Curriculum complexity: a framework for computationally efficient reinforcement learning control of soft robotic agents

a dissertation presented
by
Ana Vitoria Rodrigues Lima
to
The Department of John Paulson School of Engineering

in partial fulfillment of the requirements
for the degree of
Master of Engineering
in the subject of
Computational Science

Harvard University
Cambridge, Massachusetts
May 2023

Thesis advisor: Professor Prof. Daniel Bruder Ana Vitoria Rodrigues Lima

Curriculum complexity: a framework for computationally efficient reinforcement learning control of soft robotic agents

Abstract

In this thesis work the end objective has been to find a method to lower the computational expense for controlling soft robotic agents. Previous literature has already tackled the control question for robotic arms and rigid agents of this kind; the control of a robotic arm is a well researched question with already solutions ranging from control theory to the realm of machine learning with reinforcement learning of any kind, both online and off-line.

To our knowledge, there hasn't however been however any attempt to study in depth the control of a soft robotic agent - especially in the machine learning spectrum. With this work, we hope to present an on-line reinforcement learning approach that not only can control a soft agent, but that especially is able to cut the computational cost and computational burden necessary to control soft agents.

Contents

This thesis is dedicated to my momma Gloria and grandma Creuza - the two women that did the impossible to make it possible for me to be here today.

Olha mãe, eu não deixei a peteca cair!

Acknowledgments

There are many people to acknowledge here, but I will start from the ERP Scholarship, without which this year would not have been possible to begin with. I kindly thank hereby the Studienstiftung for having seen my path and sustained me financially this year. Then I also want to thank the Robert Wood Lab for having welcomed me, despite my completely different academic background. Hereby I also have to acknowledge my very supportive partner Caleb and my cat - who always made sure I would be taking care of myself, even in the latest coding nights.

1 Introduction

1.1 Motivation

How can a soft agent or soft robotic arm be effectively controlled? Such question poses not only control challenges but especially computational challenges. A soft robotic arm exhibits infinite degrees of freedom (DOF) due to its fundamentally different design and construction compared to traditional rigid robots. While rigid robots typically have a fixed number of joints and links, leading to a finite and discrete set of DOF, soft robotic arms are made of compliant and deformable materials that enable continuous bending, twisting, and elongation in multiple directions (Majidi, 2014). This ability to change shape and adapt to various poses is a result of the inherent compliance in their structure, which allows the arm to interact with its environment in a more fluid and versatile manner (Alspach et al., 2018).

The development of soft robotic arms with infinite DOF presents both opportunities and challenges for control and actuation. Researchers are exploring various strategies to manage the complex dynamics of soft robots, such as using distributed sensors and actuators embedded within the soft structure, or developing novel control algorithms that take into account the continuous and nonlinear behavior of these systems (Calisti et al. (2011), Marchese et al. (2014)). However, it is challenging to model such an infinite DOF system so that to simulate more complex, useful and non-trivial behaviours.

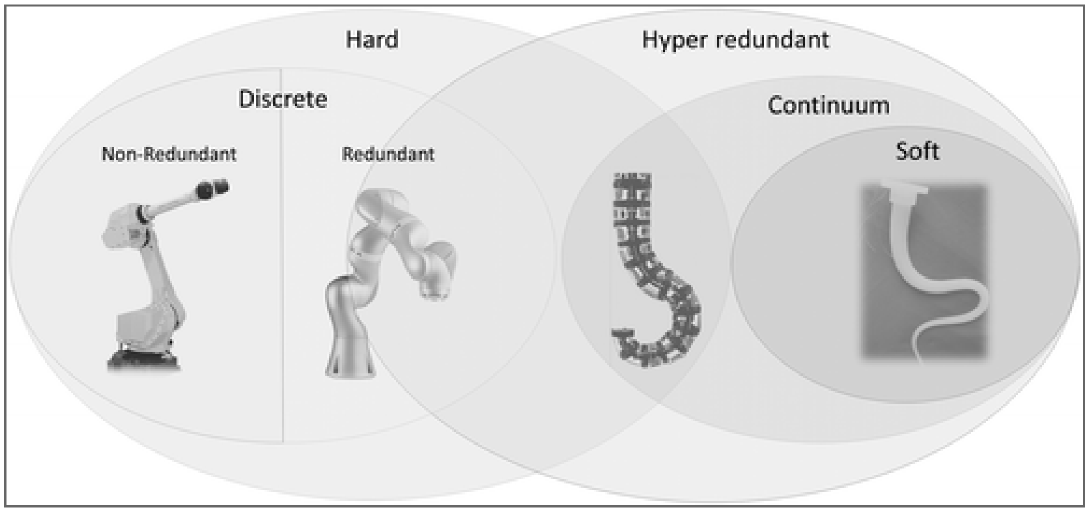

Figure 1.1: Illustration of a hard robotic arm and of a soft robotic arm. Image from George Thuruthel et al. (2018) .

A solution to this, is to discretize such infinite DOF system to a finite DOF system. In this manner, a higher amount of finite DOF corresponds to a more accurate representation of the agent, where a lower amount of finite DOF corresponds to a lower fidelity representation of the agent.

Training a reinforcement learning policy on a discretized system with a finite number of DOF is now a possible task. It is however significantly computationally expensive to do so in a high fidelity, high amount of DOF, environment.

1.2 Contribution

In this work, we want to leverage how more computationally cheaper it is to train a reinforcement learning policy on a low-fidelity system, i.e. a system with a smaller amount of finite DOF. To leverage this successfully, we want to re-use this gained knowledge gradually with more fidelity. To do this, we will borrow from the concept of curriculum learning, where we will be training on a gradually more complex environment, with the goal of achieving the same behaviour but with a fraction of the training time and a fraction of the training resources.

With this work, we hope to present an on-line reinforcement learning approach that not only can control a soft agent, but that especially is able to cut the computational cost and computational burden necessary to control soft agents.

In summary this work presents the following contributions:

- The structure of the code for the repository Complexity Curriculum, a framework where SoMoGym environments can be trained with Reinforcement Policies in a more computationally efficient manner with the usage of curriculum reinforcement learning.
- In the process of this work, documentation of the SoMoGym has been also developed and documented online in the Website of the MicroRotics Lab in this link https://www.micro.seas.harvard.edu/software .
- As part of the documentation process, demo code in Colab for this has been released as well,

so that a wider audience and more robotics labs in the field can have an easier access to utilize the SoMoGym environments for their experiments and training purposes.

Soft robots have a hard life, they have so much to learn.

Vitoria Lima

2 Background and Prior Work

2.1 Soft Robotics

Soft robotics is a rapidly growing field that focuses on the development of robots composed of flexible, compliant, and often biologically-inspired materials. This emerging discipline aims to address some of the limitations of traditional rigid robots, offering potential benefits such as increased safety, adaptability, and improved interaction with humans and the environment (Majidi, 2014). Soft robotic systems possess inherent compliance, which allows them to conform to complex shapes and interact with delicate objects without causing damage (Rus and Tolley, 2015). Furthermore, soft robots can adapt their shape and behavior to suit a wide variety of tasks, making them suitable for applications ranging from medical assistance to disaster relief (Runciman et al., 2019).

Despite these promising advantages, soft robotics also presents significant challenges, particularly in modeling and control (Santina et al., 2021). Due to their flexible nature, soft robots exhibit infinite-dimensional kinematics and nonlinear dynamics. Traditional rigid-body models and control techniques are often inadequate for accurately describing and manipulating soft robotic systems, necessitating the development of novel approaches to model and control these complex machines. One such approach is the use of data-driven policy learning, which leverages large amounts of data to learn appropriate control strategies for soft robots.

Data-driven approaches to policy learning offer a promising solution to the challenges posed by soft robotics, as they can circumvent the difficulties associated with modeling and controlling these systems analytically (Polydoros and Nalpantidis (2016), Bruder et al. (2021), Castaño et al. (2020)). By learning from data, these methods can capture the intricacies of soft robotic behavior without relying on precise mathematical models of their underlying physics. This allows for the development of effective control policies that can adapt to various situations and tasks, making them well-suited to the inherently flexible and adaptable nature of soft robots.

Soft robotics represents a cutting-edge field with significant potential for improving safety, adaptability, and human-robot interaction. However, the challenges associated with modeling and controlling these systems necessitate the development of new approaches, such as data-driven policy learning. By leveraging the power of data, researchers can overcome the complexities inherent in soft robotic systems, ultimately enabling the development of more effective and versatile robotic solutions.

2.2 Reinforcement Learning - General Background

Reinforcement learning (RL) is a subset of machine learning that focuses on training algorithms to make decisions in dynamic environments. These algorithms learn optimal behaviors through trial and error, seeking to maximize cumulative rewards or minimize cumulative costs over time (Sutton and Barto, 2018).

Fundamentally, Reinforcement Learning is about how a given agent, or multiple agents (Multi-Agent RL) interacts with a surrounding environment - whereby this agent receives a response to its action, which can be a reward or a penalty. This reward, or lack thereof, is defined by a reward function, which outpus a reward per state of the given environment. In response to this, the agent generates a policy. A policy maps the observed state to an action. In deep RL, the agent can be a neural network, which is trained to maximise this reward function. After the training stage, there is the execution stage, or better defined as *rollout* stage, i.e. when this policy is executed, or rolled out. An RL agent is considered successful when its trained policy succeeds on the rollout.

Reinforcement Learning differs itself from the rest of the machine learning field primarily by how data is collected. In machine learning, usually there is a training data set provided. This can be fully considered, like in supervised ML, or not, like in weakly supervised machine learning - but overall, a dataset is provided, albeit incomplete or not large enough at times. On the other hand, in RL the dataset is collected through interactions, discovery, and exploration through and with the environment by the agent itself.

The overall field of RL can also be seen from the perspective of a persistent trade-off between exploration and exploitation. In reinforcement learning, the trade-off between exploration and exploitation refers to the dilemma an agent faces when deciding whether to explore unknown states (exploration) or maximize rewards by choosing known best actions (exploitation). Exploration allows the agent to discover new knowledge, while exploitation leverages the knowledge gained so far to make optimal decisions. Balancing the two strategies is critical for an RL agent to achieve long-term success in its learning process.

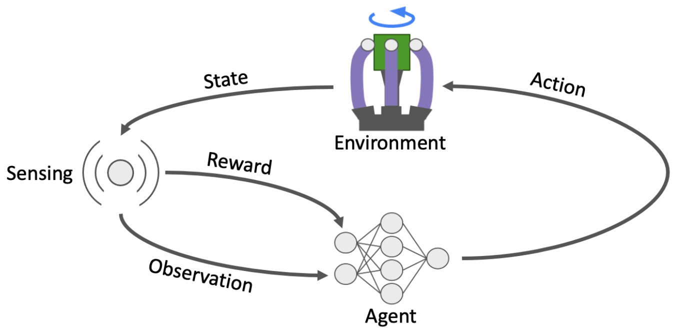

Figure 2.1: Illustration of a basic reinforcement learning setup. The agent observes a state in the environment, this is perceived by a sensor which inputs an observation and a reward to the RL agent. The agent then takes an action which is inputed in to the control inputs of the environment.

The challenge in RL is to balance exploration and exploitation to ensure that the agent learns an optimal policy (i.e., a sequence of actions that maximizes cumulative rewards). If the agent focuses too much on exploration, it might miss out on opportunities to collect high rewards from known optimal actions. However, if it solely exploits its current knowledge, it may never discover potentially better actions and become trapped in a suboptimal policy.

On a broader perspective, Reinforcement Learning can also be divided into several categories, including online and offline approaches, model-based and model-free approaches, on-policy and off-policy approaches.

Online RL

Online reinforcement learning algorithms actively interact with an environment and update their knowledge in real-time. This means the algorithm learns from each new experience, making adjustments to its decision-making process as it goes along. This approach can be beneficial in situations where the environment is continuously changing or when it is difficult to obtain large amounts of training data beforehand.

Offline RL

Offline reinforcement learning, also known as batch reinforcement learning or data-driven reinforcement learning, involves learning from a pre-existing dataset of interactions with the environment (Lange et al., 2012). This method is particularly useful when the costs of online learning are prohibitive, such as in high-stakes scenarios where mistakes can be costly, or when there is limited access to the environment for live training.

Model-Based RL

Model-based reinforcement learning approaches involve learning a model of the environment, which is then used to plan and make decisions (Moerland et al., 2022). By estimating the dynamics of the environment, the algorithm can predict the outcomes of different actions and use this information to optimize its decision-making process. Model-based approaches are often more sample-efficient than model-free approaches, as they can leverage prior knowledge of the environment to learn faster.

Model-Free RL

In contrast to model-based approaches, model-free reinforcement learning algorithms do not rely on an explicit model of the environment. Instead, they learn optimal behaviors directly from experience, using methods such as value function estimation or policy optimization (Sutton and Barto, 2018). Model-free approaches can be more computationally efficient and easier to implement, but they may require more data to learn effectively compared to model-based approaches.

On-Policy RL

On-policy reinforcement learning algorithms learn an optimal policy while following the same policy they are trying to optimize. In other words, the algorithm simultaneously explores the environment and exploits the knowledge it has gained so far, updating its policy based on its current experiences (Sutton and Barto, 2018). An example of an on-policy algorithm is the SARSA (State-Action-Reward-State-Action) method, which learns a value function for the current policy by updating its value estimates based on the actions it actually takes (Rummery and Niranjan, 1994). On-policy methods are often straightforward to implement and can provide strong guarantees of convergence. However, they can sometimes be less efficient than off-policy methods, as they may need to explore suboptimal actions to learn the optimal policy.

Off-Policy RL

Off-policy reinforcement learning algorithms, on the other hand, learn an optimal policy while following a different policy for exploration, known as the behavior policy. This separation allows the algorithm to learn about the optimal policy independently of the actions it takes during exploration (Sutton and Barto, 2018). A well-known example of an off-policy algorithm is Q-learning, which learns the value function for the optimal policy while following a different policy for exploration (Watkins and Dayan, 1992). Off-policy methods can be more sample-efficient than on-policy methods, as they can leverage past experiences from multiple behavior policies to learn the optimal policy. However, they can also be more challenging to implement and may require careful tuning of the exploration strategy to ensure sufficient learning.

2.3 Reinforcement Learning - Technical Overview

With this general overview in mind, let us now briefly give a more technical and concrete overview of RL. RL problems can often be modeled as a Markov Decision Process (MDP). An MDP is a tuple (S, A, P, R, γ), where: S is a finite set of states, A is a finite set of actions; $P(s'|s, a)$ is the state transition probability, i.e., the probability of transitioning from state s to state s' when taking action a; $R(s, a, s')$ is the reward function, i.e., the immediate reward received when transitioning from state s to state s' by taking action a; γ is the discount factor, which determines how much future rewards are worth compared to immediate rewards $(0 < \gamma < 1)$.

It is important to note that an MDP has some strong assumptions such as: *markov property*, it states that the future state of the system depends only on the current state and action, and not on the history of previous states and actions; *fully observable environment*, MDPs assume that the agent can fully observe the current state of the environment - this means that the agent has complete knowledge of the current situation and can make decisions based on this information; *stationary transition probabilities*, in an MDP, the state transition probabilities are assumed to be stationary, meaning they do not change over time - this assumption ensures that the dynamics of the environment are consistent, allowing the agent to learn a policy effectively. An extension to MDP is a Partially Observable Markov Decision Process (POMDP), which used to model decision-making problems in environments with partial observability. In POMDPs, the agent cannot directly observe the underlying state of the environment; instead, it receives observations that provide partial or noisy information about the state.

Nonetheless, MDPs capture the necessary elements of RL problems: states, actions, state transitions, rewards, and the objective of maximizing cumulative reward. Moreover, the Markov property ensures that the future state depends only on the current state and action, not on the history of past states and actions (Puterman, 1994). Given this, RL problems can often be modeled as a Markov

Decision Process (MDP), where for every timestep t and state s_t (noting that, $s_t \in S$) the agent receives a state observation s_t and a reward $r_t = r(s_t)$. Based on this, the agent chooses an action a_t, which means it is taking a step to the state s_{t+1}, which generates the reward r_{t+1}. In the training process, the agent will develop a policy π, which formally can be represented as a probability distribution over the action space given the current state:

$$\pi(a|s) = Pr(a_t = a, s_t = s)$$

There are two main types of policies in RL: deterministic and stochastic policies. A deterministic policy selects a single action for each state, while a stochastic policy selects actions probabilistically.

The choice of policy can have a significant impact on the learning process and the resulting behavior of the agent. For example, a stochastic policy may allow the agent to explore more of the environment, while a deterministic policy may be more efficient in exploiting known good actions. Overall in Reinforcement Learning, the goal is to learn a policy that maps states to actions in order to maximize the cumulative reward:

$$R_\pi = \sum_{t=0}^{T} \gamma^t r(s_t),$$

Two important concepts used in this context are the *action-value function* and the *state-value function*. Both functions estimate the expected future reward, but they differ in how they represent the information.

Action-value function (Q-function)

The action-value function, denoted as $Q(s, a)$, represents the expected return (cumulative future reward) when taking an action a in state s and following a given policy π thereafter. It is defined as:

$$Q_\pi(s, a) = E_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t = s, a_t = a]$$

$$Q_\pi(s, a) = E_\pi[R_\pi|s_t = s, a_t = a]$$

where: $E_\pi[.]$ represents the expected value given policy π; $R_{t+1}, R_{t+2}, ...$ are the rewards received at time steps $t + 1, t + 2, ...$; γ is the discount factor $(0 < \gamma < 1)$ which controls the importance of future rewards; s_t and a_t are the state and action at time step t, respectively.

The optimal action-value function, $Q * (s, a)$, is the maximum action-value function over all policies:

$$Q * (s, a) = max\pi Q\pi(s, a)$$

State-value function (V-function)

The state-value function, denoted as $V(s)$, represents the expected return (cumulative future reward) when starting in state s and following a given policy π thereafter. It is defined as:

$$V\pi(s) = E\pi[r_t + 1 + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...|s_t = s]$$

$$V\pi(s) = E\pi[R_\pi|s_t = s]$$

Notice that the state-value function does not consider specific actions, only states. The two functions are related as follows:

$$V\pi(s) = \Sigma_a \pi(a|s) * Q\pi(s, a)$$

where $\pi(a|s)$ is the probability of taking action a in state s under policy π. The optimal state-value function, $V*(s)$, is the maximum state-value function over all policies:

$$V*(s) = max\pi V\pi(s)$$

In summary, the action-value function (Q-function) represents the expected return for a specific action in a specific state, while the state-value function (V-function) represents the expected return for a specific state without considering the actions explicitly. Both functions play important roles in Reinforcement Learning algorithms, such as Q-learning and SARSA for the action-value function, and Value Iteration and Policy Iteration for the state-value function.

2.3.1 Reinforcement learning control in robotics

The field of Reinforcement Learning applied to robotic control has seen substantial progress in the last few decades. From the early foundations laid by Barto et al. (1983) to the latest advancements in model-free and model-based RL algorithms, researchers have developed numerous techniques for enabling robots to learn control policies autonomously. These approaches have demonstrated impressive results in various robotic control tasks, including manipulation, locomotion, and multi-robot systems.

In recent years, there has been significant progress in developing model-free RL algorithms for robotic control. Some notable advancements include: Deep Deterministic Policy Gradient (DDPG): Lillicrap et al. (2019) combined deep neural networks with the deterministic policy gradient algorithm to develop DDPG, enabling the use of RL for continuous control tasks. This approach has shown success in various robotic control tasks, such as manipulation and locomotion.

Furthermore, Proximal Policy Optimization (PPO), by Schulman et al. (2017b) proposed the PPO algorithm, which simplifies and improves the efficiency of trust region policy optimization. PPO has been widely adopted in robotic control applications, such as robotic arms and legged robots.

Model-based RL has also seen significant advancements in recent years. By leveraging learned models of the environment, these algorithms can potentially reduce sample complexity and improve data efficiency. Some notable works include: Chua et al. (2018) proposed the Model-Based Meta-Policy Optimization (MB-MPO) algorithm, which combines model-based RL with meta-learning. MB-MPO can rapidly adapt to new tasks, making it suitable for robotic control in changing environments. Nagabandi et al. (2017) introduced Neural Network Dynamics Models (NN-DM), a model-based RL approach that utilizes neural networks to learn dynamics models. NN-DM has demonstrated impressive results on various robotic control tasks, including manipulation and legged locomotion.

There has been a growing interest in applying RL to control multi-robot systems, enabling robots to cooperate and achieve complex tasks. Some recent works in this area include: Foerster et al. (2016) introduced the Counterfactual Multi-Agent Policy Gradients (COMA) algorithm, which enables learning of cooperative policies for multi-agent systems. This approach has shown potential in controlling multiple robots to achieve collaborative tasks. Lowe et al. (2017) proposed the Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (MADDPG) algorithm. This approach allows the learning of multi-agent policies, suitable for controlling robotic swarms in various tasks, such as exploration and search and rescue.

As the field continues to evolve, it is anticipated that further improvements in RL algorithms and their application to robotic control will be made. Key areas of ongoing research include improving data efficiency, transfer learning, and handling uncertainty in real-world environments. Ultimately, these advancements will lead to more capable, adaptable, and autonomous robots that can assist and collaborate with humans in a wide range of tasks.

2.3.2 Reinforcement learning control in soft robotics

The application of Reinforcement Learning (RL) to soft robotic control has garnered significant interest due to its potential to enable these robots to learn complex control policies autonomously. The application of RL to soft robotic control has its roots in the broader field of RL for robotic control. The foundational works on RL for robotic control, such as those by Barto et al. (1983) and Sutton et al. (1999b), laid the groundwork for applying RL to soft robotics as well. However, soft robotic control presents unique challenges, such as handling the high degree of freedom and the nonlinear, deformable nature of soft robots.

Some early work that first applied a data-driven machine learning methods to soft robotic control is George Thuruthel et al. (2017). This paper proposes a machine learning approach to develop dynamic models and trajectory optimization for a soft robotic manipulator and a trajectory optimization method for predictive control. It uses a model-free feedback controller and a neural network feedforward component to compensate for dynamic uncertainties. A direct collocation approach is used to identify optimal generalized torques and manipulator states. The simulation of the soft agent is done by discretizing the infinite DOF to 6 DOF. to It is the first demonstration of a learned dynamic model and task space controller for a soft robotic agent.

Previously, Gupta et al. (2017) explored imitation reinforcement learning to control a soft robotic hand from human demonstrations. More recently, Satheeshbabu et al. (2019) presented a model-free approach for controlling a soft spatial continuum arm using deep reinforcement learning. The authors used Deep-Q Learning to train the system in simulation and validated its efficacy and robustness on a continuum arm prototype. These however have not explored complex tasks involving contact with the environment, which is what environments and simulations with SomoGym (Graule et al., 2022) will investigate in this report.

Your past doesn't define you, it prepares you.

An algorithm that did not overfit.

3 Methods

3.1 SomoGym

Reinforcement learning training is typically known to need a simulation framework or gym, typical environments used include Pybullet from Ellenberger (2018–2019), the known OpenAI Gym from Brockman et al. (2016) and MuoJoCo from Todorov et al. (2012).

However, these frameworks are not suited to simulate nor train a soft robotic agent. SoMoGym (Graule et al., 2022) tackled this. SoMoGym is a software toolkit designed to facilitate the development and evaluation of controllers and reinforcement learning algorithms for soft agents specifically. As mentioned throughout this thesis, soft robots offer benefits over traditional rigid robots, but their compliance makes it difficult to use model-based methods in planning tasks requiring high precision or complex actuation sequences. SoMoGym provides a set of benchmark tasks in which soft robots interact with various objects and environments, allowing for the evaluation of performance on these tasks for controllers of interest and enabling the use of RL to generate new controllers, which results in facilitating effortless evaluation of control RL policies and learning frameworks.

Built using Python3, SoMoGym's environments inherit from OpenAI Gym's 'Env' class, making its structure easily comprehensible within the reinforcement learning community due to the standardized format of OpenAI Gym environments. It provides a collection of simulated environments in which soft robots execute locomotion, reaching, or manipulation tasks. The framework emphasizes easily configurable environments that support interaction with objects, allowing for experiments on the effects of varying control and robot design parameters.

Upon initialization, a SoMoGym environment uses the provided benchmark run configuration as its default. However, to initialize a custom SoMoGym environment, users can supply a run configuration dictionary containing custom general simulation parameters (such as PyBullet time step, controller update rate, maximum actuator torque, etc.) and task-specific parameters (like reward components and weights, observation components, object properties, etc.). The configuration file is in yaml format. It is important to note that in this configuration file also the amount of *segments* per actuator can be modified: this allows for determining how grained the discretization of the system will be, where a higher amount of segments indicate higher fidelity, and vice versa. Also, SoMoGym offers various observation and reward functions that can be easily selected, combined, or modified.

Internally, SoMoGym depends on the SoMo (Graule et al., 2021) framework for representing and controlling soft robots. Consequently, each continuum robot is described using a human-readable configuration file, which simplifies altering the quantity, positions, and properties of robots and other objects in each SoMoGym environment.

3.1.1 Environments

The environments that have been used for this thesis work have been the Planar Reaching environment and the Block Pushing environment.

Planar Reaching

Planar Reaching requires a 10cm continuum robot to move its tip to a target position chosen at random uniformly from its workspace. The robot consists of four serial actuators with one actively controlled bending axis per actuator. The robot is limited to deformations within the yz-plane. Gravity points in the negative z-direction. The reward is inversely proportional to the distance between the tip of the robot and the target position. A new target position is chosen at the beginning of every episode.

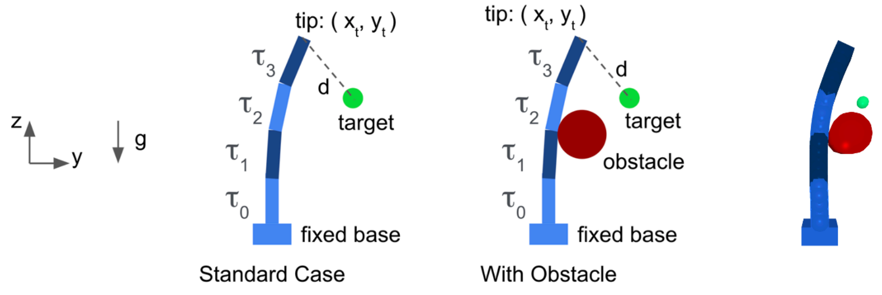

Figure 3.1: In *Planar Reaching*, a planar continuum manipulator must move its tip to a randomly chosen target position. The manipulator has four independently controlled actuators with torques τ_i for $i \in [0, 1, 2, 3]$. In *Planar Reaching with Obstacle*, the same manipulator must move its tip to a target position in the presence of a cylindrical obstacle.

Observations: target position (2 dimensions), 20 2D backbone positions (40 dimensions), 20 link velocities (40 dimensions), the distance vector between the robot tip and the target (2 dimensions), and the applied input actuator torques (4 dimensions).

Reward function: $R = w_1|d| + w_2\psi_1(d) + w_3\psi_2(d) + w_4\psi_3(d)$, where d is the distance from the manipulator tip to the target position and $\psi_\ell(d)$ (with $\ell \in \{1, 2, 3\}$) is a bonus achieved when the tip is brought into close proximity of the target. When the $d < 2cm$, $\psi_1(d) = 1$, when $d < 1cm$, $\psi_2(d) = 1$, and when $d < 0.5cm$, $\psi_3(d) = 1$. Otherwise $\psi_\ell(d) = 0$. Reward component weights $w_i \; \forall \; i \in [1, 4]$ are defined in a run configuration file.

Success metric: the manipulator tip is within 1cm of the target position.

Planar Block Pushing

A 10cm planar continuum robot must move a block from a fixed initial position to a fixed target position. The robot consists of four serial actuators with one actively controlled bending axis per actuator. The robot is limited to deformations within the xy-plane and must overcome the block's friction with the ground plane. This requires stable grasping and pushing behaviors for the agent to reliably manipulate the block across rollouts.

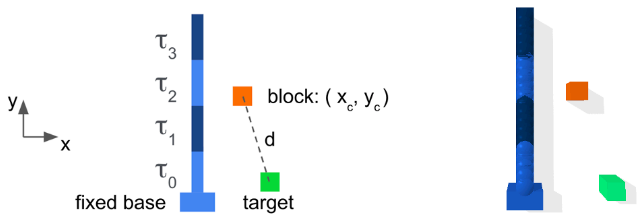

Figure 3.2: In *Planar Block Pushing*, a planar continuum manipulator is tasked to move a cube to a target position. The manipulator has four independently controlled actuators with torques τ_i for $i \in [0, 1, 2, 3]$.

Observations: block position (2 dimensions), block velocity (2 dimensions), block orientation (2 dimensions), the distance vector between the block center and the target (2 dimensions), target position (2 dimensions), 20 2D backbone positions (40 dimensions), 20 link velocities (40 dimensions), the distance vector between the robot tip and the block center (2 dimensions), and the applied input actuator torques (4 dimensions).

Reward function: $R = w_1|d| + w_2|m| + w_3\,\psi_1(d) + w_4\,\psi_2(d) + w_5\,\psi_3(d)$, where d is the distance between the block center and the target position, m is the distance between the block center and the robot tip, and $\psi_\ell(d)$ (with $\ell \in \{1, 2, 3\}$) is a bonus achieved when the block is brought into close proximity of the target. When the $d < 2cm$, $\psi_1(d) = 1$, when $d < 1cm$, $\psi_2(d) = 1$, and when $d < 0.5cm$, $\psi_3(d) = 1$. Otherwise $\psi_\ell(d) = 0$. Reward component weights $w_i \; \forall\, i \in [1, 5]$ are defined in a run configuration file.

Success metric: the block center is within 1cm of the target position.

3.2 Curriculum learning

3.2.1 Curriculum Learning Overview

Curriculum Reinforcement Learning (CRL) is a learning framework that trains an agent on a sequence of increasingly complex tasks, enabling the agent to learn more effectively by leveraging prior knowledge and building on simpler skills (Bengio et al. (2009); Graves et al. (2017)).

CRL is inspired by the way humans and animals learn complex tasks by gradually building on simpler skills and knowledge (Bengio et al., 2009). The key idea behind CRL is to construct a sequence of tasks with increasing difficulty, allowing the agent to learn more efficiently by leveraging previously learned skills. This approach can lead to faster convergence, improved generalization, and better scalability to complex environments (Graves et al. (2017); Narvekar et al. (2020)).

The essential components of CRL are the task curriculum and the learning agent. The task curriculum is a sequence of tasks $T = T_1, T_2, ..., T_n$ with increasing difficulty, where T_i denotes the i-th task. The difficulty of a task can be quantified by various measures, such as the size of the state space, the number of actions, or the complexity of the optimal policy (Narvekar et al., 2020).

Designing an effective curriculum is crucial for the success of CRL. There are two primary approaches to curriculum design: manual and automatic (Narvekar et al., 2020).

Manual Curriculum Design

Manual curriculum design relies on domain knowledge and expert intuition to construct a sequence of tasks. For example, in robot locomotion, a curriculum may start with simple walking tasks before progressing to more complex scenarios, such as obstacle avoidance and navigation. While manual design can be effective in some cases, it can be time-consuming and may not generalize well to other domains.

Automatic Curriculum Design

Automatic curriculum design methods aim to generate curricula without relying on expert knowledge. These methods can be broadly categorized into teacher-student methods, adaptive methods, and self-organized methods.

Teacher-student methods involve a separate teacher agent that designs the curriculum for the learning agent. One notable example is (Matiisen et al., 2017), where they propose a framework called Teacher-Student Curriculum Learning (TSCL) for automatic curriculum learning. The framework uses a family of Teacher algorithms that choose subtasks based on the Student's learning progress and address the problem of forgetting.

Adaptive methods adjust the curriculum based on the learning agent's performance. For instance, Florensa et al. (2018) proposed the use of a goal-reaching probability as a measure of task difficulty, allowing the curriculum to adapt to the agent's performance.

Recent advancements in CRL have led to improvements in various domains, including robotics, and computer vision. In robotics, CRL has been used to teach robots complex manipulation tasks (Kilinc and Montana, 2021), locomotion skills (Heess et al., 2017) and multi-agent collaboration (Baker et al., 2020). In computer vision, CRL has been utilized to improve object recognition (Kumar et al., 2010) and to improve training sustainability of vision transformers through progressive learning (Li et al., 2022).

3.2.2 Curriculum Learning in Soft Robotics

With this overview in mind, we can now dive into how this thesis aims at leveraging Curriculum Learning to aid learning for soft robotic agents.

Curriculum Learning aims at gradually increasing the difficulty of tasks the agent and algorithm is being trained on. This can both increase accuracy but also reduce training time. In the context of soft robotic agents, we can abstract the difficulty of the task to be the granularity of the soft robotic agent simulation.

As explained above, for a soft robotic agent to be simulated, its infinite DOF need to be discretized. Where the discretization is very high, high DOF, the simultion will have high fidelity and will be very accurate. Where the discretization is lower, low DOF, the simulation will have lower fidelity and will be less representative of a real soft agent. In SomoGym a high discretization corresponds to a higher number of segments per actuators, which can be easily specified in the configuration file of the desired environment. As the simulation in SomoGym is made to be more accurate (Figure 3.3), representative and have higher fidelity to reality, the training and execution time increase accordingly. For instance, despite having 20 segments per actuator represent more plausibly what a soft agent looks like, this will take more time to train than utilising 2 segments per actuator.

In this work we want however to leverage also the learned behaviour in low fidelity. We believe that the learned behaviour with lower fidelity can still be a meaningfully learned behaviour, despite

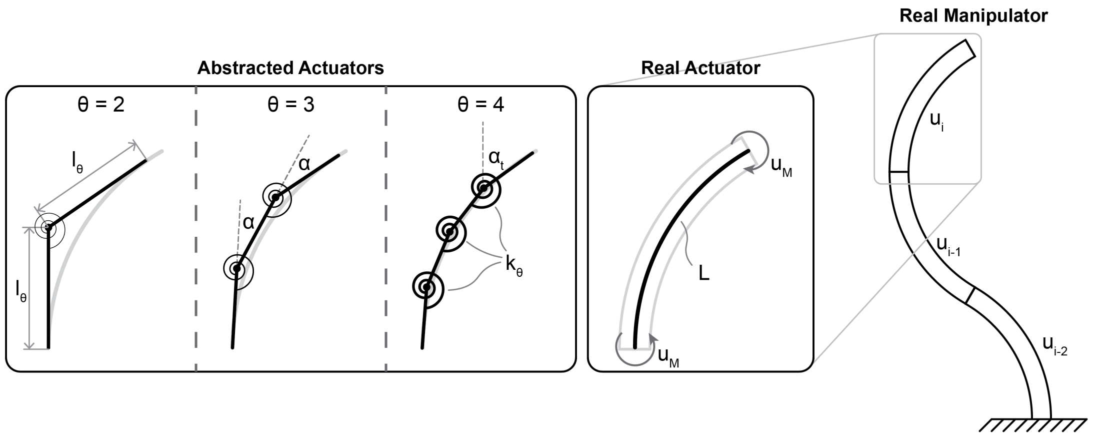

Figure 3.3: The discretization happens on an actuator level, where there is a fixed number of actuators and per actuator the number of segments within each actuator varies in an increasing fashion (Graule et al., 2023)

the goal is a learned behaviour with high fidelity. Given this, we leverage the concept of curriculum learning not to the difficulty ot the task, but to the amount of segments per actuators, i.e. to the fidelity of our simulation. This will leverage the behaviour learned with less segments, for instance 2, but will gradually refine this behaviour as the amount of segments increases, in a curriculum fashion.

3.3 PPO: Proximal Policy Optimization

As mentioned in 3.2.2, we aim to apply curriculum learning to the amount of segments per actuator. To do this, we have picked a SOTA learning algorithm which will train our policy and agent. Across these experiments, the algorithm of our choice has been PPO.

3.3.1 General Overview

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that aims to optimize policies for environments with continuous action spaces. PPO is a model-free, on-policy algorithm that uses a trust region approach to update policies, making it more stable and efficient than previous algorithms such as REINFORCE (Sutton et al., 1999a) and TRPO (Schulman et al., 2017a).

In a nutshell, PPO works by iteratively collecting data through interactions with the environment, updating the policy to improve its performance, and then repeating this process until convergence is reached. The algorithm uses a surrogate objective function that penalizes large changes to the policy, which helps prevent the policy from diverging too far from its previous values.

The PPO algorithm is comprised of two main components: the policy network and the value network. The policy network is responsible for selecting actions given the current state of the environment, while the value network is responsible for estimating the expected return for a given state. The policy and value networks are updated using a variant of stochastic gradient descent that minimizes the surrogate objective function.

PPO has been shown to outperform previous state-of-the-art algorithms on a variety of benchmarks and has been successfully applied to a wide range of tasks, including robotic control and game playing.

3.3.2 Technical Overview

Proximal Policy Optimization (PPO) is a popular reinforcement learning (RL) algorithm developed by OpenAI (Schulman et al., 2017b). It is a policy gradient method that aims to combine the benefits of both trust region policy optimization (TRPO) and vanilla policy gradient methods. PPO is designed to simplify the implementation and hyperparameter tuning process of RL algorithms, while maintaining strong empirical performance. It does so by making a few targeted modifications to the standard policy gradient formulation, allowing for efficient and stable learning.

Reinforcement learning (RL) is a type of machine learning in which an agent learns to make decisions by interacting with an environment. The agent learns to choose actions that maximize a cumulative reward signal over time. As explained above, mathematically an RL problem can be modeled as a Markov Decision Process (MDP), defined by a tuple (S, A, P, R, γ), where S is the state space, A is the action space, P is the state-transition probability function, R is the reward function, and γ is the discount factor.

Policy gradient methods are a class of RL algorithms that attempt to directly optimize a parameterized policy $\pi_\theta(a|s)$, where θ denotes the policy parameters, a is an action, and s is a state.

The objective function of PPO is defined as:

$$L_{PPO} = \mathbb{E}\left[min\left(r_t(\theta)\hat{A}_t, clip\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t\right)\right],$$

where:

- θ is the policy parameters
- $r_t(\theta) = \frac{\pi(a_t|s_t)}{\pi_{old}(a_t|s_t)}$ is the probability ratio of the action a_t taken at time step t, comparing the new policy π_θ to the old policy $\pi_{\theta old}$,
- A_t is the advantage function, estimating how much better taking action a_t is compared to the average action value in state s_t,
- $clip(x, a, b)$ is a function that clips x between a and b,
- ε is a hyperparameter controlling the degree of deviation allowed between the new policy and the old policy (typically 0.1 to 0.2).

,

The objective function encourages the algorithm to improve the policy by increasing the probability of actions with high advantage while discouraging the policy from changing too much, preventing large policy updates that could destabilize learning.

PPO is typically implemented using two neural networks, one for the policy $\pi(a|s)$ and another for the value function $V(s)$, which estimates the expected cumulative reward from state s. The algorithm alternates between collecting samples from the environment using the current policy and updating the policy and value function networks using the collected samples.

To summarize, PPO is an efficient and stable RL algorithm that constrains policy updates to ensure smooth learning progress. It balances exploration and exploitation by updating the policy based on an objective function that promotes actions with high advantage while limiting the deviation from the previous policy.

4 Experiments

The experiments performed for this work are broken down in two parts. The first is the baseline training, where no curriculum has been leveraged. In this section, per environment, the agent has been trained with PPO with 2 segments per actuator, then with 5 segments per actuator, then 10 segments per actuator, separately. This has been performed to showcase the difference in training time and to analyse the different learned behaviours from these.

The second parts of experiments consist of leveraging curriculum learning. The goal is to train continuously where in the same training session the training takes place starting from on 2 segments to then gradually achieve 15 segments.

4.1 Baseline experiments

4.1.1 Planar Reaching

In order to substantiate the difference in training across a range of segments the experiments have been performed by having 2 segments per actuator, then 5, then 10 and finally 20 segments per actuator.

This has been performed for 3 seeds (0, 100 and 200) in total, where each seed has been run with 4 threads.

The parameters that characterize this experiment on its configuration file are indicated as follows:

- `action_time: 0.01`
- `bullet_time_step: 0.001`
- `eval_freq: 50000`, which means that at each 50000 steps the policy is evaluated
- training_timesteps: 1 000 000 , meaning that per seed there were 1mio steps of training, and split over 4 threads that is 250 000.

where: `action_time` represents how often a new policy input is applied and is the inverse of the controller update rate; `bullet_time_step` represents how often the PyBullet simulation is updated; `eval_freq` is the number where at each of these steps the policy is evaluated and finally training_timesteps is the training time steps, over all threads, for training the policy.

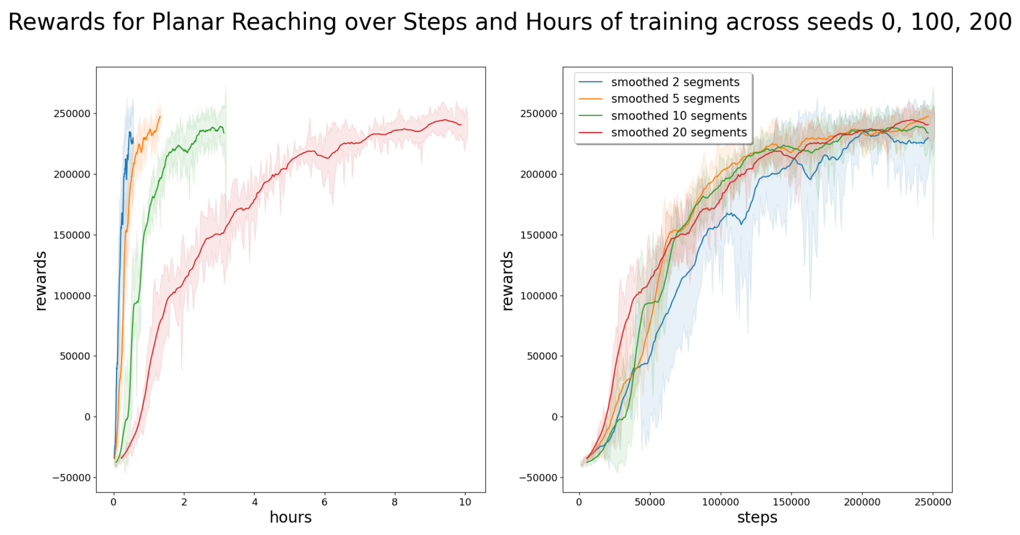

Figure 4.1: Baseline experiment for Planar Reaching with 2, 5, 10 and 20 segments.

By looking at Figure 4.1, we can see that the training of the most discretized agent, with 20 segments per actuator, reaches the same reward as the other simulations, but takes way more time in clock time, despite doing the same amount of training steps.

The training with 2 segments is performed in about half an hour, the training with 5 segments in about 1 hour, the training with 10 segments in about 3 hour, and the training with 20 segments took around 10 hours.

4.1.2 Block pushing

In order to substantiate the difference in training across a range of segments the experiments have been performed by having 2 segments per actuator, then 5, 10, then finally 20 segments per actuator.

This has been performed for 2 seeds (0 and 200) in total, where each seed has been run with 4 threads.

The parameters that characterize this experiment on its configuration file are indicated as follows:

- `action_time: 0.01`
- `bullet_time_step: 0.0002`
- `eval_freq: 50000`, which means that at each 50000 steps the policy is evaluated
- training_timesteps: 10 000 000 , meaning that per seed there were 10mio steps of training, and split over 4 threads that is 2.5mio per thread.

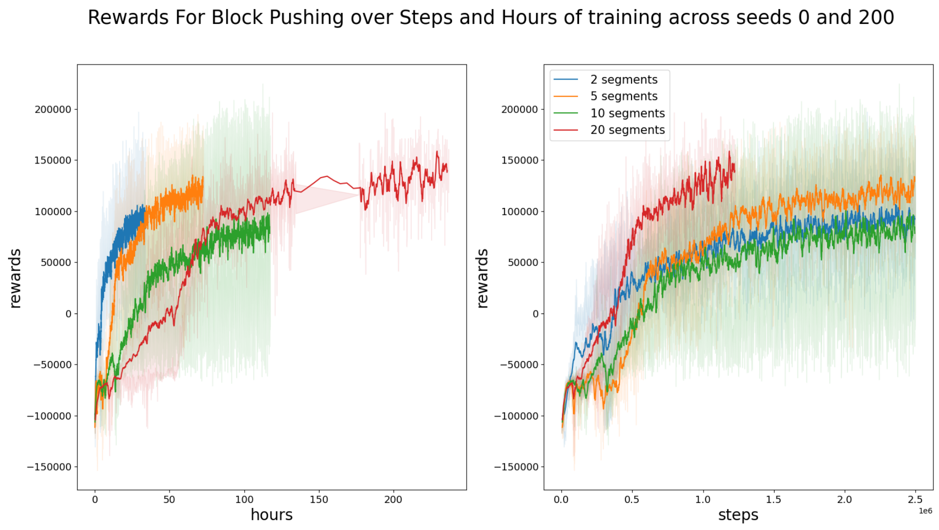

Figure 4.2: Baseline experiment for Block Pushing with 2, 5, and 10 segments.

By looking at Figure 4.2, we can see that the training of the most discretized agent, with 20 segments per actuator, approaches the reward as the other simulations, but takes way more time in clock time, despite doing the same amount of training steps. The training with 2 segments is performed in about 50 hours, the training with 5 segments in about 80 hours, the training with 10 segments took about 130 and the training with 20 segments per actuator took roughly 250 hours.

4.2 Curriculum Complexity

4.2.1 Planar Reaching

For Planar Reaching we have performed curriculum learning over 2, 4, 5, 8, 10 and 15 segments.

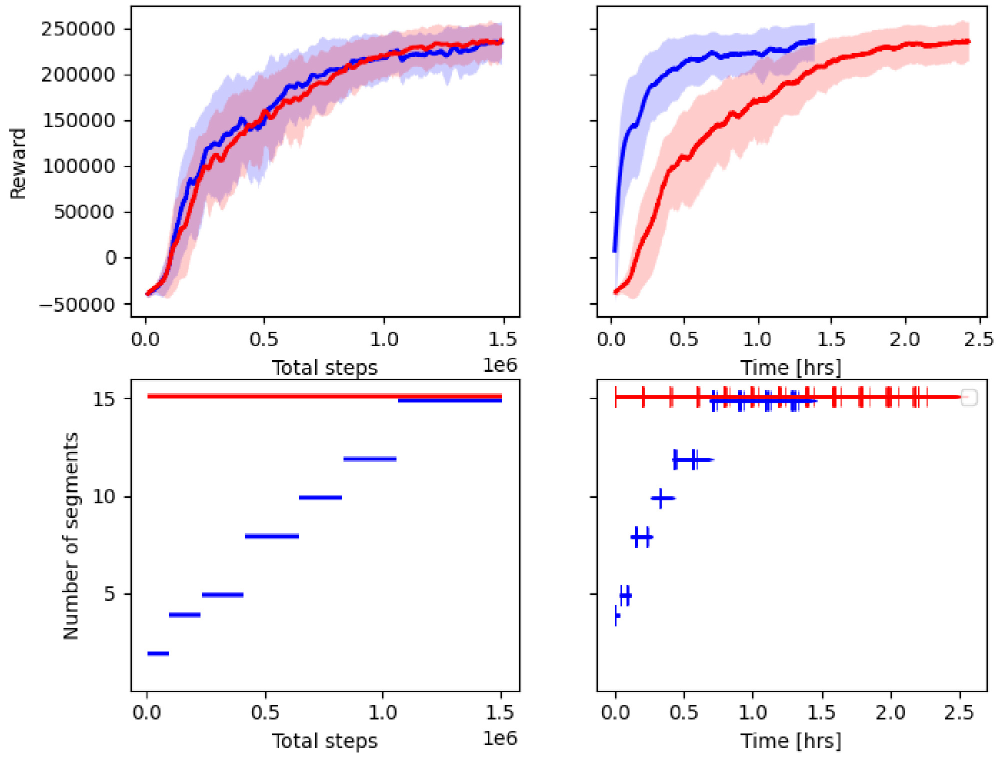

Figure 4.3: Curriculum experiment for Planar Reaching 2, 4, 5, 8, 10, 12, 15 segments.

From Figure 4.3 we can see for plots in total. In the plot of the top left we see on the x axis the number of training steps, and in the y axis the reward amount. From this plot, we can see that the curriculum training is depicted by the line in blue, and the line in red depicts training done with 15 segments for the entire training time. From this plot we can see that both lines converge to the same range of rewards. However, from the plot on the top right, we have on the x axis the training time. From this, we see that the curriculum training, blue line, finishes to train in less time than the red line, despite both converge in training.

This experiments confirms that utilizing curriculum learning for the training of a soft agent can help faster convergence. We can see that to achieve the same reward, the red line takes more time clock-wise, and that the curriculum line converges faster.

4.2.2 Block Pushing

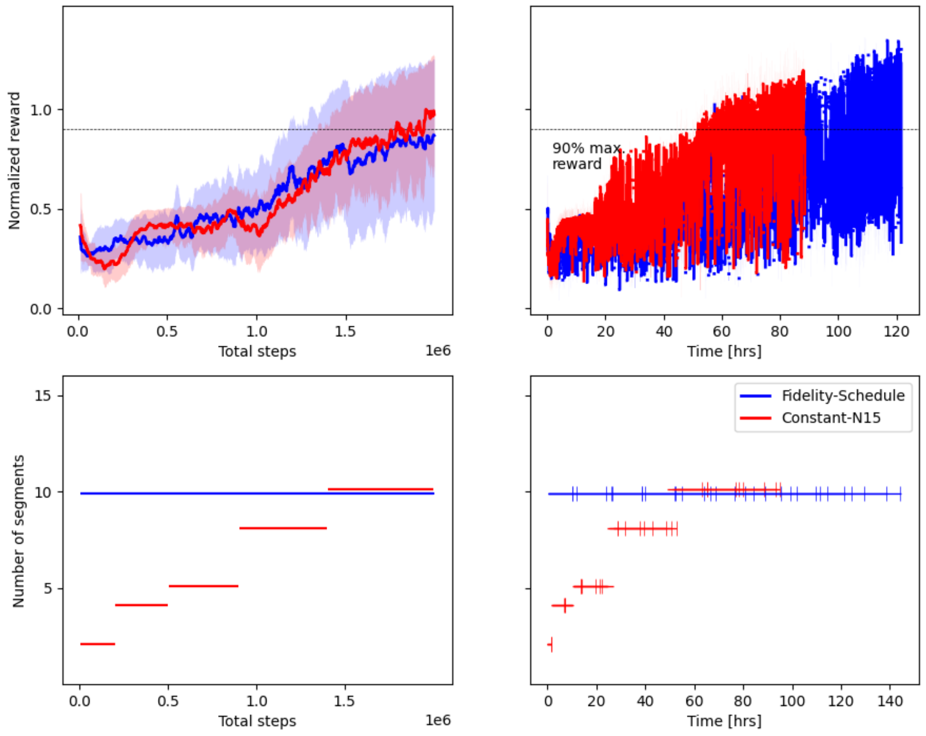

Figure 4.4: Curriculum experiment for Block Pushing 2, 4, 5, 8, 10 segments.

For the Block Pushing environment we have performed curriculum learning over 2, 4, 5, 8 and 10 segments per actuator.

From Figure 4.4 we can see for plots in total. In the plot of the top left we see on the x axis the number of training steps, and in the y axis the reward amount. From this plot, we can see that the curriculum training is depicted by the line in red, whereas the line in blue depicts training done with 10 segments for the entire training time. From this plot we can see that both lines converge to the same range of rewards. However, from the plot on the top right, we have on the x axis the training time. From this, we see that the curriculum training, red line, finishes to train in less time than the blue line, despite both converge in training.

In this experiments, we noticed that there is noticeably more variance in the training process. We think this is the case because of the nature of the task at hand. This task is quite different than simple planar reaching. In block pushing there is a contact and a concrete interaction with the environment, the block. This interaction is what we believe to induce all this variance in the training process, compared to simple planar reaching.

This experiments confirms that utilizing curriculum learning for the training of a soft agent can help faster convergence. We can see that to achieve the same reward, the blue line takes more time clock-wise, and that the curriculum line converges faster.

5 Conclusion

We introduced Curriculum Complexity, a framework for training soft agents through curriculum reinforcement learning. Based on the experiments conducted it can be concluded that curriculum reinforcement learning is a promising approach for faster and more efficient training of soft robotic agents. The experiments demonstrated that by gradually increasing the granularity of an actuator during training, the agents were able to learn more quickly and with higher accuracy. Additionally, the use of curriculum learning allowed for a reduction in computational expenses compared to traditional reinforcement learning methods.

This research has contributed to the field of soft robotics by providing a framework for computationally efficient reinforcement learning control of soft robotic agents. By leveraging curriculum complexity, this framework has shown potential for improving the control and performance of soft robotic devices in real-world scenarios.

Future directions of this work include transfer learning to actual hardware. While the experiments conducted in this dissertation were done in simulation environments, it is important to test these methods on actual hardware to ensure their effectiveness in real-world scenarios. Additionally, further research could explore how this framework could be applied to other types of soft robotic agents beyond those studied in this dissertation.

Overall, this research provides a foundation for future work in improving the control and performance of soft robotic devices through more efficient and effective training methods. By leveraging curriculum complexity and exploring transfer learning to actual hardware, researchers can continue to advance the field of soft robotics and develop new applications for these innovative devices.

References

A. Alspach, J. Kim, and K. Yamane. Design and fabrication of a soft robotic hand and arm system. In *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pages 369–375, 2018. doi: 10.1109/ROBOSOFT.2018.8404947.

B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch. Emergent tool use from multi-agent autocurricula, 2020.

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13 (5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *International Conference on Machine Learning*, 2009.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan. Data-driven control of soft robots using koopman operator theory. *IEEE Transactions on Robotics*, 37(3):948–961, 2021. doi: 10.1109/TRO.2020.3038693.

M. Calisti, M. Giorelli, G. Levy, B. Mazzolai, B. Hochner, C. Laschi, and P. Dario. An octopus-bioinspired solution to movement and manipulation for soft robots. *Bioinspiration biomimetics*, 6:036002, 06 2011. doi: 10.1088/1748-3182/6/3/036002.

M. L. Castaño, A. Hess, G. Mamakoukas, T. Gao, T. Murphey, and X. Tan. Control-oriented modeling of soft robotic swimmer with koopman operators. In *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1679–1685, 2020. doi: 10.1109/AIM43001.2020.9159033.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models, 2018.

B. Ellenberger. Pybullet gymperium. https://github.com/benelot/pybullet-gym, 2018–2019.

C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel. Reverse curriculum generation for reinforcement learning, 2018.

J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning, 2016.

T. George Thuruthel, E. Falotico, F. Renda, and C. Laschi. Learning dynamic models for open loop predictive control of soft robotic manipulators. *Bioinspiration Biomimetics*, 12, 08 2017. doi: 10.1088/1748-3190/aa839f.

T. George Thuruthel, Y. Ansari, E. Falotico, and C. Laschi. Control strategies for soft robotic manipulators: A survey. *Soft Robotics*, 5(2):149–163, 2018. doi: 10.1089/soro.2017.0007. URL https://doi.org/10.1089/soro.2017.0007. PMID: 29297756.

M. Graule, D. Bruder, T. McCarthy, A. V. R. Lima, Werfel, and R. Wood. Reducing the cost of reinforcement learning via adaptive domain discretization., 2023.

M. A. Graule, C. B. Teeple, T. P. McCarthy, G. R. Kim, R. C. St. Louis, and R. J. Wood. Somo: Fast and accurate simulations of continuum robots in complex environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3934–3941, 2021. doi: 10.1109/IROS51168.2021.9636059.

M. A. Graule, T. P. McCarthy, C. B. Teeple, J. Werfel, and R. J. Wood. Somogym: A toolkit for developing and evaluating controllers and reinforcement learning algorithms for soft robots. *IEEE Robotics and Automation Letters*, 7(2):4071–4078, 2022. doi: 10.1109/LRA.2022.3149580.

A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu. Automated curriculum learning for neural networks, 2017.

A. Gupta, C. Eppner, S. Levine, and P. Abbeel. Learning dexterous manipulation for a soft robotic hand from human demonstration, 2017.

N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver. Emergence of locomotion behaviours in rich environments, 2017.

O. Kilinc and G. Montana. Follow the object: Curriculum learning for manipulation tasks with imagined goals, 2021.

M. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper_files/paper/2010/file/e57c6b956a6521b28495f2886ca0977a-Paper.pdf.

S. Lange, T. Gabel, and M. Riedmiller. *Batch Reinforcement Learning*, pages 45–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-27645-3. doi: 10.1007/978-3-642-27645-3_2. URL https://doi.org/10.1007/978-3-642-27645-3_2.

C. Li, B. Zhuang, G. Wang, X. Liang, X. Chang, and Y. Yang. Automated progressive learning for efficient training of vision transformers, 2022.

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, 2019.

R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/68a9750337a418a86fe06c1991a1d64c-Paper.pdf.

C. Majidi. Soft robotics: A perspective—current trends and prospects for the future. *Soft Robotics*, 1:5–11, 03 2014. doi: 10.1089/soro.2013.0001.

A. D. Marchese, R. K. Katzschmann, and D. Rus. Whole arm planning for a soft and highly compliant 2d robotic manipulator. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 554–560, 2014.

T. Matiisen, A. Oliver, T. Cohen, and J. Schulman. Teacher-student curriculum learning, 2017.

T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. Model-based reinforcement learning: A survey, 2022.

A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning, 2017.

S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone. Curriculum learning for reinforcement learning domains: A framework and survey, 2020.

A. S. Polydoros and L. Nalpantidis. A reservoir computing approach for learning forward dynamics of industrial manipulators. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 612–618, 2016. doi: 10.1109/IROS.2016.7759116.

M. L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. In *Wiley Series in Probability and Statistics*, 1994.

G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. 1994.

M. Runciman, A. Darzi, and G. P. Mylonas. Soft robotics in minimally invasive surgery. *Soft Robotics*, 6(4):423–443, 2019. doi: 10.1089/soro.2018.0136. URL https://doi.org/10.1089/soro.2018.0136. PMID: 30920355.

D. Rus and M. Tolley. Design, fabrication and control of soft robots. *Nature*, 521:467–75, 05 2015. doi: 10.1038/nature14543.

C. D. Santina, C. Duriez, and D. Rus. Model based control of soft robots: A survey of the state of the art and open challenges, 2021.

S. Satheeshbabu, N. K. Uppalapati, G. Chowdhary, and G. Krishnan. Open loop position control of soft continuum arm using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5133–5139, 2019. doi: 10.1109/ICRA.2019.8793653.

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization, 2017a.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017b.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999a. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999b. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.