# SofaGym: An open platform for Reinforcement Learning based on Soft Robot simulations

Pierre Schegg[a,b], Etienne Ménager[a], Elie Khairallah[a], Damien Marchal[a], Jérémie Dequidt[a], Philippe Preux[a] and Christian Duriez[a] *

Abstract

OpenAI Gym is one of the standard interfaces used to train Reinforcement Learning (RL) Algorithms. The Simulation Open Framework Architecture (SOFA) is a physics based engine that is used for soft robotics simulation and control based on real-time models of deformation. The aim of this paper is to present *SofaGym*, an open source software to create OpenAI Gym interfaces, called environments, out of soft robot digital twins.The link between soft robotics and RL offers new challenges for both fields: representation of the soft robot in a RL context, complex interactions with the environment, use of specific mechanical tools to control soft robots, transfer of policies learned in simulation to the real world, etc. The article presents the large possible uses of SofaGym to tackle these challenges by using RL and planning algorithms. This publication contains neither new algorithms nor new models but proposes a new platform, open to the community, that offers non existing possibilities of coupling RL to physics based simulation of soft robots. We present 11 environments, representing a wide variety of soft robots and applications, we highlight the challenges showcased by each environment. We propose methods of solving the task using traditional control, RL and planning and point out research perspectives using the platform.

Keywords: SOFA, Finite Element Method, Reinforcement Learning, Imitation Learning, Monte CaRLo Tree Search, OpenAI Gym.

## 1. Introduction

The field of application of soft robotics is vast and offers many challenges including locomotion and manoeuvrability in confined environments,[1] smooth gripping of fragile objects using soft materials and using environmental contacts to solve complex tasks to cite only a few. The mechanics corresponding to the behavior of soft robots are continuum mechanics, for which there is generally no analytical solution, except in very simplified and often unrealistic cases.[2–7] However, using the Finite Element Method (FEM) or equivalent numerical methods, accurate models of soft robot deformation can be obtained. FEM is used in the physics based engine SOFA.[8] This technique can efficiently simulate soft robot behaviors, including non-linearities, heterogeneities, pneumatic cavities and the use of tendons for instance. It can also account for contacts between the robot and its environment.

Several commercial and open source simulators are available for rigid robots.[9] Multi-physics simulators for deforming structures also exist[10–12] and are sometimes used to simulate soft robots.[13–18] However those are deemed too slow and only fit for offline simulation. In practice, these tools are often used to simulate parts of the robot during the design process but rarely the whole robot in its surrounding woRLd, which requires to include a collision detection and response pipeline. The Simulation Open Frame-

*Pierre Schegg and Etienne Ménager contributed equally to this work. [a] Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France [b] Robocath, Rouen, France pierre.schegg@inria.fr, etienne.menager@inria.fr

work Architecture (SOFA)[8,19] is a FEM software first used for medical simulation but extended to soft robotics via a plugin.[20] It can handle multi-physics, collisions and a set of tools designed for soft robotics such as soft actuators and inverse models. This tool allows to model, simulate and control these robots .[21]

In recent years, RL algorithms were for instance used to learn complex behaviors in video games,[22] in environments that may be representations of physical reality, and physics based simulations.[23,24] In rigid robotics, RL algorithms were used to learn pick and place[25–27] and manipulating a cube and the Rubik's cube with a robotics hand.[28,29] There are also many examples of using RL in the soft robotics community[30] with the limitations raised above. One popular workflow is to first train the RL policy using simulation and then transfer to the real physical robot.[31] This has several advantages among which running several simulations that are faster in real time in parallel. However, transferring from simulation to reality causes several problems which are out of the scope of this paper,[32,33] though it is widely accepted that using a simulator that is more faithful to reality will ease this transfer process.

OpenAI Gym[34] has become the reference for RL providing both interfaces (called gym environments in the RL community) allowing to compare different learning algorithms' performances and a standardized way of communicating with these environments. More specifically, OpenAI aims to provide a fair common ground to compare RL algorithms and keep track of the best performing ones in a leaderboard.[35] These environments include common control tasks such as swinging a pendulum, algorithmic tasks, tasks based on Atari videogames and robotics using the MuJoCo[36–39] physics engine. OpenAI and many other projects also provide baselines,[?,?,40,41] namely implementations of common RL algorithms which are designed to be used with gym environments. Finally both OpenAI Gym environments and baselines are Open Source.

There also exists Gym-based environments based on physical simulations used for robotics, such as Mujoco[42] and PyBullet,[43] and which have the capacity to simulate deformable objects,[44–47] but the use cases of those environments are mostly in learning to manipulate deformable objects and not simulating soft robots. This is also the case of the PlasticineLab[48] and SoftGym[49] environments, where the goal is to modify soft structures with rigid grippers or tools. Simulators, such as ChainQueen[50] or DiffTaichi,[51] offer differentiable physical simulators to simulate flexible robots and perform learning. Although these approaches provide real-time simulations, they are often based on simplified models and do not offer a wide variety of soft actuators, deformable structures and constitutive laws. To the authors' best knowledge there is no easy, complete and Open Source way to couple soft robot simulation and Gym environments, meaning the control of a soft-body deformed by specific soft actuators in a Gym environment. This is what this paper and the associated source code aims to provide.

SofaGym allows to use the Gym API directly with SOFA:: SOFA makes it possible to take into account faithful mechanical modelling based on FEM calculations, the SoftRobots plugin includes many soft actuation methods, and Gym makes it possible to manipulate environments using a reference framework and directly exploit baseline algorithms. The paper's contributions are:

- A new and open source platform allowing to train and test RL algorithms on numerical twins of soft robots, simulated with their environment.
- 11 examples of Gym environments based on robots representing a wide variety of tasks classical robotics (grasping, manipulation, locomotion and high level control) and showcasing the challenges discussed in the previous part.
- Examples of trained RL agents used to solve those tasks intended as a first baseline to compare new results to, including some recent, state of the art algorithms never applied to soft robotics.
- For the first time, RL approach is combined with model order reduction of a FEM model. Potential and limitations of such approach are discussed.

- A discussion over the results, including transferring a policy learned in simulation to reality, and the potential of this platform for both the soft robotics and RL communities. We also discuss how learning methods and model based control can complement each other, specifically using learning or planning for high level control and inverse model for low level control as well as using learning or planning techniques to overcome non convexities and local minima, which are limiting the inverse model optimization.

## 2. Background

### *2.1. Background on RL*

RL is a type of machine learning. The goal of RL is to learn the sequence of decisions to make in order to reach a certain goal. An important class of such problems is made of Markov decision problems (MDP). An MDP is defined by a set of states $S$, with $s \in S$, a set of actions $A$, with $a \in A$, a transition function $s_{t+1} \sim \mathbb{P}(s_t, a_t)$, a reward function $r_t \sim R(s_t, a_t)$, and an objective function $J$. Though strictly speaking the set of states of an MDP is discrete, it is rather common to consider a continuous state space in RL, likewise, for the set of actions which may also be continuous. In the case of soft robotics, the action space can be constituted by the command of different actuators, whether they are rigid (servo-motor, linear actuator, etc) or specific to soft robotics (cable, pneumatic cavity, etc). The state can be composed of information obtained from sensors, such as contact sensors, or visual information. In the case where a simulator is used for training, other information such as gradients or data from mechanical models can be used. The solution of such a problem is a policy $\pi(a|s)$ which represents the probability to perform action $a$ in state $s$. In RL, it is very customary to consider the following objective function:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}[\Sigma_{t=0}^{T}\gamma^t r(s_t, a_t)] \qquad (1)$$

where $\gamma \in [0, 1)$ is a discount factor[52] , $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ...)$ is a trajectory sampled with the policy $\pi$ and $T$ is the horizon, and $t \in \mathbb{N}$ are decision instants that indicate an order of completion of the actions, in the sense that the effect of an action is considered immediate and that the execution of $a_t$ is completed before $t + 1$. In this case, it can be shown that the optimal policy $\pi^*$, that is the one that optimizes $J$, is deterministic and stationary, that is in any state $s$, there is one best action, or a set of equivalently best actions, leading to the same value of $J$.

If we know all the elements defining the MDP, obtaining the optimal policy consists in solving an optimization problem, which is known as a planning problem. In RL, we suppose that we know $S$, $A$, and $J$, but we know neither the transition function, nor the reward function. In this case, the algorithm has access to pieces of information regarding these two functions by interacting with its environment: these interactions reveal samples of these two functions. Learning an optimal policy may be achieved through a trial-and-error approach: the agent progressively learns the consequences of its actions by iteratively trying one action in its current state, and observe its consequences (the transition to the next state, and the reward).

To be able to deal with large state spaces, the policy is represented using a function approximator with parameters $\theta$, hence the notation $\pi_\theta$. It is very customary to use a neural network for that purpose, hence the $\theta$ are the network weights. In RL, the classical approach to solve an MDP is $Q$-learning using a neural network to represent the state-action value function $Q$,[53–57] leading to Deep Q-Learning (DQN),[58] or an actor-critic approach, such as the state-of-the-art algorithms Proximal Policy Optimization (PPO)[59] and Soft Actor Critic (SAC).[60] Recent algorithms, like Actor with Variance Estimated Critic (AVEC)[61] that can be combined with SAC and PPO, allow to improve both sample efficiency and stability of these algorithms.

By definition, a state contains all the necessary information to be able to decide which action should be performed in order to optimize the objective function. In a real application, it is difficult to be certain that we indeed have all the necessary information. For that reason, one distinguishes the notion of an "observation" from the notion of state. Ideally, the observation is a state but usually, one uses the information it has access to, the

observation available about the current situation, and hopes that this is indeed a state.

### *2.2. Background on planning*

Planning algorithms aim to create a plan to reach an objective. Planning is usually done "offline" in the sense that the creation of the plan is done before the execution of the sequence of actions. Since the plan must be searchable before the sequence of actions is carried out, a model of the environment must be available, that is the knowledge of the definition of the MDP, $S$, $A$, the transition function, the reward function, and the objective function. In this paper, we use a simulator as a model of reality and we will consider tree-search techniques.

As an example of a tree-search technique, Monte Carlo Tree Search (MCTS) [62] is a planning algorithm used in sequential decision making. The algorithm considers the tree of possibilities: the root $s_0$ of the tree is the initial configuration of the environment; each node $s_i$ is the configuration obtained by starting from the previous configuration $s_{p(i)}$, where $p(i)$ corresponds to the parent of $i$ in the tree, and applying an action. The general idea is to select the most promising leaf in the sense of the reward function, simulate an action from this leaf to create a new leaf, retrieve the expected reward from this new leaf and propagate it to the rest of the tree. The tree is traversed until a final state is found.

Many tree search algorithms have been developed.[63] Recently, many stemmed from the Multi-Armed Bandit community. Some examples include Upper Confidence Bounds applied to trees (UCT),[64] Open-Loop Optimistic Planning (OLOP) [65] and Optimistic Planning for Deterministic systems (OPD) .[66] The general idea always remains to expand a search tree in order to find the sequence of actions which will solve the task.

### *2.3. Gym API*

In the RL community, an environment consists of all the elements introduced in the previous section: an action space, an observation space, a reward function and a transition function. Less formally, an environment in RL corresponds to a means of simulating the interactions between an agent and an external world and of retrieving information of interest for learning such as the state of the agent $s_t$, or the immediate reward $r_t$.

The Gym framework allows to interact with an environment using a very small set of simple functions, like $step(a)$ to simulate the effect of the agent performing action $a$ and to recover information from the environment (state, reward, flag to indicate the end of the episode), $reset()$ to restart the environment and $render()$ to give a visual representation of the current state.

Using this interface provides a single generic way (API) to interact with many environments. This is what SofaGym allows when the environment is a SOFA simulation, i.e. the simulated robots and their surroundings.

### *2.4. Background on soft robot simulation*

The modeling of soft robots is based on solid mechanics and relies on the sum of forces that apply to the system, including internal forces (deformations), inertial forces, external forces (gravity, contacts, etc.) and forces related to the actuation (motor torques, pressure in the cavities, etc.).

We can keep a unified formalism for rigid and deformable solids, by adapting the parametrization. In the following we suppose that the state of the robot is represented by a position vector $\mathbf{q}$ and a velocity vector $\mathbf{v}$. Depending on the type of model we are using $\mathbf{q}$ can represent the 3D position of nodes (for a volume FEM mesh or mass spring models for instance) but also Frame positions (for beams, shells or rigid parts of the robot). We also use sometimes reduced coordinate (for Cosserat models or articulated systems). SOFA contains many numerical models of deformations.

In SOFA, the dynamics of the robot are written:

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} = \mathbf{g}(\mathbf{q}) - \mathbf{f}(\mathbf{q},\mathbf{v}) + \mathbf{H}_a^T\boldsymbol{\lambda}_a + \mathbf{H}_c^T\boldsymbol{\lambda}_c \quad (2)$$

where $\mathbf{M}(\mathbf{q})\dot{\mathbf{v}}$ represents the inertial forces, $\mathbf{g}(\mathbf{q})$ the gravity forces, $\mathbf{f}(\mathbf{q},\mathbf{v})$ the internal forces (based on the constitutive laws of the materials). The actuator forces $\mathbf{H}_a^T\boldsymbol{\lambda}_a$ and contact forces $\mathbf{H}_c^T\boldsymbol{\lambda}_c$ are using Lagrange Multipliers.

To integrate the equation of the dynamics over time, several difficulties need to be considered: (i) the stability of the deformable models is not guaranteed with explicit integration schemes, except with very small time steps, (ii) collisions between the robot and its environment create non-smooth events (even on deformable solids) that need to be carefully integrated over time, (iii) the computation time needed to integrate a numerical time step needs to be limited in comparison with the size of this time step (note that if it is possible to make them equal, a real-time simulation is possible). These different limitations have led to the use of "time stepping" implicit integration schemes of quite low order.[8]

Lagrange Multipliers are in particular used for the computation of both direct and inverse models of soft robots, as in most cases, the efforts exerted by the actuators result from a solving process, computed by optimization. The same problem arises with contact: contact forces are not known in advanced and need to be solved by dedicated solvers.[21] Contact points between models are detected thanks to collision detection algorithms which often uses surface models of the robot and of its environment. Contact response uses a formalism of complementarity constraints based on Signorini's law and friction is modeled with Coulomb's law, accounting for stick and slip cases, which are necessary to model correctly locomotion, grasping and manipulation.

## 3. Challenges of applying RL to soft robotics

### *3.1. Soft robotics is a unique problem for RL*

From an RL point of view, the control of soft-robots offers numerous challenges. First, the observation space must contain enough information to take into account the deformations of the different parts of the environment (external objects or robot) and the contacts between the environment and the robot. Contacts are very important in the context of soft-robotics because they condition the shape of the robot. Taking this information into account leads to a large observation space that is sometimes difficult to parameterise. Indeed, the compact representation of soft robots is an open question. An example of this is the question of finding a compact representation of the deformations of a robot without considering all the points that compose it. In this paper we explore the idea of using deformation modes rooted in a reduced model. Second, the dimension of the action space of a soft-robot is often much larger than that of a rigid robot. This action space makes it possible to control the deformation of a robot with flexible or non-flexible motorisation. Generally, the soft robot is under actuated. This leads to very complex interactions with the environment, and the result of each action is strongly conditioned by the collision and deformation state of the robot. Indeed, a simple contact with the environment can completely change the result of an action and lead to very different configurations of the robot. In addition, in general, redoing an action in the other direction does not return the robot to its previous state, which leads to a notion of non-reversibility of actions. These two facts lead to long training times. It is therefore necessary to be able to overcome this problem by inventing appropriate methods, particulaRLy in terms of sample efficiency.

### *3.2. Limits of existing control methods*

Soft robots can be controlled with traditional control techniques, such as inverse models.[67] These models do not require learning time but are limited by strong mathematical assumptions on the optimization domain, notably convexity assumptions. On the contrary, RL is able to handle problems without strong assumptions about the spaces and can therefore be more suitable for instance in problems where contacts lead to change of the optimization domain. In some cases, RL and inverse model control will converge to the same solutions. In those cases, RL will run in real time but requires prior training while optimization techniques compute online. An interesting perspective could be to use the simulator as a model in model based RL. Another promising approach, inspired by rigid robotics ,[68] could be to use information from the simulation in the RL models by doing Differentiable Dynamic Programming. In the authors' opinion, one of the main challenges of applying RL in soft-robotics is to find when to use traditional control schemes and when to use tools from learning, combining the advantages of both approaches. In this paper we demonstrate using RL as a *high level control* tool to find a tra-

jectory in the observation space, and an inverse model to actuate the motors and follow the trajectory.

### *3.3. Need for fast and accurate simulation*

RL requires many interactions between the agent and the environment to learn a good policy. To limit the learning time, rapid interactions between the agent and the environment are required. Whether in soft or classical robotics, the advantage of performing learning in simulation is well established: parallelization and fast computing speeds up learning, there is no risk of altering a physical robot which can be costly, and it is possible to perform critical tasks such as medical operations. There are fast simulators in the case of manipulation of soft objects[48,49] , which allow to realise quick learning. But in critical tasks, such as medical tasks, or in tasks requiring a real representation of deformations, accurate and faithful simulators are also needed. The balance between accuracy and speed has to be evaluated. Two major challenges of RL in soft robotics are to provide fast, accurate, and faithful simulation tools, and to design sample efficient learning algorithms. FEM simulation, which has been tested and validated in the medical field and real time control in soft robotics,[69] enables to obtain faithful and accurate simulations, sometimes at the expense of simulation speed. An increased simulation accuracy, as well as domain randomization techniques are known to facilitate the transfer of policies learned in simulation to the real woRLd.[28,29] It is also possible to run parallel episodes. This allows to collect data faster and can to some extend compensate for slower simulations. Some RL algorithms, such as RL algorithms that use a replay buffer, can take advantage of this parallelization and can thus provide better temporal performance than running a single real-time simulation.

### *3.4. Transferring from simulation to reality*

Transfer learning is a well-known problem in robotics. It consists in learning a strategy in simulation and then applying it to the real physical robot robot. Discrepancies usually appear as a simulation is never a perfect model and unmodeled external factors have to be taken into account. Mathematically modeling a soft robot is a complex task. Using FEM simulation allows to represent a large diversity of shapes, geometries and deformations, but the quality of the simulation often depends on the calibration or estimation of several parameters (Young's modulus, Poisson's ratio, friction coefficients, etc). A slight variation of this calibration can lead to very different configurations of the robot when there are contacts or threshold effects. RL tends to exploit weaknesses in the simulation and edge cases: the more accurate the simulation, the easier it is to transfer the strategy learned in simulation to the real robot. In this paper we show how one can use a combination of planning and inverse model control to successfully transfer a policy planned in simulation to the real physical robot.

### *3.5. Lack of a unified platform and benchmark*

As stated in the introduction, OpenAI Gym provides a series of standard environments for the community to design and experiment algorithms on, and to compare the results on a fair common benchmark. This type of platform helps drive research and allows to compare the performances of either new algorithms on a known problem, or known algorithms on a new task. At the moment, no such platform exists for applying RL to a wide variety of deformable robots, with the capacity of being easily extended to new robots and applications. SofaGym aims to provide such a platform, implementing several environments and setting a baseline for future research to compare to. SofaGym also allows to easily create new environments from FEM simulations to offer new and interesting problems to the community.

### *3.6. Taking into account the adaptability of soft robots*

Using RL to control soft robots gives two levels of adaptation. Firstly, RL algorithms allow to generalize to different situations. Secondly, the structure of the robot itself is flexible. These levels of adaptation combine and any adaptation performed from a mechanical point of view by the body of the robot is will be added to the adaptation performed at the learning level. To the author's opinion, this makes learning based strategies well

adapted to flexible robots since they require less precise strategies (the inaccuracies being compensated by the structure of the robot). This is an important point to take into account when transferring from simulation to reality, since a part of the adaptation to the imperfections of the simulation and the control will be compensated directly by the physical structure of the robot. These imperfections must still be in the adaptation range of the robot, hence the need to have faithful simulations that take into account the deformations of the flexible robot.

To take full advantage of this adaptation of flexible robots, the reward function must be well designed. Indeed, it is possible to force some points of the robot to reach a given position, a speed, to have a certain behavior. However, by directly constraining the robot's points, we do not take advantage of the robot's ability to deform and adapt. It is more appropriate to focus the reward function on the interaction of the robot with its environment: for a manipulator the position of the object to be manipulated, a zone in which to stay, a maximum force to be applied to an object, etc. The definition of the reward function is therefore a challenge in itself: it must allow to solve the given task while leaving the possibility for the robot to adapt itself to its environment.

## 4. Materials and Methods: RL Environments

In this section we exhibit 11 Gym environments representing different common applications in soft robotics (grasping, manipulation, planning and locomotion) and showcasing the challenges exposed in section 3 (controlling deformable bodies, handling collisions and contacts, soft actuation, advantages and limits of training simulation, advantages of learning and planning techniques over traditional control schemes and their complementarity and the problem of transferring from simulation to reality). These tasks and challenges are illustrated by different robots, shown in Figure 1. The aim is to showcase the versatility of SofaGym by demonstrating that it can be used to create a Gym environment out of any SOFA simulation of a robot, covering a variety of actuators, sensors and tasks.

For the soft robotics community, these environments are intended as examples of how to use SofaGym on a variety of SOFA simulations and to encourage readers to create new environments out of their own simulations and applications, thus submitting novel and interesting problems to the community and allowing other research groups to compare their results on the same problem. The environments presented in this article are proposed as a set of first tasks in a virtual playground to test, combine, compare and validate various algorithms, observation spaces, action spaces and rewards.

For the RL community, we present a selection of environments which represent challenging tasks in soft robotics. The focus is put both on the diversity of applications and on representing a variety of challenges. This could enable the development of new concepts and algorithms to solve those tasks. By open sourcing the platform we also enable the modification of the observation and action spaces of every environment and the engineering of the reward function.

For each environment, we present the challenges posed by the environment and we list research perspectives involving the environment or a simple modification of it. Some of those challenges are explored in section 5, other perspectives are detailed in the discussion. Since SofaGym is an open-source platform, all results can be verified and improved by users, and the challenges presented in the previous section can be deeply explored.

### *4.1. Baseline Environments*

*4.1.1 Overview* We designed 3 small scale environments to showcase specific challenges posed by soft robotics. These environments were designed to run fast and to serve as a starting point for discussion on RL in soft-robotics field. One of them was adapted from a classic Gym environment to include deformable elements: the *CartStem* environment corresponds to the classical gym *CartPole* environment with à flexible pendulum. This environment shows the importance of deformation in soft robot control. The *CartStemContact* is an environment designed to show the importance of taking into account contacts between soft-robot and objects for the resolution of a placement task. The last environ-

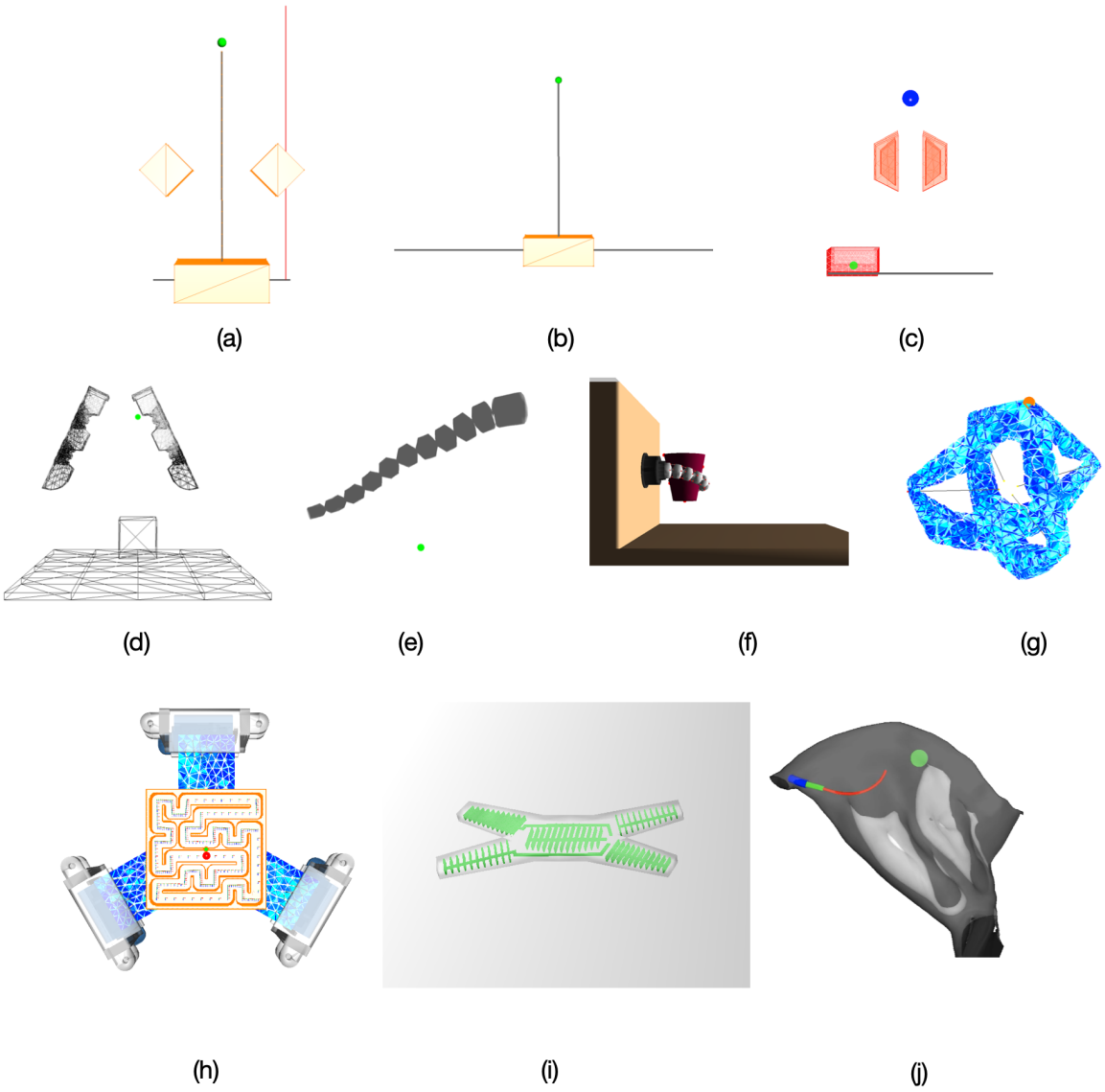


**Figure 1. Examples of environments developed in SofaGym. (a), (b), and (c) are environments of the baseline. (a) CartStemContactEnv: a stem is moved horizontally with a cart and must use the contacts to place its tip, in green on the image, at the horizontal level of the red line. (b) CartStemEnv: a flexible stem must be kept vertical by the horizontal movement of a cart on which it is fixed. (c) CatchTheObjectEnv: a ball must be dropped into a moving cart. The fall of the ball can be prevented by using a pneumatic gripper. (d) GripperEnv: A cable actuated gripper is used to bring a deformable cube to a certain height, given by the green point in the figure. (e) TrunkEnv: the Truck must be activated so that its end reaches a goal, represented in green on the figure. (f) TrunkCupEnv: the Trunk is used to move and orient a flexible cup. (g) DiamondEnv: the end of the robot, in orange on the figure, must be brought to a certain position. (h) MazeEnv and SimpleMazeEnv: in MazeEnv a tripod robot is used to orient a maze that has to be solved by bringing a ball to a certain position. The SimpleMazeEnv is simply the maze that can be oriented. (i) MultigaitEnv and ReducedMultigaitEnv: in both cases the robot is moved along the horizontal axis. The difference of the two environments is given by the use or not of a reduced model to represent the robot. (j) ConcentricTubeRobotEnv: it is necessary to bring the tip of the smallest of the three tubes to a certain position in a nasal cavity.**

ment, *CatchTheObject* environment, illustrates pneumatic actuation. These setups are shown in figure 1.a, 1.b, 1.c.

*4.1.2 Technical considerations* The most important technical point is the actual simulation time needed to perform a simulation time step. Here we use the RTF (Real-Time Factor) to quan-

tify this simulation time. The RTF is given by the formula $RTF = \frac{\Delta t_{simu}}{\Delta t_{real}}$, where $\Delta t_{real}$ secondes is the real time to simulate $\Delta t_{simu}$ secondes of the simulation. If we have $RTF \geq 1$, we have $\Delta t_{simu} \geq \Delta t_{real}$ and we simulate more time than the computation time. A real-time simulation has a $RTF = 1$. For all environments we compute the RTF with $dt = 0.01s$ and when the simulation is stable with a larger $dt$ we also compute the RTF for this value. In fact, it is possible to improve the RTF by increasing the $dt$ value, paying attention to the stability of the systems. The mean is computed over one episod. These results are given to give an order of magnitude of the temporal performance of the environments. The length of the episodes is given in Appendix C. The following table summarises the RTFs[1] of the different baseline environments.

| Environment | Mean RTF | |
|---|---|---|
| | dt = 0.01 | dt = 0.05 |
| *CartStemContact* | 3.5 | 17.7 |
| *CartStem* | 3.6 | 18.6 |
| *CatchTheObject* | 1.2 | 5.2 |

The environments proposed in the baseline therefore run faster than real time. Thus, the baseline environments are fast environments that allow easy testing of RL algorithms in the context of soft robotics.

*4.1.3 Challenges and research opportunities* These environments allow to illustrate the functioning of SofaGym on simple examples and to highlight some basic RL challenges in soft-robotics. The *CartStemContact* environment highlights a problem that could not be solved by convex optimization: the contacts allow to delimit three convex zones and the RL algorithm allows to switch from one zone to another. The *CartStem* environment is made up of a flexible part: the reward and the agent's state cannot be represented in the same way as for the associated rigid environments. In addition, the parameters of the robot (rigidity of the stem, force of the actuation) have to be designed to simulate a realistic behavior. In *CatchTheObject* environment, the pneumatic actuation leads to slower simulations as the entire deformation of the cavities has to be simulated. Efficient algorithms must therefore be found to solve the corresponding tasks.

Even if these environments are simple, they make it possible to think about other challenges such as transfer learning and how to use a learned strategy for an environment which differs slightly from the learning environment. For example, it would be easy to change the size of the pendulum or the material composition of the different robots in order to see how the algorithms can adapt to these changes. This could represent a first step towards designing a more robust policy and the transfer to a real robot.

### *4.2. Grasping example: GripperEnv*

*4.2.1 Overview* The Gripper environment offers two different simulations. In both, the objective is to grasp a deformable cube and bring it to a certain height. The closer the cube is to the target, the greater the reward. In the first scenario the actuation is discrete and in the second it is continuous.

The simulated gripper consists of two soft cable-actuated fingers that can translate and rotate in the simulated environment. The fingers have 3 phalanx and are made out of silicone.[70] The fingers have 4 embedded cavities with sensors that measure the pressure inside them so that these readings can be used to infer the fingers shape during the deformation. The two fingers are rigidly attached at their base and a cable passes through each of them. The fingers are actuated by pulling the cables. Forces of random intensity and direction are applied to the cube to test the robustness of the grip. The setup is shown in figure 1.d.

*4.2.2 Technical considerations* In the same way as for the previous environments, it is possible to calculate the RTF of this environment.

| Environment | Mean RTF | |
|---|---|---|
| | dt = 0.01 | dt = 0.03 |
| *GripperEnv* | 0.15 | 0.5 |

[1] All calculations are done on the same machine (macOS Big Sur, Intel Core i7 quad-core 3.1GHz processor, 16GB 2133 MHz LPDDR3 memory), under the same conditions.

*4.2.3 Challenges and research opportunities* Soft grippers are the most widespread application of soft robotics nowadays and they remain challenging because of the contact between the gripper and the objects to be grabbed. As was developped in section 3.6. soft grippers are the robots which best illustrate the adaptability of flexible robots. Indeed, the gripper's claws adapt to the objects to be gripped when the forces applied are sufficiently large. A slight change in the force applied to the object will change the gripper's deformation but not affect the overall grip. The reward must be designed to take into account the adaptability of the robot. In addition, it is very easy to change the geometry of the gripper, so long as it is cable actuated. We include a simple geometry from the SoftRobots plugin examples[21] in the environment as a comparison. Changing the object to grasp is also simple, we include a dragon figurine from the standard SOFA meshes. This environment could be used to experiment with co-designing a finger geometry and a control policy. It is also interesting to test what is required for a policy learned to grasp one object to transfer to a new object. This new object can be either rigid or deformable. Finally, soft grippers are easy to fabricate and assemble. This makes this environment a good foundation to investigate the challenges posed when transferring a policy from simulation to reality.

### *4.3. Placement and manipulation examples: DiamondEnv, TrunkEnv and TrunkCupEnv*

*4.3.1 Overview* We propose here three environments in which the objective is to place either the effector of the robot or a manipulated object at a given position and orientation. The objective is therefore to learn the inverse model of the robot. The *DiamondEnv* environment implements a flexible parallel robot[71] made of silicone and operated by 4 cables placed in the middle of each leg. The *TrunkEnv* environment implements a flexible robot inspired by an elephant trunk.[67] This robot is composed of a flexible body made out of silicone actuated by 8 cables fixed all along the robot. In the *TrunkCupEnv*, this Trunk robot is holding a deformable cup.[72] The goal is to manipulate this cup to bring it's center of gravity to a target position and orientation. These setups are shown in figure 1.e, 1.f and 1.g.

*4.3.2 Technical considerations* In the same way as for the previous environments, it is possible to calculate the RTF of these environments. The following table summarises the RTFs of the different environments.

| Environment | Mean RTF | |
|---|---|---|
| | dt = 0.01 | dt = 0.05 |
| *DiamondEnv* | 0.11 | 0.6 |
| *TrunkEnv* | 0.5 | 2.3 |
| *TrunkCupEnv* | 0.1 | 0.8 |

*4.3.3 Challenges and research opportunities* These environments show that the inverse model can be learned through training techniques but this requires a lot of training time and samples. We provide the equivalent simulations implementing inverse model control to allow for an easy comparison between the learned policies and the strategy chosen by constrained optimization. These environments are intended to showcase the value of inverse model control and to encourage research on how to combine learning techniques with existing control schemes rather than replacing them entirely.

### *4.4. High level control examples: MazeEnv and SimpleMazeEnv*

*4.4.1 Overview* This environment consists of a small rigid sphere, a rigid maze and a tripod soft robot.[73] The maze is attached to the tripod. Deforming the membrane of the tripod changes the orientation of the maze. The ball is moved by gravity by modifying the maze's orientation and needs to navigate to one of the 4 corners. The setup is shown in figure 1.h.

In the complete Maze environment, the actions control directly the servomotors which deform the tripod's membrane. In the simplified version, we consider only the maze and the actions bring the maze directly in a certain orientation. This is not a soft robotics environment per se, it was designed to investigate how RL and planning can interact with traditional control schemes.

*4.4.2 Technical considerations* In the same way as for the previous environments, it is possible to calculate the RTF of these environments. The following table summarises the RTFs of the different environments.

| Environment | Mean RTF | |
|---|---|---|
| | dt = 0.01 | dt = 0.05 |
| *MazeEnv* | 0.02 | 0.06 |
| *SimpleMazeEnv* | 1.26 | 9.0 |

The Maze environment is slow: the two elements, tripod and Maze, taken separately rotate in real time. The link between the two causes the drop in performance of the simulation. It is important to pay attention to the collision model as the time step increases.

*4.4.3 Challenges and research opportunities* The movement of the ball in the labyrinth is entirely guided by the contacts between these two elements. The mesh of the labyrinth can be constructed automatically using generation tools such as Gmsh,[74] which makes it very easy to generate random variations of this environment. The inverse model of the Tripod robot supporting the maze is easy to solve. Though this does not allow to directly solve the maze and navigate the ball to one corner, it makes it natural to split the problem in two. This is also justified by the computation times of the full scene and the reduced scene. Thus one can solve the maze learning or planning the orientations of the maze (*SimpleMaze*) and then use the inverse model to compute the servomotor positions associated with these maze orientations. Moreover the Tripod robot is easy to fabricate and a tutorial is available using 3D printing and silicone casting.[75] The mesh can also be 3D printed. This makes this environment a good candidate to explore transferring from simulation to reality. We showcase an open loop example of policy transfer in this paper. Extracting the state from sensor data and simulation registration would be needed to close the loop.

### *4.5. Locomotion examples: MultigaitEnv and ReducedMultigaitEnv*

*4.5.1 Overview* The MultigaitEnv provides a digital twin of the multigait soft robot[76] in contact with the floor. In this environment the goal of the agent is to move forward in the $x$ direction as far as possible in a given time. The robot is actuated by inflating or deflating 5 cavities, one in the center and one in each leg. The setup is shown in figure 1.i.

To improve the performance of the MultiGait simulation, Model Order Reduction (MOR) technique[77] is used. MOR technique is based on a projection of a simulated model into a lower dimensional space. Given the geometrical details of the MultiGait robot, its FEM model involves a high-dimensional mesh which would be expensive to solve in simulation. The idea is to define a base of a reduced model such that the evolution of the FEM problem can be approximated by looking at the solution in the reduced space. This technique consists of two phases. The first is to find the basis using the POD (Proper Orthogonal Decomposition). This is equivalent to finding the different modes that allow to describe all possible positions of the robot. In the second phase, this basis will be used to solve the simulation problem. In the case of the Multigait, there are 63 modes (3 rigid translational modes, and 60 modes of deformation sorted from most to least important). We call this environment ReducedMultigaitEnv.

*4.5.2 Technical considerations* In the same way as for the previous environments, it is possible to calculate the RTF of this environment. The following table summarises the RTFs of the different environments.

| Environment | Mean RTF |
|---|---|
| *MultigaitEnv* | 0.007 |
| *ReducedMultigaitEnv* | 0.15 |

Once again, we can see that the reduction of the model allows us to have better temporal performances, with a factor of 20. This shows the interest of using mechanics tools in simulation environments. In this environment, increasing the time step causes instability problems for the same solver configuration (contact distance, alarm distance, etc.).

*4.5.3 Challenges and research opportunities* With multigait, we have explicit mechanical modelling. Contrary to the two previous parts where

there were mechanical tools to solve the task, the mechanics is used here as a tool to model the robot and solve the simulation. The idea of this environment is therefore to see how to exploit this mechanical representation of the robot to support learning, for example by using MOR deformation modes to represent the state of the robot. As the Multigait robot is a classic of soft-robotics, it is also possible to compare the results obtained in simulation with a hand crafted baseline. Finally, the interest of having two scenes, reduced scene and full scene, is to see if it is possible to transfer the learning from one to the other. The use of MOR has some limitations, such as a limited number of deformation and translation modes, which must be taken into account both in the learning and in the transfer of strategy.

### *4.6. Navigation example in a medical context: ConcentricTubeRobotEnv*

*4.6.1 Overview* This environment proposes a Concentric Tube Robot (CTR) with 3 concentric tubes. The goal is to bring the tip of the smallest tube to a target position inside a nasal cavity. Each tube can be pushed, pulled or rotated independently at its proximal base, outside the patient. The setup is shown in figure 1.j.

*4.6.2 Technical considerations* In the same way as for the previous environments, it is possible to calculate the RTF of this environment. When contacts are limited, the RTF of this scene is 2.5. However, when the contacts become very numerous, which necessarily happens when the robot is inserted further into the cavity, the RTF can drop by a factor of 10.

| Environment | Mean RTF |
|---|---|
| *CTREnv* | 0.25 to 2.5 |

*4.6.3 Challenges and research opportunities* This environment showcases a medical use case, which has become one of the major applications of soft robotics. In this environment the robot has to navigate in a tight space, which means there will be numerous collisions and contacts between the robot and its surroundings. Since the actuators are far from the tip of the robot, the elastic behavior of the tubes play an important role. The stick-slip transitions in contacts also play an important role in navigation. One can also extract the contact forces between the robot and the nasal cavity which allows to explore safe RL. Finally, since quadratic programming relies on local optimization, this is one of the use cases where using the QP alone will often fail to solve the trajectory and end up in a local minimum. This can be solved by using a high level controller, such as trajectory optimization, in conjunction with the inverse model. We view this environment as another opportunity to explore how learning or planning algorithms can interact with traditional control.

### *4.7. Note on performance*

One of the main limitations of using FEM simulation is its computational cost. SofaGym is no exception to this rule. However this uncovers new research challenges. On the learning side it is paramount to design algorithm that are as data efficient as possible, and that can learn from off policy data. Another interesting challenge is to determine what parts of the task to learn and how to integrate the learned elements with traditional control schemes.

On the simulation side, many improvements can be made, and any gain in simulation speed will have an immediate impact on the learning time. One of the most computationally costly steps is the collision pipeline as is demonstrated by the *ConcentricTubeRobotEnv*. Model Order Reduction tools also greatly increase the speed of the simulation as demonstrated by the *MultigaitEnv* and *ReducedMultigaitEnv*, however this makes the problem of transferring from simulation to reality harder since the reduced model will be less accurate.

### *4.8. Note on implementation*

In order to enable the use of RL and planning algorithms in SOFA, two tools have been implemented. These elements are presented in detail in the Appendix B. It should be noted that these modifications have been implemented for the current version of SOFA and that they are likely to evolve with time.

In order to use planning algorithms, all intermediate configurations of the environment must be saved so that we can return to these configurations if they look promising. SOFA does not allow this step to be done easily, and a server/worker system has been implemented: each configuration is stored in a worker, which can be called by the server if needed.

In order to limit the size of the manipulated SOFA scenes, we have separated the mechanical model of the scene elements from the visual model. The computation nodes contain only the mechanical models, and a viewer allows to retrieve the information from these nodes in order to display them in a visual model of the scene.

## 5. Results: Solving classical soft robotics tasks using SofaGym and RL

In this section we will tackle some of the challenges proposed in the previous section using SofaGym. First, we will compare classical RL algorithms on the baseline environments. Then we will discuss different RL tasks that question the challenges of RL in soft-robotics, namely Behavior Cloning (BC), RL using a mechanical model to represent the agent's state, planning and strategy transfer between simulated and real robots in open loop. The interest here is twofold. The first is to show a wide variety of learning methods based on SofaGym environments applied to classical tasks in soft robotics. Our goal is to demonstrate that the SofaGym platform can easily be interfaced with baseline state of the art learning algorithms and allows to solve the presented tasks. Secondly, the interest is to show how SofaGym can address the challenges we have highlighted in the previous sections. We do not solve here all the challenges but give an idea of how the platform allows to deal with these challenges.

### *5.1. Solve the baseline environments with classical RL algorithms*

The examples in the baseline can be solved using different classical algorithms. The choice was made to use the state-of-the-art algorithms PPO and SAC, which illustrate two different classes of algorithm (on-policy and off-policy). In order to show that SofaGym environments can be used with any algorithm, we also chose to solve these examples with the AVEC algorithm.[?] AVEC is coupled both with SAC and PPO, and we call these algorithms SAC-AVEC and PPO-AVEC. Note that the implementation of the PPO and the SAC are given by the stable-baseline and the implementation of AVEC is given by the author of the corresponding article. The training curves for each of these examples are shown in Figure 2.

We have based the tuning of the hyperparameters on what worked well[?] in the case of rigid examples, by performing a rough grid search. This does not represent an exhaustive search of hyperparameters but allows to have a comparison of basic algorithms on simple soft-robotics examples. For reproducibility these hyperparameters are given in Appendix C. For each environment and each algorithm, the trainings are performed on 6 different seeds, and the results are averaged on 5 non-deterministic tests. All the configuration elements of the environnements and algorithms can be easily modified.

For the environment *CartStem*, we simulated 80 learning steps of 0.1 second each (i.e. a simulation of 8 seconds). This allows us to get closer to the behavior of the Gym CartPole which performs 200 learning steps for a $dt$ of 0.027 seconds, which corresponds to 5.4 seconds of simulation. As for the Gym CartPole, we have a reward of $+1$ when the stem is sufficiently vertical, i.e. when the deformation is low and the tip of the stem remains in a cone above the cart. The simulation is stopped when the sphere goes out of the cone defined previously or when the cart goes out of its given path (i.e. it goes too much to the right or to the left). The initial inclination of the stem is random. The actuation is discrete and corresponds to a constant positive or negative force applied to the cart. Only the algorithms based on PPO that implements discrete actions are used in this environment; our current SAC implementation deals only with continuous actions.

For the environment *CartStemContact*, the *timer* is 30 time steps of 0.3 seconds. The reward is given by the horizontal distance between the tips and the goal (represented by a red line in the figure 1). The episode stops when the *timer* is

reached or when the distance is lower than a certain threshold, here 0.15 cm. The initial position of the goal and the cart is random. The actuation is continuous and corresponds to a position control of the cart.

For the environment *CatchTheObject*, the *timer* is 30 time steps of 0.1 seconds. The reward is given by the distance between the ball and the goal in the cart (represented by a green dot in the figure 1). The episode stops when the ball is in the cart, the ball is under the cart or the timer is reached. The initial vertical position of the ball and the initial position of the cart, as well as the initial direction of the cart are random. When the cart reaches one end of the path, it slightly waits (2.5 seconds) before starting again in the other direction. The actuation is continuous and corresponds to a pressure control of the gripper.

First it is possible to see that we manage to solve the given tasks with RL algorithms, even if not all of them compute a good strategy, i.e. solve the task with the highest possible cumulative reward. For example, in the case of *CartStemContact*, the strategy found by PPO gets a worse cumulative reward than the one found by SAC. The use of AVEC reduces the effective learning time by a factor of at least 2.5, which is interesting in the case of environments that are long to simulate. For exemple, we can solve the *CartStem* in 6 hours (over 6 seeds) with PPO-AVEC but in 18 hours with PPO. The all computation time (for all the seeds and all the algorithms) for the *CartStemContact* is about 19 hours and for the *CatchTheObject* is about 23 hours. Durations are measured on CPU[2]. However, the results obtained in terms of reward are sometimes lower than those obtained with the algorithms without AVEC. For example, in the case of *CartStemContact*, the cumulative reward obtained by SAC-AVEC is 1.5 points lower than the one obtained with SAC. These poorer performances could come from a poor setting of the hyperparameters. A more precise search of these hyperparameters could lead to a better performance of the AVEC algorithms. We did not consider the effect of the AVEC *bias* hyperparameter in this study. Fine tuning this hyperparameter might alleviate the performance gap.

For the *CartStem* the solution found by the algorithm consists in pushing the stem towards the side where it bends to limit its fall, slowing down before arriving at the edge (to avoid stopping the episode prematurely) and limiting the movement of the cart in order to not disturb the stem which is then only subject to its own weight. To obtain a movement closer to the CartPole, it would be necessary to modify the weight and stiffness parameters of the stem as well as the force that can be applied to the cart. This corresponds to one of the challenges which consists in finding mechanical parameters allowing to have a behavior as realistic as possible. Here we consider a state made of the position and the horizontal speed of the stem's tip and of the the cart: indeed it is not possible to have an angle of the stem as for the CartPol. This representation could be improved to take into account more accurately the deformation of the stem, for example by using information from the Cosserat[78] beam model used for the simulation. This corresponds to another challenge of soft-robotics, that is, figuring out to represent the robot state. We will develop this idea later.

For the *CartStemContact*, the PPO algorithms are not very stable. The oscillations can be due to the exploration of the learning algorithms or to the dynamics of the systems. SAC algorithms find strategies that lead to a higher cumulative reward and manage to find a solution to the task in most cases. When the goal is behind an obstacle but close to its extremity, the learning algorithms have difficulty reaching it. There are two ways to get closer to the goal: directly by moving closer to the obstacle, or indirectly by using the opposite obstacle to deform the bar, as is shown in the video linked to this article. The definition of the reward ensures that the minimum cumulative reward is reached in the direct approach, even if the goal is not achieved. It is likely that the results on SAC can be improved with other reward, hyperparameters and more training. Thus when the contact is necessary to reach the goal, i.e. when the goal is far behind the opposite obstacle, the robot has no problem, but it hesitates more when the solution can be reached by almost not using contacts. This hesitation can come from the representation of the contacts and the robot: here the state is constituted by the horizontal position of

[2]Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, 40 cores, one simulation per core, 64GB RAM, no GPU

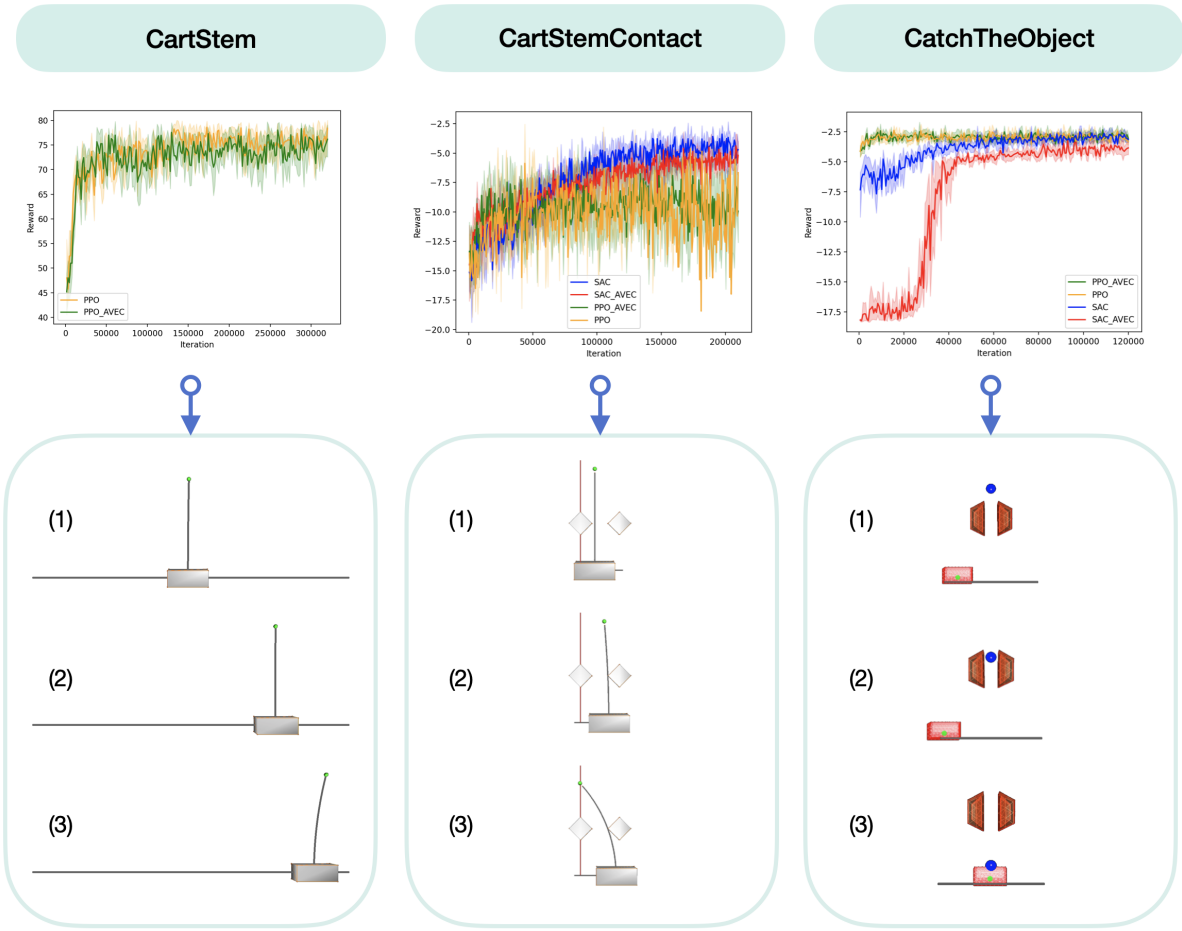


**Figure 2. Results of the baseline. Each task can be solved with an RL algorithm. AVEC based algorithms are faster (in absolute time), but sometimes not as good in terms of reward. PPO can not solve the CartStemContact task and the results strongly oscillate. For the CartStem: (1) applying an action makes the stem bend in one direction, even if it is initially vertical; (2) the robot rectifies it by going to the same side; (3) and then slows down so as not to go out of the allowed path, the stem being then subjected to its own weight. For the CartStemContact: (1) the goal is behind one of the obstacles; (2) the robot comes to rest on the opposite contact; (3) to be able to reach the goal thanks to its deformation. For the CatchTheObject: (1) the ball is subjected to its own weight; (2) the gripper catches the ball while waiting for the cart to pass under it; (3) the gripper releases the ball to reach the goal.**

the centers of the contacts and their dimensions, as well as the horizontal position of the robot, the tips and the goal. Taking into account the contacts in another way can be a way to improve the results. We can see that we are able to overcome the convex space breaks imposed by the contacts, which is not possible for convex optimization methods. This topic is related to the challenge of the limitation of convex optimization methods[67] to control soft-robots.

For the *CatchTheObject*, *PPO*, *SAC* and *PPO-AVEC* algorithms converge to the same result and the performances seem to be good. *SAC-AVEC* does not obtain the same cumulative reward value after 120,000 iterations (about -4 versus -3 for the other algorithms). However, it is possible to see on the curve that the learning continues, and the result may not be the final result. The presence of cavity leads to a slower environment than the other environments of the baseline. This is where *AVEC* algorithms are very interesting: they allow to obtain the same results with *PPO-AVEC* in much shorter times. This illustrates perfectly a challenge we have stated above: it is necessary to find sample efficient algorithms when the simulation of the environments is slow, i.e. when the RTF is close or worse than real time ($RTF \leq 1.5$).

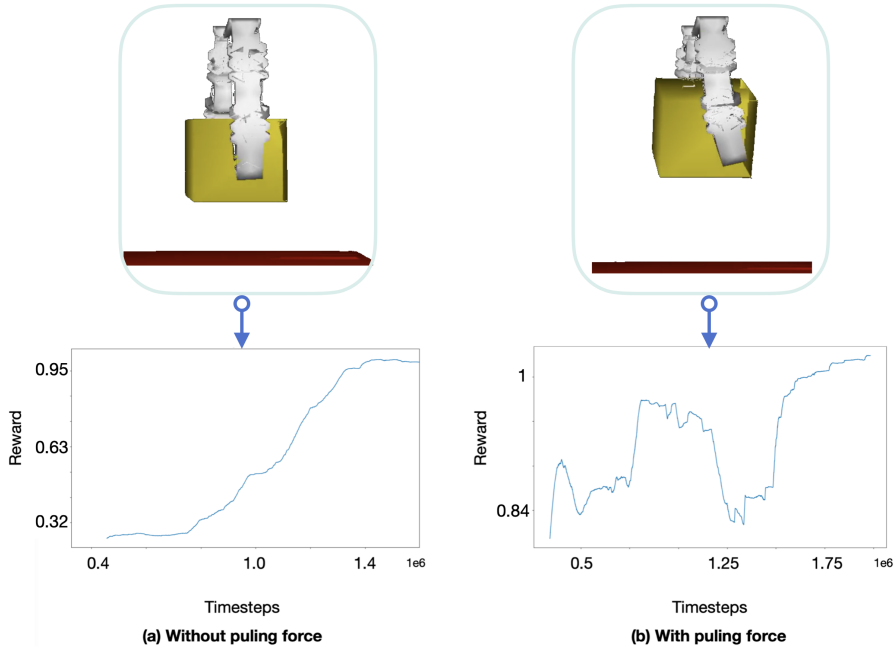


**Figure 3. Grasp learned with and without vertical pulling forces. The addition of force shows that the gripper is able to use both fingers to effectively catch the cube. This is confirmed by the curve rewards against timestep. The interest of this study is to show that the gripper is able to adapt to the occurrence of a random force on the cube. Even in this configuration, the gripper is able to achieve similar performance to that achieved without disturbance. Rewards are normalized to the largest value of the plot.**

### *5.2. Use RL to grasp a flexible object with GripperEnv*

The previous section showed that it is possible to solve basic tasks using RL algorithms. This section shows that it is possible to exploit more advanced RL concepts to solve complex soft-robotics tasks such as grasping tasks. To do that, the problem is reformulated as a control policy to pick up the object and raise it to a certain goal height. Our method is inspired by the OpenAI paper for learning in-hand dexterous manipulation[79] and is based on the use of PPO.

The first idea was to learn to grasp a single object (a cube) by learning to lift it. We designed our value and policy network to have some shared layers to increase the training speed. We tried both including and excluding an LSTM layer. The hyperparameters are given in Appendix C. This configuration led the gripper to converge towards a policy that successfully grabs the cube and takes it to the desired height. However, we noticed that the grip was weak as the gripper was only displacing one finger and using the second one as support. To motivate firmer grips, we added a random force that pulls the cube down at each time step. If the grip is good, the gripper is able to maintain the cube and continues moving with no issues. Effectively, adding this force made the gripper hold the cube firmly. In both cases, the policy converges towards a discounted sum of rewards of around 190. Even though adding the pulling forces introduced a local maximum during training, the agent does not get stuck inside it. These elements are shown in figure 3, where the reward is smoothed with a moving average.

To speed up training, we moved towards using BC. BC treats the problem of imitation learning as a supervised learning problem. That is to say, given expert trajectories (observation-action pairs), the policy network is trained to reproduce the expert behavior: for a given observation, the action taken by the policy must be the one taken by the expert. To apply this method, we coded by hand an expert that generates good grasps. Then we pre-train our policy network for 3000 epochs on this data before running the standard PPO algorithm. By removing LSTM, normalizing reward and state and using ReLu activation, the policy learns to center the gripper before grabbing the

cube and lifting it. Other than speeding up the learning process, pre-training guides the agent to learn what we would expect it to perform. In fact, in the pure RL case, the agent tends to do unnecessary planar movements after grabbing the cube, which we do not observe in the pre-trained version. These elements are shown in figure 4, where the reward is smoothed with a moving average.

We can now imagine future works on the gripper. So far the gripper learns how to grasp a specific object. However, we can imagine that the gripper grasp different objects. If the gripper had appropriate knowledge about the shape of the handled object, it could learn to generalize to different objects. To extend this work it would be necessary to learn to grab any object from its 3D mesh. One solution could be to construct a good representation of the 3D mesh. For example we can imagine to sample points from the mesh by weighted random selection that selects triangles in the mesh with probability proportional to their areas, and then to select random points inside them. Then, we can use Dynamic Graph [80] with edgeConv (dynamic or fixed) to process these points, extract features and create a good representation of the object. The state of the system would then be the representation of this point cloud (replacing the position of the cube) and the other elements defined in GripperEnv. In addition, constraints on the deformation of the object can be added in the reward to take into account maximum applicable deformations on the objects.

### 5.3. *Compare Quadratic Programming and planning methods with TrunkEnv, DiamondEnv and ConcentricTubeRobotEnv*

Some of the tasks presented in SofaGym have already been solved using optimization, namely TrunkEnv[67] and DiamondEnv.[71] The inverse simulations used in those articles are included in SofaGym in order to compare the performances of learning and planning algorithms to that of quadratic programming (we use QP to refer to the specific quadratic programming algorithm[81] for the inverse problem in soft robotics). In both cases, the goal is to move the effector of the robot (tip of the trunk or the top of the diamond robot) to a predefined target position.

For the purpose of this article we solved the diamond placement task with planning too, specifically using the OPD[66] algorithm. Both quadratic programming and planning manage to place the tip of the robot close to the target in 100% of the cases. Due to the need to construct the planning tree at each planning step, OPD takes around 10 times longer to solve the same trajectory as shown in Table 1. However, since the QP relies on local optimization it can fall into local minima and the mean error of using OPD is around 4 times smaller. The behaviour is similar on the trunk placement task.

While a learned policy could probably match or outperform inverse model control, it would require a long prior training, which quadratic programming does not need. To the authors opinion these examples show that not every problem should be solved using learning or planning techniques. Using these techniques to complement traditional control schemes as we illustrate in section 5.4 seems to be a more promising research direction.

There are however cases where quadratic programming and other optimization based methods will fail to solve the task. One such case is when contacts between the robot and its surroundings make the optimization space non-convex. In those cases, quadratic programming will often get stuck in local minima. This phenomenon is illustrated in the *CartStemContact* environment and the *ConcentricTubeRobot* environment.

RL algorithms can be used in several different ways. In the case where the environment is stochastic, they allow to generalize and obtain a robust controller. But RL algorithms can also be used in the deterministic case to find trajectories to solve a task when optimization techniques like QP do not work. This is the case for example with the *CartStemContact* environment. Suppose initialization and the goal being fixed, the goal being behind the right obstacle and the cart between the two obstacles. There is no randomness in the environment. When we apply the QP from the initial position, the cart moves to the right to minimize the distance to the goal (minimise the objective, given the mechanical state of the robots) and stays blocked at the obstacle. In order to reach the goal

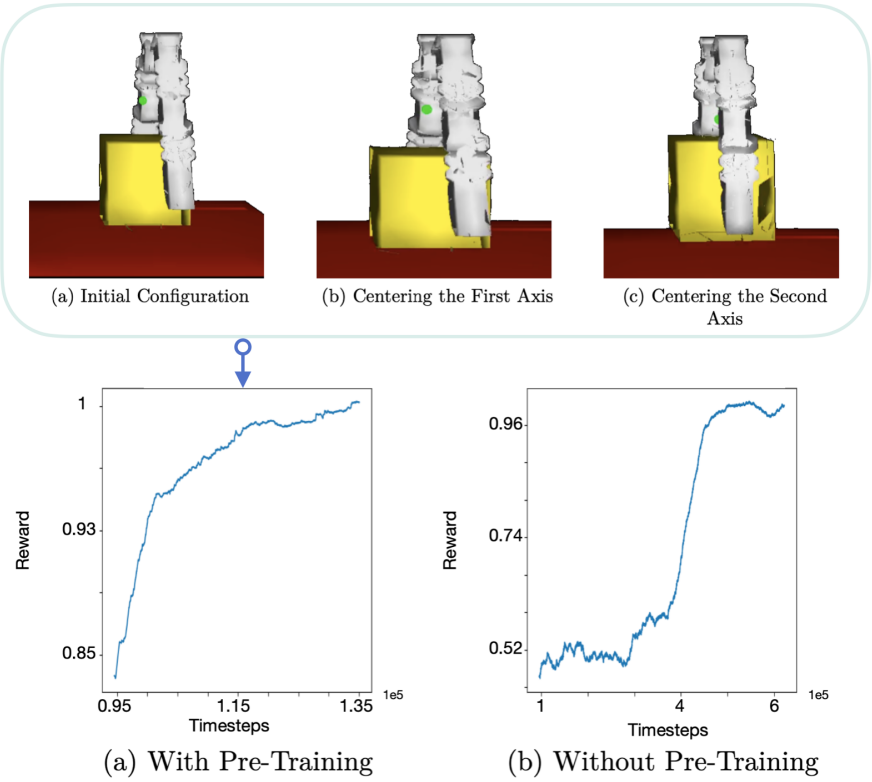


**Figure 4. Learned Policy with BC. BC allows learning to be guided by a known behavior. In the example of the gripper, a hand-designed strategy is implemented to center the gripper on the cube before catching it. The learning starts from this strategy in order to generalize it to different positions of the cube. The steps implemented by the gripper after learning are: starting from any initial position; centering the first axis; centering the second axis; moving the caught cube. In addition the BC speed up the training, as shown in the learning curve with and without Pre-training (rewards over timestep). The goal of this study is to show the advantage of pre-training. The agent performs similaRLy with and without pre- training but learning is much faster with pre-training. Rewards are normalized to the largest value of the plot.**

| Algorithm | Mean Error | Mean Computation Time |
|---|---|---|
| *Quadratic Programming* | 13.24mm | 3.93s |
| *Planning (OPD)* | 3.13mm | 38.24s |

Table 1. Comparison of the error and computation time using Quadratic Programming (QP) and planning (OPD) to solve the Diamond task.

using the QP, the bar must first be in contact with the opposite obstacle. A RL or planning algorithm allows to find a trajectory in which the bar comes to rest on the opposite obstacle in order to be able to touch the goal, which is not possible to do with the QP. In the case of the *CartStemContact* environment we show how learning a policy can solve the task in section 5.1. We solved the *ConcentricTubeRobot* environment using the same planning algorithm[66] as previously. The planning algorithm explores trajectories in the action space to navigate the tip of the robot in the cavity and reach the goal point. The planning algorithm achieves 100% success rate in placing the tip of the robot at different targets in the nasal cavity while Inverse Model control is trapped in local minima. However, the algorithm does not run in real time as it has to stop and plan at every step.

In the case of the trunk manipulating the soft cup (Figure 1.f), Quadratic Programming allows to solve the positionning and orientation problem in real-time. However, finding a way to wrap the trunk around the cup in a way that creates a strong grasp but also gives good controlability

and allows a large range of motion for the cup is not an easy task. Indeed, the position in which the trunk catches the cup conditions the movements of the cup. If the goal is not in the set of positions reachable with this configuration of the trunk, the Quadratic Programming remains stuck in a local minimum since it does not change the way the trunk catches the cup. Though learning how to manipulate the cup will probably result in the same conclusion as for the positioning of the Trunk or the Diamond, learning how to efficiently grasp an object with the Trunk robot and change the grasping if necessary seems to be an exciting research opportunity.

In cases where there is no stochastic effect or break in the optimization space (e.g. with contacts), optimization techniques are the preferred ones. This is what we show with the *Diamond* and *Trunk* examples: planning techniques and optimization techniques lead to the same results, but the optimization techniques are more efficient. If there is uncertainty in the modeling task or some configurations that are not handled by optimisation methods, RL helps to increase robustness of controllers. Domain randomization can help to design controllers that can perform a task reliably, even in unexpected scenarios. We show this behavior whit *CartStemContact* or *Gripper* examples.

### *5.4. Combine planning and inverse model in a sim2real approach with MazeEnv*

To further explore the link between planning and the inverse model, we propose to solve the maze by combining the advantages of both approaches. In order to learn only what is necessary, we will use planning to find an orientation sequence to solve the maze and then we will use an inverse model allowing the tripod to orientate the maze according to this sequence.

First, we used the tree search algorithm OPD[66] to plan trajectories in action space which allows to go from the initial position in the center of the maze to each of the 4 corners. We used the SimpleMaze environment since we just use the orientation of the maze during this first step. In this use case we use tree search planning as a high level control. We then use the inverse model of the tripod robot to convert the maze rotations into servomotor positions.

Using those servomotor positions we can then control the physical tripod to execute the action sequences in open loop and solve the task of navigating the maze in the real woRLd. The use of an inverse model for real robot control has already been proven, as in the case of Trunk control.[69]

In conclusion, the resolution of the maze is composed of three steps: first solve the maze, i.e find the sequence of orientation that solves the maze with the SimpleMazeEnv, then use inverse model to control the MazeEnv and finally use the servomotor position to realize the control of the real maze. An illustration of the two last steps is shown in figure 5.

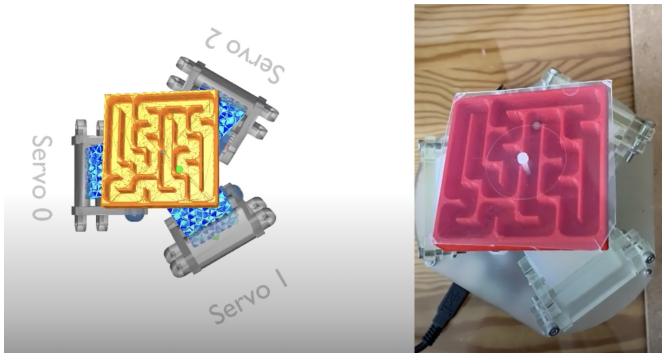


**Figure 5. Solving the maze in simulation with inverse model and in real woRLd with open-loop control.**

This is a simple example for transferring a policy from simulation to reality. Indeed the system dynamics are quite simple and the timestep between taking two actions is long enough to allow the ball to settle in its position before taking the next action. Still, we believe that this example shows how a learned controller can be used in conjunction with more traditional robotics tools to learn a task in simulation and transfer it to reality.

We can imagine future works on the MazeEnv. The maze is solved in simulation and we propose here to use inverse model to solve the problem in the real woRLd. This model is used in an open-loop way and could be extended to a close-loop way. In order to achieve this transfer, it would be necessary to go from the state given by SOFA (position and orientation of objects) to states that can be obtained thanks to real sensors. One possibility would be to use visual data provided by camera: a state learning process would have to be

carried out in order to switch from the image to a state that can be used by our RL algorithm.

### *5.5. Use mechanical modelling with ReducedMultigaitEnv*

The two previous examples have shown the interest of combining inverse model and planning techniques. However, mechanical modelling can be used for more than just direct control. In this section, we want to show that a mechanical model can be used as a representation of the agent's state in an RL problem.

The question of interest is whether a strategy can be learned that allows the MultiGait robot to move along the x-axis as efficiently as a hand-defined strategy. This hand-defined strategy consists of inflating the rear legs, then the central cavity, then the front legs, and finally deflating in the same order. This strategy is shown in the Figure 6.

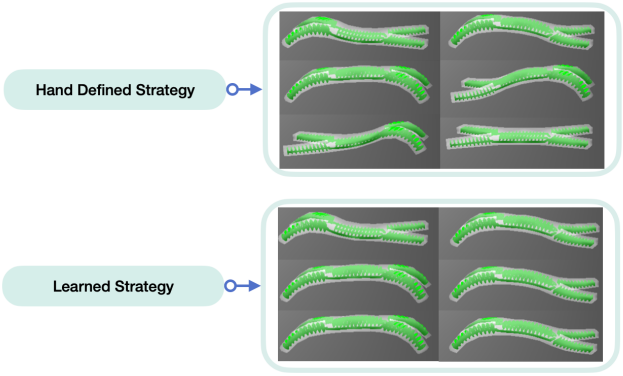


**Figure 6. Steps in the handwritten and learned strategy for solving the locomotion task with the multigait.**

The ReducedMultigaitEnv environment uses a simulation based on a reduced model of the robot. This reduced model is composed of the translation and deformation modes of the robot. These modes can be used to represent the robot, indeed the weight of each mode in the current configuration of the robot is a compact description of the robot state. However, it should be noted that this mechanical model is a reduced model with some limitations: the simulation is only accurate within the scope of the model. In this example, the model does not involve any rotation modes, thus they should not be used in the simulation or in the training. To take this into account, we limit the search to symmetrical actions (joint action of the two front legs and joint action of the two back legs). If the robot is allowed to use asymmetric actuation when the simulation does not allow it, the results are no longer accurate.

To find the strategy, we use PPO with discrete actions to inflate or deflate the cavities. We consider an episode of 6 steps, corresponding to the duration of the hand defined strategy. If $x_t$ is the position of the robot's center of mass at episode $t$, we consider for this task the immediate reward $r_t = max(0, x_t - x_{t-1})$ with $x_0$ the initial robot position. The hyperparameters are given in Appendix C. This task and this environment are deterministic.

The table 2 summarizes the results obtained for the task. The learned strategy inflates all the cavities and then uses the front legs of the robot to move forward. These different steps are represented in the figure 6. The learned strategy is different the one defined by hand and it allows to go further in the x direction. Both strategies are also shown and compared in the accompanying video.

In this example, we used the intensity of 32 deformation modes as the state of the environment. This shows one possible way to reduce the dimension of the state in a convenient and automatic manner, while retaining enough information to learn efficient policies. In the learned policy, the robot uses the contacts of its back legs with the ground to make its front legs slide and move forwards. This strategy is only valid if this technique is possible, i.e. for a given friction coefficient.In the perspective of the transfer between simulation and reality, the friction coefficient can be imposed by changing the environment's surface, which would allow to get closer to the simulation conditions. Moreover, it is not possible to know from the simulation result alone if the learned policy is indeed more efficient than the hand-written one in the real woRLd. Establishing this requires to test and compare the performance of both open loop policies on a physical robot. This is left as future work.

| Strategy | Cumulative Reward | Final x position | |
|---|---|---|---|
| | | Reduced model | Complete model |
| *Hand Defined* | 5.38 | 3.24 cm | 3.56 cm |
| *Learned* | 6.33 | 4.51 cm | 6.21 cm |

Table 2. Comparison of the cumulative reward and final x position, for complete and reduced model, obtained with hand defined and learned strategy, for an episode of size 6.

As a first validation of the policy found on the reduced model, we tested these strategies with the complete model of the robot. With the complete model, the hand-defined strategy achieves 3.56 cm and the learned strategy 6.21 cm. Though the trend is the same, i.e. the learned strategy is better in terms of distance than the hand-defined strategy, there is a difference between the results obtained with the reduced model and those obtained with the complete model. This can be explained by the fact that the model reduction gives a close solution of the deformations, but there are important differences on the boundary conditions (the contacts) due to small errors made in the reduced model. In particular, the reduced model removes the local deformations that are important for contact. In this experiment, the results are rather favorable because the learned strategy remains valid on the complete model and it is obtained in much less time. The reduced environment on which the policy is learned is 21 times faster than the environment with the complete model.

Validating the strategy learned on the reduced model by a simulation on the complete model is undoubtedly an essential step to avoid cases where the learning would exploit errors introduced by the reduction.

For the Multigait, further work should be done to propeRLy choose the simulation parameters and to match the simulation actions with the actions that can be performed on the physical robot. These elements would be necessary for a transfer learning step. Randomization could also be applied to the friction coefficient to try to learn a more robust policy. The environment can be made more challenging by switching to continuous and asymetric actions. This would however require to compute a new reduced model including rotation modes. The task could also be made more complex by giving a random position as an objective rather than simply moving forward. Finally, another element that has not been treated here is the justification (or not) of the use of the MOR to represent the robot state. Indeed, other states could be used to describe the robot: an interesting study would be to investigate the impact of these representations on the learning behaviour. To carry out this study, we would need different SOFA scenes using MOR to generalize the interest of this representation.

## 6. Disussion and Perspectives

RL algorithms have been popularized on the ability to make decisions and select optimal actions from input data. The particular context of robotics is the fact that these input data correspond to real sensor measurements and that the decisions taken by the algorithm will possibly transform the state of the robot and especially of its environment. The even more particular context of soft robotics comes from the fact that the state space to be observed for the robot is very large and that actuation is often redundant.

As presented in the paper, there are many open questions when applying RL to soft robot control: what part of the robot state should be observed on a soft robot? How can one ensure that the robot state is observable? How can an optimal action strategy then be found? How does the robot perceive its environment? Can active strategies to better perceive the environment be learned? Techniques other than RL can also be explored: control theory approaches based on a reduced size model and a state observer for instance.[82] But these methods do not easily comply with contacts of the robot on its environment.

In this paper, we propose a way to explore this still very open field of research, through the use of physical simulation. The interest of physical simulation is that it allows to easily experiment learning algorithms in simplified environments, with

possibly simplified models of robots possibly in contact with an environment. One can thus relatively easily generate large amounts of data, which is often the key to these algorithms, without being blocked by experimental feasibility.

The difficulty is often to pass the validation step, also called Sim2Real. But our approach is based on FEM whose validation methods are well known in engineering sciences. The constraint is rather on the compromise between computation time and accuracy. We see, on the relatively simple examples of this article, that the volumes of simulated data necessary to obtain an efficient and stable control policy (when it is obtained) are already very large and the numerous parameters of these algorithms are difficult to adjust. If one increases the size of the simulated models for more accuracy, the time required for the learning algorithms could become off-putting. It is therefore useful to have an environment like SofaGym where one can adjust the trade-off between accuracy and computation time.

Moreover, the use of this framework has allowed the development of original approaches, presented in this paper, which are avenues for the scientific challenge of soft robot control: reducing the size of models (as in *ReducedMultigaitEnv*) or combining learning with inverse models (as in *MazeEnv*). SofaGym is thus an adaptive tool to explore learning methods on soft robots in environments of variable complexity. It is an open tool, available to the community, on which we already propose a set of easily accessible and reproducible examples (that could be extended in the future) on which we can compare approaches.

## 7. Conclusion

SofaGym is a new and open source platform. It is intended to help bridge the gap between soft robotics and RL by allowing to seamlessly train RL algorithms on simulated soft robotics tasks, and by allowing soft robotics researchers to integrate new tasks and robots as environments. We highlighted numerous challenges and research opportunities created by applying RL to soft robotics, for both the RL and the soft robotics communities. These challenges are then illustrated by 11, ready to use Gym environments, based on simulations of well-known soft robots. We then present examples of trained RL agents to set a first performance baseline for future research to compare to, to show one way of tackling some of the challenges showcased in the environments, but also to start a discussion on which tasks or subtasks are worth learning. We use these last examples to illustrate what is to us the most exciting research perspective: combining learning and planning techniques with existing control schemes. We finally discuss how the problem of perceiving the environment and reconstructing a state from sensor data is an open question and in the authors' opinion an exciting research perspective for both fields.

In this paper we focus on three major elements.

First, the simulation speed is an important issue. To address this problem, we proposed three different approaches: the use of sample efficient algorithms, demonstrated by the AVEC algorithm used for the baseline, the use of a reduced model for the simulation, as shown with the Multigait robot, and the learning of only one part of the problem in the case of the Maze. There are still many ways to improve simulation speed, including the use of GPU[83] or Machine Learning[84,85] based solvers, which will be explored in the future.

The second important point is the transfer from simulation to reality. The validation of the learning on a physical robot is a crucial point. We have done a validation on the maze but it is cleaRLy a simple and favourable case. On the Multigait robot, the transfer from the reduced to the complete model gives promising results, but should be further researched by designing an experiment on a physical robot. Recreating the state from sensor data is also a crucial point for successful closed loop transfer from simulation to reality. We did not explore this point in the examples but we give perspectives in the discussion.

The last point concerns the idea of combining learning and mechanical techniques to control robots. We have highlighted the interest of classical controllers with the examples of the Trunk and Diamond robots and we show one way to combine a planning controller with a traditional inverse model controller to solve the maze. The general idea is to learn a high level control (global

motion) and to perform a low level control (actuator motion) using known and mastered control techniques.

The SofaGym platform was designed to work with any SOFA simulation of a soft robot with little adjustment (see appendix A). The aim of this paper is to present the contribution of a unified open source framework for research mixing RL, planning and soft-robotics with SOFA. Relying on SofaGym allows to model and simulate complex and realistic environments while avoiding learning constraints of real robots (learning time, access to simulation data that are not easily accessible in reality, etc). The source code is available [3] in the hope that other researchers using RL and SOFA may use it and build their own applications on it. The platform is open source so that the community has full access to the tool, can appropriate it and improve it. In the long term, it is even possible to compare learning methods for certain tasks, using robot simulations as a reference.

[3] here we will add the github url of the source code

## A. Tutorial: Integrating a new SOFA Simulation to create a new SofaGym Environment

SofaGym is intended as a software infrastructure which allows to create a gym environment out of any SOFA simulation. In this Appendix we outline the main required components.

This section assumes that the reader has access to a working SOFA simulation which will be turned into the new environment.

SofaGym environments are comprised of three python files:

- An Environment
- A Toolbox
- A SOFA Scene

The Environment file implements the new Environment class, which inherits from *AbstractEnv*. In this file the user should define the default configuration of the environment and implement 3 methods. In the *__init__()* method the user should define the action space and observation space. The *reset()* method should reset the environment to it's initial state and start a new simulation. The initial state can include random element. Finally the *get_available_actions()* method should output the actions which the agent can choose from, given the state of the environment.

The Toolbox file should implement at least three methods. *get_state* returns the state of the system given access to the root node of the SOFA simulation. *get_reward* returns the reward associated with the last action chosen given access to the root node of the SOFA simulation. *apply_action* takes as an argument the action chosen by the agent and adds an animation to the simulation in order to run the corresponding transition.

The SOFA Scene is your simulation scene with an extra Mechanical Object representing the goal to reach and two extra controllers. The first controller is used to move the goal Mechanical Object. The second is used to compute the reward based on elements of the simulation.

These are the minimum elements needed to get a new environment working. Additional methods should be implemented to access more features such as visualization or storing intermediate states to the memory. This information is available in the SofaGym documentation.

SofaGym is open source and available on github[4]. If you use this tutorial to create new and challenging environments, please consider submitting a Pull Request to make those environments available to a larger community.

## B. Technical considerations

Planning algorithms require being able to return to intermediate nodes at any time to extend them and simulate new transitions. This means that it is necessary to store the simulation state corresponding to every node in order to be able to launch a simulation from this point and run the next transition. In the case of RL, such a feature is also useful to visualize intermediate steps taken by a policy.

It is currently not possible to load a SOFA simulation from a stored state other than the initial state and run the simulation. This is an intrinsic limitation of SOFA due to how variables are shared, and not shared, between the C++ simulation framework and the SofaPython3 interface which is used in SofaGym. SOFA allows access to collect and store the positions and velocities of each object. A new simulation can then be created and these stored positions and velocities can be applied to objects, this allows to render visually a scene from a stored state. However, because many variables required for the simulation, such as collision responses and internal solver variables, can not be accessed and thus can not be stored, it is not possible to run the simulation from a stored state. SOFA is open source and our team is working on adding this feature in the future.

To bypass this limitation we propose two workarounds. These are intended as temporary and a simpler architecture could be designed once the feature allowing to load a scene from a stored state and run the simulation is integrated into SOFA.

First, we separate the calculation elements and the visual elements into two different sofa scenes. The scene with the calculation elements is the mechanical simulation of the robot and contains the mechanical solvers and the mechanical mod-

[4]here we will add the github uRL of the source code

els of objects. It is used to compute the physical behaviour and interactions of the objects and returns the state: positions, velocities and other relevant information and data specific to each environment. The visual scene only contains the visual models of the objects. The state is passed either directly from the mechanical simulation to this scene with a geometrical correspondence or loaded from memory, and the visual scene is used to visually display the state. This visual scene allows to render the mechanical behavior during training, inference or planning. It also allows to visualize a state stored in memory for offline inspection.

Second, we designed a server-worker architecture. The server takes care of distributing the calculations between several clients and centralizes the results. Each worker $i$ is associated with an action sequence $A_i = [a_{i1}, ...., a_{in}]$. Given an action sequence $A = [a_1, ...., a_n]$ and a new action $a_{n+1}$, the server searches for the client with the action sequence $A_i = A$. This client forks and the child executes the new action $a_{n+1}$. The parent and son processes are referenced to by the server as two separate clients and both action sequences $[a_1, ...., a_n]$ and $[a_1, ...., a_n, a_{n+1}]$ can be accessed. These clients are full SOFA *calculation* simulations. They store the full state of the simulator since they are complete simulations, and thus simulation can be resumed from this state. The main drawback of this system is its RAM consumption. If too many clients stay opened the OS will start memory swapping which results in a major loss of performance.

In practice, when training RL agents, there are only as many clients opened at one time as there are parallel processes used for training. In this cases the positions and velocities are stored in ROM to allow for offline rendering and inspection but it is not possible to run transitions again from such a stored state. Since the SOFA simulations presented in this article run on CPU and do not leverage multi-threading, usually the training of RL agents in parallel will be done on a vectorized environment comprised of as many environments as there are CPU cores available. This results in a relatively small number of clients opened at the same time (lesser or equal to 40 in our experiments), which does not cause memory issues.

When running planning algorithms however, the number of opened clients is equal to the number of nodes in the planning tree, which is in the order of $b^d$ where $b$ is the branching factor (number of discrete actions) and $d$ is the depth of the tree. This can quickly fill the RAM and cause memory swapping. In order to avoid this, a cleaning system is implemented to close clients which are the less likely to be expanded again. If these closed clients are required later on, it is necessary to load a simulation from the initial state and simulate the entire action sequence leading to this node before being able to simulate the next transition.

## C. Hyperparameters

The hyperparameters used for the baseline are given in the next two tables. We found that the setting in Table 2 work well for all the environments:

| **Hyperparameter** | **Value** |
|---|---|
| $\gamma$ | 0.99 |
| *Learning rate* | 1e-4 |
| *Number of layers* | 3 |
| *Size of layers* | 512 |
| *AVEC bias coefficient* | 0 |

Table 3. General hyperparameters to solve the baseline.

The batchsizes and the scale factors used in the environments of the baseline are given in Table 3:

| **Environment** | **Batchsize** | **Scale factor** |
|---|---|---|
| *CartStem* | 256 | 10 |
| *CartStemContact* | 200 | 30 |
| *CatchTheObject* | 256 | 10 |

Table 4. Specific hyperparameters to solve the baseline.

The hyperparameters used for the gripper are given in Table 4:

| **Hyperparameter** | **Value** |
|---|---|
| *LSTM size* | 128 |

Table 5. Hyperparameters to solve the gripper.

The hyperparameters used for the multigait are given in Table 5:

| Hyperparameter | Value |
|---|---|
| *Policy network* | MLP |
| *Number of layers* | 3 |
| *Sizes of layers* | [256, 128, 64] |
| *Number of mode (simulation)* | 63 |
| *Number of mode (state)* | 32 |

Table 6. Hyperparameters to solve the multigait.

The episode size for each environment is given in Table 5:

| Environment | Max episode size |
|---|---|
| *CartStemContact* | 30 |
| *CartStem* | 80 |
| *CatchTheObject* | 30 |
| *Gripper* | 250 |
| *Trunk* | 250 |
| *TrunkCup* | 250 |
| *Diamond* | 50 |
| *Maze* | 250 |
| *SimpleMaze* | 50 |
| *Multigait / ReducedMultigait* | 6 |

Table 7. Episode size for each environments.

## References


[1] M. Runciman, A. Darzi, and G. P. Mylonas, "Soft robotics in minimally invasive surgery," *Soft robotics*, vol. 6, no. 4, pp. 423–443, 2019.

[2] T. Greigarn and M. C. Cavusoglu, "Task-space motion planning of MRI-actuated catheters for catheter ablation of atrial fibrillation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Chicago, IL, USA), pp. 3476–3482, IEEE, Sept. 2014.

[3] M. Khoshnam and R. V. Patel, "A pseudo-rigid-body 3R model for a steerable ablation catheter," in *2013 IEEE International Conference on Robotics and Automation*, (Karlsruhe, Germany), pp. 4427–4432, IEEE, May 2013.

[4] Y. Ganji, F. Janabi-Sharifi, and A. N. Cheema, "Robot-assisted catheter manipulation for intracardiac navigation," *International Journal of Computer Assisted Radiology and Surgery*, vol. 4, pp. 307–315, June 2009.

[5] Hesheng Wang, Weidong Chen, Xiaojin Yu, Tao Deng, Xiaozhou Wang, and R. Pfeifer, "Visual servo control of cable-driven soft robotic manipulator," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Tokyo), pp. 57–62, IEEE, Nov. 2013.

[6] A. D. Marchese, K. Komorowski, C. D. Onal, and D. Rus, "Design and control of a soft and continuously deformable 2D robotic manipulation system," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, (Hong Kong, China), pp. 2189–2196, IEEE, May 2014.

[7] V. Falkenhahn, A. Hildebrandt, R. Neumann, and O. Sawodny, "Model-based feedforward position control of constant curvature continuum robots using feedback linearization," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, (Seattle, WA, USA), pp. 762–767, IEEE, May 2015.

[8] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik, *et al.*, "Sofa: A multi-model framework for interactive physical simulation," in *Soft tissue biomechanical modeling for computer assisted surgery*, pp. 283–321, Springer, 2012.

[9] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, vol. 9, pp. 51416–51431, 2021.

[10] I. ANSYS, "Ansys Mechanical." https://www.ansys.com/fr-fr/products/structures/ansys-mechanical, 2021.

[11] D. Systemes, "Abaqus." https://www.3ds.com/fr/produits-et-services/simulia/produits/abaqus/derniere-version/, 2021.

[12] C. Inc, "Comsol Multiphysics." https://www.comsol.com/comsol-multiphysics, 2021.

[13] J. Cao, L. Qin, J. Liu, Q. Ren, C. C. Foo, H. Wang, H. P. Lee, and J. Zhu, "Untethered soft robot capable of stable locomotion using soft electrostatic actuators," *Extreme Mechanics Letters*, vol. 21, pp. 9–16, 2018.

[14] Z. Wang, R. Kanegae, and S. Hirai, "Circular shell gripper for handling food products," *Soft Robotics*, 2020.

[15] E. B. Joyee and Y. Pan, "A fully three-dimensional printed inchworm-inspired soft robot with magnetic actuation," *Soft robotics*, vol. 6, no. 3, pp. 333–345, 2019.

[16] M. Rogóż, H. Zeng, C. Xuan, D. S. Wiersma, and P. Wasylczyk, "Light-driven soft robot mimics caterpillar locomotion in natural scale," *Advanced Optical Materials*, vol. 4, no. 11, pp. 1689–1694, 2016.

[17] N. Cheney, R. MacCurdy, J. Clune, and H. Lipson, "Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding," *ACM SIGEVOlution*, vol. 7, no. 1, pp. 11–23, 2014.

[18] S. Kriegman, A. M. Nasab, D. S. Shah, H. Steele, G. Branin, M. Levin, J. Bongard, and R. Kramer-Bottiglio, "Scalable sim-to-real transfer of soft robot designs," *CoRR*, vol. abs/1911.10290, 2019.

[19] SOFA Consortium, "Simulation open framework architecture." https://github.com/sofa-framework/sofa, 2020.

[20] Team DEFROST (INRIA/CRISTAL) Lille, "Softrobots plugin for sofa." https://github.com/SofaDefrost/SoftRobots, 2019.

[21] E. Coevoet, T. Morales-Bieze, F. Largilliere, Z. Zhang, M. Thieffry, M. Sanz-Lopez, B. Carrez, D. Marchal, O. Goury, J. Dequidt, *et al.*, "Software toolkit for modeling, simulation, and control of soft robots," *Advanced Robotics*, vol. 31, no. 22, pp. 1208–1224, 2017.

[22] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, "Gotta learn fast: A new benchmark for generalization in rl," *arXiv preprint arXiv:1804.03720*, 2018.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] D. Hafner, T. P. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," *CoRR*, vol. abs/1811.04551, 2018.

[25] L. Pinto and A. Gupta, "Supersizing Self-supervision: Learning to Grasp from 50K Tries and 700 Robot Hours," *arXiv:1509.06825 [cs]*, Sept. 2015. arXiv: 1509.06825.

[26] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection," *arXiv:1603.02199 [cs]*, Aug. 2016. arXiv: 1603.02199.

[27] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," *CoRR*, vol. abs/1606.07419, 2016.

[28] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning Dexterous In-Hand Manipulation," *arXiv:1808.00177 [cs, stat]*, Jan. 2019. arXiv: 1808.00177.

[29] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving Rubik's Cube with a Robot Hand," *arXiv:1910.07113 [cs, stat]*, Oct. 2019. arXiv: 1910.07113.

[30] S. Bhagat, H. Banerjee, Z. Ho Tse, and H. Ren, "Deep Reinforcement Learning for Soft, Flexible Robots: Brief Review with Impending Challenges," *Robotics*, vol. 8, p. 4, Jan. 2019.

[31] H. Choi, C. Crump, C. Duriez, A. Elmquist, G. Hager, D. Han, F. Hearl, J. Hodgins, A. Jain, F. Leve, C. Li, F. Meier, D. Negrut, L. Righetti, A. Rodriguez, J. Tan, and J. Trinkle, "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward,"

*Proceedings of the National Academy of Sciences*, vol. 118, no. 1, 2021.

[32] R. A. Brooks, "Artificial life and real robots," *Proceedings of the First European Conference on artificial life*, pp. 3–10, 1992.

[33] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of Real-World Reinforcement Learning," *arXiv:1904.12901 [cs, stat]*, Apr. 2019. arXiv: 1904.12901.

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

[35] OpenAI, "Openai leaderboard." https://github.com/openai/gym/wiki/Leaderboard, 2020.

[36] J. Schulman, A. Gupta, S. Venkatesan, M. Tayson-Frederick, and P. Abbeel, "A case study of trajectory transfer through non-rigid registration for a simplified suturing scenario," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4111–4117, 2013.

[37] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[38] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," *arXiv preprint arXiv:1708.05144*, 2017.

[39] H. Zheng, P. Wei, J. Jiang, G. Long, Q. Lu, and C. Zhang, "Cooperative heterogeneous deep reinforcement learning," *arXiv preprint arXiv:2011.00791*, 2020.

[40] OpenAI, "Openai baselines." https://github.com/openai/baselines, 2020.

[41] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." https://github.com/hill-a/stable-baselines, 2018.

[42] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033, IEEE, 2012.

[43] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning." http://pybullet.org, 2016–2021.

[44] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," *CoRR*, vol. abs/1806.07851, 2018.

[45] Y. Wu, W. Yan, T. Kurutach, L. Pinto, and P. Abbeel, "Learning to manipulate deformable objects without demonstrations," *arXiv preprint arXiv:1910.13439*, 2019.

[46] D. Seita, P. Florence, J. Tompson, E. Coumans, V. Sindhwani, K. Goldberg, and A. Zeng, "Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks," *arXiv preprint arXiv:2012.03385*, 2020.

[47] E. Tagliabue, A. Pore, D. Dall'Alba, E. Magnabosco, M. Piccinelli, and P. Fiorini, "Soft tissue simulation environment to learn manipulation tasks in autonomous robotic surgery," *2020 IEEE International Conference on Intelligent Robots and Systems (IROS). IEEE*, 2020.

[48] Z. Huang, Y. Hu, T. Du, S. Zhou, H. Su, J. Tenenbaum, and C. Gan, "Plasticinelab: A soft-body manipulation benchmark with differentiable physics," 04 2021.

[49] X. Lin, Y. Wang, J. Olkin, and D. Held, "Softgym: Benchmarking deep reinforcement learning for deformable object manipulation," in *CoRL*, 2020.

[50] Y. Hu, J. Liu, A. Spielberg, J. B. Tenenbaum, W. T. Freeman, J. Wu, D. Rus, and W. Matusik, "Chainqueen: A real-time differentiable physical simulator for soft robotics," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6265–6271, 2019.

[51] Y. Hu, L. Anderson, T.-M. Li, Q. Sun, N. A. Carr, J. Ragan-Kelley, and F. Durand, "Difftaichi: Differentiable programming for physical simulation," *ArXiv*, vol. abs/1910.00935, 2020.

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

[53] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[54] G. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," tech. rep., University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[55] L.-J. Lin, *Reinforcement learning for robots using neural networks*. PhD thesis, Carnegie Mellon University, 1992.

[56] V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural networks*, vol. 3, no. 6, pp. 671–692, 1990.

[57] C. K. Tham, *Modular on-line function approximation for scaling up reinforcement learning*. PhD thesis, University of Cambridge, 1994.

[58] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[59] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017.

[60] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018.

[61] Y. Flet-Berliac, R. Ouhamma, O.-A. Maillard, and P. Preux, "Learning Value Functions in Deep Policy Gradients using Residual Variance," in *ICLR 2021 - International Conference on Learning Representations*, (Vienna / Virtual, Austria), May 2021.

[62] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez Liebana, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4:1, pp. 1–43, 03 2012.

[63] S. LaVallée, *Planning Algorithms*. Cambridge University Press, 2006. available freely online at http://lavalle.pl/planning/.

[64] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Proc. European Conference on Machine Learning (ECML)*, vol. 2006, pp. 282–293, 09 2006.

[65] E. Leurent and O. Maillard, "Practical open-loop optimistic planning," *ArXiv*, vol. abs/1904.04700, 2019.

[66] J.-F. Hren and R. Munos, "Optimistic planning of deterministic systems," in *European Workshop on Reinforcement Learning*, pp. 151–164, Springer, 2008.

[67] E. Coevoet, A. Escande, and C. Duriez, "Optimization-based inverse model of soft robots with contact handling," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 02 2017.

[68] R. Budhiraja, J. Carpentier, C. Mastalli, and N. Mansard, "Differential dynamic programming for multi-phase rigid contact dynamics," *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pp. 1–9, 2018.

[69] E. Coevoet, A. Escande, and C. Duriez, "Soft robots locomotion and manipulation control using fem simulation and quadratic programming," in *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 739–745, IEEE, 2019.

[70] S. Escaida Navarro, B. Hein, S. E. Navarro, S. Nagels, H. Alagi, L.-M. Faller, O. Goury, T. Morales-Bieze, H. Zangl, B. Hein, R. Ramakers, W. Deferme, G. Zheng, and C. Duriez, "A Model-based Sensor Fusion Approach for Force and Shape Estimation in Soft Robotics,"

*IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5621–5628, 2020.

[71] Z. Zhang, J. Dequidt, A. Kruszewski, F. Largilliere, and C. Duriez, "Kinematic modeling and observer based control of soft robot using real-time finite element method," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5509–5514, 10 2016.

[72] E. Coevoet, A. Escande, and C. Duriez, "Soft robots locomotion and manipulation control using fem simulation and quadratic programming," in *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*, pp. 739–745, 2019.

[73] F. Vanneste, O. Goury, J. Martínez, S. Lefebvre, H. Delingette, and C. Duriez, "Anisotropic soft robots based on 3d printed meso-structured materials: Design, modeling by homogenization and simulation," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 01 2020.

[74] Geuzaine, Christophe and Remacle, Jean-Francois, "Gmsh."

[75] Inria Lille, "Build your own tripod." https://handsonsoftrobotics.lille.inria.fr/index.php/tripod/, 2021.

[76] R. Shepherd, F. Ilievski, W. Choi, S. Morin, A. Stokes, A. Mazzeo, X. Chen, M. Wang, and G. Whitesides, "Multigait soft robot," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 108, pp. 20400–3, 11 2011.

[77] O. Goury and C. Duriez, "Fast, generic, and reliable control and simulation of soft robots using model order reduction," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1565–1576, 2018.

[78] Y. Adagolodjo, F. Renda, and C. Duriez, "Coupling numerical deformable models in global and reduced coordinates for the simulation of the direct and the inverse kinematics of soft robots," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3910–3917, 2021.

[79] O. M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, pp. 20 – 3, 2020.

[80] Y. Wang, Y. Sun, Z. Liu, S. Sarma, M. Bronstein, and J. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions on Graphics (TOG)*, vol. 38, pp. 1 – 12, 2019.

[81] F. Largilliere, V. Verona, E. Coevoet, M. Sanz-Lopez, J. Dequidt, and C. Duriez, "Real-time control of soft-robots using asynchronous finite element modeling," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2550–2555, 2015.

[82] M. Thieffry, A. Kruszewski, C. Duriez, and T.-M. Guerra, "Control design for soft robots based on reduced-order model," *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 25–32, 2018.

[83] H. Courtecuisse, J. Allard, P. Kerfriden, S. P. Bordas, S. Cotin, and C. Duriez, "Real-time simulation of contact and cutting of heterogeneous soft-tissues," *Medical Image Analysis*, vol. 18, pp. 394–410, Feb. 2014.

[84] A. Mendizabal, E. Tagliabue, J.-N. Brunet, D. Dall'Alba, P. Fiorini, and S. Cotin, "Physics-based deep neural network for real-time lesion tracking in ultrasound-guided breast biopsy," in *Computational Biomechanics for Medicine* (K. Miller, A. Wittek, G. Joldes, M. P. Nash, and P. M. F. Nielsen, eds.), (Cham), pp. 33–45, Springer International Publishing, 2020.

[85] A. Mendizabal, P. Márquez-Neila, and S. Cotin, "Simulation of hyperelastic materials in real-time using deep learning," *Medical Image Analysis*, vol. 59, p. 101569, 2020.