FACULTY OF ENGINEERING AND ARCHITECTURE

Robotic folding with CURL in simulation

Vicent Roig Server

Supervisor: Prof. dr. ir. Francis wyffels
Counsellors: Andreas Verleysen, Victor-Louis De Gusseme

Faculty of Engineering and Architecture
Master's dissertation submitted in order to obtain the academic degree of
Master of Engineering Informatics
Academic year 2020-2021

GHENT UNIVERSITY

Permission of use

Vicent Roig Server
Gent, January 11th, 2021

Preface

I would like to thank several people that helped me to complete this master dissertation as the conclusion of my Master in Engineering Informatics.

Thanks to my supervisors Prof. dr. ir. Francis wyffels, Andreas Verleysen and Victor-Louis De Gusseme for giving me the opportunity to join your laboratory on these special days and helping me to develop this thesis.

Last but not least, I want to express my profound gratitude to my parents for their endless support to make this possible.

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.

In the experiments accomplished in the thesis, it has been seen that the contrastive method used takes the information efficiently, in other words, has a fast convergence. However, the most challenging parts are to design a reward function that allows the algorithm to link the visual observations with the rewards given and obtain good image observations to give as much information as possible to the algorithm enabling a precise interpretation of the environment.

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.

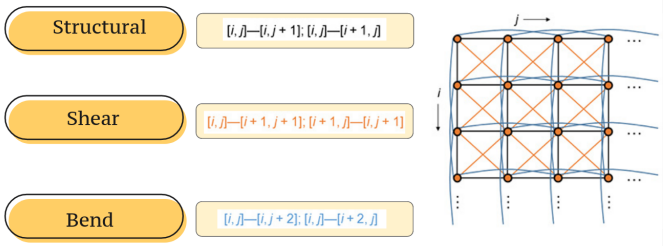


Figure 1: Mass-spring model. Where i and j are the positions of the particles in the mesh.

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

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

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:

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

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.

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

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-).

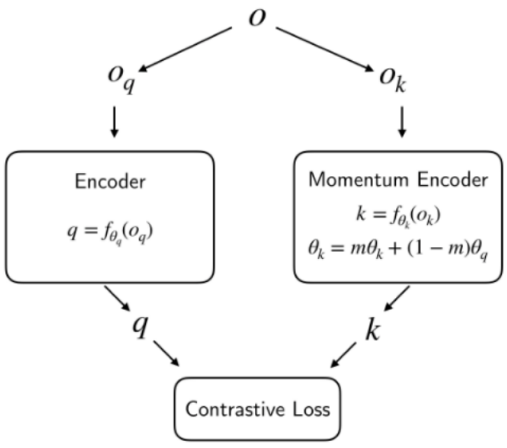


Figure 2: Schema of Contrastive Representation Learning [6].

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].

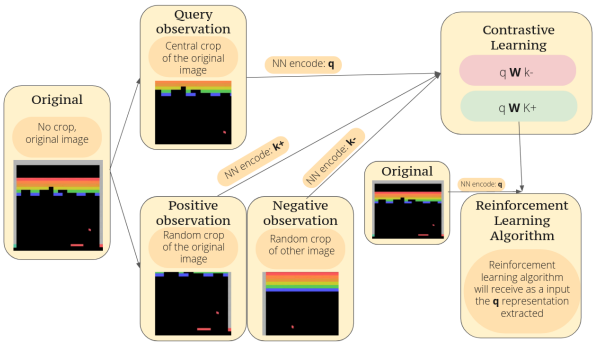


Figure 3: Schema of CURL

D. CURL in folding task

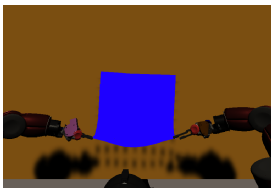
By implementing CRL highly increases the sample-efficiency of the RL algorithm by reducing the time to train the encoder. However, it is also important a good setup for the RL algorithm. The design of the reward function or the camera features are crucial for a good operation of the algorithm.

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.

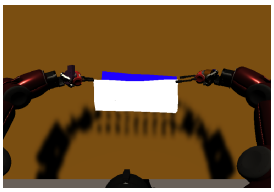
IV. Folding task

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 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:



(a) Step one.



(b) Step two.

Figure 4: Steps to accomplish the folding task.

- 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
$(\beta_1, \beta_2) \rightarrow (f_\theta, \pi_\Psi, Q_\Phi)$	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.

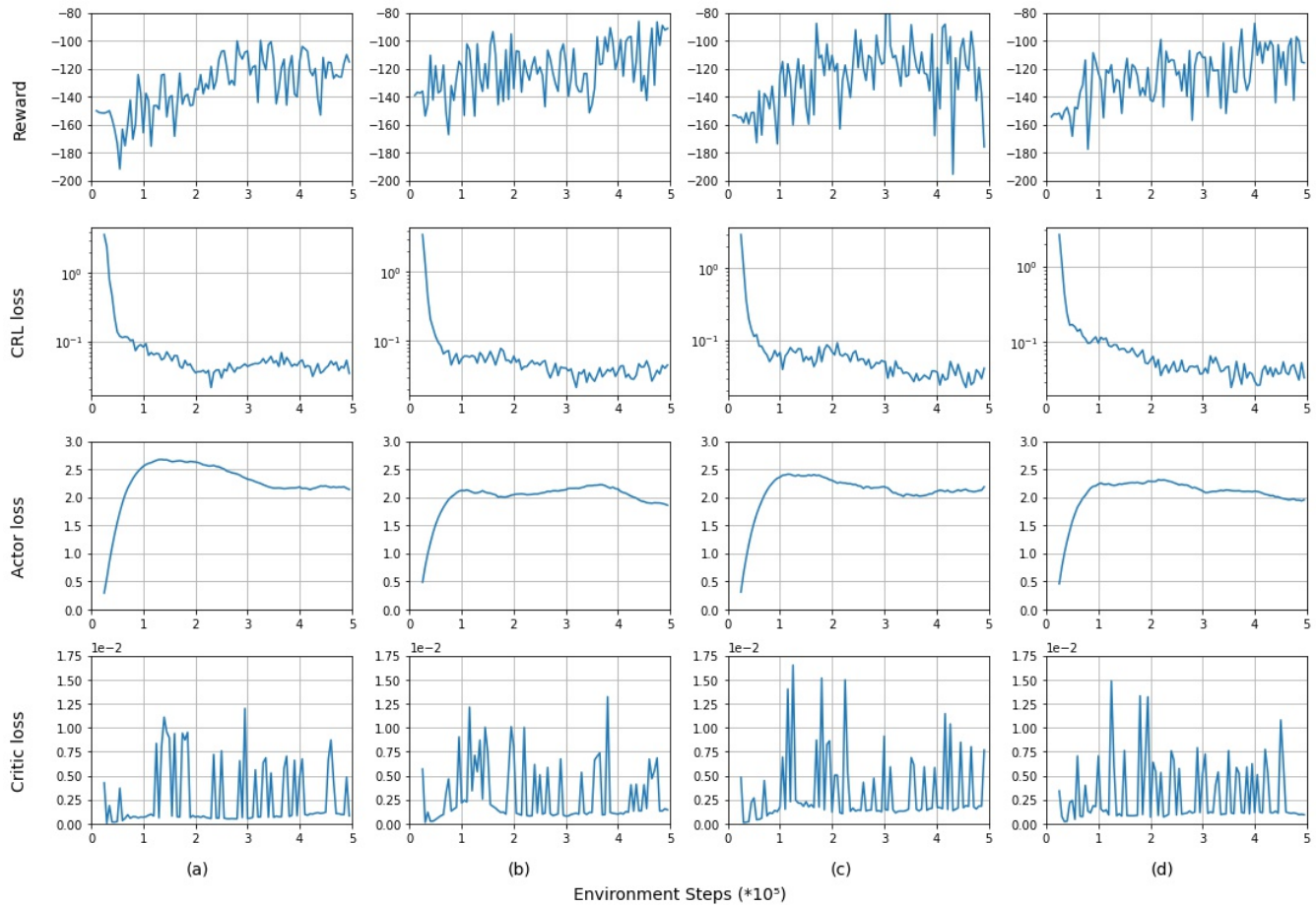
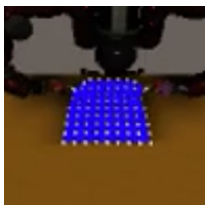
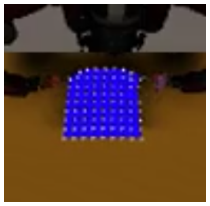


Figure 5: In this plot there are the training curves of the CURL algorithm. In each column, the visual observations are taken from different camera positions related to the figure 6. We can see that the CRL loss converges very fast meaning that extracts good features from the observations. On the other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

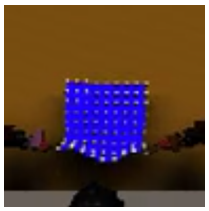


Camera position (a).

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.

List of contents

List of Figures

List of Tables

1

Introduction

1.1 Context

Object manipulation with minimal human intervention has been an objective in robotics that has diverse applications and huge economical opportunities [2] [3]. 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 [4].

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 [5] [6] 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) [1] 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.

1.2 Research goal

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.

The goal of this thesis is cloth folding using visual observations as an input for a DRL algorithm. The thesis wants to check whether the robot can learn to recognize a piece of cloth and fold it accordingly to the visual observations that it receives as an input.

We want to experiment with the CURL algorithm [1] which is the first RL algorithm that accelerates visual recognition using Contrastive Representation Learning (CRL) [7] [8]. CURL coupple a RL algorithm with a CRL where CRL learns a representation from images by extracting the features using Contrastive Learning [9] and gives that information to the RL algorithm.

As a simulation engine, Unity has been chosen because it has a package called Machine Learning Agents (ML-Agents) that enables games and simulations to serve as environments for training intelligent agents. Those agents can be trained using reinforcement learning, imitation learning, neuroevolution, or other machine learning methods through a Python API.

2

Simulating cloth

2.1 Approaches to cloth simulation

The objective is to generate a convincing cloth simulation. However, for this purpose, physics has to be simplified depending on what is your objective or what part of the cloth's characteristics it is needed to focus on.

2.1.1 Continuous Model

Modeling an object as a continuum assumes that the substance of the object fills the space that it occupies allowing to represent the cloth as a heterogeneous textile (making it more realistic) and permits the use of low resolution models without losing basic material properties.

The equation that describes [10] the continuous theory is the Euler-Cauchy stress principle. Using a tensor it is possible to describe all the deformations that the object is suffering.

2.1.2 Discrete Model

The discrete model discretizes the cloth in a polygonal mesh. The vertices of the mesh are called particles and the topology of the mesh defines how particles interact with one another. Once the mesh that describes the cloth is known, forces on each particle are computed depending on its position and velocity as well as the position and the velocity of their neighbors. When the force (F) for each particle has been computed, Newton's equation of motion that governs the motion of particles is applied to know the trajectory of each particle with mass m at position x computed by:

$$F = \frac{d}{dt}(mv) = m\frac{d^2x}{d^2t} \tag{2.1}$$

Different discrete models differ in their methods of computing the forces [10]. In this thesis we use the mass-spring model.

2.2 Modeling cloth with mass-spring particle system

In this section, it is explained how we modeled the cloth simulation that this thesis will use in the robotic folding task. The cloth simulation is built following the subsections of section 2.2 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`.

2.2.1 Mass-Spring Model

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 2.1. 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.

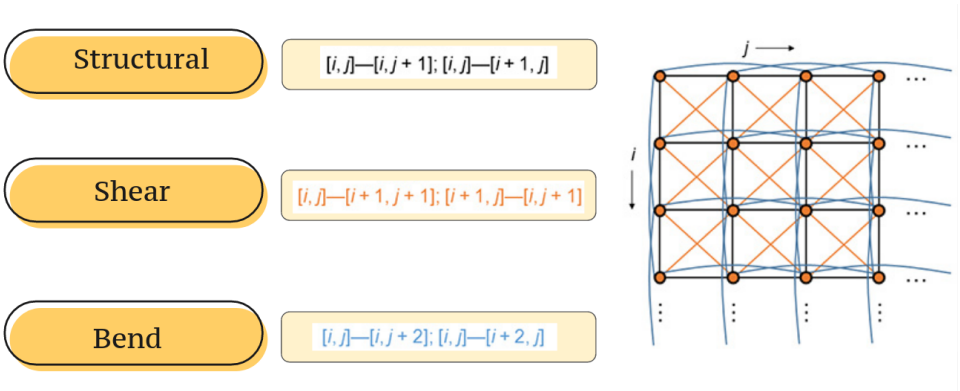


Figure 2.1: Mass-spring model. Where i and j are the positions of the particles in the mesh.

The elastic forces executed by linear springs between two particles x_i and x_j are given by equation 2.2:

$$\vec{F_{ij}^{e}} = k_{ij}(\left\|\vec{r_{ij}}\right\| - l_{ij})\frac{\vec{r_{ij}}}{\left\|\vec{r_{ij}}\right\|} \tag{2.2}$$

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:

$$\vec{F_{ij}^{d}} = d_{ij}(\vec{v_i} - \vec{v_j}) \tag{2.3}$$

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:

$$\vec{F^{g}} = -m\vec{g} \tag{2.4}$$

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:

$$\vec{F^w} = \rho \; A \; \vec{v^2} \; \vec{n_A} \tag{2.5}$$

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:

$$\vec{F^e} + \vec{F^d} + \vec{F^g} + \vec{F^w} = m\frac{d^2x}{d^2t} \tag{2.6}$$

2.2.2 Integration method

Once the total force for each particle is computed, it is time to apply Newton's equation of motion which governs the motion of the particles to know the trajectory of each particle. A numerical integrator is needed to solve these second order ordinary differential equation. The integration methods are divided in two groups:

- Explicit method: It calculates the state of a system at a later time from the state of the system at the current time.
- Implicit method: It finds a solution by solving an equation involving the current state of the system and the state in the past.

Verlet Integrator 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. To obtain the equation of this Verlet integration method, the Taylor series of $x(t + \Delta t)$ and $x(t - \Delta t)$ are computed until second order:

$$x(t + \Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)\delta t^2 + O(\Delta t^3) \tag{2.7}$$

$$x(t - \Delta t) = x(t) - v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 - O(\Delta t^3) \tag{2.8}$$

And then adding equations (2.7) and (2.8) the Verlet equation is found:

$$x(t + \Delta t) = 2x(t) + x(t - \Delta t) + a(t)\Delta t^2 \tag{2.9}$$

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

It is important to note that it is necessary to compute the velocity at every step (by dividing the amount of distance traveled between the delta time) because this velocity is needed to calculate the damping force.

$$v(t) = \frac{x(t) - x(t - \Delta t)}{\Delta t} \tag{2.10}$$

2.2.3 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) [11].

2.2.4 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.

2.2.4.1 Cloth-cloth collisions

Optimized spatial hashing for collision detection for deformable objects, is the technique chosen to optimize the collision detection [12]. 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.

Primitive hashing

First of all, the discretization of the points (nodes/particles of the mesh) and triangles has to be done to map them into the hash table. Given the cell coordinates (i, j, k) and a grid cell size of n, the index h of the hash table is:

$$h = H(i, j, k) = (x\ p_1 \oplus y\ p_2 \oplus z\ p_3)\ mod\ n \tag{2.11}$$

where H is the hash function defined by [12] and p_1, p_2, p_3 are large prime numbers: 73856093, 19349663, 83492791, respectively.

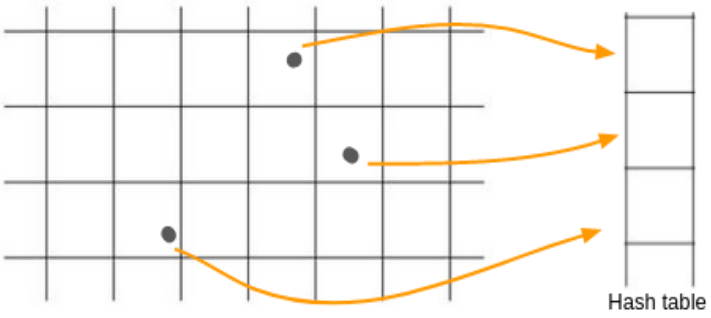


Figure 2.2: Points lie on one grid cell and are hashed into a single hash table entry.

Spatial hashing of points is easy to implement because they lie only on one single cell at a time. Given a point, its position (x, y, z) is divided by the grid size and rounded down to the next integer:

$$(i, j, k) = ([\frac{x}{s_{grid}}], [\frac{y}{s_{grid}}], [\frac{z}{s_{grid}}]) \tag{2.12}$$

To hash the triangles, first, the minimum bounding box is computed. Then, following the points procedure, the values are divided by the grid size and rounded down to the next integer. If a point and a triangle are mapped to the same hash index and the point is not part of the triangle, an intersection test has to be performed.

Intersection test: Point-Triangle intersection

The 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 [12]. 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.

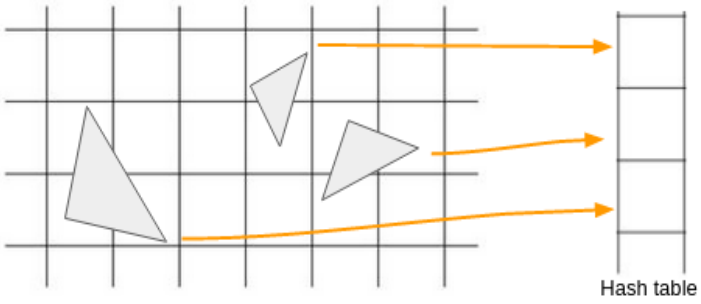


Figure 2.3: Triangles may lie on several grid cells and they could be hashed into one or more hash table entries.

2.2.4.2 Collision response

To provide more realism, once a collision is detected, it is possible to add physical responses to the collisions like friction force or energy dissipation.

Friction

The friction force is tangential to the plane to be able to simulate non-sliding contact.

$$\vec{F_s} = \vec{F_T} - k_f \|F_n\| \, \vec{u_t} \tag{2.13}$$

where $\vec{u_t} = \frac{F_t}{\|F_t\|}$ is the tangential direction of the force, $k_f(F/m)$ is the friction coefficient, $F_t(F/m)$ and $F_n(F/m)$ are the tangential force and normal force respectively and $F_s(F/m)$ is the friction sliding force simulated.

Impact dissipation

In an inelastic collision, the dissipated energy depends on the dissipation constant (k_d). If the constant is one, all the energy is dissipated and as a result, the particles will get stocked. On the contrary, if the constant is zero, the collisions will be completely elastic without losing any energy.

$$\vec{v} = \vec{v_t} - k_d \, \vec{v_n} \tag{2.14}$$

where $v_t(m/s)$ and $v_n(m/s)$ are the tangential velocity and normal velocity of the particle before the collision respectively and $v(m/s)$ is the velocity after the collision.

Total response: Implementation in model

After doing the projection of the particle, the equation 2.15 is used to implement friction and impact dissipation. As a result, the particle has a new realistic position and energy.

$$x = (v_t - k_f \|v_n\| \frac{v_t}{\|v_t\|} - k_d \, v_n) \, \Delta t \tag{2.15}$$

2.2.4.3 Plane-cloth collisions

Plane-cloth collisions are those that deal with the collisions between the patch of cloth and the table where the cloth is folded. They are dealt differently because it doesn't require as much computation time as the cloth-cloth collisions.

All the particles are checked if they are above or over the plane. To do so, the dot product between the normal of the plane and the vector distance that go from the plane to the particle is calculated. If the dot product is negative then a correction has to be done.

$$\vec{r} \cdot \vec{n} < 0 \tag{2.15}$$

Where $\vec{r} = p_{particle} - p_{plane}$ and $\vec{n}$ is the normal of the plane.

If the particle satisfy the condition in equation 2.15, projection is done to correct position of the particle. Friction force and an impact dissipation are applied to simulate a more real situation.

2.2.4.4 Robot-cloth collisions

The collisions between the patch of cloth and the robot are handled using Unity collision system. The robot, as well as the nodes of the mesh, have colliders associated to detect collisions. When a collision is detected, the function "OnCollisionEnter" is activated and the position of the first point of collision is saved. The function "OnCollisionStay" is called upon every time that the particle remains inside the robot collider. Inside this

last function, the depth of the penetration is computed with the actual position and the position of the first point of contact. Finally, the position of the particle is corrected by adding the amount of penetration.

2.2.5 Order in calculations

The simulation has to do some steps to calculate the next position of the particle. The order of these steps is very important for the correct running of the simulation. The steps are as follows:

1. Calculate total forces: First of all, the calculation of the forces that affect every particle is computed.
2. Verlet Integrator: The calculation of the next position of the particles is calculated using Verlet Integrator.
3. Collisions correction: The corrections due to the collisions are applied.
4. Solve constraints: The corrections to avoid the superelasticity are done.

The wrong order of these calculation steps can induce instabilities in the simulation.

2.2.6 Constants range and limitations

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. In equations 2.16 and 2.17 there is the superelasticity correction.

$$if(ParticleA.isActive)ParticleA.Position- = 0.5 \cdot direction \cdot f \tag{2.16}$$

$$if(ParticleB.isActive)ParticleB.Position+ = 0.5 \cdot direction \cdot f \tag{2.17}$$

Where f is the amount that the spring distance has increased or reduced compared with the last step, the direction is the direction from $ParticleB$ to $ParticleA$, and 0.5 is the constant that can be changed in the range of 0.01 to 0.5.

On the other hand, the collision response is not acting as expected. The implementation of friction and dissipation works but is limited. The range of the constants is from 0 to 1 but the difference is only noticed when it is in one of these numbers. To avoid further problems the friction and dissipation are deactivated. As a substitute, some particles get stuck on the plane as soon as they get in contact. This prevents the cloth from slipping, allowing limited dynamic movement.

The fact that even when the constants (elastic and damping) are zero the behavior of the cloth remains working and the explained problems with collision correction constants, leads us to believe that the magnitude order of the superelasticity correction is higher than the force implemented by the springs and the collision correction. 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.

Now, we have the cloth simulation ready to be used by the robot for the folding task. One important thing to have present in the future chapters are the limitations explained in this subsection to avoid further problems at the time to couple the robot with the programmed cloth simulation explained.

3

Contrastive methods for reinforcement learning

3.1 Reinforcement Learning

Reinforcement learning (RL) is the area of machine learning that deals with sequential decision-making. RL problem can be formalized as an agent that has to make decisions in an environment in order to maximize the notion of cumulative rewards. One important aspect of RL is that an agent has to learn a good behavior using trial-and-error experience. This makes the agent learn new behaviors incrementally. The main important actors of RL are:

- Agent: is the main actor. Is the learning character that observes the environment, selects and performs actions.
- Action: Is a set of possible movements that the actor can perform.
- Environment: Is the world where the agent can move and interact.
- State: Is a description of the state of the environment.
- Reward: Is the reward given by the environment, in other words, is the feedback that the environment gives to the agent.

- Policy: Is a probability distribution over actions given states.
- Value: Is the long term return of a state given a policy.
- Observation: Is the input information that represent what the agent is seeing at that point in time.

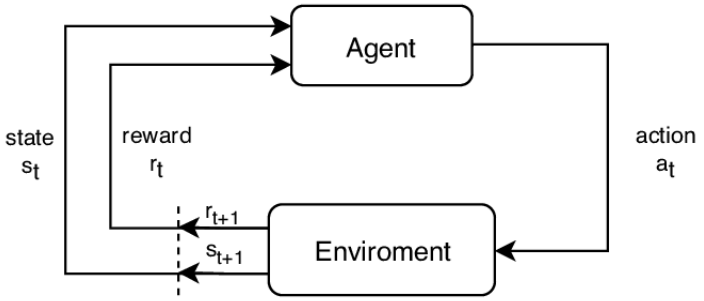


Figure 3.1: Diagram of the reinforcement learning setting.

As we can see in figure 3.2, at every step of interaction, the agent observes the environment and based on that, select and perform an action. Finally, after performing the action, the agent receives a reward from the environment.

The RL algorithms can be divided in [13]:

- The value-based learning algorithms create an action-value function $Q^{\pi}(s, a)$ with all the action-state samples received. With this value function, it extracts a policy $\phi(a_t|s_t)$ that will take the best action at a certain state given. The advantages of value-based methods are that the sample efficiency is high, the variance of the action-value function is small and that it is difficult to fall into a local maximum. The disadvantages are that it cannot be applied in the continuous action space problems.
- The policy-based learning algorithms create directly a policy $\pi(a_t|s_t)$ mapping all the pairs action-space. This policy gives the best action at a certain state given. The advantages are that it has better convergence and it can be applied into continuous and discrete action space problems.
- On-policy methods improve the policy that is used to take actions. Selects an action based on the current policy, executes the action and uses the next received data to update the policy.

- Off-policy methods improve the action-value function following a specific policy. Selects an action following and independent policy, executes the action and uses the next received data to update the action-value function. The policy to take actions can be deterministic, discrete probability distribution for discrete action spaces, or stochastic, continuous probability distribution for continuous action spaces.

- Finally, there are the actor-critic methods that combine value-based methods and policy-based methods. It uses the value-based method to learn an action-value function $Q^{\pi}(s, a)$ to improve the sample efficiency and also it uses the policy-based method to learn a policy function. Actor-critic methods absorb the advantages and the disadvantages of both methods.

Deep Reinforcement Learning (DRL) [14], combines Deep Learning (DL) and RL creating a RL process where DL determines the action to take at every step. In DRL the state is represented by high-dimensional representations like images or videos that provide information about the agent context. DRL algorithms that learn from visual observations are inefficient [5] [6] because first, they have to extract the state representation from the visual observations and then perform the RL.

3.2 Soft Actor Critic (SAC)

Soft Actor Critic(SAC) [15] 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 3.1.

$$\mathcal{L}(\psi) = -\mathbb{E}_{a_t \sim \pi}[\, Q^{\pi}(s_t, a_t) - \alpha \log \pi_{\psi}(a_t|s_t)] \tag{3.1}$$

At the same time the critics Q_{θ_1} and Q_{θ_2} are trained to maximize the Bellman equation (3.2) 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 3.1.

$$\mathcal{L}(\phi_i, \mathcal{D}) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}}[\, \frac{1}{2}(\, Q_{\theta_i}(s_t, a_t) - r(s_t, a_t) - \gamma \mathbb{E}_{s_{t+1} \sim p}[\, V(s_{t+1})]\,)^{\,2}] \tag{3.2}$$

3.3 Contrastive Representation Learning (CRL)

Contrastive representation learning (CRL) [7] is the process of learning a representation from high-dimensional input data by extracting the features using contrastive learning [8] [9]. Contrastive Learning is a mechanism to learn representations that obey similarity constraints in a dataset organized by similar and dissimilar sets as we can see in the example of figure 3.2.

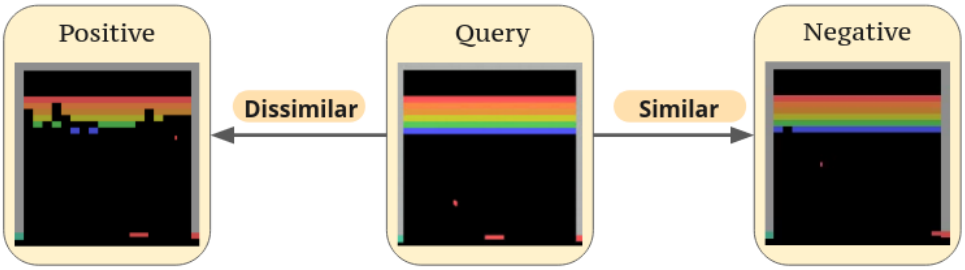


Figure 3.2: Example of similar and dissimilar pairs.

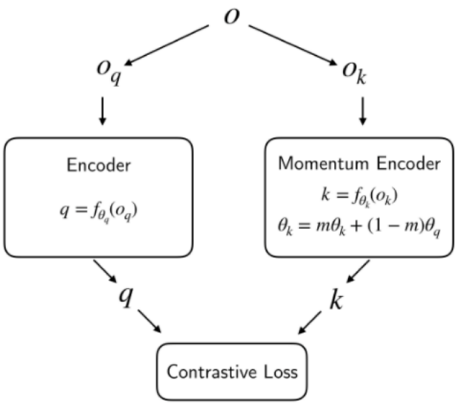


Figure 3.3: Schema of Contrastive Representation Learning [1].

CRL, trains a visual representation encoder network that extracts representation vectors from augmented data samples. In the figure 3.3, we can see how the CRL algorithm [16][17][18] 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.

$$\mathcal{L}_q = \log \frac{exp(sim(q, k_+))}{exp(q, k_+) + \sum_{i=0}^{K-1} exp(sim(q, k_i))} \tag{3.3}$$

The contrastive loss 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-).

3.4 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 3.4 is possible to see that CURL trains the encoder q that is shared with the RL algorithm.

The CURL (Srinivas, A., 2020) [1] implementation, is coupled with SAC [15] 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 [18].

3.5 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

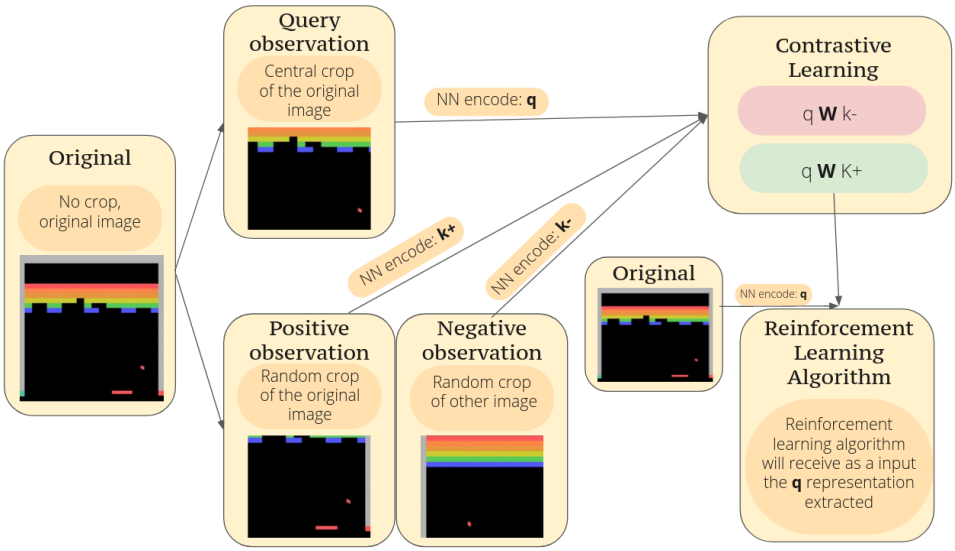


Figure 3.4: Schema of CURL

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.

The next chapter explains how to couple the programmed cloth simulation with the CURL algorithm explaining the design of the reward function used and the setup of the experiment framework using a package of Unity called ML-Agents.

4

Environment for learning to fold in simulation

4.1 Simulation robot in the cloth environment

In this section, it is explained how to setup the robot simulation, the cloth simulation and the CURL algorithm.

4.1.1 Robot model

Once the cloth simulation is physically plausible, the next step in the thesis is taken. This consists of using a robot to move the cloth on the scene. Baxter is imported into Unity as a Unified Robot Description Format (URDF) to use its arms for the cloth folding task.

The URDF is an XML specification to describe a robot physically. In that file are represented three structures:

- Kinematic and dynamic description of the robot.
- Visual representation of the robot.
- Collision model of the robot.

4.1.2 Arm control

To control the arms of the robot, an inverse kinematics script is used (in both hands) which allows moving the arm while it follows a target [19]. It has three important components:

- Number of bounds: Total number of joints that has the arm.
- Pole position: The pole position is a Transform that allows creating a preference of how the bones move.
- Target position: The target position is the position where we want the gripper to be placed.

4.1.3 Cloth detection

The mechanism of how the gripper of the robot can grasp the cloth is out of the scope of this thesis. For this reason, a magnetic behavior between the gripper of the robot and the cloth is implemented. When the gripper is close enough, the cloth gets automatically attached to the gripper. This magnetic behavior can be deactivated manually in the case that it is needed to leave the cloth somewhere in the scene. To know if the cloth is attached to the gripper, collisions are checked between both of them. If there are no collisions, it means that the cloth is not attached to the gripper.

This magnetic behavior is implemented using Unity collisions and by changing the hierarchy of the game objects in the scene. When the function *OnCollisionStay* is activated (activated when a collision between two Unity rigid bodies is currently happening), it means that the gripper and the cloth are colliding. When this happens, the cloth is attached as a child in the hierarchy of the gripper. If the magnetic behavior is deactivated, the cloth is detached from the gripper hierarchy and the function *OnColliderStay* is not activated, the cloth is not attached to the gripper. In this case, the Unity function *OnColliderExit* is called, the last time that the collision has happened.

4.1.4 Agent

The folding task behavior that we want to train is designed as simply as possible to make things easier for the learning process. Instead of implementing two agents (multi-agents), it is implemented only one agent with an action space of size six. The first three positions of the vector are the incremental distance added to the current position of the left arm

and the last three positions of the vector are the incremental distance added to the current position of the right arm.

As an action, continuous movement has been chosen because it allows more freedom of movement as well as more softness than the discrete space.

Finally, to get the visual observations a camera is placed in the environment. This camera can be moved to another position. In the training, the camera position has been used as a hyperparameter training for the folding task with four different camera positions.

4.1.5 Unity: ML-Agents

The Unity Machine Learning Agents (ML-Agents) is a package of Unity that enables games and simulations to serve as environments for training intelligent agents [20]. Agents can be trained using reinforcement learning, imitation learning, neuroevolution, or other machine learning methods through a Python API.

The learning environment contains the Unity scene and all the game characters. The Unity scene provides the environment in which agents observe, act, and learn. The learning environment contains two Unity components to organize the Unity scene:

- Agents: It is attached to a Unity game object and generates its observations, performing the actions that it receives and assigning a reward.
- Behaviour: defines specific attributes of the agent such as action space and observation space. Behavior can be thought of as a function that receives observations and rewards from the agent and returns actions.

It is needed to define three entities at every moment of the training:

- Observations: The input observations are images generated from the cameras attached to the agent and that represent what the agent is seeing at that point in time.
- Actions: Are what the reinforcement learning algorithm sends to the agent based on the input observations sent to the algorithm. Actions can be discrete or continuous depending on the environment necessity.

- Reward: Is a scalar value indicating how well the agent is doing. The reward signal is provided every time that the agent receives an action to send the information to the algorithm regarding how well or badly it is doing.

Every learning environment will always have one agent for every character in the scene. While each agent must be linked to a behavior, it is possible for agents that have similar observations and actions to have the same behavior.

4.1.6 Python

ML-Agents package contains a low-level Python API for interacting and manipulating the learning environment. This API is contained in the *mlagents−envs* Python package. This communicator can be used to use Unity as the simulation engine to implement different machine learning algorithms. We can see in figure 4.1 the exchange of information through the API.

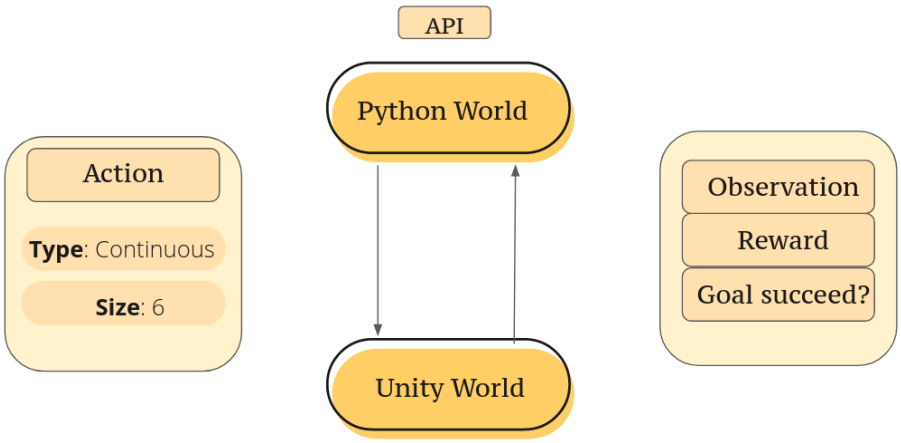


Figure 4.1: Exchange of information between Unity and Python using API

4.1.7 Implementing CURL in Unity

In the current thesis the Python API connects the CURL algorithm with the cloth simulation and the robot simulation implemented in Unity.

As we can see in figure 4.1, the observations from the Unity world, taken by the camera in Unity, are sent to the algorithm in the Python world, through the API, then these

observations are processed by the algorithm and the action resultant is sent again through the API to the Unity world.

In the implementation it is used a wrapper to customize and extend the environment functionalities. If the environment is wrapped, it gains new functionalities. For example, the Unity environment gives observations, but it is needed to accumulate them providing to the agent the N last observations to allow it to learn the temporal dimension. Another point is that the Python API from ml-agents is not very easy to use because is low-level connection. In this thesis, all the functionalities are wrapped following the ideas of [21].

4.2 Folding task

In this section, it is explained how the reward function works, the setup in Unity and some limitations of the implementation.

4.2.1 Cloth folding overview

Before designing the reward function is needed to look into the folding task to understand how it works to able to design an accurate reward function. In the scope of this thesis, a square patch of cloth has to be folded. Viewed from an upper lever there are two clear steps:

- First step: The robot has to get one side of the cloth attached to the gripper.
- Second step: While the robot has one side of the cloth attached to the gripper has to reach the goal position.

4.2.2 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 the robot losing it. Nevertheless, as we can see in the schema of figure 4.3, to implement a more general behavior the reward function will always check if the cloth is attached to the gripper. In figure 4.2, we can see the two basic steps of the reward function:

(a) Step one.

(b) Step two.

Figure 4.2: Steps to accomplish the folding task.

- 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 [22], 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 [15], the reward function is normalized to make the algorithm more efficient. The normalization is done by divide all the reward values by 500.

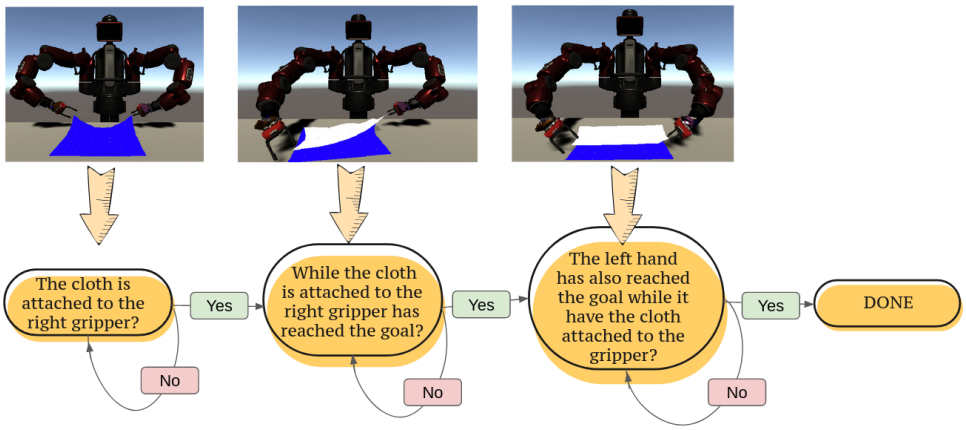


Figure 4.3: Schema of the reward function explanied in section 4.2.2.

4.2.3 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.

4.2.4 Implementation problems

Now, we have the setup of the experimental environment that consists of the cloth and the robot simulation with CURL algorithm.

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`.

5 Results

5.1 Experimental framework

Using the setup explained in chapter 4, 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.

As explained in section 4.1.4, the camera position has been used as a hyperparameter for the training of CURL algorithm. In chapter 3, 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 5.1 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 5.1 are taken from the CURL implementation [1].

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
$(\beta_1, \beta_2) \rightarrow(f_\theta, \pi_\Psi, Q_\Phi)$	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 5.1: Hyperparameters for CURL [1]

5.2 SAC results

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

In figure 5.1 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

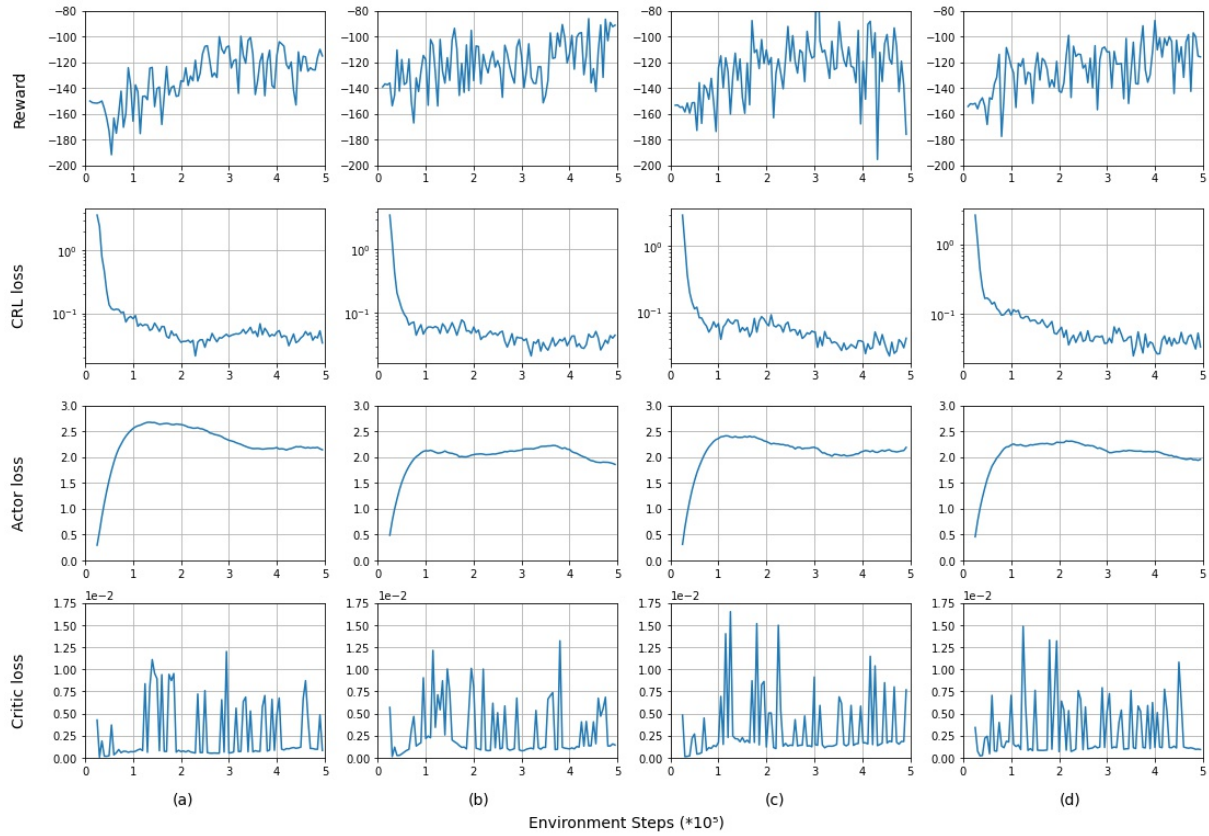


Figure 5.1: In this plot there are the training curves of the CURL algorithm. In each column, the visual observations are taken from different camera positions related to the figure 5.2. We can see that the CRL loss converges very fast meaning that extracts good features from the observations. On the 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 5.2: Examples of perspective of the different camera positions related to the training curves.

that receive as an input. Probably because the images are in two dimensions and the environment is in three dimensions.

As explained in section 3.2, 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.

5.3 CRL results

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

In figure 5.1 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. As it is explained in sections 3.3 and 3.4, 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, explained in section 3.3, which has good results.

5.4 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 [1]. 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) [23] 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 [1] with the RLAD paper [23], we can see that RLAD [23] has better results than CURL [1]. However, this always will depend on the task and the training

preferences. As it was already presented by Chen et al. 2020 [22] visual augmentations is a way to extract more value to unsupervised learning.

6

Conclusions and future work

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, as explained in section 2.2.6, the constants (spring constants and collision 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.

6.1 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.

Bibliography

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

[2] 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.

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

[4] 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.

[5] 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.

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

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

[8] 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. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2020.3031549

[9] 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, 2006, pp. 1735–1742.

[10] M. Wacker, B. Thomaszewski, and M. Keckeisen, "Physical models and numerical solversfor cloth animations," 2005.

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

[12] 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.

[13] H. Zhang and T. Yu, *Taxonomy of Reinforcement Learning Algorithms*. Singapore: Springer Singapore, 2020, pp. 125–133. [Online]. Available: https://doi.org/10.1007/978-981-15-4095-0_3

[14] 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. [Online]. Available: http://dx.doi.org/10.1561/2200000071

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

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

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

[18] 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.

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

[20] 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.

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

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

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