Robotic folding with CURL in simulation

Vicent Roig Server
Supervisors: Francis wyffels, Andreas Verleysen, Victor-Louis De Gusseme

Abstract—**Humans are able to learn from visual observations how to solve a task. However, in reinforcement learning for robot manipulation of deformable objects, it is not common the use of visual observations as an input information. Long training periods and data-inefficiency make visual observations not suitable for these types of learning tasks. The algorithm that uses visual observations, apart from learning the task, has to learn to extract the state information from the images. Nevertheless, it is not always possible to give state observations to the algorithms especially in real robot manipulation tasks because the external objects are not controlled as in the simulated environments. For this reason in this thesis it is wanted to experiment with new methods that extract state information from images efficiently using contrastive learning. Those new algorithms, allow faster learning using visual observations rather than using state observations as an input.**

Nowadays there are different types of cloth with a huge range of shapes, sizes and colors. If we want to use a robot for cloth folding it is possible to pre-program a robot to fold a specific piece of cloth. However, if the size of the piece of cloth changes, then it will be necessary to reprogram the robot to accomplish the folding task. It is not possible to have a specific program for all the pieces of cloth on the planet. The only solution is to train the robot to identify the piece of cloth using visual observations and then the robot has to learn how to fold it according to the cloth properties.

I. Introduction

Object manipulation with minimal human intervention has been an objective in robotics that has diverse applications and huge economical opportunities [1] [2]. Moving heavy products from one point to another or helping humans to deal with everyday problems are some of the possible applications. Object manipulation has two important steps: grasp the object and then manipulate it. Although manipulation and grasping for rigid objects is a well-studied field, it is still in the earlier stages for deformable objects. The reason is caused by the physical difference between them. Rigid objects are easy to model and easy to grasp thanks to the impossibility to change the shape and the limited degrees of freedom of movement. However, deformable objects modeling is difficult because they have many degrees of freedom and complex dynamics. Also, the fact to grasp them changes the deformation of the object making the problem more difficult. In addition to that manipulation challenge, there is the grasping problem that depending on the way that the deformable object is grasped, will cause different deformations making very difficult the use of modeling methods to estimate the cloth state and the grasping point [3].

This thesis is focused on the manipulation part omitting the grasping problem. For this reason, the simulation always starts with the grasping task already done. In cloth manipulation for deformable objects, data-efficiency is very important to have precise information of movements and deformations. Deep Reinforcement Learning (DRL) has emerged as a viable method to learn robotic controllers that can acquire complex behaviors from high-level specifications such as images. Two of the main important things of DRL are the reward function that guides the agent towards learning the long-term objective and the neural network that codify the visual observations as an input information.

Reinforcement learning (RL), is a category of machine learning and artificial intelligence where intelligent machines can learn from their actions similar to the way humans learn from experience. Inherent in this type of machine learning is that an agent is rewarded or penalized based on their actions. Actions that get them to the target outcome are rewarded through a series of trial and error, making this technology ideal for dynamic environments. RL algorithms that learn from images are inefficient [4] [5] compared to the algorithms that use physical states of the environment as an input for the algorithm. The reason is easy, the agent has to take the state information that is intrinsically in the images and later on do the RL with that extracted data. It is simple to understand that extracting the data from the image is more difficult rather than receiving the state directly. However, the difficulty to model deformable objects makes the use of visual observations the best input data option to learn manipulation tasks.

In this thesis, it has been used a new algorithm called Contrastive Unsupervised Representations for Reinforcement Learning (CURL) [6] developed by (Aravind Srinivas, 2020), that extracts the information from images allowing the neural networks to learn faster to characterize the observations making use of contrastive learning.

II. Simulating cloth

A. Cloth model

In this section, it is explained how we modeled the cloth simulation that we will use in the robotic folding task. The cloth simulation is built following the subsections as a steps for programming the simulation. It has been used Unity environment with $C\#$ code language. The code is available in https://github.com/vicent44/RobotAgents.

The mass-spring model is a discrete model that consists of particles in a rectangular mesh interconnected with three types of linear springs displayed in figure 1. Structural, shear and bend:

- Structural: This spring is used to counteract the tension and maintain the basic structure of the mesh. Implements resistance to stretching.

- Shear: This spring is needed to allow shearing forces. Implements resistance to shearing.
- Bend: This spring is needed to allow bending in the cloth. Implements resistance to bending.

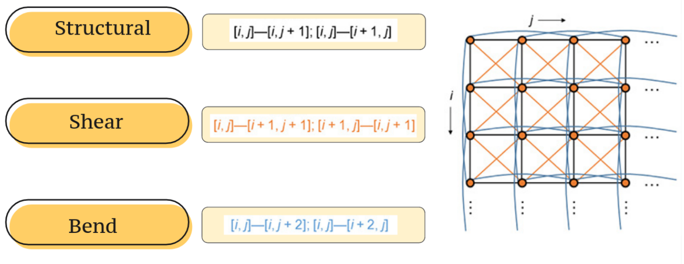

Structural
[i, j]—[i, j + 1]; [i, j]—[i + 1, j]
Shear
[i, j]—[i + 1, j + 1]; [i + 1, j]—[i, j + 1]
Bend
[i, j]—[i, j + 2]; [i, j]—[i + 2, j]
j
i

Figure 1: Mass-spring model. Where i and j are the positions of the particles in the mesh.

The elastic forces executed by linear springs between two particles x_i and x_j are given by equation 1:

$$\vec{F^e_{ij}} = k_{ij}(\|\vec{r_{ij}}\| - l_{ij})\frac{\vec{r_{ij}}}{\|\vec{r_{ij}}\|} \tag{1}$$

where $k_{ij}(N/m)$ is the elastic constant of the spring, $l_{ij}(m)$ is its rest length and $\vec{r_{ij}} = r_i - r_j$ is the distance between the two particles x_i and x_j. The two constants depends on the type of the spring.

In the mass-spring system a viscous force is used to allow energy dissipation due to internal friction, in other words, to reduce the kinetic energy of the oscillations. These forces depend on the velocity of the particles that take part of the spring:

$$\vec{F^d_{ij}} = d_{ij}(\vec{v_i} - \vec{v_j}) \tag{2}$$

where d_{ij} is the damping constant.

In the model, it is also important to add external forces like gravity and air force to make the model more realistic. For the gravity, a negative constant force is implemented in the vertical axis:

$$\vec{F^g} = -m\vec{g} \tag{3}$$

where g is $9.8m/s^2$. For the wind force, another constant force is implemented. However, this force depends on the area exposed to the wind. For this reason, the area of the mesh is divided into triangles and the force is calculated for each triangle taking into account the direction of the wind and the normal of the triangle to calculate the direction of the force. Then, this force is divided by the number of nodes of the triangle. The wind force implemented is:

$$\vec{F^w} = \rho\ A\ \vec{v^2}\ \vec{n_A} \tag{4}$$

where $\rho(kg/m^3)$ is the density of the cloth, $v(m/s)$ is the velocity of the wind, $A(m^2)$ is the area where the force is applied and $\vec{n_A}$ is the normal of the triangle.

Finally, the equation to solve for each particle is:

$$\vec{F^e} + \vec{F^d} + \vec{F^g} + \vec{F^w} = m\frac{d^2x}{d^2t} \tag{5}$$

As an integrator, has been chosen Verlet integrator which is an implicit method that is frequently used in computer graphics to calculate trajectories of particles since it provides very good numerical stability as well as time reversibility which enables to make corrections in the solutions provided by the method. The Verlet method used, only depends on the previous position, the current position and the total force.

$$x(t + \Delta t) = 2x(t) + x(t - \Delta t) + a(t)\Delta t^2 \tag{6}$$

where $a(t)$ is the acceleration (F/m).

B. *Cloth collisions*

The mass-spring model only describes the mechanical behaviors of the cloth. The forces to avoid collisions are not included.

One common way of reacting to collisions is to generate forces that eventually separate colliding objects. It is a penalty based system that applies a force to the point where the contact has been produced to simulate a real collision and avoid interpenetration. The problem with this reaction is that it is very difficult to know the correct order of force to apply. By using a force too strong the objects will become unstable and by using a force too weak the objects will penetrate each other without solving the collision problem. Another technique to solve collisions consists of using projection as a collision reaction moving the penetrated point out of the collided object.

It is important to say that there are two important types of collisions: cloth-cloth collisions also known as self-collisions and cloth-object collisions. By their names, it is easy to deduce that cloth-cloth collisions prevent the cloth from going through itself and cloth-object collisions prevent the cloth from penetrating other objects. In this thesis, different techniques are used to solve those collisions.

Optimized spatial hashing for collision detection for deformable objects, is the technique chosen to optimize the collision detection [7]. First of all, it is needed to subdivide the space into uniform rectangular regions. Each region maps the object primitives that are partially or fully contained on it into a hash table. Then, the intersection between object primitives is calculated when they share the same hash table cell. This clustering highly reduces the number of necessary collision tests, testing only the primitives that share the same table cell in view of the fact that those have high probability of collision. The size of the hash table has high influence on the performance of the collision detection algorithm. Larger hash tables reduce the possibility of mapping different positions on the same hash index. On the other hand, large tables suffer the problem that the performance decreases due to memory management.

The main importance of that model relies on the use of an appropriate discretization model as well as an adequate intersection test. In this project, since the cloth is simulated as a mesh of triangles, a point-triangle test has been used for cloth-cloth collision detection.

In this project, since the cloth is simulated as a plane, point-triangle intersection test is performed by calculating the barycentric coordinates of the triangle concerning the projection of the point into the triangle. Then, is calculated the dot product between the normal of the triangle and the vector distance between the barycentric point and the particle position. If the dot product is positive, the particle has to be pulled away in the direction of the normal triangle. Otherwise, it has to pull away to the opposite direction [7]. In addition, this correction of the particles is also done to the particles that form the triangle to provide deformation. In that case, the corrected distance is multiplied by the barycentric distance of each vertex of the triangle and the direction is the contrary of the one used to move the point.

C. Superelasticity

The particles might be correctly placed initially but after one integration step the distance between them might become invalid which would produce an unrealistic cloth behavior. So, as a solution, we use a direct manipulation of the position of the particles which involves pulling them closer in case of excessive elongation or pushing them away in case of reduced elongation (depending on whether the erroneous distance is too small or too large) [8].

III. Contrastive methods for reinforcement learning

A. Soft Actor Critic (SAC)

Soft Actor Critic(SAC) [9] is an off-policy actor-critic DRL algorithm based on the maximum entropy reinforcement learning. The actor aims to maximize the expected reward while also maximizing entropy. That is, to succeed at the task while acting as randomly as possible. The actor samples stochastically actions from policy π_ψ and is trained to maximize the expected return measured by critics Q_{ϕ_i} as is shown in equation 7.

$$\mathcal{L}(\psi) = -\mathbb{E}_{a_t \sim \pi}[\, Q^\pi(s_t, a_t) - \alpha \log \pi_\psi(a_t|s_t)] \tag{7}$$

At the same time the critics Q_{θ_1} and Q_{θ_2} are trained to maximize the Bellman equation (equation 8) using the distribution of the states and actions sampled previously. The use of two Q-functions improve the performance in most of the training environments. The minimum of both of them is used in equation 7.

$$\begin{aligned}\mathcal{L}(\phi_i, \mathcal{D}) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}[\, \frac{1}{2}(\, Q_{\theta_i}(s_t, a_t) - r(s_t, a_t) - \\ \gamma \mathbb{E}_{s_{t+1}\sim p}[\, V(s_{t+1})]\,)^2]\end{aligned} \tag{8}$$

B. Contrastive Representation Learning (CRL)

Contrastive representation learning (CRL) [10] is the process of learning a representation from high-dimensional input data by extracting the features using contrastive learning [11] [12]. Contrastive Learning is a mechanism to learn representations that obey similarity constraints in a dataset organized by similar and dissimilar sets.

CRL, trains a visual representation encoder network that extracts representation vectors from augmented data samples. In the figure 2, we can see how the CRL algorithm [13][14][15] uses two data-augmented versions of an observation O called O_q (query observations), which are a central crop of the observation O, and O_k (key observations) that contains the positive and the negative crops, which are a random crop of the observation O and a random crop of other observation O of another time step respectively. Then, the cropped observations are encoded. The key observations are encoded with a momentum encoder network ($f_{\theta_k}(O_k)$) sampled by the averaged version of the query observations encoder network ($f_{\theta_q}(O_q)$), which means that only one encoder network has to be trained. Once the encoder network is applied to the query observation we have as a result a vector representation called query (q) and once the momentum encoder network is applied to the keys observations we have as a result a vector representations key positive (k+) and key negative (k-). The loss is interpreted as the log-loss of K-way softmax classifier that tries to classify the query as key positive.

$$\mathcal{L}_q = \log \frac{exp(sim(q, k_+))}{exp(q, k_+) + \sum_{i=0}^{K-1} exp(sim(q, k_i))} \tag{9}$$

The contrastive loss (equation 9) is used as an unsupervised objective function for training the encoder network that represents the queries and the keys. Contrastive loss is a function whose value is low when the query (q) is similar to the positive key (k+) and dissimilar to the negative key (k-).

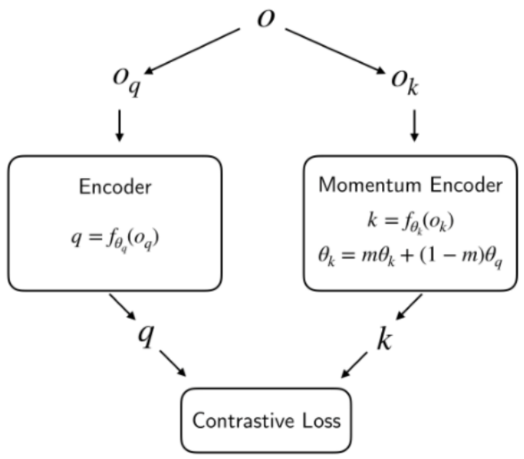


Figure 2: Schema of Contrastive Representation Learning [6].

C. *Contrastive Unsupervised representations for Reinforcement Learning (CURL)*

Contrastive Unsupervised Representation for Reinforcement Learning (CURL), implements CRL in RL framework to deal with learning inefficiency from visual observations. CURL uses a form of CRL that maximizes agreement between augmented versions of the same observation, where each observation is a stack of temporally sequential frames.

CURL is a generic framework that can be plugged into any reinforcement learning algorithm that learns from visual observations. The RL policy and value function are built using the query encoder which is jointly trained with the contrastive and reinforcement learning objectives. In figure 3 is possible to see that CURL trains the encoder q that is shared with the RL algorithm.

The CURL (Srinivas, A., 2020) [6] implementation, is coupled with SAC [9] and uses as a factor for the discriminative objective the inner product $sim(q,k) = q^T \ W \ k$ between query-key pairs, where W is a parameter matrix to learn [15].

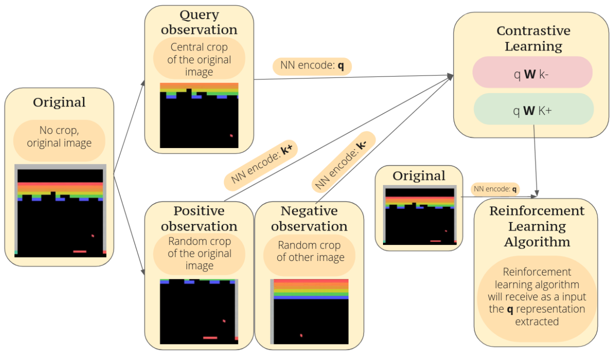

Original
No crop, original image
Query observation
Central crop of the original image
NN encode: **q**
Contrastive Learning
q **W** k-
q **W** K+
NN encode: **k+**
NN encode: **k-**
Original
NN encode: **q**
Positive observation
Random crop of the original image
Negative observation
Random crop of other image
Reinforcement Learning Algorithm
Reinforcement learning algorithm will receive as a input the **q** representation extracted

Figure 3: Schema of CURL

D. *CURL in folding task*

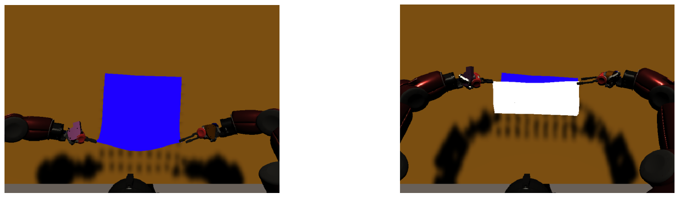
By implementing CRL highly increases the sample-efficiency of the RL algorithm by reducing the time to train the encoder. However, it is also important a good setup for the RL algorithm. The design of the reward function or the camera features are crucial for a good operation of the algorithm.

The camera to get the visual observations will be important. Setting up the camera in a position that the agent will not be able to see all the dimensions of the environment will reduce the amount of information that the agent will get as an input. For this reason, we will have to experiment with different camera positions. Another important thing is the design of the reward function, which have to allow the agent to learn by linking the rewards given with the visual observations.

IV. Folding task

A. *Reward function*

Once the basics of cloth folding are known, it is time to design the reward function. It is important to simplify the task. For this reason, the folding always starts with the cloth attached to the gripper of the robot to avoid the possibility of



(a) Step one. (b) Step two.

Figure 4: Steps to accomplish the folding task.

the robot losing it. Nevertheless, to implement a more general behavior the reward function will always check if the cloth is attached to the gripper. In figure 4, we can see the two basic steps of the reward function:

- First step: While the cloth is attached to the gripper, the robot has to reach the goal position which is on the other side of the cloth. If the gripper reaches that point then a reward of 25 is added. To stimulate the gripper to stay in the goal position while the other arm is finding the goal position, a negative reward of -25 is set if the gripper moves away from the goal.
- Second step: If both grippers reach the goal then a reward of 100 is added and the episode finishes.

Until the agent finishes the task, to stimulate the agent to keep moving to find the shortest path to the goal, at each step a reward of -1 is set.

For the detection of the goal position, it is implemented a Unity object attached to the particles (to make it dynamic) that detect collisions once the grippers are near the goal position.

In the earlier experiments, the agent was not able to find the goal position because the reward function was sparse and the robot had difficulties to know where to go. After reading the paper of Albert Zhan, 2020 [16], we realize that to help the agent to understand the images it was necessary to implement a dense reward $r = -d$ where d is the euclidean distance.

Finally, taking into account the recommendations in the paper of Tuomas Haarnoja, 2018 [9], the reward function is normalized to make the algorithm more efficient. The normalization is done by divide all the reward values by 500.

B. *Action mask*

To avoid unreal movements and to reduce the volume of possible movements of the arms of the robot, it has been set some boundary conditions:

- Table plane: This plane is to avoid that the arms of the robot go lower than the position of the table.
- Robot plane: This plane is to avoid the arms going to the back of the robot.
- Other planes: These planes are to define a cube around the grippers of the robot to define a finite volume of movement.

To do so, before each action is updated in the robot arms, it is checked if the update positions have some collisions between some of these objects. In the positive case, the action is ignored and a reward of -1 (normalized dividing by 500) is added. On the other hand, if the new position does not collide with any of these game-objects then the new position is updated.

V. Discussion

It is important to say that the cloth constants which are elastic and damping, have almost no effect. For spring constant higher than 10.000 the simulation becomes unstable whereas in the range from 0 to 10.000 it is not possible to observe any difference. For the damping constant there is the same situation, in the range from 0 to 50 it is not possible to observe any difference but for a constant higher than 50 the simulation becomes unstable. This means that in order to change the characteristics of the cloth, instead of modifying the elastic and damping constants, the superelasticity correction constant is the one that has to be changed. Reducing it makes the cloth softer and increases its stiffness.

The fact that even when the constants (elastic and damping) are zero the behavior of the cloth remains working, leads us to believe that the magnitude order of the superelasticity correction is higher than the force implemented by the springs. Is especially notable that if the elastic and the damping constants are zero, which means that the forces of newton are zero, the patch of cloth has still the behavior of the mass-spring model. In other words, the superelasticity correction cancels all the forces.

The cloth and the robot simulation have different simulation time steps and these produce difficulties at the time to couple them. The cloth simulation requires more calculations every time step making the time step of the cloth simulation larger than the time step of the robot simulation that does not require the same amount of calculations. This causes that when the robot moves the patch of cloth, this one takes some time to react than the robot and the movement is slower. That difference is also present in the implementation of the training part because the movements of the robot have to be very small to couple the simulation time step of the robot and the cloth. As far as we know, this only slows down the training process and does not have any implications on the learning part.

The code of the implementation of the cloth simulation, the robot simulation and the CURL is available at https://github.com/vicent44/RobotAgents.

VI. Results

A. Experimental framework

We now proceed to train the robot folding task. For the training, a computer provided by the University of Gent with a GPU Titan Xp was used.

The camera position has been used as a hyperparameter for the training of CURL algorithm. In the contrastive methods section, we explained that CURL is an algorithm that couples contrastive representation learning (CRL) with an RL algorithm, which is SAC. For this reason, in figure 5 we have the results of how are performing both parts, the CRL part with the contrastive loss (CRL loss) plots and the RL part with the critic loss, actor loss and reward plots from four different camera positions.

The hyperparameters used for the CURL implementation (hyperparameters from CRL and SAC) that are in table 1 are taken from the CURL implementation [6].

Hyperparameter	**Value**
Observation rendering	[84,84]
Observation sampling	[64,64]
Stacked frames	4
Batch size	32
Replay buffer size	100.000
Hidden units	256
Evaluation episodes	4
Optimizer	Adam
$(\beta_1, \beta_2) \rightarrow (\alpha)$	0.9
$(\beta_1, \beta_2) \rightarrow (\mathrm{f}_\theta, \pi_\Psi, Q_\Phi)$	0.9
Learning rate (α)	0.001
Learning rate $(f_\theta, \pi_\Psi, Q_\Phi)$	2e-4
Q function EMA τ	0.01
Critic target update freq	2
Convolutional layers	4
Number of filters	32
Non-linearity	ReLU
Encoder EMA τ	0.05
Latent dimension	50
Discount γ	0.99
Initial temperature	0.1
Initial steps	20.000

Table I: Hyperparameters for CURL [6]

B. SAC results

In this section, we will discuss the results of the RL training part of the CURL algorithm.

In figure 5 we can observe that for all the camera positions the critic loss is very close to zero with some sparse peaks, which can be interpreted as the agent exploring unseen parts of the environment, where the action-value function is not well known. At the start of the training, it is normal to see a lot of peaks, but they are supposed to decrease as the training and learning progresses. As we can see in the plots, the peaks are not tending to disappear and this can be interpreted as the agent learning is not progressing. One of the causes could be that the agent is not able to link the visual observations with the actions taken causing the agent to explore places that have already been explored. The agent is not able to perform the actions that he wants to according to the visual observations that receive as an input. Probably because the images are in two dimensions and the environment is in three dimensions.

SAC trains the actor to maximize the expected return of its actions. If the actor loss increases, actions with higher reward are taken. However, we can see in the actor loss plots that, after the stabilization part, the gradient is almost null meaning that the learning is not progressing.

Finally, the reward function in all the camera positions tends to converge. However, after the analysis done above with the actor loss and the critic loss, we can say that the agent is moving randomly in the environment without any learning progress. In https://github.com/vicent44/RobotAgents there are some video examples of the behavior.

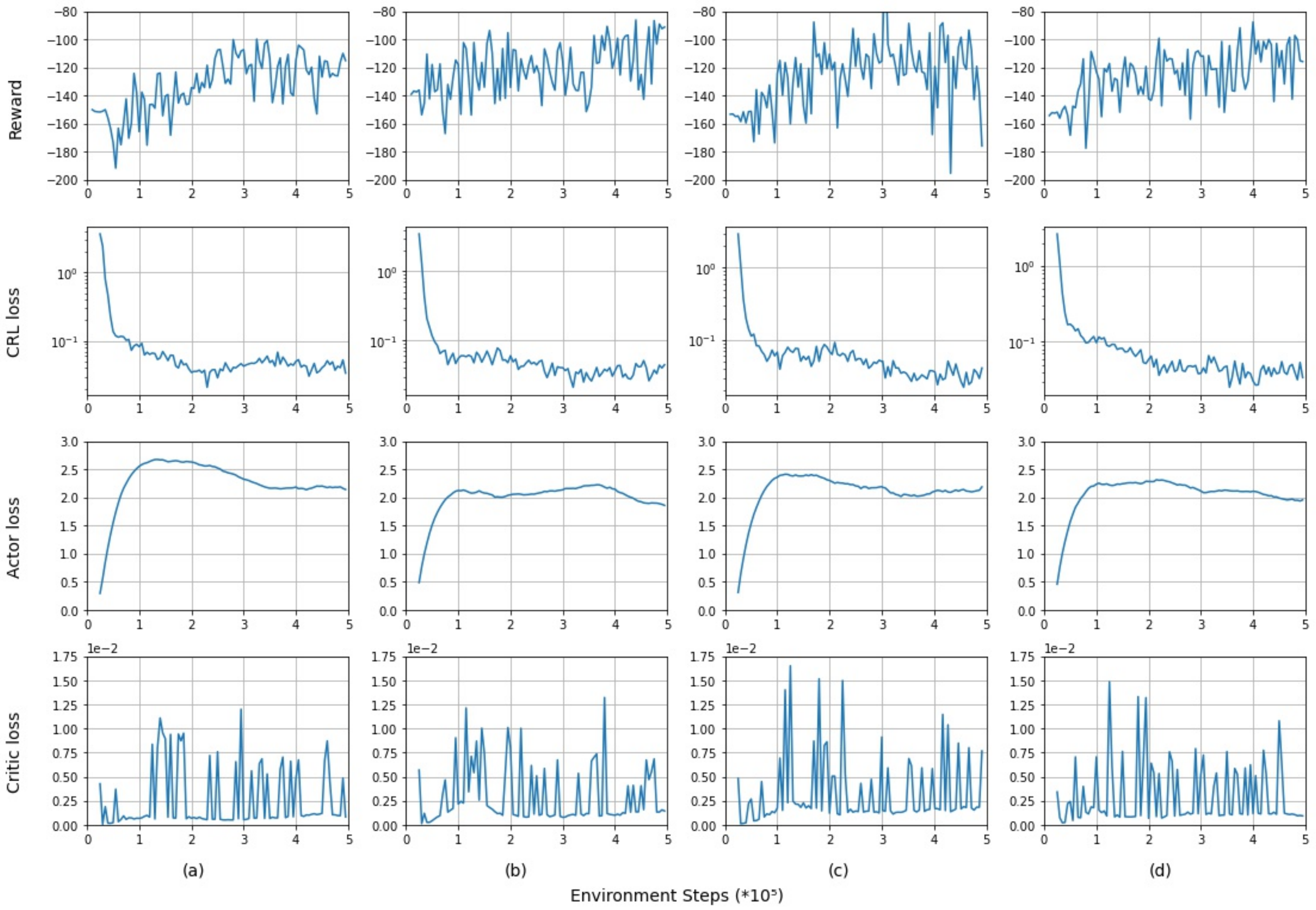
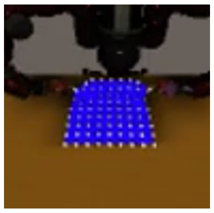
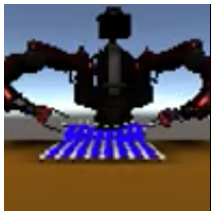
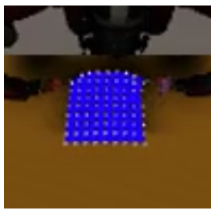


Figure 5: In this plot there are the training curves of the CURL algorithm. In each column, the visual observations are taken from different camera positions related to the figure 6. We can see that the CRL loss converges very fast meaning that extracts good features from the observations. On the other hand, the reward and the critic loss do not converge meaning that the training is not performing as expected.


Camera position (a).


Camera position (b).


Camera position (c).

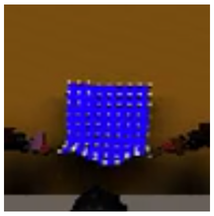

Camera position (d).

Figure 6: Examples of perspective of the different camera positions related to the training curves.

C. CRL results

In this section, we will discuss the results of the contrastive learning part of the CURL algorithm.

In figure 5 in the CRL loss plots, we can observe that the CRL loss converges very fast with independence of the camera position. Contrastive representation learning is able to extract the features from the images efficiently. The contrastive loss (CRL loss) is a function whose value is low when the query (q) is similar to the positive key (k+) and dissimilar to the negative key (k-). In other words, when the features extracted q matches more with the features extracted $k+$ and less with the features extracted $k-$. However, this is not something that surprises us. The extraction of the features from the images is not related to the camera position, is related to the contrastive loss function, which has good results.

D. Discussion results of CURL

In the CURL implementation, there are some important observations. The authors of CURL have recently published an update to the CURL paper [6]. In that update, they say that after some experiments (after the first release of the paper) they realized that the data-efficiency archived in CURL implementation is more related to the data augmentations rather than for the contrastive learning part. Reinforcement Learning Augmented Data (RLAD) [17] consists of training directly the RL algorithm with augmented observations rather than use the augmented observations to extract features performing CRL and feeding the RL neural network that encodes the observations. Comparing the results of the CURL paper [6] with the RLAD paper [17], we can see that RLAD [17] has better results than CURL [6]. However, this always will depend on the task and the training preferences. As it was already presented by Chen et al. 2020 [16] visual augmentations is a way to extract more value to unsupervised learning.

VII. Conclusion

This thesis successfully implemented the cloth and the robot in simulation. It was possible to couple the unity environment (cloth and robot simulation) with the Python environment (CURL algorithm) using ML-Agents. Nevertheless, there are some limitations. The connection between the cloth simulation and the robot simulation is difficult because they have different simulation time step. The cloth simulation requires more calculations every time step making the time step of the cloth simulation larger than the time step of the robot simulation that does not require the same amount of calculations. The result of that is that when the robot moves the patch of cloth, this one takes some time to react than the robot and the movement is slower. That difference is also present in the implementation of the training part because the movements of the robot have to be very small to couple the simulation time step of the robot and the cloth. As far as we know, this only slows down the training process and does not have any implications on the learning part. Another limitation is that, the constants (spring constants) of the cloth simulation are not working as expected. This means that it is not possible to experiment with different cloth characteristics reducing considerably the scope of the possible results. A possible solution to the cloth simulation problems, instead of programming our own cloth simulation, would be to use a specific program like Blender to obtain the cloth simulation.

Finally, as a conclusion, we can say that the most challenging part was to design a reward function that allows the RL algorithm to link the visual observations with the actions taken. This enables a precise interpretation of the environment by the RL algorithm meaning a good learning. From the generated data it is possible to say that the algorithm is not able to accomplish the task. The problem relies on the image observations. The fact that the visual observations are in two dimensions, makes the agent not to get one axis information. For example, if the camera observation direction is in the $Y - axis$, when the robot moves the gripper in that axis the observations will not have the depth of the movement. In this scenario, it is difficult to differentiate two different positions in the same axis, especially in our case that the movement of the gripper is slow. A possible solution for the dimensional observations problem would be to give, as an input, images from different camera positions rather than use always the same camera position. That would give the depth that one camera position can not.

A. Future work

On the one hand, it is important to keep experimenting with the current simulation. Different camera positions, as well as different initial positions of the cloth, are good features to experiment with. Besides, implementing a mechanism to give as an input a mix of all different camera visualizations already implemented to provide a 3-dimensional perspective would be crucial to improve the data-efficiency. In addition, increasing the quality of the cloth simulation to allow experiments with different cloth characteristics, design a good reward function and solve the coupling problem between the robot simulation and the cloth simulation is very important to achieve good results in this thesis.

On the other hand, the next step of this thesis is to apply this project to a real robot environment. That step is difficult, especially at the time to design a new reward function for the new environment. The main problem in the real world is that it is not possible to have control of all the space as easy as in the simulation environment. We assume that in the setup of the real robot, with a real piece of cloth, we do not have any mechanism to show the robot when the task is accomplished and design a good reward function will be very tricky. Introducing some detectors like sensors in the gripper of the robot or in the cloth would be a good solution for the robot to know when the goal is accomplished.

References

[1] J. Sanchez, J. A. Corrales Ramon, B. C. BOUZGARROU, and Y. Mezouar, "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: A survey," *The International Journal of Robotics Research*, vol. 37, pp. 688 – 716, 06 2018.

[2] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," 2016.

[3] S. Donaire, J. Borràs, G. Alenyà, and C. Torras, "A versatile gripper for cloth manipulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6520–6527, 2020.

[4] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski, "Model-based reinforcement learning for atari," 2020.

[5] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," 2016.

[6] A. Srinivas, M. Laskin, and P. Abbeel, "Curl: Contrastive unsupervised representations for reinforcement learning," 2020.

[7] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," *VMV'03: Proceedings of the Vision, Modeling, Visualization*, vol. 3, 12 2003.

[8] T. Jakobsen, "Advanced character physics."

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[10] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, “Supervised contrastive learning,” 2020.

[11] P. H. Le-Khac, G. Healy, and A. F. Smeaton, “Contrastive representation learning: A framework and review,” *IEEE Access*, vol. 8, p. 193907–193934, 2020.

[12] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality reduction by learning an invariant mapping,” in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, vol. 2, pp. 1735–1742, 2006.

[13] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” 2020.

[14] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum contrast for unsupervised visual representation learning,” 2020.

[15] O. J. Hénaff, A. Srinivas, J. D. Fauw, A. Razavi, C. Doersch, S. M. A. Eslami, and A. van den Oord, “Data-efficient image recognition with contrastive predictive coding,” 2020.

[16] A. Zhan, P. Zhao, L. Pinto, P. Abbeel, and M. Laskin, “A framework for efficient robotic manipulation,” 2020.

[17] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, “Reinforcement learning with augmented data,” 2020.

[18] A. Gruslys, W. Dabney, M. G. Azar, B. Piot, M. Bellemare, and R. Munos, “The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning,” 2018.

[19] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, “An introduction to deep reinforcement learning,” *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, p. 219–354, 2018.

[20] A. Aristidou and J. Lasenby, “Fabrik: A fast, iterative solver for the inverse kinematics problem,” *Graphical Models*, vol. 73, pp. 243–260, 09 2011.

[21] Keavnn, “Rls: Reinforcement learning research framework for unity3d and gym.” https://github.com/StepNeverStop/RLs, 2019.

[22] B. H. Heidelberger, “Consistent collision and self-collision handling for deformable objects.” Dissertation submitted to ETH Zurich, 2007.

[23] M. Wacker, B. Thomaszewski, and M. Keckeisen, “Physical models and numerical solversfor cloth animations,” 2005.

[24] H. Zhang and T. Yu, *Taxonomy of Reinforcement Learning Algorithms*, pp. 125–133. Singapore: Springer Singapore, 2020.

[25] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, “Unity: A general platform for intelligent agents,” 2020.