same initial configuration using the same random seed.

To quantify local exploration, we estimated the local entropy from 100 samples at 100 joint configurations as before. The task space standard deviation σ_v and the null space standard deviation σ_0 were chosen according to

$$\sigma_v = \sqrt{\frac{n}{m}\sigma_{\text{old}}^2 - \frac{n-m}{m}\sigma_0^2}. \tag{4.6}$$

Here, we use a baseline standard deviation of $\sigma_{\text{old}} = 1$, and 90 % NSR means that the null space standard deviation is $\sigma_0 = (1 - 0.9)\sigma_{\text{old}}$. For σ_{old} and σ_0, equation (4.6) then determines σ_v. For the global exploration experiment, we sampled 3 trajectories from 100 initial joint configurations each. The global task space entropy was then estimated using a histogram method with 400 bins.

As can be seen in Figure 4.8, local (Figure 4.8a) and global (Figure 4.8b) portray consistent results: the task space entropy is larger for a stronger NSR. However, the entropy gain for NSR over 90 % is only marginal.

Figure 4.9 shows the end-effector traces for 0 % and 90 % NSR using the same random seed. The trace of the 90 % sampling shows higher end-effector velocities and, hence, explores more areas of the task space, which explains the increased global entropy.

4.6.3. Impact on Learning

The $p = 0$ Jacobian method with 100 % NSR showed superior results in the previous experiments that focused purely on task space exploration. However, the ultimate goal of exploration is accumulating knowledge about the environment to learn a new task. We already brought up concerns that strong NSR could hinder learning success. In the final experiments of this section, we investigate the impact of different NSR percentages on learning. We train the agents with PPO (subsection 3.1.4) on the sparse reward-reaching tasks described previously and evaluate their final performance, success rate, and action effort. In the following experiments, we approach the question of whether employing TAAS instead of pure joint space sampling is beneficial in robot RL.

Planar Reaching

The evaluation metrics after 0.5 million training steps for the planar reaching experiment are presented in the tables 4.1, 4.2, and 4.3. Additional results for the planar reaching and the TIAGo reaching experiments can be found in appendix A.2. For each documented

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	92 %	100 %	100 %	0 %	100 %
0.7	96 %	96 %	100 %	0 %	96 %
0.5	84 %	92 %	96 %	0 %	92 %
0.3	64 %	80 %	84 %	0 %	88 %

Table 4.1.: Planar reaching: success rates.

standard deviation (Std) and null space reduction (NSR) configuration, we accumulated the data of 25 randomly initialized trials (random goal and random initial joint configuration) and provide averages and 95 % confidence intervals. The Std value corresponds to σ_{old} in equation (4.6) and indicates an equal mean squared action norm of across all untrained agents with the same Std.

Table 4.1 lists the success rates where we consider a trial successful if the average cumulative return is at least 100. We observe that generally, the larger the Std, the more likely it is to learn the task successfully. Surprisingly, the success rate with a Std of 1 and 0 % NSR is lower than the success rate of the same configuration with a Std of 0.7. We consider this an outlier without statistical significance. Quite drastically, we see that with 99.9 % NSR, the agent was never able to learn the task successfully independent of the Std level. Since 100 % NSR leads to comparable success rates as 50 % NSR, the low performance of 99.9 % NSR is probably caused by numerical instabilities in the gradient calculation. 50 % NSR and 100 % NSR lead to the same number or more successful trials for all Std values compared to joint space exploration, and 90 % NSR improves the success rate even further. Notably, with 90 % NSR, at already 0.7 Std, all trials learned successfully, which suggests that this configuration explored most efficiently.

Looking at the average performance (cumulative, discounted reward) in Table 4.2, the values largely follow the same pattern as the success rates. E.g., a larger Std means generally better performance, and the performance of the 99.9 % trials implies no improvement to a random actor. However, the average performances of the 90 % NSR runs are always below the performances of the 50 % NSR runs and 100 % runs. This observation indicates that even though the 90 % NSR actors can learn to solve the task more often, they are also negatively influenced by numerical instabilities. A Std of 1 and 50 % NSR lead to the best performance.

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	82.8 (±10.4)	90.3 (±1.5)	85.2 (±3.1)	2.7 (±1.4)	88.1 (±3.3)
0.7	84.8 (±7.6)	85.2 (±7.7)	83.2 (±4.0)	2.0 (±0.8)	85.7 (±7.7)
0.5	73.1 (±13.6)	80.3 (±10.4)	79.4 (±8.1)	1.8 (±0.9)	81.6 (±10.4)
0.3	54.7 (±17.3)	66.6 (±14.8)	63.4 (±13.6)	3.0 (±2.1)	74.4 (±12.2)

Table 4.2.: Planar reaching: average performance.

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	1.06 (±0.03)	1.01 (±0.05)	0.76 (±0.04)	2.25 (±0.11)	1.27 (±0.23)
0.7	0.83 (±0.04)	0.82 (±0.04)	0.65 (±0.03)	1.70 (±0.09)	0.99 (±0.15)
0.5	0.66 (±0.03)	0.66 (±0.04)	0.56 (±0.04)	1.38 (±0.17)	0.71 (±0.05)
0.3	0.44 (±0.03)	0.47 (±0.03)	0.43 (±0.03)	0.95 (±0.12)	0.56 (±0.07)

Table 4.3.: Planar reaching: root mean squared action norm.

By comparing our observations with the reported values in Table 4.3, we see that the lower performance values of the 90 % NSR runs correspond to a lower root mean squared action norm. A NSR of 50% or 90% reduces the final action norm compared to 0% NSR. The configuration with a Std of 0.7 and 90 % NSR leads to the lowest root mean squared action norm while successfully learning the task in all trials. For 100 % NSR and the not learning 99.9 % NSR, we observe higher values and higher uncertainties compared to most other trials.

TIAGo Reaching

Compared to the planar reaching task, the TIAGo reaching task is more difficult to learn because of its higher-dimensional task and action space, the smaller target area, joint limits, collisions, and a higher control frequency. The increased difficulty lets us evaluate how much the previous findings can be generalized to more complex scenarios, and if

the importance of null space exploration changes with the additional challenges. In the following tables, we report the same metrics as before after 1 million training steps.

In contrast to planar reaching, none of the success rates in Table 4.4 reaches 100 %, confirming the increased difficulty of the task. As before, the runs with 99.9 % NSR were not able to learn the task, and the success rates generally tend to be better with higher Std. However, we observe a performance drop for 90 % NSR when going from a Std of 0.7 to 1.0. Moreover, the 90 % NSR success rates are all inferior compared to 0 %, 50 %, and 100 % NSR success rates, which could be a sign of numerical instabilities. A 50 % NSR seems to be the best choice in terms of success rate for this experiment. For each Std, it outperforms the 0 % NSR baseline, and its variant with a Std of 1 reports the highest success rate overall.

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	80 %	92 %	12 %	0 %	88 %
0.7	72 %	88 %	40 %	0 %	56 %
0.5	60 %	72 %	36 %	0 %	56 %
0.4	56 %	60 %	40 %	0 %	60 %
0.3	60 %	64 %	32 %	0 %	56 %

Table 4.4.: TIAGo reaching: success rates.

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	66.9 ($\pm$14.6)	77.0 ($\pm$10.6)	7.0 ($\pm$6.3)	0.1 ($\pm$0.1)	66.4 ($\pm$12.0)
0.7	59.0 ($\pm$16.2)	68.1 ($\pm$13.3)	20.7 ($\pm$10.7)	0.0 ($\pm$0.1)	45.9 ($\pm$15.4)
0.5	49.4 ($\pm$17.3)	55.4 ($\pm$16.3)	19.1 ($\pm$9.2)	0.1 ($\pm$0.1)	49.1 ($\pm$15.1)
0.3	48.3 ($\pm$16.9)	50.0 ($\pm$16.3)	15.4 ($\pm$9.8)	0.0 ($\pm$0.1)	44.5 ($\pm$17.2)

Table 4.5.: TIAGo reaching: average performance.

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	2.32 ($\pm$0.22)	1.73 ($\pm$0.20)	2.24 ($\pm$0.17)	5.13 ($\pm$1.08)	5.68 ($\pm$1.53)
0.7	1.62 ($\pm$0.18)	1.36 ($\pm$0.18)	1.43 ($\pm$0.16)	4.39 ($\pm$1.54)	3.94 ($\pm$1.53)
0.5	1.13 ($\pm$0.11)	0.99 ($\pm$0.13)	1.10 ($\pm$0.09)	3.79 ($\pm$1.42)	2.63 ($\pm$0.79)
0.4	1.02 ($\pm$0.14)	0.92 ($\pm$0.11)	0.91 ($\pm$0.08)	3.46 ($\pm$1.28)	1.88 ($\pm$0.46)
0.3	0.81 ($\pm$0.07)	0.75 ($\pm$0.10)	0.72 ($\pm$0.06)	2.44 ($\pm$0.87)	1.30 ($\pm$0.31)

Table 4.6.: TIAGo reaching: root mean squared action norm.

Table 4.5 shows that higher success rates explain the higher average performance. Nonetheless, while (0.5, 50 %) and (0.7, 0 %) have the same success rates, the performance of the latter is higher, and while (0.7, 50 %) and (1.0, 0 %) have a comparable average performance, the success rate of (0.7, 50 %) is considerably higher. These discrepancies can be explained by the difference in the root mean squared action norm (Table 4.6): the 50 % NSR runs consistently lead to a lower root mean squared action norm compared with 0 % NSR. This observation indicates that successful task completion is less null space dependent than optimal performance. For the 100 % NSR runs, we find the same phenomenon of increased mean squared action norms in Table 4.6 as before.

4.7. Discussion

In this chapter, we focused on local task space exploration and how kinematic knowledge can be used to improve it. We derived an optimal sampling strategy within a local maximum entropy framework and found that it is strongly linked to existing Jacobian methods. By defining a common framework for Jacobian methods with a power parameter p, we could make the connecting explicit. The optimal strategy with $p = 0$ was proven to be equivalent to joint space exploration with null space reduction (NSR). Based on the theory, we developed the task-space-aware action sampling (TAAS) method that uses kinematic knowledge and combines task space and null space exploration.

The empirical evaluation confirmed our theoretical findings. The Jacobian method with $p = 0$ showed its superiority, and NSR increases exploration efficiency in task space.

However, the learning results show a more nuanced picture. Even though reducing null space sampling could improve success rates and performance when applied to 50 %, the trials without null space exploration could only improve over joint space sampling on the simpler planar task. It appears that more complex scenarios require more exploration of the null space actions, but it remains unclear how much NSR is beneficial and when it hinders learning successfully.

While in the exploration experiments, 100 % NSR explored most efficiently, the mean squared action norm after training was higher compared to other methods. This is of particular interest since we were originally interested in reducing the mean squared action norm. However, the high norms are most likely caused by interpolations of the neural network approximator that cannot be corrected for due to missing null space exploration. Therefore, it is necessary to either have some degree of null space exploration or let the network only predict actions along the task space axes.

When NSR was applied to a stronger degree but below 100 %, we observed decreased performance measures. This observation is indicative of numerical instabilities in the optimization process but is unrelated to the principle of TAAS. We conjecture that the different scales of the probabilities over the task and the null space lead to poorly conditioned probability ratios during policy optimization. The fact that we observed this instability even for 90 % NSR in the higher dimensional TIAGo reaching experiment supports this conjecture because the standard deviation along each null space axis is smaller than for the planar reaching task. We suggest adapting the PPO clipping objective to work with differently scaled distributions or internally reweighing the probabilities to improve the conditioning.

Overall, the benefit of TAAS for RL is limited. As it is a local method that uses step-wise Gaussian noise for exploration, it widens the spectrum of efficiently solvable RL problems only marginal. This problem motivates the search for global task space methods that make more use of the available task knowledge.

5. Global Task Space Exploration

In the previous chapter, we looked at the local task space exploration problem and developed an algorithm (TAAS) that exploits kinematic knowledge locally to further exploration efficiency. However, we also experienced that for complex robotics tasks, such an exploration method is still insufficient. Reaching a rewarding state can be a rare event, and associating the right actions leading to successful executions is hard. Building on top of the success of RRL, we now focus on global task space exploration that can potentially overcome the limitations of local exploration methods and incorporate additional task knowledge.

5.1. Intuition

Consider the scenario in Figure 5.1. The TIAGo robot is positioned in front of a shelf. It is supposed to learn a control policy that moves the end-effector to a goal region inside the shelf. Only if the the end-effector reaches the goal region, the agent receives a reward of $+1$. However, the position of the goal region is not known, which makes the application of a simple, engineered goal-reaching controller impossible. Hence, we cannot apply RRL to the problem. At the same time, falling back to pure Gaussian action noise exploration is undesirable. The likelihood of reaching a goal state inside the shelf only by chance is low, and we lose the potential benefit of incorporating the prior knowledge available.

Ideally, we would like to apply a learning algorithm that follows two heuristics. First, the exploration should focus on the shelf area. Since we know that the goal position is inside the shelf, trajectories that lead away from the shelf have no chance of reaching the goal at all. We would prefer an algorithm that exploits this prior knowledge. Second, once the goal area is identified, the exploration should focus on trajectories that lead to that goal rather than ending up at a different spot inside the shelf and not receiving any reward.

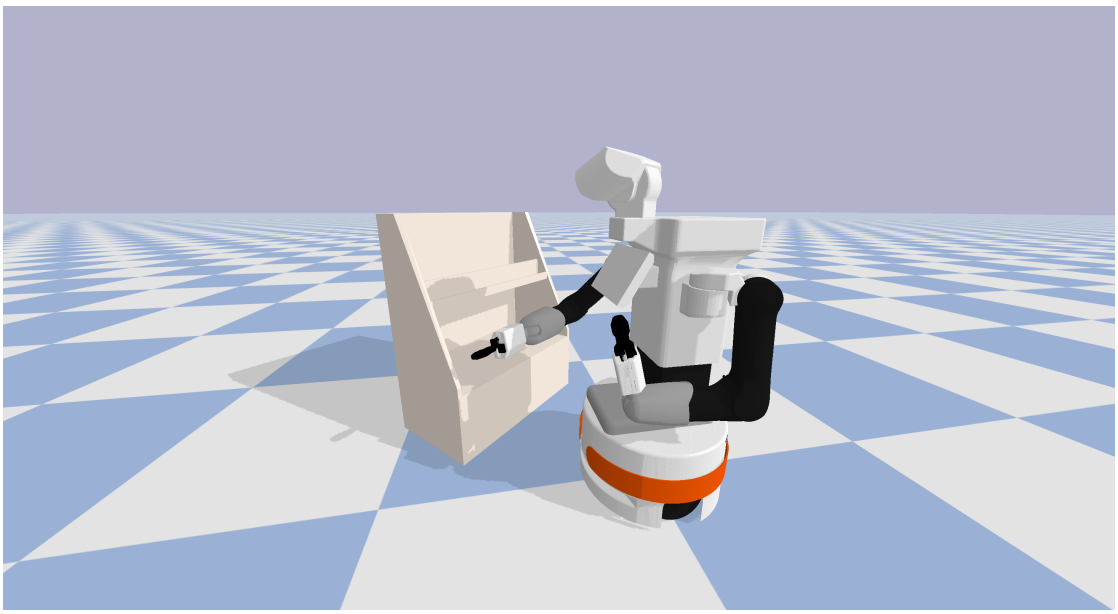


Figure 5.1.: The TIAGo robot is supposed to learn reaching an unknown goal position inside the shelf. Hence, it is preferable to explore trajectories leading toward the shelf.

The idea of splitting the exploration problem into reaching the goal and optimizing the trajectory is closely related to the RRL perspective. A base controller ensures reaching the goal, and the residual policy optimizes the execution to fulfill other criteria. Nonetheless, implementing our idea requires a substantial modification to the RRL framework. Namely, we need the base controller to actively explore the environment. Only this modification can provide well-informed global task space exploration.

In the following sections, we present a new RL agent design that couples local action exploration with global task space exploration. We show how the new method can be integrated into an existing policy optimization algorithm. Finally, we evaluate the new method in simulation on the TIAGo robot.

5.2. Task Space Biasing (TSB)

Task space biasing (TSB) uses two trainable policies as indicated in the overview scheme in Figure 5.2. The global bias policy φ provides a task space target $\boldsymbol{x}$ once per episode. This target indicates which part of the task space the exploration should concentrate on in the current episode. At each time step, using kinematic information about the robot, $\boldsymbol{x}$ is transformed to a joint space action bias $\boldsymbol{b}$ that drives the end-effector movement toward $\boldsymbol{x}$. The local policy π outputs a residual action that is added to $\boldsymbol{b}$, and the superposition $\boldsymbol{u}$ is executed on the robot.

The schematic includes a context state $\boldsymbol{c}$. This context state includes information that might change from episode to episode, which gives the flexibility of a context-dependent task space bias. As motivated previously, the bias policy should make use of prior knowledge. Its initialization can follow expected goal positions or other hints.

Comparing Figure 5.2 with the scheme of RRL (Figure 2.2), we can note certain similarities. The residual policy in RRL is replaced by the local policy in TSB, acting in largely the same way, and the base controller in RRL can be found as a combination of the bias policy and the Jacobian method incorporating prior knowledge and guiding the local policy. However, an important distinction has to be made. Unlike in RRL, the bias policy φ is a learnable, stochastic policy that actively explores the task space. Thus, we can view TSB

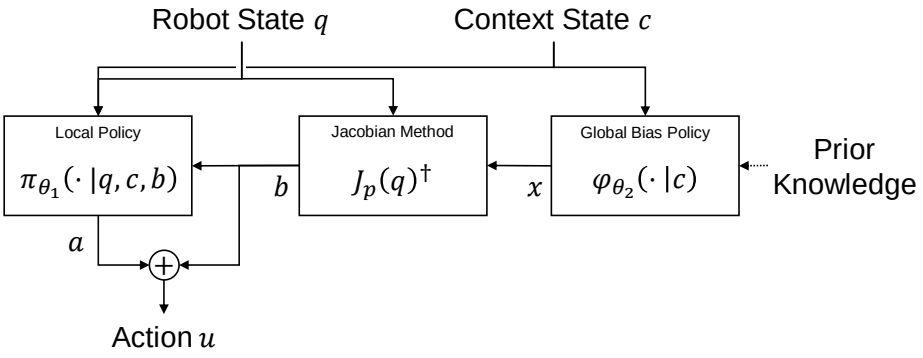


Figure 5.2.: Task space biasing (TSB) uses a global bias policy φ that exploits prior knowledge to guide exploration. The task space target $\boldsymbol{x}$ is transformed to joint space using a Jacobian method, yielding the bias action $\boldsymbol{b}$. The local policy outputs the residual action $\boldsymbol{a}$.

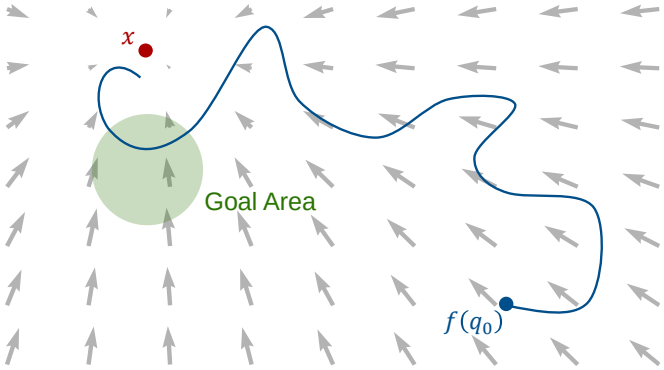


Figure 5.3.: Illustration of TSB working principle. The sampled task space target x (red) becomes the center of a bias vector field (gray) for one episode that causes more exploration toward and around x. A hypothetical task space trajectory starting from $f(q_0)$ is drawn in blue. Target samples close to the goal area (green) likely result in higher rewards and are reinforced throughout the training.

as both a special case of RRL since it focuses explicitly on task space exploration and a generalization of it that opens the space for learnable base controllers.

The exploration mechanism is illustrated in Figure 5.3. For each episode, a task space target $\boldsymbol{x}$ is sampled that acts as an attractor for the end-effector. The attraction is realized by the Jacobian method that outputs a bias action driving the end-effector toward $\boldsymbol{x}$. Meanwhile, the local policy explores in action space at every time step, resulting in a variant trajectory approaching $\boldsymbol{x}$. If $\boldsymbol{x}$ happens to be sampled close to the goal area, the local exploration makes it likely to receive a reward. Over time, the learning process will reinforce highly rewarding task space targets and focus on points near the goal area. At the same time, the local policy is supposed to learn to avoid obstacles and optimize the goal-reaching trajectory.

5.2.1. Parameterization and Action Sampling

We assume that the state $\boldsymbol{s}$ is a concatenation $[\boldsymbol{q}|\boldsymbol{c}]$ of a robot state $\boldsymbol{q}$ and a context state $\boldsymbol{c}$ that includes episode-specific information. At the beginning of an episode, $\boldsymbol{x}$ is sampled from the global bias policy

$$\boldsymbol{x} \sim \varphi_{\boldsymbol{\theta}_2}(\cdot|\boldsymbol{c}).$$

We implement $\varphi_{\boldsymbol{\theta}_2}$ as a Gaussian distribution where the mean $\boldsymbol{\mu}$ is the sum of the output of a neural network $\boldsymbol{g}(\boldsymbol{c})$ and an a priori defined center $\boldsymbol{x}_{\text{prior}}$

$$\boldsymbol{\mu} = \boldsymbol{x}_{\text{prior}} + \boldsymbol{g}(\boldsymbol{c}).$$

The covariance has the structure $\sigma^2\boldsymbol{I}$. This parameterization allows us to easily incorporate prior knowledge by choosing an initial exploration center $\boldsymbol{x}_{\text{prior}}$ and an exploration *radius* σ. The free network parameters and the standard deviation σ are contained in the trainable parameter vector $\boldsymbol{\theta}_2$. Given $\boldsymbol{x}$ and the robot state $\boldsymbol{q}$, the Jacobian method determines a bias action $\boldsymbol{b}$ as follows. First, a task space direction vector $\boldsymbol{d}$ with a maximum norm of 1 is found:

$$\boldsymbol{d}(\boldsymbol{x}, \boldsymbol{q}) = \begin{cases} \boldsymbol{x} - \boldsymbol{f}(\boldsymbol{q}) & \text{if} \quad \|\boldsymbol{x} - \boldsymbol{f}(\boldsymbol{q})\| \leq 1 \\ \frac{\boldsymbol{x}-\boldsymbol{f}(\boldsymbol{q})}{\|\boldsymbol{x}-\boldsymbol{f}(\boldsymbol{q})\|} & \text{if} \quad \|\boldsymbol{x} - \boldsymbol{f}(\boldsymbol{q})\| > 1 \end{cases}.$$

The norm clipping ensures that the bias term will not dominate the final action, and the local policy can still explore even if $\boldsymbol{x}$ is far from the current end-effector position $\boldsymbol{f}(\boldsymbol{q})$. Then, we get the bias action

$$\boldsymbol{b}(\boldsymbol{q}, \boldsymbol{x}) = \boldsymbol{J}_p(\boldsymbol{q})^\dagger \boldsymbol{d}(\boldsymbol{x}).$$

To avoid problems with kinematic singularities and bias the action in a natural way that follows the manipulability ellipsoid, we use the Jacobian of power $p = 0$ $\boldsymbol{J}_0(\boldsymbol{q})$.

The residual action $\boldsymbol{a}$ is sampled from the local policy

$$\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}_1}(\cdot|\boldsymbol{q}, \boldsymbol{c}, \boldsymbol{b}(\boldsymbol{q}, \boldsymbol{x}))$$

that observes the full state and the bias. The bias is included to let the local policy react to potentially undesirable bias actions and stay closer to a Markovian setting. $\varphi_{\boldsymbol{\theta}_1}$ is a trainable Gaussian distribution with a diagonal covariance matrix and a neural network, predicting the mean action.

5.2.2. Integration into PPO

Since a task space target $\boldsymbol{x}$ is sampled once per episode and influences all actions samples $\boldsymbol{u}_i$ within a trajectory, the trajectory distribution $p_{\boldsymbol{\theta}}(\tau)$ changes to

$$\begin{aligned} p(\tau) &= p_0(\boldsymbol{s}_0)\varphi_{\boldsymbol{\theta}_2}(\boldsymbol{x}|\boldsymbol{s}_0)\prod_{i=0}^{T-1} p(\boldsymbol{s}_{i+1}|\boldsymbol{s}_i,\boldsymbol{u}_i)p(\boldsymbol{u}_i|\boldsymbol{s}_i,\boldsymbol{x}) \\ &= p_0(\boldsymbol{s}_0)\varphi_{\boldsymbol{\theta}_2}(\boldsymbol{x}|\boldsymbol{s}_0)\prod_{i=0}^{T-1} p(\boldsymbol{s}_{i+1}|\boldsymbol{s}_i,\boldsymbol{u}_i)\pi_{\boldsymbol{\theta}_1}(\boldsymbol{a}_i|\boldsymbol{s}_i,\boldsymbol{x}). \end{aligned}$$

Now, in a similar fashion as in subsection 3.1.3, we restructure the performance difference lemma

$$\begin{aligned} &\mathbb{E}_\tau\left[\sum_{t=0}^{T-1}\gamma^t A^{\text{old}}(\boldsymbol{s}_t,\boldsymbol{u}_t)\right] \\ &= \int p_0(\boldsymbol{s}_0)\varphi_{\boldsymbol{\theta}_2}(\boldsymbol{x}|\boldsymbol{s}_0)\sum_{t=0}^{T-1}\gamma^t\left(\prod_{i=0}^{T-1} p(\boldsymbol{s}_{i+1}|\boldsymbol{s}_i,\boldsymbol{u}_i)\pi_{\boldsymbol{\theta}_1}(\boldsymbol{a}_i|\boldsymbol{s}_i,\boldsymbol{x})\right)A^{\text{old}}(\boldsymbol{s}_t,\boldsymbol{u}_t)d\tau \\ &= \frac{1-\gamma}{1-\gamma^T}\int p_0(\boldsymbol{s}_0)\varphi_{\boldsymbol{\theta}_2}(\boldsymbol{x}|\boldsymbol{s}_0)d^{\pi_{\boldsymbol{\theta}_1}}(\boldsymbol{s}|\boldsymbol{s}_0,\boldsymbol{x})\pi_{\boldsymbol{\theta}_1}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{x})A^{\text{old}}(\boldsymbol{s},\boldsymbol{u})d\boldsymbol{a}d\boldsymbol{s}d\boldsymbol{x}d\boldsymbol{s}_0 \\ &= \frac{1-\gamma}{1-\gamma^T}\mathbb{E}_{\substack{\boldsymbol{s}_0\sim p_0,\ \boldsymbol{x}\sim\varphi_{\boldsymbol{\theta}_2}(\cdot|\boldsymbol{s}_0),\\ \boldsymbol{s}\sim d^{\pi_{\boldsymbol{\theta}_1}}(\cdot|\boldsymbol{s}_0,\boldsymbol{x}),\ a\sim\pi_{\boldsymbol{\theta}_1}(\cdot|\boldsymbol{s},\boldsymbol{x})}}\left[A^{\text{old}}(\boldsymbol{s},\boldsymbol{u})\right] \end{aligned}$$

where $d^{\pi_{\boldsymbol{\theta}_1}}(\cdot|\boldsymbol{s}_0,\boldsymbol{x})$ is now conditioned on the initial state and the sampled target $\boldsymbol{x}$. With the common approximations, we arrive at the adapted surrogate loss

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{\substack{\boldsymbol{s}_0\sim p_0,\ \boldsymbol{x}\sim\varphi_{\boldsymbol{\theta}_2}(\cdot|\boldsymbol{s}_0),\\ \boldsymbol{s}\sim\boldsymbol{u}^{\pi_{\boldsymbol{\theta}_{1,\text{old}}}}(\cdot|\boldsymbol{s}_0,\boldsymbol{x}),\ \boldsymbol{a}\sim\pi_{\boldsymbol{\theta}_1}(\cdot|\boldsymbol{s},\boldsymbol{x})}}\left[A^{\text{old}}(\boldsymbol{s},\boldsymbol{u})\right] \\ &= \mathbb{E}_{\substack{\boldsymbol{s}_0\sim p_0,\ \boldsymbol{x}\sim\varphi_{\boldsymbol{\theta}_{2,\text{old}}}(\cdot|\boldsymbol{s}_0)\\ \boldsymbol{s}\sim\boldsymbol{u}^{\pi_{\boldsymbol{\theta}_{1,\text{old}}}}(\cdot|\boldsymbol{s}_0,\boldsymbol{x}),\ \boldsymbol{a}\sim\pi_{\boldsymbol{\theta}_{1,\text{old}}}(\cdot|\boldsymbol{s},\boldsymbol{x})}}\left[\frac{\pi_{\boldsymbol{\theta}_1}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{x})\varphi_{\boldsymbol{\theta}_2}(\boldsymbol{x}|\boldsymbol{s}_0)}{\pi_{\boldsymbol{\theta}_{1,\text{old}}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{x})\varphi_{\boldsymbol{\theta}_{2,\text{old}}}(\boldsymbol{x}|\boldsymbol{s}_0)}A^{\text{old}}(\boldsymbol{s},\boldsymbol{u})\right]. \end{aligned}$$

The clipping objective follows naturally. However, instead of clipping the probability ratio at once, we perform the clipping for the local and the bias policy separately to avoid that one policy dominates the update. We use a clipping threshold of $\epsilon = 0.1$ leading to approximately the same maximum clipping ratio as before: $(1-0.1)^2 \approx (1-0.2)$.

5.3. Empirical Evaluation

With the following experiment, we investigate if and to what extent the new TSB method can improve on regular model-free RL and compare it to the *ideal* RRL scenario. Furthermore, we explore differences between the RRL base controller and the TSB bias policy. Lastly, we assess the robustness of TSB to sub-optimal bias initializations.

We evaluate the performance on the TIAGo robot in simulation as already seen in Figure 5.1. The task is to reach a goal area with a 10 cm radius inside the shelf with the end-effector of the right arm. If the end-effector is within the goal radius, the agent receives a reward of 1. The position of the goal area changes for each episode, and the agent observes the robot joint angles and the goal position. The action frequency is 100 Hz. We use a horizon length of 500 and train the policies for 3 million steps.

We compare TSB with two baselines. The **RL** baseline uses plain PPO for training. It corresponds to pure local exploration without incorporating prior knowledge. The **RRL** baseline is considered the *ideal* scenario because it directly exploits the knowledge of the goal position. As the base controller, we use the same Jacobian method as for TSB. All methods use identical architectures for the local policies, and the hyperparameters resulted from a small grid search and are documented in appendix A.3.3. The global bias policy of the TSB method is initialized such that its mean is at the center of the shelf, and two times its standard deviation covers all expected goal positions.

5.3.1. Comparison to RL and RRL

Figure 5.4 summarizes the results from the comparison of TSB to the two baselines RL and RRL. We present the average learning curves and the 95 % confidence intervals after 25 trials. Unsurprisingly, before training (at time step 0), the initial performance of RL is the lowest, and RRL performs the best. The well-informed initialization of the bias policy in TSB improves initial performance compared to the RL baseline but cannot reach RRL performance at all. However, after 0.75 million training steps, the TSB learning curve surpasses the RRL learning curve, and its final performance is significantly better than those of the two baselines. This observation is particularly remarkable since RRL does not need to learn the mapping from the context variable to the actual goal position. The RL baseline cannot overcome the lack of integrated prior knowledge.

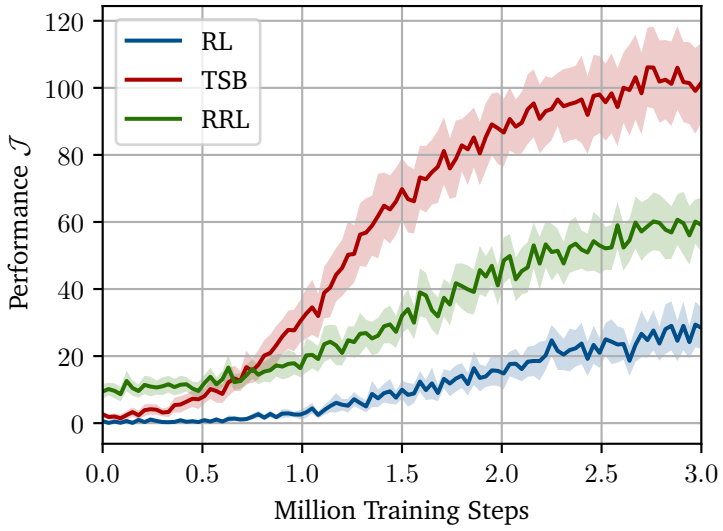


Figure 5.4.: Mean learning curve with 95 % confidence interval for TSB (red) and the baselines RL (blue) and RRL (green). TSB outperforms both baselines by a large margin.

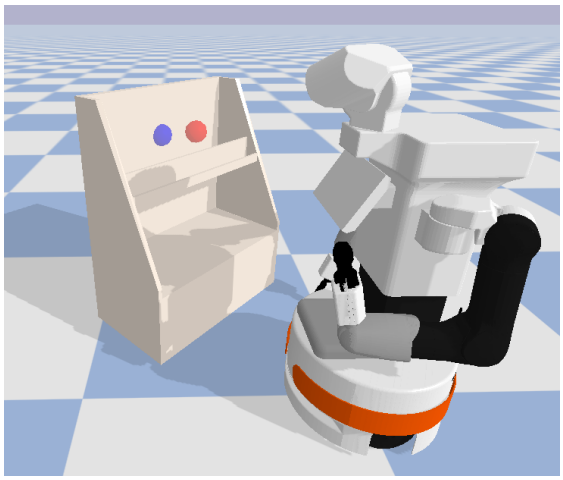
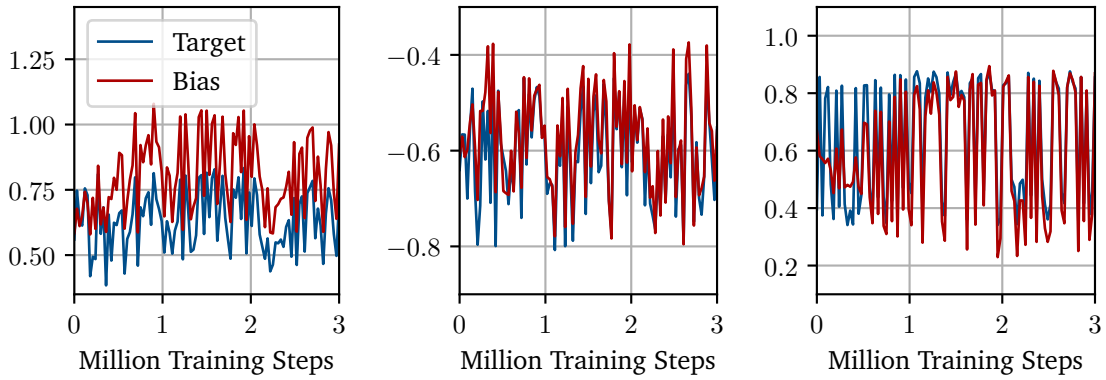


Figure 5.5.: The position of the learned bias in blue and the target position in red are offset. In relation to the target, the bias lies opposite to most initial end-effector positions.

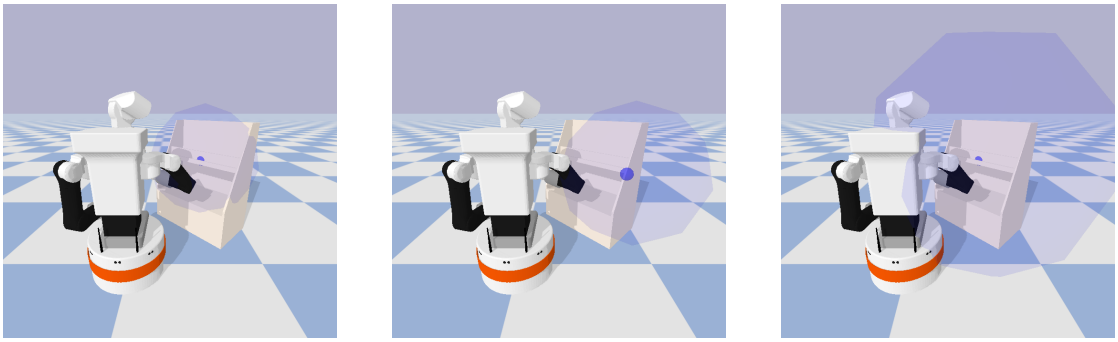


(a) x (b) y (c) z

Figure 5.6.: The mean of the learned bias distribution and the position of the target along the indicated axis. While at the end of training, the bias tracks the y and z coordinates closely, bias and target are offset along x.

To understand how TSB uses the bias, Figure 5.5 depicts the goal position in red and the position of the learned bias in blue. Surprisingly, the positions are clearly offset. The bias is slightly shifted away from the goal to a position opposite to most initial end-effector positions. Even though we only report one instance, we could observe this phenomenon arising in the majority of trials. This offset positioning causes the bias action to drive the end-effector to some extent faster toward the goal but also slightly too far.

The observed offset in Figure 5.5 corresponds to an offset along the x axis in the world coordinates of the simulator. This x offset can also be noticed in Figure 5.6a. Along the other axes, Figure 5.6 reports good tracking performance. However, besides the offset, the goal tracking along x appears to be accurate as well. A final accuracy level seems to be reached after around 0.75 million steps. Beyond that point, there is no perceivable accuracy gain. Notably, at the same time, the learning curve of TSB in Figure 5.4 surpasses the RRL performance.



(a) Well-Informed (b) Shifted (c) Large Variance

Figure 5.7.: We compare different bias distribution initializations to assess the robustness of the method. The figures show the center and two times the standard deviation of the initial bias distribution.

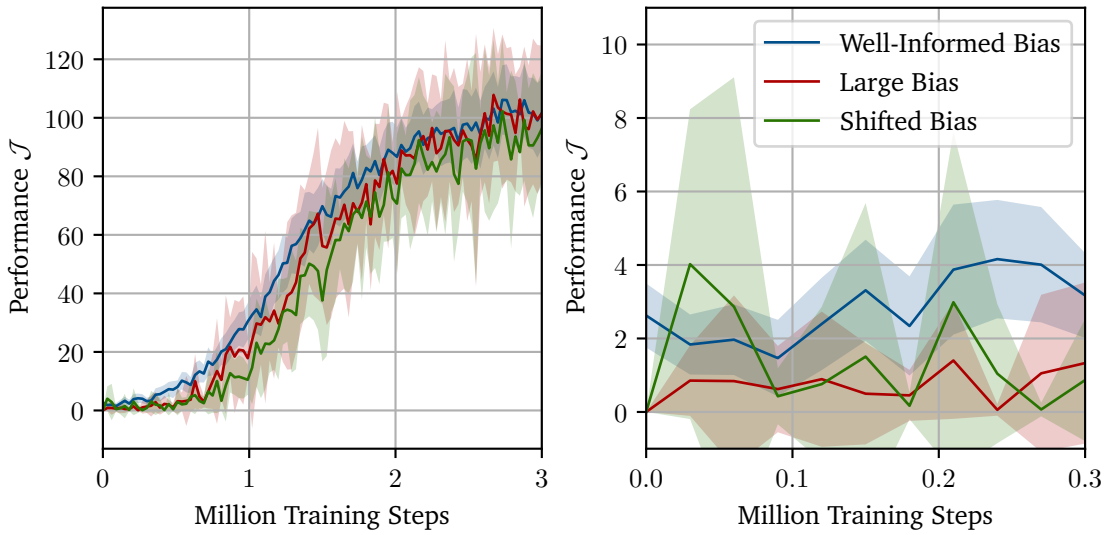


Figure 5.8.: Mean learning curves and 95 % confidence interval for different bias initialization. The right plot shows the same data but focuses on the first 300 thousand training steps. All initializations perform on par.

5.3.2. Robustness to Bias Initialization

After comparing TSB against the two baselines, we evaluate the robustness of the method to sub-optimal bias initializations. The different initial distributions are visualized in Figure 5.7, showing the center and two times the standard deviation. The well-informed initialization is the one we used in the previous experiment. In the shifted initialization, the center is misplaced to the edge of the shelf, and in the large variance initialization, the standard deviation is double the well-informed standard deviation.

We evaluated the learning of the sub-optimal initializations using 5 trials. Figure 5.8 reports the learning curves and confidence intervals with a focus on the first 300 thousand training steps on the right. In the beginning, the well-informed initialization results in a higher performance, and it seems to take longer for the sub-optimally initialized policies to improve. Nevertheless, the initial performance gap diminishes as training progresses, ultimately resulting in comparable final performances.

5.4. Discussion

This chapter described a new approach, task space biasing (TSB), to incorporate prior knowledge to guide local and enable global exploration. We showed how the existing state-of-the-art RL algorithm PPO can be adapted to work with TSB through an alternative surrogate loss. The comparison of the structure of TSB and RRL displayed that the TSB method has comparable components but extends the framework by making the base controller learnable.

We evaluated the new method on a challenging sparse-reward and goal-conditioned reaching task and found that it outperforms plain PPO and RRL by a large margin, even though RRL used a goal-reaching Jacobian method. We analyzed the learned bias and found that it was offset to the goal position in a consistent way. This observation indicates that the gained flexibility of TSB compared to RRL enabled local action biasing that is more beneficial than pure goal-reaching. Instead of driving the end-effector to the goal position directly, the bias policy provides a trajectory around which the local policy can explore and optimize more easily.

In a second comparison, we assessed the robustness of TSB to sub-optimally initialized bias distributions. Even though the initial performance dropped compared to the well-informed initialization, the final performance was equally high independently of the initial bias

distribution. This result assures some robustness of TSB to sub-optimal bias initializations. However, we do not expect a completely misplaced initial bias distribution to lead to comparable results. On the other hand, we experienced that the bias is converging rapidly compared to the overall training, which suggests the use of TSB in scenarios with less prior knowledge available. There, the bias policy could relatively quickly find a promising region to explore, even if the initial bias distribution is wide.

The quick convergence of the bias opens a new question. It remains unclear whether the performance benefit resulted from the interplay between optimizing the bias and the local policy, or if it solely came from the bias policy being optimized on a variant of the MDP that accounts for local exploration noise. If the latter turned out to be true, it would be advisable to optimize the bias first, freeze it, and then optimize the local policy in the style of RRL. This investigation, however, is beyond the scope of this thesis.

Even though the TSB method and the experiments focused on goal reaching, and the context state was the goal position, the developed framework is far more general. Trivially, the context could be a textual or visual description of the goal, but the framework can be directly applied as long as the global bias policy is conditioned only on information available at the first time step of an episode. This generality opens the possibility of making any base controller adaptable during the learning process and, hence, improves the integration of prior knowledge into RL.

6. Conclusion

In this thesis, we studied how task space methods can be integrated into robot RL to further exploration. We approached this overarching question from a local and a global perspective on task space exploration. The theoretic findings and intuitive ideas were used to develop methods exploiting the task structure, namely task-space-aware action sampling (TAAS) and task space biasing (TSB). In both cases, we found that incorporating prior knowledge about the robot and its environment can improve exploration efficiency and learning performance. However, we experienced that a global exploration mechanism is far more effective than an optimized local one. While TAAS could improve on joint space exploration, we do not believe that it can extend the spectrum of tasks solvable by RL significantly. Instead, a global approach, like the proposed TSB method, may do.

In light of the promising results of the TSB method, we see great potential in fusing policies exploring on a global level with local exploration. Hence, we suggest further research to investigate the extension of RRL base controllers to adjustable bias policies. Moreover, the interplay between a local and a global policy connects our work, at least conceptually, to hierarchical RL. Here, an interesting path could be making the connection explicit and combining hierarchical RL with action superposition.

In the local exploration part, we evaluated exploration and learning independently. Understanding the exploration during the learning process, however, remains an open challenge. It could be worth studying which characteristics of the observed data lead to successfully learning a task and which experiences improve learning speed or mislead the optimization. These insights could further improve exploration strategies and learning speed.

This thesis fits into a line of research that tries bridging the gap between learning and engineered control. We believe that strong inductive biases are essential for the emergence of intelligent robots and encourage combining intuition and established techniques from learning and engineering in new creative ways.

Bibliography

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Aug. 2017.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," Aug. 2018.

[3] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Information Fusion*, vol. 85, pp. 1–22, 2022.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," July 2019.

[5] D. Korenkevych, A. R. Mahmood, G. Vasan, and J. Bergstra, "Autoregressive Policies for Continuous Control Deep Reinforcement Learning," Mar. 2019.

[6] A. Raffin, J. Kober, and F. Stulp, "Smooth Exploration for Robotic Reinforcement Learning," June 2021.

[7] O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius, "Pink Noise Is All You Need: Colored Noise Exploration in Deep Reinforcement Learning," in *The Eleventh International Conference on Learning Representations*, Feb. 2023.

[8] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics*. Advanced Textbooks in Control and Signal Processing, London: Springer, 2009.

[9] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual Reinforcement Learning for Robot Control," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6023–6029, May 2019.

[10] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual Policy Learning," Jan. 2019.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series, Cambridge, Massachusetts: The MIT Press, second edition ed., 2018.

[12] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in Applied Mathematics*, vol. 6, pp. 4–22, Mar. 1985.

[13] T. Yang, H. Tang, C. Bai, J. Liu, J. Hao, Z. Meng, P. Liu, and Z. Wang, "Exploration in Deep Reinforcement Learning: A Comprehensive Survey," July 2022.

[14] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep Exploration via Bootstrapped DQN," July 2016.

[15] I. Osband, J. Aslanides, and A. Cassirer, "Randomized Prior Functions for Deep Reinforcement Learning," Nov. 2018.

[16] T. Tan, Z. Xiong, and V. R. Dwaracherla, "Parameterized Indexed Value Function for Efficient Exploration in Reinforcement Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 5948–5955, Apr. 2020.

[17] I. Osband, B. Van Roy, D. Russo, and Z. Wen, "Deep Exploration via Randomized Value Functions," Sept. 2019.

[18] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction," May 2017.

[19] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying Count-Based Exploration and Intrinsic Motivation," Nov. 2016.

[20] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning," Dec. 2017.

[21] E. Hazan, S. Kakade, K. Singh, and A. V. Soest, "Provably Efficient Maximum Entropy Exploration," *ArXiv*, Dec. 2018.

[22] L. Lee, B. Eysenbach, E. Parisotto, E. Xing, S. Levine, and R. Salakhutdinov, "Efficient Exploration via State Marginal Matching," Feb. 2020.

[23] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter Space Noise for Exploration," Jan. 2018.

[24] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy Networks for Exploration," July 2019.

[25] T. Rückstieß, M. Felder, and J. Schmidhuber, "State-Dependent Exploration for Policy Gradient Methods," in *Machine Learning and Knowledge Discovery in Databases* (W. Daelemans, B. Goethals, and K. Morik, eds.), vol. 5212, pp. 234–249, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[26] B. Mazoure, T. Doan, A. Durand, R. D. Hjelm, and J. Pineau, "Leveraging exploration in off-policy algorithms via normalizing flows," Sept. 2019.

[27] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming Exploration in Reinforcement Learning with Demonstrations," Feb. 2018.

[28] G. Zuo, Q. Zhao, J. Lu, and J. Li, "Efficient hindsight reinforcement learning using demonstrations for robotic tasks with sparse rewards," *International Journal of Advanced Robotic Systems*, vol. 17, p. 172988141989834, Jan. 2020.

[29] G. Bellegarda and K. Byl, "Training in Task Space to Speed Up and Guide Reinforcement Learning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2693–2699, Nov. 2019.

[30] M. Kaspar, J. D. M. Osorio, and J. Bock, "Sim2Real Transfer for Reinforcement Learning without Dynamics Randomization," Feb. 2020.

[31] F. Al-Hafez and J. J. Steil, "Redundancy Resolution as Action Bias in Policy Search for Robotic Manipulation," in *Proceedings of the 5th Conference on Robot Learning*, pp. 981–990, PMLR, Jan. 2022.

[32] M. Alakuijala, G. Dulac-Arnold, J. Mairal, J. Ponce, and C. Schmid, "Residual Reinforcement Learning from Demonstrations," June 2021.

[33] T. Staessens, T. Lefebvre, and G. Crevecoeur, "Adaptive Control of a Mechatronic System Using Constrained Residual Reinforcement Learning," *IEEE Transactions on Industrial Electronics*, vol. 69, pp. 10447–10456, Oct. 2022.

[34] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, May 1992.

[35] S. Kakade and J. Langford, "Approximately Optimal Approximate Reinforcement Learning," in *International Conference on Machine Learning*, July 2002.

[36] J. Schulman, N. Heess, T. Weber, and P. Abbeel, "Gradient Estimation Using Stochastic Computation Graphs," Jan. 2016.

[37] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," Apr. 2017.

[38] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," Oct. 2018.

[39] B. Siciliano, "Kinematic control of redundant robot manipulators: A tutorial," *Journal of Intelligent and Robotic Systems*, vol. 3, no. 3, pp. 201–212, 1990.

[40] G. Strang, *Introduction to Linear Algebra*. Wellesley: Wellesley-Cambridge Press, 4th. ed ed., 2009.

A. Appendix

A.1. Derivation of Jacobian Pseudo-Inverse Method

The objective is to find the least squares solution $\dot{\boldsymbol{q}}$ that solves the differential kinematics equation (3.3) and it reads

$$\min \frac{1}{2}\dot{\boldsymbol{q}}^\intercal \dot{\boldsymbol{q}}, \quad \text{s.t. } \boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}.$$

We formulate the constraint problem by means of Lagrangian optimization with the vector valued Lagrangian multiplier $\boldsymbol{\lambda}$ leading to the Lagrangian

$$\mathcal{L} = \frac{1}{2}\dot{\boldsymbol{q}}^\intercal \dot{\boldsymbol{q}} + \boldsymbol{\lambda}^\intercal (\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} - \boldsymbol{v})$$

Deriving with respect to $\dot{\boldsymbol{q}}$ and setting the gradient to $\mathbf{0}$ gives

$$\nabla_{\dot{\boldsymbol{q}}}\mathcal{L} = \dot{\boldsymbol{q}} + \boldsymbol{J}(\boldsymbol{q})^\intercal \boldsymbol{\lambda} = \mathbf{0} \quad \Leftrightarrow \quad \dot{\boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})^\intercal \boldsymbol{\lambda}.$$

We now solve for $\boldsymbol{\lambda}$ by substituting $\dot{\boldsymbol{q}}$ in the constraint

$$\boldsymbol{v} = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^\intercal \boldsymbol{\lambda} \quad \Leftrightarrow \quad \boldsymbol{\lambda} = \left(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^\intercal\right)^{-1} \boldsymbol{v}.$$

Finally, we obtain the solution

$$\begin{aligned}\dot{\boldsymbol{q}} &= \boldsymbol{J}(\boldsymbol{q})^\intercal \left(\boldsymbol{J}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})^\intercal\right)^{-1} \boldsymbol{v} \\ &= \boldsymbol{J}(\boldsymbol{q})^\dagger \boldsymbol{v}.\end{aligned}$$

A.2. Extended Evaluation of Learning with TAAS

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	82.8 (±10.4)	90.3 (±1.5)	85.2 (±3.1)	2.7 (±1.4)	88.1 (±3.3)
0.7	84.8 (±7.6)	85.2 (±7.7)	83.2 (±4.0)	2.0 (±0.8)	85.7 (±7.7)
0.5	73.1 (±13.6)	80.3 (±10.4)	79.4 (±8.1)	1.8 (±0.9)	81.6 (±10.4)
0.4	67.2 (±14.6)	71.5 (±13.7)	75.7 (±10.0)	2.6 (±2.1)	79.3 (±10.3)
0.3	54.7 (±17.3)	66.6 (±14.8)	63.4 (±13.6)	3.0 (±2.1)	74.4 (±12.2)
0.2	40.3 (±17.9)	56.7 (±16.7)	48.2 (±16.9)	1.4 (±0.8)	53.1 (±17.2)
0.1	33.1 (±15.7)	36.4 (±17.5)	28.5 (±14.7)	2.4 (±1.5)	39.3 (±16.0)

Table A.1.: Planar reaching: average performance (cumulative, discounted reward) with 95 % confidence interval for different base standard deviations (std) and null space reductions.

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	92 %	100 %	100 %	0 %	100 %
0.7	96 %	96 %	100 %	0 %	96 %
0.5	84 %	92 %	96 %	0 %	92 %
0.4	80 %	84 %	92 %	0 %	92 %
0.3	64 %	80 %	84 %	0 %	88 %
0.2	48 %	68 %	60 %	0 %	64 %
0.1	48 %	44 %	44 %	0 %	56 %

Table A.2.: Planar reaching: success rates for different base standard deviations (std) and null space reductions.

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	1.06 (±0.03)	1.01 (±0.05)	0.76 (±0.04)	2.25 (±0.11)	1.27 (±0.23)
0.7	0.83 (±0.04)	0.82 (±0.04)	0.65 (±0.03)	1.70 (±0.09)	0.99 (±0.15)
0.5	0.66 (±0.03)	0.66 (±0.04)	0.56 (±0.04)	1.38 (±0.17)	0.71 (±0.05)
0.4	0.57 (±0.03)	0.57 (±0.03)	0.50 (±0.03)	1.07 (±0.14)	0.67 (±0.07)
0.3	0.44 (±0.03)	0.47 (±0.03)	0.43 (±0.03)	0.95 (±0.12)	0.56 (±0.07)
0.2	0.33 (±0.03)	0.34 (±0.03)	0.32 (±0.02)	0.99 (±0.19)	0.41 (±0.05)
0.1	0.21 (±0.02)	0.20 (±0.02)	0.17 (±0.01)	0.85 (±0.20)	0.27 (±0.04)

Table A.3.: Planar reaching: root mean squared action norm and 95 % confidence interval for different base standard deviations (std) and null space reductions.

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	66.9 (±14.6)	77.0 (±10.6)	7.0 (±6.3)	0.1 (±0.1)	66.4 (±12.0)
0.7	59.0 (±16.2)	68.1 (±13.3)	20.7 (±10.7)	0.0 (±0.1)	45.9 (±15.4)
0.5	49.4 (±17.3)	55.4 (±16.3)	19.1 (±9.2)	0.1 (±0.1)	49.1 (±15.1)
0.4	42.1 (±16.3)	49.8 (±17.4)	18.5 (±9.7)	0.1 (±0.3)	49.0 (±16.3)
0.3	48.3 (±16.9)	50.0 (±16.3)	15.4 (±9.8)	0.0 (±0.1)	44.5 (±17.2)
0.2	33.1 (±16.6)	37.4 (±17.1)	12.0 (±8.8)	0.1 (±0.2)	35.9 (±16.4)
0.1	11.1 (±10.0)	24.2 (±13.7)	6.7 (±5.5)	0.6 (±1.1)	18.4 (±12.2)

Table A.4.: TIAGo reaching: average performance (cumulative, discounted reward) with 95 % confidence interval for different base standard deviations (std) and null space reductions.

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	80 %	92 %	12 %	0 %	88 %
0.7	72 %	88 %	40 %	0 %	56 %
0.5	60 %	72 %	36 %	0 %	56 %
0.4	56 %	60 %	40 %	0 %	60 %
0.3	60 %	64 %	32 %	0 %	56 %
0.2	44 %	48 %	16 %	0 %	48 %
0.1	20 %	40 %	16 %	0 %	32 %

Table A.5.: TIAGo reaching: success rates for different base standard deviations (std) and null space reductions.

Std / NSR	0 %	50 %	90 %	99.9 %	100 %
1.0	2.32 (±0.22)	1.73 (±0.20)	2.24 (±0.17)	5.13 (±1.08)	5.68 (±1.53)
0.7	1.62 (±0.18)	1.36 (±0.18)	1.43 (±0.16)	4.39 (±1.54)	3.94 (±1.53)
0.5	1.13 (±0.11)	0.99 (±0.13)	1.10 (±0.09)	3.79 (±1.42)	2.63 (±0.79)
0.4	1.02 (±0.14)	0.92 (±0.11)	0.91 (±0.08)	3.46 (±1.28)	1.88 (±0.46)
0.3	0.81 (±0.07)	0.75 (±0.10)	0.72 (±0.06)	2.44 (±0.87)	1.30 (±0.31)
0.2	0.58 (±0.06)	0.56 (±0.06)	0.53 (±0.05)	2.67 (±0.92)	1.03 (±0.34)
0.1	0.36 (±0.06)	0.33 (±0.05)	0.33 (±0.09)	2.46 (±0.89)	0.62 (±0.14)

Table A.6.: TIAGo reaching: root mean squared action norm and 95 % confidence interval for different base standard deviations (std) and null space reductions.

A.3. Experiment Specifications

A.3.1. Planar Reaching

Environment Specification	
Robot Link Lengths	$[1, 1, 1, 1]$
Joint Constraints	-
Action Space	Velocity commands for every joint
Action Limits	$[\pm\pi, \pm\pi, \pm\pi, \pm\pi]$
Observation Space	Sine and cosine of every joint angle
Goal Radius	0.5
Goal Position	Uniformly distributed over the reachable space of the manipulator
Initial Joint Configuration	Uniformly distributed over full joint space
Control Frequency	10 Hz
Discount Factor γ	0.99
Horizon Length	300

Table A.7.: Planar reaching Environment specifications.

Hyperparameters	
Learning Rate	0.0003
Clipping Ratio	0.2
Entropy Coefficient	0
GAE λ	0.98
Number Policy Update Epochs	10
Number Episodes Per Update	10
Policy and Value Network	64 hidden neurons, ReLU activation
Network Initialization	To reduce the impact of initialization, we initialize the weights between [-0.1, 0.1] and set the last layer bias to 0 for the policy network in all experiments.

Table A.8.: Planar reaching hyperparameters.

A.3.2. TIAGo Reaching

Environment Specification	
Lower Joint Limits	$[-1.17, -1.17, -0.78, -0.39, -2.09, -1.41, -2.09]$
Upper Joint Limits	$1.57, 1.57, 3.92, 2.35, 2.09, 1.41, 2.09]$
Action Space	Velocity commands for every joint
Action Limits	$[\pm 0.975, \pm 0.975, \pm 1.175, \pm 1.175, \pm 0.975, \pm 0.88, \pm 0.88]$
Observation Space	Joint angles
Goal Radius	0.1
Goal Position	Uniformly distributed in a spherical region in front to the robot
Initial Joint Configuration	Uniformly distributed over 80 % of joint space
Control Frequency	25 Hz
Discount Factor γ	0.99
Horizon Length	500
Gravity	Due to insufficient gravity compensation by the Pybullet simulator, we needed to set gravity to zero for all TIAGo experiments.

Table A.9.: TIAGo reaching environment specifications.

Hyperparameters	
Learning Rate	0.0003
Clipping Ratio	0.2
Entropy Coefficient	0
GAE λ	0.98
Number Policy Update Epochs	10
Number Episodes Per Update	10
Policy and Value Network	64 hidden neurons, ReLU activation

Table A.10.: TIAGo reaching hyperparameters.

A.3.3. TIAGo Shelf Environment

Environment Specification	
Lower Joint Limits	$[-1.17, -1.17, -0.78, -0.39, -2.09, -1.41, -2.09]$
Upper Joint Limits	$1.57, 1.57, 3.92, 2.35, 2.09, 1.41, 2.09]$
Action Space	Velocity commands for every joint
Action Limits	$[\pm 0.975, \pm 0.975, \pm 1.175, \pm 1.175, \pm 0.975, \pm 0.88, \pm 0.88]$
Observation Space	Joint angles and goal position
Goal Radius	0.1
Goal Position	Uniformly distributed inside the shelf in each episode
Initial Joint Configuration	Uniformly distributed over 80 % of joint space in each episode
Control Frequency	100 Hz
Discount Factor γ	0.995
Horizon Length	500

Table A.11.: TIAGo shelf environment specifications.

Hyperparameters	
Initial Standard Deviation	1.0
Learning Rate	0.0003
Clipping Ratio	0.2 (TSB: $0.1 \cdot 0.1$)
Entropy Coefficient	0.004
GAE λ	0.98
Number Policy Update Epochs	10
Number Episodes Per Update	20
Policy Network	256 hidden neurons, ReLU activation
Value Network	64 hidden neurons, ReLU activation
Bias Network (TSB only)	no hidden layer, ReLU activation

Table A.12.: TIAGo shelf environment hyperparameters.

Abbreviations and Symbols

List of Abbreviations

Notation	Description
DoF	degrees of freedom
eSVD	extended singular value decomposition
KL	Kullback-Leibler divergence
MDP	Markov decision process
NSR	null space reduction
PPO	proximal policy optimization
Q-function	state-action value function
RL	reinforcement learning
RRL	residual reinforcement learning
SAC	soft actor-critic
Std	standard deviation
SVD	singular value decomposition
TAAS	task-space-aware action sampling
TRPO	trust region policy optimization

TSB	task space biasing
UCB	upper confidence bound

List of Symbols

Notation	Description
$\boldsymbol{a}, \boldsymbol{u}$	(local) action
$\boldsymbol{b}$	bias action
$A(\boldsymbol{s}, \boldsymbol{a}), A_t$	advantage
$\mathcal{D}$	data buffer
$\det \boldsymbol{A}$	determinant of $\boldsymbol{A}$
$\underline{\det} \boldsymbol{A}$	pseudo-determinant of $\boldsymbol{A}$
γ	discount factor
$p_0(\boldsymbol{s})$	initial state distribution
$d^\pi(\boldsymbol{s})$	discounted state distribution of policy π
$u^\pi(\boldsymbol{s})$	undiscounted state distribution of policy π
$E(p)$	effort
$H(p)$	entropy
$\mathbb{E} f(x)\,[]$	expectation
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution
$\boldsymbol{\mu}$	mean of Gaussian distribution
$\boldsymbol{\Sigma}$	covariance matrix of Gaussian distribution
σ	standard deviation
T	horizon length

$\mathcal{L}$	loss function
$\boldsymbol{I}_n$	n-dimensional identity matrix
$\boldsymbol{J}_p(\boldsymbol{q})$	Jacobian of power p
$\boldsymbol{A}^\dagger$	pseudo-inverse of A
$\text{Tr}\left[\boldsymbol{A}\right]$	trace of A
$\mathcal{J}(\boldsymbol{\theta}), \mathcal{J}(\pi)$	reinforcement learning objective
$\boldsymbol{\theta}$	parameter vector
$\pi(\boldsymbol{a}\|\boldsymbol{s})$	(local) policy
$\varphi(\boldsymbol{s})$	base controller
$\varphi(\boldsymbol{x}\|\boldsymbol{c})$	bias policy
$r(s_t, a_t, s_{t+1})$	reward
$R(\tau)$	cumulative discounted reward
$\mathcal{A}$	action space
$\mathcal{S}$	state space
$\boldsymbol{s}$	state
$\boldsymbol{c}$	context state
$\boldsymbol{q}$	joint angles, robot state
$\boldsymbol{e}$)	task space error
$\boldsymbol{x}, \boldsymbol{v}$	task space vectors
τ	trajectory
$V(\boldsymbol{s})$	value function

Figures and Tables

List of Figures

2.1 Overview of exploration paradigms in deep RL. 4

2.2 In residual reinforcement learning (RRL), an action u is the superposition of an action a sampled from the residual policy π_θ and a base action b from the predefined base controller φ. 8

3.1 In reinforcement learning (RL), the agent learns by interacting with the environment. The agent observes states, and its policy acts accordingly. The environment responses with a state transition and a reward signal. The RL algorithm updates the policy to maximize the cumulative reward. 11

3.2 The Jacobian matrix of a 2 DoF manipulator. The blue and red arrow illustrate how the blue and red joint can move the end-effector locally. They are the columns of the Jacobian matrix. The manipulability ellipsoid and its principle axes are depicted in grey. 18

3.3 Kinematics of a 2 DoF manipulator. The blue joint rotates with 0.6 rad/s and the red one with -0.5 rad/s. The scaled column vectors of the Jacobian superpose and result in an end-effector velocity illustrated with the black arrow. 20

3.4 Comparison of the Jacobian pseudo-inverse and the Jacobian transpose method in joint space (left) and task space (right). The Jacobian pseudo-inverse action in blue moves the end-effector along the direct task space path to the goal in green. The Jacobian transpose action corresponds to a gradient step minimizing the task space error in joint space. 21

3.5 Null space actions of a 4 DoF manipulator. While the joint configuration changes, the end-effector remains at the same task space position. 22

3.6 An affine transformation of a Gaussian distribution (in green) can be decomposed into the marginalization over the null space of the transformation (blue) and a scaling and translation operation (red). 25

3.7 Illustration of the extended singular value decomposition (eSVD) of a 2 by 3 matrix. The arrows correspond to the singular vectors. 27

4.1 Focusing on effective actions in task space increases task space entropy for single step exploration without increasing action effort. End-effector positions are shown in blue. The red ellipse corresponds to the standard deviation in task space. 29

4.2 Jacobian methods of power p transform a target velocity (black circle) differently. The resulting velocities are depicted by the arrows, and the power of the corresponding Jacobian method is indicated. The principle axes of the manipulability ellipsoid are shown in gray. 34

4.3 Task-space-aware action sampling (TAAS) constructs a local kinematics model (the Jacobian) to split exploration noise into a task space part and a null space part. 38

4.4 Simulation Environments for local exploration experiments. 40

4.5 Relationship between the mean action norm and the local task space entropy. Average and 95 % confidence interval from 100 random joint configurations. 41

4.6 Relation between the power of a Jacobian method and local task space entropy for a mean action norm of 0.25. Average and 95 % confidence interval from 100 random joint configurations. 42

4.7 Task space sample patterns for different Jacobian methods and joint space exploration with equal mean squared action norm (0.5) at the same joint configuration (depicted on the left). 200 action samples are drawn each. The principle axes of the manipulability ellipsoid are shown in gray. 43

4.8 Relationship between null space reduction (NSR) and task space entropy. Average and 95 % confidence interval are reported. 44

4.9 End-effector traces starting from the same initial configuration using the same random seed. 44

5.1 The TIAGo robot is supposed to learn reaching an unknown goal position inside the shelf. Hence, it is preferable to explore trajectories leading toward the shelf. 52

5.2 Task space biasing (TSB) uses a global bias policy φ that exploits prior knowledge to guide exploration. The task space target $\boldsymbol{x}$ is transformed to joint space using a Jacobian method, yielding the bias action $\boldsymbol{b}$. The local policy outputs the residual action $\boldsymbol{a}$. 53

5.3 Illustration of TSB working principle. The sampled task space target $\boldsymbol{x}$ (red) becomes the center of a bias vector field (gray) for one episode that causes more exploration toward and around $\boldsymbol{x}$. A hypothetical task space trajectory starting from $\boldsymbol{f}(\boldsymbol{q}_0)$ is drawn in blue. Target samples close to the goal area (green) likely result in higher rewards and are reinforced throughout the training. 54

5.4 Mean learning curve with 95 % confidence interval for TSB (red) and the baselines RL (blue) and RRL (green). TSB outperforms both baselines by a large margin. 58

5.5 The position of the learned bias in blue and the target position in red are offset. In relation to the target, the bias lies opposite to most initial end-effector positions. 58

5.6 The mean of the learned bias distribution and the position of the target along the indicated axis. While at the end of training, the bias tracks the y and z coordinates closely, bias and target are offset along x. 59

5.7 We compare different bias distribution initializations to assess the robustness of the method. The figures show the center and two times the standard deviation of the initial bias distribution. 60

5.8 Mean learning curves and 95 % confidence interval for different bias initialization. The right plot shows the same data but focuses on the first 300 thousand training steps. All initializations perform on par. 60

List of Tables

4.1 Planar reaching: success rates. 46

4.2 Planar reaching: average performance. 47

4.3 Planar reaching: root mean squared action norm. 47

4.4 TIAGo reaching: success rates. 48

4.5 TIAGo reaching: average performance. 48

4.6 TIAGo reaching: root mean squared action norm. 49

A.1 Planar reaching: average performance (cumulative, discounted reward) with 95 % confidence interval for different base standard deviations (std) and null space reductions. 69

A.2 Planar reaching: success rates for different base standard deviations (std) and null space reductions. 69

A.3 Planar reaching: root mean squared action norm and 95 % confidence interval for different base standard deviations (std) and null space reductions. 70

A.4 TIAGo reaching: average performance (cumulative, discounted reward) with 95 % confidence interval for different base standard deviations (std) and null space reductions. 70

A.5 TIAGo reaching: success rates for different base standard deviations (std) and null space reductions. 71

A.6 TIAGo reaching: root mean squared action norm and 95 % confidence interval for different base standard deviations (std) and null space reductions. 71

A.7 Planar reaching Environment specifications. 72

A.8 Planar reaching hyperparameters. 73

A.9 TIAGo reaching environment specifications. 74

A.10 TIAGo reaching hyperparameters. 75

A.11 TIAGo shelf environment specifications. 76

A.12 TIAGo shelf environment hyperparameters. 77