Alma Mater Studiorum - Università di Bologna

SCUOLA DI INGEGNERIA
DIPARTIMENTO di
INGEGNERIA ELETTRICA, ELETTRONICA E INFORMAZIONE
"Guglielmo Marconi"
DEI
Master Degree in Automation Engineering

Robotic Manipulation of Deformable Linear Objects: A Model-Free Control Using Reinforcement Learning Algorithms

Master Degree Thesis
in
Autonomous and Mobile Robotics

Supervisor:
Prof. Gianluca Palli

Co-Supervisor:
Kevin Galassi

Candidate:
Mohamed Aboraya

Academic Year
2022/2023
III Session

Abstract

This thesis focuses on the robotic manipulation of deformable linear objects (DLOs) in applications such as assembling deformable wire harnesses and cables in manufacturing. Addressing the limitations in existing studies, this research presents a comprehensive investigation into the perception of DLOs using both single-arm and dual-arm robots. DLOs pose challenges in both perception and manipulation for automated robotic systems due to their lack of distinctive features and intrinsic deformability. The research explores the modeling of DLOs and employs reinforcement learning techniques to tackle tasks such as unknotting, untangling, and shape control. The goal is to contribute to the understanding and application of reinforcement learning in solving challenges related to DLO manipulation. In this work we investigate a new observation method of DLOs and compare its efficiency in relative to the one that is popular in the litrature (images). Then also we see the significance of using different reinforcement learning methods with these observation methods. The use of an RL model like SAC to control a the DLO through a better action space that allows the agent to adabt easily with the environment. Also, the use of an observation space that is more reliable than the images while the robot is interacting with that environment. Finally we present a reliable reward function that can be used for training the agent on the tasks such as achieving a target configuration and a target orientation. The use of this reward function not only has a positive effect on the new observation space investigated here, but also on the one that is popular in the literature.

Contents

List of Figures

Chapter 1

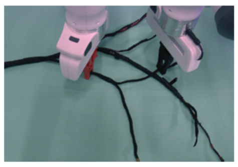
Introduction

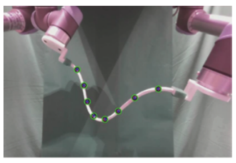
The global goal of engineering research is to reach general methods that leverage physical objects to solve real-world problems. An example of a complex task that now it's gaining more interesting in the industries are the manipulation of deformable linear objects (DLOs) such as cables or wires. These materials are used in various application such as aereospace or automotive and are a valuable and important part of the costs of the final product. Unfortunately, there exist very few automatic solutions that can be used to address the problem, therefore this field of research it's been very explored in the last years. In this thesis, it's been evaluated how to address the problem of cable manipulation from a machine learning perspective, in particular the proposed solution is based on a reinforcement learning.

1.1 DLO modeling

The manipulation of deformable linear objects (DLOs) has always been a major interest of many researchers and engineers. A huge problem in this area is the modeling of deformable linear objects that posses the main characteristics of the real world counterparts. In addition, in the literature, they are usually categorized as *uniparametric* deformable objects [1]. DLOs is part of a generic class of deformable linear objects that consists of wires, cables, strings, ropes, and elastic tubes. They are commonly found in industrial scenarios such as automotive [2] [3], and aerospace [4] industries. DLOs currently represent a complex task for automated robotic systems, both at perception and manipulation levels. Perceiving DLOs presents difficulties due to their inherent lack of distinctive features and deformable characteristics. DLOs are characterized by small dimensions in terms of diameters [5], making an additional challenge concerning their 3-D perception capabilities with most sensors. Unlike dealing with deformable sheets which has larger density in their 3d perception capabilities. From the manipulation side, the DLOs intrinsic deformability results in a high dimensional state space with complex nonlinear dynamics[6]. More information about the model used in the thesis can be found in sec.2 of the chapter relative to the reinforcement learning environment.

(a) unknotting a robe


(b) untangling a wire harness


(c) shape control of a cable

(d) shape control of a rope

Figure 1.1: Example of manipulation of DLOs a)[7], b) authors, c)[8], d)[9]

1.2 Reinforcement Learning techniques

Reinforcement learning is one of the three main machine learning approaches that are currently used nowadays, alongside supervised learning and unsupervised learning as in Figure 1.2. In supervised learning a function that maps some input to output is being learnt by considering a dataset of samples that include the inputs and their corresponding outputs (labels). The AI model, represented by a neural network, is trained on that dataset with respect to a loss function to minimize. The training process is considered successful if it can correctly predict the output (label), especially in an unseen set of input. In contrast to supervised learning, unsupervised learning involves extracting patterns or structures from input data without explicit feedback or labeled outputs. Instead of being provided with labeled examples, the algorithm must infer the underlying structure or distribution within the data on its own. This can involve tasks such as clustering, where similar data points are grouped together, or dimensionality reduction, which aims to capture the most relevant features of the data while reducing its complexity. Unsupervised learning is particularly useful for tasks where labeled data is scarce or expensive to

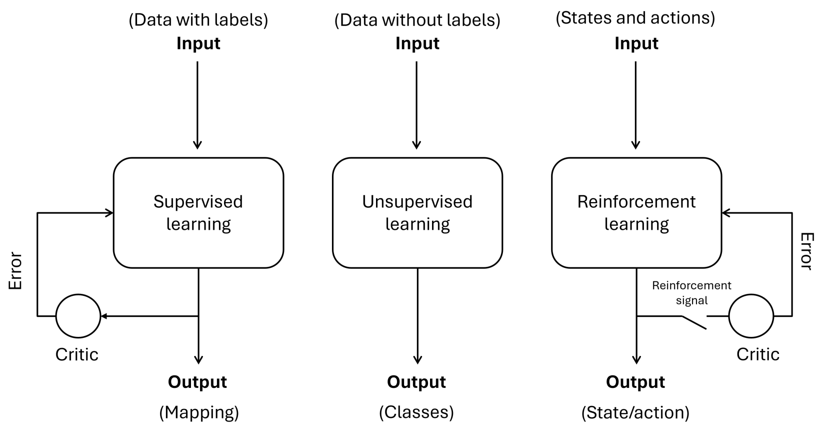


Figure 1.2: The three main approaches in machine learning: Supervise learning, Unsupervised learning and Reinforcement learning.

obtain, and it plays a crucial role in tasks such as anomaly detection, data compression, and exploratory data analysis.

In the case of reinforcement learning (RL), we consider an agent that is actively interacting with an environment. Through its interactions, it is possible that it may influence the environment that it operates in. At each action, the agent receives a reward based on its behavior and the new goal of the optimization, it's now to maximize the expected reward through the interaction with the environment. The "dataset" we need to consider, are the actions our agent took and the accumulated rewards it got by taking those actions. Another challenge here is that the dataset is dynamic. For instance, the agent acts in a certain way, then the collection of some data of the actions that the agent executed and then performing an optimization (e.g. do more of the actions that led to a successful result). But as a result of this optimization, the behavior of the agent is changed. Thus, a collection of the data is needed to evaluate the agent performance. Using the Fig 1.3 as reference, the agent performs an action A_t, as it is the learning entity that makes decision and takes actions in the environment. The external system with which the agent interacts is the environment, which responds to the agent's action, providing feedback in the form of rewards R_t and new states S_t. Where the state S_t is the representation of the current situation or configuration of the environment. The Acton A_t it the set of possible moves or decisions that the agent can take in a given state. The reward R_t is a numerical value that the environment provides to the agent as feedback based on the action taken in a particular state. The agent's objective is to maximize the cumulative reward over time. The strategy or mapping from

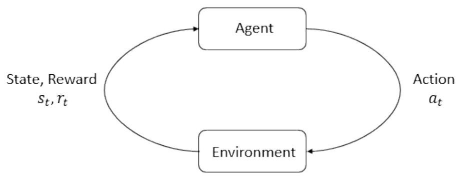


Figure 1.3: Flowchart of an RL algorithm [10]

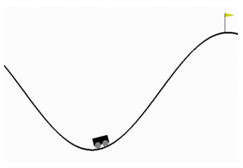
states to actions is called a policy, that policy is used by the agent to make decisions. The goal is to learn an optimal policy that leads to maximum rewards. Another aspect that is always used in RL is the value function, which is a function that estimates the expected cumulative reward of being in a particular state or taking a specific action. It helps the agent to evaluate the desirability of different states or actions. The learning process of the agent is by trail and error, adjusting its policy or value function based on the feedback received from the environment in the form of rewards. This trial and error makes the agent face a dilemma of choosing between exploring new actions to discover their effects and exploiting known actions to maximize immediate rewards. RL problems are often formulated as Markov Decision Processes (MDPs), which consist of states, actions, transition probabilities, rewards, and a discount factor. A single run or sequence of interactions between the agent and the environment, starting from an initial state and ending in a terminal state is called an episode.

Reinforcement Learning is known to be more sample inefficient, where a "sample" is considered a single interaction with the environment, in fact reinforcement learning needs a lot of samples/interactions with the environment to be able to solve a task. This sample-efficiency can in part be explained by the fact that humans can leverage a lot of their previous knowledge (priors) when they encounter a new task. A human can for example reuse some of the knowledge and skills of previous games and/or concepts they already acquired from other experiences throughout their life. An RL-agent in contrast, starts the learning process without any assumptions. Another Problem to be considered with RL is the exploration-exploitation trade-off. Whenever an RL-agent is trained, the agent always needs some time to explore, it needs to explore the action space by executing actions that it hasn't taken before, in order to discover how to solve the problem. On the contrary, the agent can't always take random actions, because these random actions might lead to nothing. Sometimes we want the agent to leverage what it has already learned to try and optimize further. This exploration-exploitation trade-off needs to be automated to strike a good balance between letting the agent explore and taking actions for which it already knows what they will lead to. A further fundamental problem with RL is the so called Sparse-reward problem. As the name implies, this problem occurs when our RL-agent receives sparsly reward during the interaction with the environment, meaning that it won't receive a reward for all the state visited. Differently, in a dense-reward scenario, the agent will receive a reward different from zero for each action. Considering the mountain car environment [11] in OpenAI Gym which is a classic reinforcement learning problem that involves a car positioned between two hills Figure 1.4b.

The goal is to enable the car to reach the flag at the top of the right hill. The state space is continous and defined by two variables, the position and the velocity, and the action space is also continous, representing the force applied to the car with range from -1 to 1. The reward for this example is sparse-reward as the agent recieves a reward of -1 for each time step until the episode terminates. The goal of that reward design is to encourage the agent to reach the flag at the top of the right hill as quickly as possible. Reaching the flag yeilds a reward of 0. The episode terminates if the care's position exceeds some value 0.6. The car dynamics are simulated based on physics. When the agent applies force it moves the car, the gravity and friction affect its motion. Starting state is a random position with zero velocity. The agent must learn to apply the right amount of force in the right direction to move the car up the left hill and reach the flag on the right hill. however the challenge lies in the fact that the car's engine is not strong enough to reach the flag directly, so the agent needs to learn to build momentum by going back and forth between the hills.

An example of an environment with a dense reward function is the "pendulum-v0" environment in Open-AI Gym Figure 1.4a. The agent in that environment controls a pendulum, and the goal is to swing it up and balance it at the upright position. The state space is continuous and consists of the pendulum's angle an angular velocity. The action space is also continuous, representing the torque applied to the pendulum. The reward function is dense and is designed to encourage the agent to balance the pendulum. The agent receives negative rewards based on the angular distance from the upright position and the angular velocity. The goal is to maximize the cumulative reward over time. The episode terminates after a specified number of steps. The dynamics are governed by the physics of a simple pendulum The starting state for this system is random at the beginning of the episode. The agent needs to learn a policy that applies torque in a way that minimizes the angular distance from the upright position and stabilizes the pendulum. The dense reward function provides more informative feedback to the agent, allowing it to learn a nuanced strategy for controlling the pendulum.

(a) Pendulum demo



(b) Mountain car demo

Figure 1.4: Example of environment with Dense (Left) and Sparse (right) rewards

Chapter 2

Training Environment

The first element of a reinforcement learning based application is the definition of the environment. The environment is fundamental since it's define how the agent can interact with the world and define the reward based on the agent's action. From a practical implementation perspective, nowadays, the Open AI-gym[12] library is become the standard frameworks used for this application, and it is used to create the interactive environment for the simulation and the learning process of the RL-Agent. It's worth noticing that we could design a whole environment without the Open AI-gym framework, however gym provides a generic stable interface that can be easily integrated with other different RL- libraries. It offers a standardized interface for interacting with environments, which makes it easier to compare and reproduce results across different algorithms and research papers. Reproducibility and sharing of an environment created in OpenAI Gym is easy and well-known in the research community, enabling others to reproduce your results and build upon your work. Also, some RL libraries like stable-baseline[13], RLlib or tf-agents[14] can be easily integrated with OpenAI-Gym environments and basic to advanced RL algorithms can be used to train the agents with ease (without coding from scratch).

OpenAI Gym is an open-source toolkit developed by OpenAI that provides a standardized environment for developing and testing reinforcement learning algorithms. It offers a variety of environments with different tasks and challenges, making it a popular choice for researchers, educators, and developers interested in reinforcement learning. OpenAI Gym defines a set of environment, each representing a specific task or problem for reinforcement learning. Examples include classic control problems like Cartpole or MountainCar and more complex environment based on the Atari 2600 games. Gym provides a simple interface for interacting with environments, making it easy to experiment with different algorithms. This interface includes methods for taking actions, receiving observations, and obtaining rewards. Environments in Open-AI Gym have well-defined observation spaces, which represent the information available to the agent at each time step. Observations can be continuous or discrete. Similarly, environments have action spaces that define the possible actions an agent can take. Actions can also be continuous or discrete. The reward system is a critical component in reinforcement learning, and OpenAI Gym provides a reward mechanism to evaluate the performance of agents. The goal of reinforcement learning is often to design agents that maximize cumulative rewards over time. Open-AI Gym is widely used as a benchmark for testing and comparing different reinforcement learning algorithms. This allows researchers and developers to access the performance of their methods on standardized tasks. The Users can create custom environments by implementing the Gym's environment interface. This flexibility makes it easy to adapt the toolkit to new problems or experimental setups. It provides a variety of reinforcement learning algorithms as well as baseline for benchmarking. This allows users to compare their custom algorithms with established ones. The toolkit is compatible with popular reinforcement learning libraries, such as TensorFlow[15] and PyTorch[16], making it convenient for users to integrate their favorite machine learning frameworks. Open-AI Gym has extensive documentation, tutorials, and an active community. This makes it accessible for users at various levels of expertise, from beginners to experienced researchers.

To create a custom environment, we just need to override existing function signatures in the gym with our environment's definition. These functions that we necessarily need to override are:

- `__init__()`: This function initializes the environment with default values.
- `reset()`: This function is for resetting the environment to the default settings.
- `step()`: This function executes how the environment will change once the agent takes an action. Usually, the reward function is also incorporated or called within `step()`.
- `render()`: For rendering the environment.

1 Pick and Place simulation

In this research, we tackle the problem of a robotic pick and place of a deformable linear object in which the robot has to decide which action to perform to straight a cable or a rope. Since The environment is following the same structure of the environments in Open-AI Gym, then in this section we shall shed the light on how the aspects of this structure is formulated.

The initialization of the environment in the Algorithm 1 starts with loading the parameters of the model dynamics.

The controller for the dynamics has the following parameters, the maximum number of steps for moving to target position is 1000, the maximum displacement to consider at the moving node is 0.01,the maximum workspace before calling the reset function is (2.0, 2.0) for planar motion, and the maximum moving velocity is 0.1. The position controller is necessary in the simulation since the RL agent chooses only the final displacement of the chosen node as an action in each step executed in the environment. The parameters for this position controller are a proportional gain $K_p = 700$, integral gain $K_i = 500$, feedforward gain $K_f = 0.75$, duration of the force profile in seconds $ff_{len} = 0.2$, the time step size in seconds for the force profile $ff_{\delta_t} = 0.0005$, the maximum number of steps for moving to target position is 1000, the maximum displacement to consider at the moving node is 0.1,the maximum workspace before calling the reset function is (2.0, 2.0) for planar motion. After loading the parameters the DLO model is constructed and initialized to a random shape with a randomness parameter σ which in this case has a value of 0.3. Then the target position is chosen to be some shape which is selected at random out of 4 shapes which are horizontal, vertical, and two diagonal angles. Finally the definition of the observation space and the action space, which is mandatory for this custom environment that is constructed based on the standard structure of OpenAI Gym.

The step function 2 in this simulation is then used for the interaction between the agent and the simulated environment. When the agent supplies the action to this function the action is scaled from the normalized input (usually all the action elements are between zero and one), and the model is moved based on this action which is a selection of a specific node to be moved for some distance in each direction of the planar surface on which the DLO is set. Consequently to this action there is a new observation that is returned from the step function as a feedback to the agent, in addition a reward is returned also from the function to be used by the RL algorithm to evaluate the agent action. At the end there is a condition to check if the environment should be terminated because we reached the final goal or near that goal by some tolerance.

The reset function 3 is to reset the simulated model, update the target shape and set the settings to the default parameters as it was in the initilization function. The reset function returns the initial state/obseravation to start the episode. the function is always called after the end of the episode.

Algorithm 1 PickAndPlaceEnv Constructor

```
0: procedure __INIT__(params, max_attempt=100, env_param=None, render=False)
1: SUPER(PickAndPlaceEnv) {Initialize superclass}
2: max_attempt ← max_attempt
3: attempt ← 0
4: GNN_RL_MODEL_Path ← os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

5: params_path ← os.path.join(GNN_RL_MODEL_Path, "params", params)
6: with open(params_path) as file:
7:     param_dict ← yaml.load(file, Loader=yaml.FullLoader)
8: n_nodes ← param_dict['model']['nodes']
9: max_disp ← 0.5
10: sigma ← 0.3
11: model ← DLOModel(params=params_path, rendering=rendering)
12: target_pos ← _generate_target_shape(random_int(low = 0, high = 4))
13: model.free(verbose=verbose, skip=100, steady_state_threshold=0.00005)

14: action_space ← Box(low=0, high=1, shape=(3,), dtype=float32)
15: observation_space ← Dict({'node_poses': Box(low=-inf, high=inf, shape=($N_{nodes}$+2, 2)),
15:         'target_poses': Box(low=-inf, high=inf, shape=(model.nodes+2, 2))})
15: end procedure=0
```

Algorithm 2 PickAndPlaceEnv Step Method

```
0: procedure STEP( action)
1: attempt ← attempt + 1
2: old_pos ← pos.copy()
3: {Compute scaled action from normalized input}
4: scaled_action ← {'node_id' : round(action[0] * (n_nodes - 1)),
5:     'dx': action[1] * max_disp * 2 - max_disp,
6:     'dy': action[2] * max_disp * 2 - max_disp}
7: moving_node ← scaled_action['node_id']
8: dx ← scaled_action['dx']
9: dy ← scaled_action['dy']
10: if render: then
11:     expected_target_pos ← old_pos[moving_node] + np.array([dx, dy])
12: end if
13: {Apply control action}
14: model.move_idx(node_idx=scaled_action['node_id'], dx=dx, dy=dy)
15: pos ← model.pos[:,:2].copy()
16: {Collect the new Obs / Reward / Info}
17: observation ← _get_obs()
18: reward ← _get_reward()
19: info ← _get_info()
20: {check if the target shape is reached}
21: truncated ← False
22: terminated ← False
23: if reward < 0.2: then
24:     terminated ← True
25:     reward ← 20
26:     attempt ← 0
27: else if attempt >= max_attempt: then
28:     truncated ← True
29:     attempt ← 0
30: end if
31: return observation, reward, terminated, truncated, info
31: end procedure=0
```

Algorithm 3 PickAndPlaceEnv Reset Method

```
0: procedure RESET( seed=None)
1: if seed is not None: then
2:     seed(seed)
3: end if
4: target_pos ← _generate_target_shape()
5: model.reset(time=True, shape=True, sigma=sigma)
6: model.free(verbose=verbose, skip=100, steady_state_threshold=0.00005)

7: free_evolution()
8: pos ← model.pos[:, 0:2].copy()
9: attempt ← 0
10: info ← _get_info()
11: return _get_obs(), info
11: end procedure=0
```

2 DLO Model

Deformable Objects are complex mechanical objects to model, usual technique involve the use of a system of masses connected by spring and dampers. Based on this concept both DLOs and planar objects has been successfully modeled and used in learning application as clothes [17]. The simiulation model used in this research is build based on the work of [18]. Where the model is a nonlinear series of linear system (mass-spring-damper). The masses of the nodes in the model are connected by a series of springs that reflect the internal stiffness of the cable as in eq2.2. In addition the bending stiffness between the segments of the DLO is modeled by placing a torsional spring at each node as in eq2.3. Stability of the model is improved by adding viscous friction proportional to the velocity of the node, which is included as a damping term. The generic dynamics in eq2.1 for each node shows the evolution of the second order system. Here p is the node position coordinates, k_d is a damping constant, f_i^s is the force due to the axial effects, and f_i^b are the forces due to the bending effects. The axial force in 2.2 shows the interaction of spring forces between a node the succissive on. where link i is known for current length which is l_i and an initial length l_i^0. the unit vecto of node i is represented by u_i and β_i is calculate in eq.2.4. The model is very close to the real-world problem, even though it is a little bit different in the sense that the physical design of the robot has a gripper, which usually interacts with a segment of the robot. The decision to perform actions at a single point is not the case for the real-world problem but this is a valid assumptions with our

N_{nodes}	20
K_d	100
K_b	0.01
K_s	999.0
l	0.5 m
m	0.2 Kg
δ	0.001
K_f	0.05

Table 2.1: Cable model parameters used for the experiments

MDP hidden state.

At each iteration of time d_t, the forces in eq.2.2 and eq.2.3 acting on each mass are evaluated using the updated position that is calculated through the integration of the acceleration, it's possible to obtain the new position of each mass.

The final dynamics of the DLO are defined by its parameters that were empirically calculated. The parameters used can be found in Tb.2.1

$$m_i \ddot{p}_i = -k_d \dot{p}_i + f_i^s + f_i^b \tag{2.1}$$

$$f_i^s = -k_s(l_i - l_i^0)u_i + k_s(l_{i+1} - l_{i+1}^0)u_{i+1} \tag{2.2}$$

$$\begin{aligned} f_i^b = k_b \frac{\beta_{i-1}}{l_i \sin \beta_{i-1}} u_i \times (u_{i-1} \times u_i) \\ -k_b \frac{\beta_i}{l_i \beta_i} u_i \times (u_i \times u_{i+1}) \\ -k_b \frac{\beta_i}{i_{i+1} \sin \beta_i} u_{i+1} \times (u_i \times u_{i+1}) \\ +k_b \frac{\beta_{i+1}}{l_{i+1} \sin \beta_{i+1}} u_{i+1} \times (u_{i+1} \times u_{i+1}) \end{aligned} \tag{2.3}$$

$$\beta_i = \arctan \frac{\|u_{i+1} \times u_i\|}{\langle u_{i+1}, u_i \rangle} \tag{2.4}$$

3 Observation Space

The observation space represent the information the agent can be used to learn to solve the environment. In the literature the observation space that is

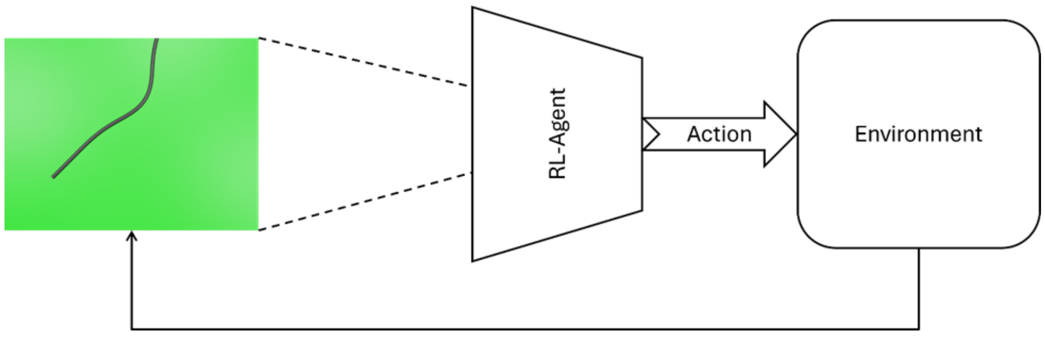


Figure 2.1: Image observation diagram

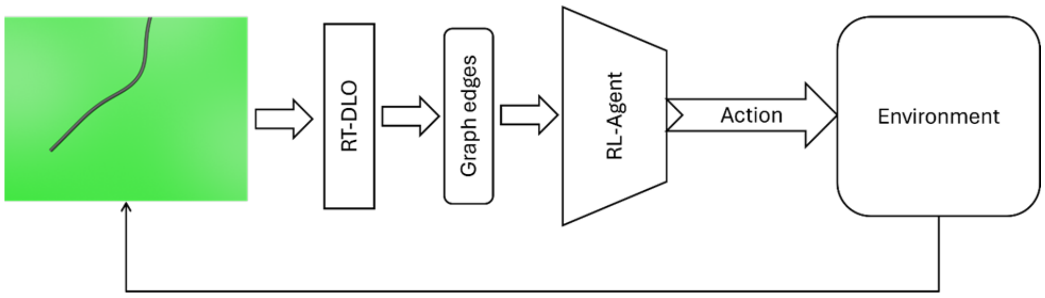


Figure 2.2: Tensor observation diagram

commonly adopted are images where the images are used with convolutional neural networks to estimate the best action to achieve the best next state. Images are commonly used because of the availability of camera devices with diverse characteristics and settings that offer high-resolution data. Apart from camera, there are different types of devices that are used to perceive the DLOs like photoreflectors, or capacitive-based tactile sensors, and force/torque sensors, however these types yield comparatively less rich data than camera-based sensors. In the presented work there are two types of perceptions that are studied and evaluated. The first one is the actual RGB image that is used as an observation for the environment, which can be associated with some perception problems like occlusion. The second one is a graph edges which are selected based on topological reasoning. In figure 2.1 you can refer to the former observation type and for the later you can refer to the figure 2.2.

Image observation can be handled efficiently with CNNs to infer most of the information from the image. On the other hand, graph edges are the 2D points of each node in the image assuming we are only dealing with 2D environment for the sake of simplicity, but it is originally designed to handle the 3D case. The 2D points have more flexibility to be dealt with since it can be handled more efficiently with MLPs to have an observation

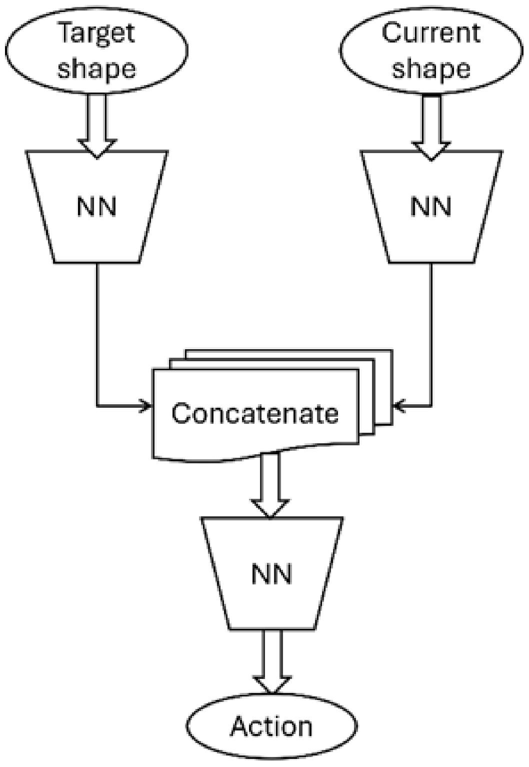


Figure 2.3: scheme for multi-input observation architecture with the NN, either in case of MLPs, or in case of CNNs

that can represent the history of previous observations in a more compact representation. And these observations can be rendered to represent a simpler image without bad effects like occlusion. For both types of observations, we created a multi-input observation for the RL-agent in Figure 2.3 This multi-input observation is a group of the target shape that the agent is supposed to follow to achieve the task and the target shape that represents the current state of the DLO. We also tried to make a single-input observation by calculating the Euclidean distance between the nodes of the current shape and their corresponding nodes in the target shape, that in the case of the observation is a graph of 2d points. But in the case of images as an observation the single input is a little bit involved, as the target shape is rendered to be shown in the same image with target position and orientation that to be reached by the end of the task. By the end of this experiment the multi-input observation is much more efficient in improving the learning curve for the agent.

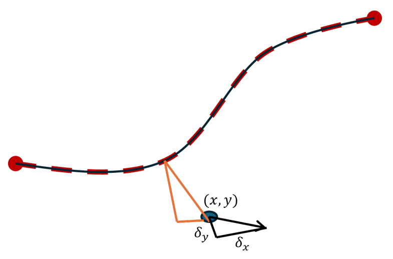
4 Action Space

The action space depends entirely on the observation from the robotic system. For instance, the image observation is always an image from a constant robotic configuration with constant camera settings (camera matrix), and this observation represents the workspace of the robot and the position and the configuration of the DLO inside that space. Hence, the action space, as in the work of Yan, Vangipuram, Abbeel, and Pinto[19], is the absolute position of the node and the perturbation of that node in the configuration space as defined in eq2.5. Where the absolute position can be pixel point in the image, while the perturbation is how far that node is supposed to be moved relative to its current position. This action space can be considered complete in case of dealing the fabric that is supposed to have larger pixel density in the image, but this is not the case for the DLOs as it has small pixel density in the image, so further consideration needs to be given. The action on the DLOs in that case is performed on the nearest node on the DLO to the chosen (x,y) point by the agent, and it is the nearest in the sense of Euclidean distance. This can be demonstrated in fig.2.4a, as the euclidean distance is shown by the orange triangle, and the action that corresponds to the nearest node on the DLO.

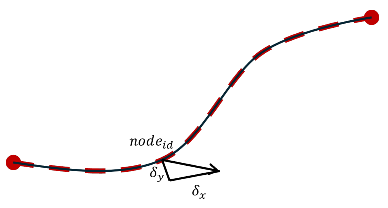
$$a = (x, y, \delta x, \delta y). \tag{2.5}$$

The case when we know the nodes position from the observation that is acquired by the RT-DLO SDK becomes easier to be handled in the action space, because the absolute position is of each node is well known, since that feedback gives the action position of the node in an ordered array, we can use this to make the action space for the agent to be reduced to some extent and become simpler than the one that is proposed for the other observation space that is explained previously. Then the action space proposed in this research for this type of observation is defined in eq2.6. The node to be moved is directly chosen as explained in 2, then the perturbations in both planar directions. An example of the mentioned action is shown in fig.2.4b, and in that example the node is selected directly since it's absolute position is known to the robot's workspace. This simpler action space has effect on the evolution of the agent as it will be shown on the next chapter of results.

$$a = (node_{id}, \delta x, \delta y). \tag{2.6}$$



(a) action taken using image observation



(b) action taken using tensor of points observation

Figure 2.4: Example of action space using image (Left) and using tensor of points (right)

5 Reward Function

For this experiment we used multiple reward functions, some of them are used to make the robot follow the position, orientation, and configuration of the target shape, while others are used to make the robot follow only the configuration of the target shape, and others are used to make the robot follow both orientation and configuration of the target shape. The sum of absolute differences was used to make the robot follow the position, orientation, and configuration of the target shape. This reward function is calculated based on the simulation model that is used during the training process, in this model the position of the nodes is known in advance for both the current observation and the target observation. This function is shown in equation 2.7 This function is always negative function to urge the robot to finish the episode as soon as possible while reaching the target shape.

$$R = -\sum_{i=0}^{n} |x_i - y_i| \tag{2.7}$$

The delta gain difference is another reward function that is used to improve the learning process and it is calculated as the difference between the current sum of absolute differences and the previous one, so the reward is then to penalize the action if it the current absolute difference is not better than the previous one, and to reward the action if otherwise. This function has an order of magnitude better than absolute difference reward function in the sense of the stability of the learning process with less standard deviation.

$$R = \sum_{i=0}^{n} |x_{i,t-1} - y_{i,t-1}| - \sum_{i=0}^{n} |x_{i,t} - y_{i,t}| \tag{2.8}$$

An additional reward function is been evaluated, this reward is obtained by the summation of two term, a first term $P_{i,j}$ which depends on the cables position and an orientation term $O_{i,j}$ which instead depend on the orientation of the objects with respect to the target. The two term are given with a negative sign to help the agent to solve the problem in the minor possible steps.

$$reward = -P_{i,j} - O_{i,j} \tag{2.9}$$

The first part in eq2.10 of this function deals with the shape of the DLO and how far is it from the reference/target shape in terms of the relative position between the nodes in the DLO. Based on formation control theory of distributed systems and the rigidity condition of the formation as elaborated in [20], when the Euclidean distance between each node and all the other nodes in the model is well-defined as a constraint, then the condition of the rigidity of the shape is complete. Therefore, the distance for each node with all the other nodes is compared with the corresponding counterparts in the target shape, the summation of the difference in these distances between the actual shape and the target shape expresses the error in the shape formation.

$$P_{i,j} = \sum_{j=0}^{n} \sum_{i=0}^{n} (|||x_j - x_i||_2 - ||y_j - y_i||_2|) \tag{2.10}$$

The second part of this reward function in eq2.11 is concerned with the orientation of the actual shape in relative to the current shape. It is based on a metric which is the comparison of the absolute angle of the line connecting the two ends of the actual DLO with the absolute angle of the corresponding line between the two ends of the target DLO.

$$O_{i,j} = |\arctan 2(x_{01} - x_{n1}, x_{00} - x_{n0}) - \arctan 2(y_{01} - y_{n1}, y_{00} - y_{n0})| \tag{2.11}$$

This constraint is fine for orientation, since the shape formation is accounted for in the first part. The promising results in the result chapter are based on this reward function.

Chapter 3

Reinforcement Learning Algorithm

The algorithms that were used in this research are a Proximal Policy Optimization (PPO) [21] which is an on-policy gradient method and it is model-free, Soft-actor critic [22] which is an Off-policy gradient method, and it is model-free, and PlaNet [23] which is a model-based method. On-policy methods learn the value or policy function based on the same policy that is being used to generate the data, while the off policy method learn the value or policy function based on a different policy than the one used to generate the data. The agent in the on-policy method learns from its own actions while following its current policy, which indicates that the data used for learning comes from the behavior of the agent under its current policy. On the other hand, the agent in off-policy method learns from experiences generated by a different policy, often using historical data or samples generated by a different agent or policy, the advantage here is that the agent can learn from data collected by a different, possibly more explorative, policy. Figure 3.1 shows that difference in graphical representation.

Regarding accounting for building an explicit model or representation of the environment's dynamics, including transition probabilities and rewards, which is called model-free method, has its own characteristics as the agent uses the learned model to simulate possible future trajectories, enabling it to plan and make decision without directly interacting with the environment. Model-based methods can potentially leverage the acquired model to optimize decision-making. On the contrary, model-free methods directly learn a policy or value function from interacting with the environment without explicitly building a model of the environment's dynamics, these methods focus on learning optimal policies or value functions based on observed experiences (state-action pairs and rewards) rather than trying to understand or represent

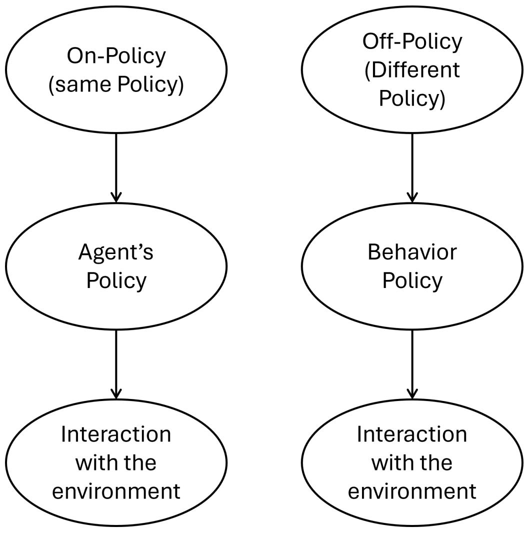

On-Policy (same Policy)
Agent's Policy
Interaction with the environment
Off-Policy (Different Policy)
Behavior Policy
Interaction with the environment

Figure 3.1: Diagram indicates difference between On-Policy and Off-Policy methods

the underlying dynamics of the environment. The choice between model-free and model-based methods depends on the specific characteristics of the problem, the availability of accurate models, and the trade-off between exploration-exploitation. Model-free methods are often preferred when the environment is complex or when obtaining an accurate model is challenging, while model-free methods can be advantageous when an accurate model is available and planning is crucial for decision-making. Figure 3.2 !!! shows that difference in a graphical representation

PPO is designed to address some of the limitations and challenges associated with traditional policy optimization methods. It was introduced by OpenAI and has gained popularity for its stability and efficiency. It aims to optimize policies in reinforcement learning by iteratively updating them to maximize the expected cumulative reward. PPO belongs to the class of algorithms known as policy optimization methods, these methods directly optimize the policy of an agent, which defines the probability of an agent, which defines the probability distribution over actions given a particular state. It prevents large policy updates that might destabilize the learning process by introducing “trust region” approach, this helps to ensure a more stable and controlled learning process. Additionally, it uses clipped surrogate objective function that constraints the policy update, avoiding excessively large policy changes, and ensuring that the policy does not deviate too far from the previous iteration. Moreover, it uses multiple iterations of collecting samples from the environment, computing advantages, and updating the

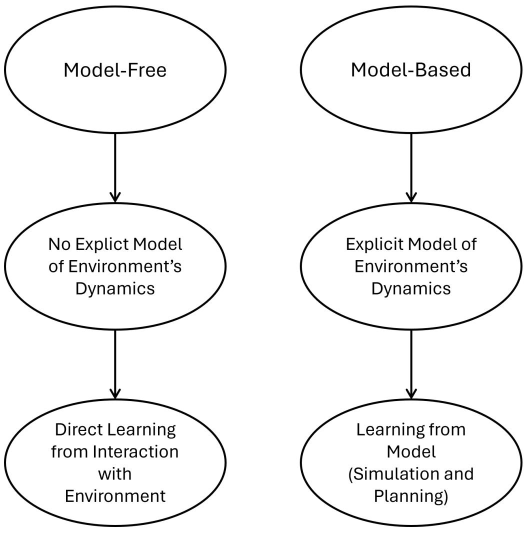

Model-Free
Model-Based
No Explict Model of Environment's Dynamics
Explicit Model of Environment's Dynamics
Direct Learning from Interaction with Environment
Learning from Model (Simulation and Planning)

Figure 3.2: Diagram indicates difference between model-free and model-based approaches

policy. This process is repeated to iteratively improve the policy. Compared to some other policy optimization methods PPO is known for its stability and sample efficiency. The use of the clipped surrogate objective and trust region constraint contributes to this stability. Deep reinforcement learning tasks can be solved using PPO, as it can be combined with deep neural networks to handle high-dimensional state spaces and complex environments. PPO can be viewed in algorithm 4.

SAC is always used for training agents to learn optimal policies in continuous action spaces. It is known for its effectiveness in handling complex and high-dimensional environments. Earlier it was mentioned that SAC is an off-policy algorithm, meaning that it learns from a replay buffer that stores experiences collected from the agent's interaction with the environment. This allows for more efficient use of past experiences. SAC employs a variant of Q-learning called soft Q-learning, which involves minimizing the soft value function, where that "soft" aspect refers to using a temperature parameter that controls the degree of softness in the maximization operation. Furthermore, another distinctive feature of SAC is the inclusion of an entropy term in the objective function, where this entropy regularization encourages the agent to explore more diverse and leads to a better trade-off between exploration and exploitation. The actor-critic architecture is used in this algorithm. The actor is responsible for selecting actions, while the critic evaluates the value of the chosen actions, and both are neural networks that are trained jointly. Stability of the learning is achieved through using the

Algorithm 4 Proximal Policy Optimization (PPO)

```
1: Initialize policy parameters $\theta$
2: Set the number of iterations $T$
3: Set the number of policy updates per iteration $K$
4: Set the clipping parameter $\epsilon$
5: for $t = 1$ to $T$ do
6:     for $k = 1$ to $K$ do
7:         Collect samples by running the policy: $\{(s_i, a_i, r_i)\}$
8:         Compute advantages: $A(s_i, a_i) \approx Q(s_i, a_i) - V(s_i)$
9:         Compute surrogate objective:
               $$L(\theta) = \mathbb{E}_t \left[ \min \left( \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{\text{old}}}(a_i|s_i)} \cdot A(s_i, a_i), \text{clip} \left( \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{\text{old}}}(a_i|s_i)}, 1 - \epsilon, 1 + \epsilon \right) \cdot A(s_i, a_i) \right) \right]$$
10:        Update policy using gradient ascent: $\theta \leftarrow \theta + \alpha \nabla_\theta L(\theta)$
11:    end for
12: end for=0
```

target networks, those target networks are slowly updated versions of the main actor and critic networks, which helps in reducing the variance of the value estimates. Similarly to the idea behind double Q-learning SAC uses two Q-networks (critics) to estimate the state-action value function, which helps mitigate overestimation bias in the value estimates. It can be used when the agent learns form a fixed dataset of experience, so that making it suitable for scenarios where data collection is expensive or limited. It can be viewed in Algorithm 5.

PlaNet (Planning Network) is an algorithm that combines model-based and model-free approaches. It was introduced by researchers at DeepMind. It is designed to handle high dimensional visual inputs and efficiently learn policies for tasks with sparse and delayed rewards. Imagination-based planning is utilized in this algorithm, where it learns a world model to simulate possible future states and uses these simulations for planning and decision-making. The algorithm employs a stochastic latent variable model to capture uncertainty in the environment. The model includes a deterministic component (deterministic transition function), and a stochastic latent variable that capture unobservable dynamics. Additionally, it employs variational inference to learn the latent variables, allowing it to infer the distribution of latent variables given observations and rewards. PlaNet optimizes policies through a form of rollout policy optimization. It generates multiple action sequences using its learned world model and selects actions based on these

Algorithm 5 Soft Actor-Critic (SAC)

```
 1: Initialize actor parameters $\theta$, critic parameters $\phi$, target parameters $\theta'$, $\phi'$
 2: Initialize replay buffer $D$
 3: Set temperature parameter $\alpha$, discount factor $\gamma$, target update rate $\tau$
 4: Set maximum episodes $N$
 5: for $n = 1$ to $N$ do
 6:     Observe initial state $s$
 7:     for each time step do
 8:         Select action $a$ from the policy $\pi_\theta(s)$ with added noise
 9:         Execute $a$, observe next state $s'$ and reward $r$
10:         Store transition $(s, a, r, s')$ in $D$
11:         for each update step do
12:             Sample a batch of transitions $(s_i, a_i, r_i, s'_i)$ from $D$
13:             Compute target values:
                    $$y_i = r_i + \gamma \min_j Q_{\phi'}(s'_i, a'_j) - \alpha \log(\pi_\theta(a_i|s_i))$$
14:             Update critic:
                    $$\phi \leftarrow \phi - \eta \nabla_\phi \frac{1}{|D|} \sum_i [Q_\phi(s_i, a_i) - y_i]^2$$
15:             Update actor:
                    $$\theta \leftarrow \theta + \eta \nabla_\theta \frac{1}{|D|} \sum_i \alpha \log(\pi_\theta(a_i|s_i)) - Q_\phi(s_i, a_i)$$
16:             Update target networks:
                    $$\theta' \leftarrow \tau\theta + (1 - \tau)\theta', \quad \phi' \leftarrow \tau\phi + (1 - \tau)\phi'$$
17:         end for
18:     end for
19: end for=0
```

sequences. The combination of model-free reinforcement learning with model-based approach is achieved through the use of a policy optimization objective to fine-tune the policy based on real interaction with the environment. The design of that algorithm allows it to be uncertainty-aware, enabling it to explore more effectively in environments with space and delayed rewards. The uncertainty in the model is used to guide the exploration. Memory buffer of past experiences is maintained in PlaNet to facilitate learning from both real interactions and simulated experiences generated by the learned world model. It has the capability of handling high-dimensional visual inputs, making it suitable for tasks with image-based observations. PlaNet was demonstrated to be effective in a range of challenging tasks, including robotic control, and playing video games. It showcases how model-based methods can be integrated with model-free reinforcement learning to handle complex, high-dimensional environments and improve sample efficiency. A high-level overview of that approach is presented in 6.

Algorithm 6 PlaNet (Simplified)

```
1: Initialize policy parameters $\theta$, world model parameters $\phi$, buffer $D$
2: Set planning horizon $H$, number of rollouts $K$, optimization steps $L$
3: for each iteration do
4:    Collect real interactions: $\mathcal{D}_{\text{real}} = \{(s_t, a_t, r_t)\}$
5:    Update world model $\phi$ using $\mathcal{D}_{\text{real}}$
6:    for $k = 1$ to $K$ do
7:       Generate rollouts using world model: $\mathcal{D}^k_{\text{sim}} = \text{rollout}(\phi, \theta, H)$
8:       Update policy $\theta$ using $\mathcal{D}^k_{\text{sim}}$
9:    end for
10:   Sample batch from $\mathcal{D}_{\text{real}}$ and $\mathcal{D}^k_{\text{sim}}$
11:   Update world model and policy using variational inference and policy
      optimization
12:   Store experiences in buffer: $D \leftarrow D \cup \mathcal{D}_{\text{real}} \cup \mathcal{D}^k_{\text{sim}}$
13: end for=0
```

Chapter 4

Results

1 Results on The Evaluations

In this experimental research we conducted a wide range of experiments to explore the results of using a new observation type which is a tensor of points2.2 compared to the one that is used in most of the literature2.1 about controlling the DLO. The aspects of this comparison can be clearly viewed in the first 6 figures in the current chapter. These figures represent the data collected after training an RL-agent in the framework of SAC algorithm5 to compare between the two observations. The PPO algorithm4 was also used to obtain similar data for our study, but it was proven that SAC is better of such non-linear environment with large execution time. As the PPO is an on-policy algorithm that is updated by the gradient of the action taken, it had a tendency to fall in a local minima without a noticable improvement in the results. On the contrary, the SAC algorithm which is an off-policy with interchanging dynamics between exploration and exploitation of the environment, has proven to be an asset to evaluate those observations and get a clear view about the significance of one type over the other. The mean reward of multiple evaluation trials was one of the metric used in the assessment. In figure4.1 it can be shown that in the case of the tensor observation space, the model is evolving with a trend towards a better value which is in this case the less penalty it can achieve during an episode. On the contrary, the model is still not confident about its actions so for the same number of steps the metric is fluctuating without a trend and this can be clearly seen on the red line which is a running average of the actual values. The most minimum value of the reward is reached in the results with the image observation, which is another indicator of the argument that the feedback of a tensor of points is better than the image.

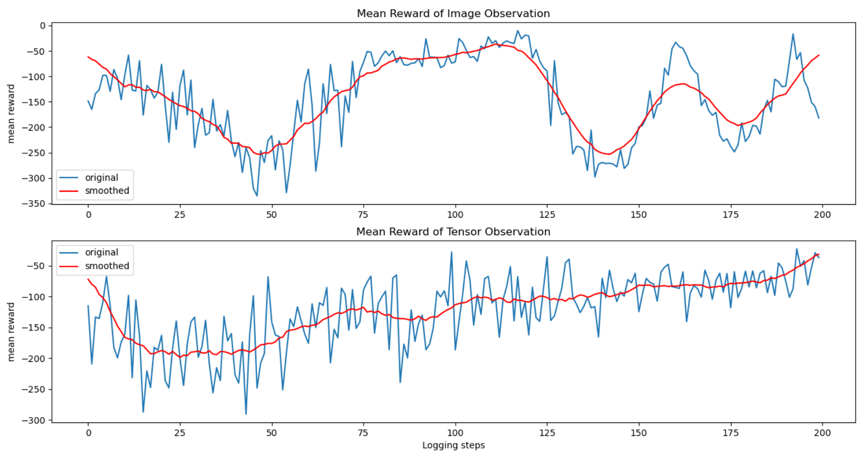


Figure 4.1: figures show the mean evaluation reward after each learning step. a) the upper figure is for the type of observation in images. b) the lower figure is for the type of observation in tensors

2 Results on Episode Length

The episode length during evaluation was another aspect to consider to see if the agent is trying to finish the penalization as soon as possible and end the episode earlier than the maximum number of steps that are allowed in an episode. Figure4.2 shows that the tensor of points is again making a better learning performance in this setup. one aspect of this figure is that the running average for the lower graph (tensor of points) is better than the one of the top graph (image observation), as it shows that the agent is trying to finish the episode more earlier towards the end of the training steps.

3 Results on The Rollout and The Networks

The rollout mean reward can be seen in figure4.3, which shows that both types of observation helps the model to evolve when it even does the exploration. The rollout in SAC is the batch of episodes in which the agent is exploring the environment with some exploitation actions also. unlike the evaluation episodes in which the agent uses a deterministic policy, the rollout involve some exploration to new action-state pairs. The agent reaches by the end of the training a value in case of the tensor observation that is much lower than the other one, and the difference is a significant one.

The episode length in the rollout is almost the same in terms of evolution

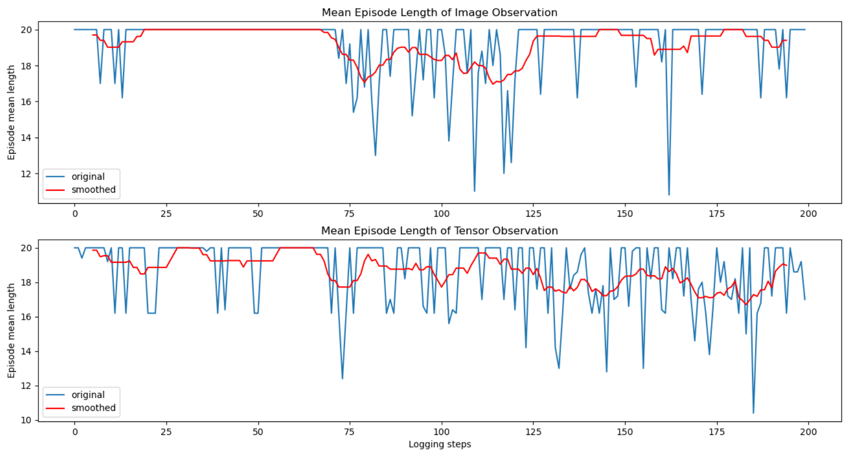


Figure 4.2: figures show the mean episode length after each learning step. a) the upper figure is for the type of observation in images. b) the lower figure is for the type of observation in tensors

shape of the two curves, but the tensor observation has reached a slightly less lower episode length by the end of the episode.

The actor loss in figure4.5 for both cases is the same but the case for the image observation the scale is much larger. This corresponds to the behaviour that is noticed in the mean reward for the evaluation steps, or in the mean reward for the rollout. As that plot and the one of the critic loss in figure4.6 indicate that the agent sensitivity to observation space is high, and the learning process for this problem can be elongated or saturated to some performance based on the obsrvation space in hand.

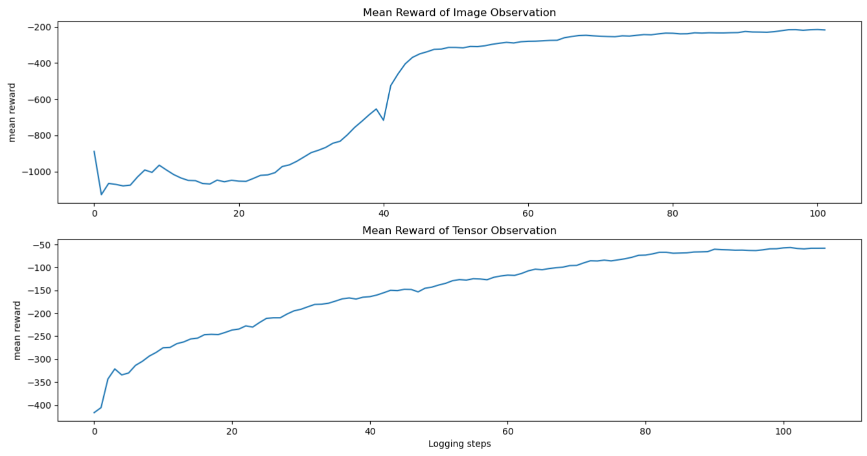


Figure 4.3: figures show the mean reward of a rollout after each learning step. a) the upper figure is for the type of observation in images. b) the lower figure is for the type of observation in tensors

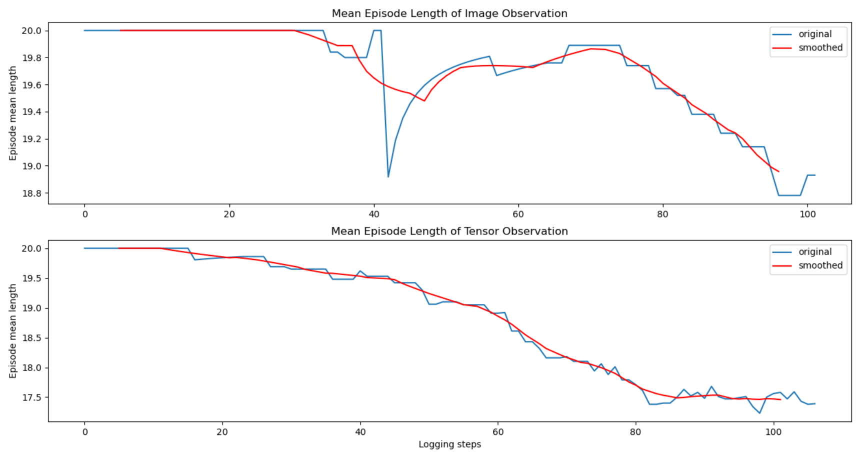


Figure 4.4: figures show the mean episode length after each learning step in a rollout. a) the upper figure is for the type of observation in images. b) the lower figure is for the type of observation in tensors

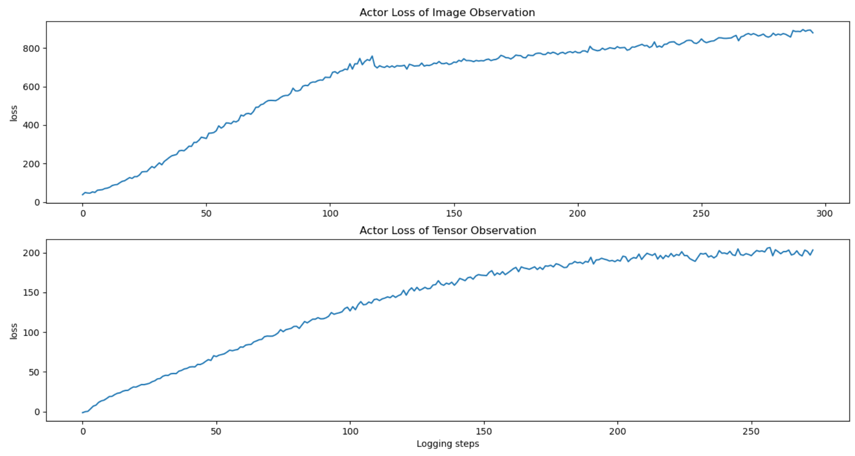


Figure 4.5: figures show the actor loss after each learning step. a) the upper figure is for the type of observation in images. b) the lower figure is for the type of observation in tensors

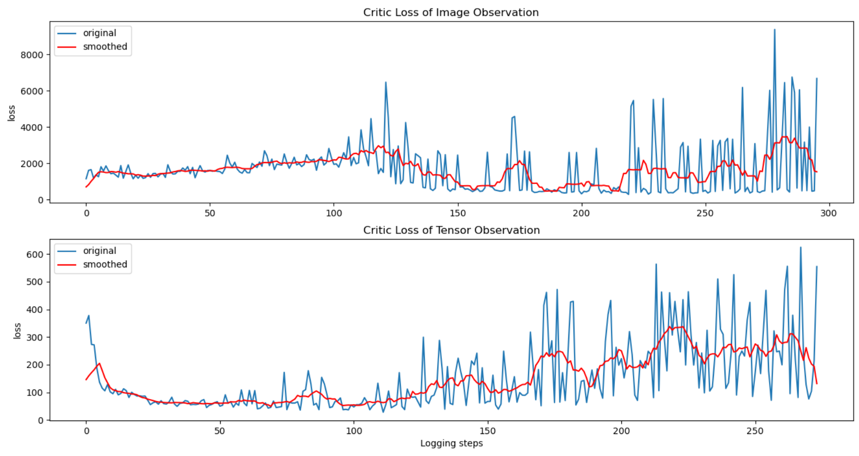


Figure 4.6: figures show the critic loss after each learning step. a) the upper figure is for the type of observation in images. b) the lower figure is for the type of observation in tensors

Chapter 5

Conclusion and Future work

The work presented here has shown that the use of a new observation space like tensor of points that can represent the perception of the RL agent about the environment is promising. In the mentioned results we can notice that the RL model is very sensitive to the observation space and can perform to better or the worst based on this information. Also the use of reward function has its impact on that model and can make the training process for that model eaither very lengthy and struggling or easier and straightforward. We presented here a new reward function that splits the problem of DLO manipulation into two parts, where one of them deals with configuration of the DLO and the other one deals with the orientation of the DLO. This reward function showed uplifting results for both types of observation space. It also showed an enhanced results when both SAC and PlaNet were trained on that reward function in relative to the other reward functions (sum of absolute differences and delta gain). The future work is to use this reward function with an on-policy algorithm like PPO for both types of observation. Also, the RL agent used in this research were hardly achieving a high tolerance results. It will be crucial to explore fine-tuning techniques for the algorithms used here. It was due to some hardware limitations very hard to address this problem.

Bibliography

[1] Jose Sanchez et al. “Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey”. In: *The International Journal of Robotics Research* 37.7 (2018), pp. 688–716.

[2] Jerome Trommnau et al. “Overview of the state of the art in the production process of automotive wire harnesses, current research and future trends”. In: *Procedia CIRP* 81 (2019), pp. 387–392.

[3] Xin Jiang et al. “Robotized assembly of a wire harness in a car production line”. In: *Advanced Robotics* 25.3-4 (2011), pp. 473–489.

[4] Ankit Shah, Lotta Blumberg, and Julie Shah. “Planning for manipulation of interlinked deformable linear objects with applications to aircraft assembly”. In: *IEEE Transactions on Automation Science and Engineering* 15.4 (2018), pp. 1823–1838.

[5] Konrad P Cop et al. “New metrics for industrial depth sensors evaluation for precise robotic applications”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 5350–5356.

[6] Naijing Lv, Jianhua Liu, and Yunyi Jia. “Dynamic modeling and control of deformable linear objects for single-arm and dual-arm robot manipulations”. In: *IEEE Transactions on Robotics* 38.4 (2022), pp. 2341–2353.

[7] Wen Hao Lui and Ashutosh Saxena. “Tangled: Learning to untangle ropes with rgb-d perception”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 837–844.

[8] Mingrui Yu et al. “Global model learning for large deformation control of elastic deformable linear objects: An efficient and adaptive approach”. In: *IEEE Transactions on Robotics* 39.1 (2022), pp. 417–436.

[9] Robert Lee et al. “Sample-efficient learning of deformable linear object manipulation in the real world through self-supervision”. In: *IEEE Robotics and Automation Letters* 7.1 (2021), pp. 573–580.

[10] OpenAI. *Spinning Up in Deep Reinforcement Learning*. https://spinningup.openai.com/er [Accessed: March 8, 2024].

[11] Andrew William Moore. *Efficient Memory-based Learning for Robot Control*. Tech. rep. University of Cambridge, 1990.

[12] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[13] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

[14] Sergio Guadarrama et al. *TF-Agents: A library for Reinforcement Learning in TensorFlow*. `https://github.com/tensorflow/agents`. [Online; accessed 25-June-2019]. 2018. URL: `https://github.com/tensorflow/agents`.

[15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[16] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[17] Daniel Seita et al. "Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 9651–9658. DOI: `10.1109/IROS45743.2020.9341608`.

[18] Alessio Caporali et al. "Deformable Linear Objects Manipulation with Online Model Parameters Estimation". In: *IEEE Robotics and Automation Letters* (2024).

[19] Wilson Yan et al. "Learning predictive representations for deformable objects using contrastive estimation". In: *Conference on Robot Learning*. PMLR. 2021, pp. 564–574.

[20] Javier Alonso-Mora, Stuart Baker, and Daniela Rus. "Multi-robot formation control and object transport in dynamic environments via constrained optimization". In: *The International Journal of Robotics Research* 36.9 (2017), pp. 1000–1021.

[21] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[22] Tuomas Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: `1801.01290 [cs.LG]`.

[23] Danijar Hafner et al. *Learning Latent Dynamics for Planning from Pixels*. 2019. arXiv: `1811.04551 [cs.LG]`.