Towards a Sample Efficient Reinforcement Learning Pipeline for Vision Based Robotics

Pierre BELAMRI  
Research Intern CAOR  
MINES ParisTech, PSL Research University  
Paris, France  
pierre.belamri@mines-paristech.fr

Maxence MAHÉ  
Research Intern CAOR  
MINES ParisTech, PSL Research University  
Paris, France  
maxence.mahe@mines-paristech.fr

Jesùs BUJALANCE MARTIN  
CAOR  
MINES ParisTech, PSL Research University  
Paris, France  
jesus.bujalance_martin@mines-paristech.fr

Abstract—Deep Reinforcement learning holds the guarantee of empowering self-ruling robots to master enormous collections of conduct abilities with negligible human mediation. The improvements brought by this technique enables robots to perform difficult tasks such as grabbing or reaching targets. Nevertheless, the training process is still time consuming and tedious especially when learning policies only with RGB camera information. This way of learning is capital to transfer the task from simulation to the real world since the only external source of information for the robot in real life is video. In this paper, we study how to limit the time taken for training a robotic arm with 6 Degrees Of Freedom (DOF) to reach a ball from scratch by assembling a pipeline as efficient as possible. The pipeline is divided into two parts: the first one is to capture the relevant information from the RGB video with a Computer Vision algorithm. The second one studies how to train faster a Deep Reinforcement Learning algorithm in order to make the robotic arm reach the target in front of him. Follow this link to find videos and plots in higher resolution: https://drive.google.com/drive/folders/1_lRlDSoPzd_GTcVrxNip10o_spDp?usp=sharing

Index Terms—Robotic, Reinforcement Learning, Computer Vision, imitation learning

I. INTRODUCTION

Today, technological advances have allowed robots to manipulate objects and perform tasks just as well or even better than humans [1]. This is evidenced by the increasing automation of assembly plants and even the use of robotic arms on Mars rovers. The applications are almost infinite, from surgical medicine to the dismantling of nuclear power plants and the handling of radioactive waste. However, programming these robots is tedious and time consuming. In addition, they cannot adapt to a new environment which limits their usage and their interaction with humans. Therefore, one of the research challenges is to make the collaborative robots as adaptive as possible. It is necessary that the behavior of these robots is not defined in a fixed way and written by a programmer.

Recent breakthroughs especially in Deep Reinforcement Learning (DRL) have led to uncommon capacities in self-governing tasks such as reaching a target or even grabbing only square objects. Reinforcement Learning (RL) algorithms learn how to act by trial and error [2]. By succeeding or failing, the agent gets short-terms rewards that add up to a long-term reward. The ultimate goal is to define the best sequence maximizing the long-term reward. DRL incorporates deep learning in the learning process allowing agents to make decisions from unstructured input data without manual engineering of the state space [3]. DRL has mastered and surpassed humans in well known classical games such as Chess and Go and video games such as Dota5 [4].

However, reinforcement learning methods need hundreds of hours of training and important computational power making
them extremely sample inefficient. It’s why, today, robotic agents learn their policy in a simulation environment such as RLBench in order to test millions of steps more easily [5], [6]. This accelerates the training. Nevertheless, one of the drawbacks of simulation is a policy with a high variance, hardly transferable to tasks in the real world. Indeed, the robot learns inside a simulation environment where it can know its coordinates and the coordinates of the target in real time. Therefore it is able to locate itself in space and more easily learn how to reach. It is not possible to know this much information in real life training only with cameras and the joints’ positions.

In this paper, we propose a training pipeline using off policy RL algorithm to efficiently learn how to reach a target. Our pipeline is in three stages: 1) an imitation learning process. 2) a computer vision algorithm locates on the camera stream the target and the robot’s head. 3) a DRL algorithm is trained based on these observations, to learn how to reach. This pipeline is easy to adapt to any reaching and grasping situation since it relies only on imitation learning and camera observation and is also easy to modify. In order to assemble the most efficient pipeline we had to study different configurations with different algorithms, this benchmark was an important part of our work and is detailed in this paper. In addition, for security purposes, we studied in RLBench [6], a simulated environment in order to try different configurations and not to damage the real robot. The ultimate goal is to transfer what was learnt in the simulated environment, in real life with as few changes as possible. It is why our simulated environment was very close to reality with cameras placed around the robot in order to recreate the real world configuration.

II. RELATED WORK
A. Deep Reinforcement Learning (DRL)

Deep reinforcement learning overcame the difficulties encountered when dealing with high dimensional state-space where previous RL methods had issues to choose the most efficient task. Thanks to its neural network, DRL achieved state of the art results on difficult tasks such as self driving cars [7] to poker game [8]. It is especially useful to learn tasks from pixel inputs such as in video games [9] or simulation [6]. However the combination of off-policy learning and high-dimensional, nonlinear function approximation with neural networks presents a major challenge for stability and convergence. In this paper we studied one of the more recent off policy algorithms in DRL which is the Soft Actor Critic method [10] that achieved state of the art results. The choice of an off-policy algorithm is because off-policy algorithm are more sample efficient for difficult tasks such as robotic manipulation.

Soft Actor Critic (SAC) The Soft Actor Critic algorithm is an off-policy method relying on two major pillars. The first one is the actor critic architecture with a separate policy and value function, the second is the maximization of the entropy in order to favor stability and exploration. Actor-Critic algorithms have an actor network and a critic network which updates the actor network with a predefined frequency [11]. In other words, it alternates between evaluation and improvement. Off policy actor-critic obtains faster training times than on-policy. However, it is very unstable and difficult to train [12]. To answer this problem SAC introduces the soft actor critic by adding the maximization of the entropy to the algorithm in addition to maximizing the expected return. The result is better stability and faster training with state of the art results, regardless of the policy parameterization.

Algorithm 1 Soft Actor-Critic

Inputs: The learning rates, $\lambda_\pi$, $\lambda_Q$, and $\lambda_V$ for functions $\pi_\theta$, $Q_\phi$, and $V_\psi$ respectively; the weighting factor $\tau$ for exponential moving average.

1: Initialize parameters $\theta$, $\psi$, and $\tilde{\psi}$.
2: for each iteration do
3: (In practice, a combination of a single environment step and multiple gradient steps is found to work best.)
4: for each environment setup do
5: $a_t \sim \pi_\theta(a_t | s_t)$
6: $s_{t+1} \sim P(s_{t+1} | s_t, a_t)$
7: $D \leftarrow D \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
8: for each gradient update step do
9: $\tilde{\psi} \leftarrow \psi - \lambda_V \nabla_{\psi} J_V(\psi)$.
10: $w \leftarrow w - \lambda_Q \nabla_{w} J_Q(w)$.
11: $\theta \leftarrow \theta - \lambda_\pi \nabla_{\theta} J_\pi(\theta)$.
12: $\psi \leftarrow \tau \psi + (1 - \tau) \tilde{\psi}$.

B. Imitation Learning

Imitation Learning aims to mimic the behavior of an expert for a certain task. One way to leverage demonstration data for an off-policy DRL agent is to initialize the agent’s memory with demos showing him how to succeed the task [15]. Implementing imitation learning in a reinforcement learning process enables the agent to learn faster and therefore decreases the training time [16]. The most natural form of imitation consists in creating a memory buffer of perfect behaviors and feed to the agent in order for him to take examples from it. In this paper, we use demos from the simulation environment. For each demo we record the visual representation and the proprioception of the robot (joints’ positions).
C. Single Shot Detector

Single Shot Detector (SSD) is a computer vision algorithm used for its capacity to process frames in real-time. Unlike Mask-R-CNN [17] and Faster-R-CNN [18] running respectively at 5 fps and 7 fps, SSD can run at more than 50 fps without losing in precision [19]. The backbone of the algorithm can change according to needs, in this paper, we use a MobileNetV1 [20] backbone in order to be implemented easily without needing high computational power. The SSD network feeds forward the image only once and uses the output feature maps of the backbone network as the inputs for its layers. In addition, SSD uses Atrous convolution [21] instead of conventional convolution which enables the network to have fewer number of parameters and therefore be faster.

D. Contrastive Learning

Contrastive Learning (CL) is a machine learning technique used to extract high dimensional features from an unlabeled data set by learning which data points are similar or different [22]. This self-supervised learning method works by maximizing a contrastive loss function if two pairs of images are similar and by minimizing it if they are different. It relies on data augmentation (crop, color distortion...) in order to create a positive pair. The images (positive and negative pairs) are fed to a neural network and a projection head. Then the similarity function tells if the images are similar [23]. The loss function used in this paper is the InfoNCE loss [24].

III. Overview

To provide an overview of our work, we have to begin introducing the reaching task, followed by the environment in which we are performing this task, and then, an overview of the pipeline will be presented, before studying in detail the different parts of it.

A. Environment

CoppeliaSim [25] is a simulator commonly used in the robotics community. Within CoppeliaSim, there are a number of options for physics simulators, like Bullet. For this we decided to use and control it with Python thanks to the PyRep [26] toolkit, to which we added the learning environment RL-Bench [6], which offered an initial structure for reinforcement learning. It is in this environment that we carried out the training on the reach task.

B. Reach Task

The reaching task is a basic element of robotic manipulation. It consists in the robot’s gripper to reach the target while satisfying Cartesian space constraints. We aim to use reinforcement learning to train a UR3 on one goal attainment task, during which a policy function is trained to learn to map current states to the robot’s next action to reach the target goal. An example of a trained goal attainment is shown in Fig.3. Many recent studies have targeted position-only reaching [27] or situations wherever the tip effector is forced to reach the target from one direction, usually the upper vertical [28] one. Such simplification will greatly simplify tasks however is tough to generalize to interact with objects that need to be reached in a certain way.

C. Pipeline

Before studying in detail the different parts of our training pipeline, her is an overview of it.

First, the training process starts with the pixel inputs from the camera being analyzed by a computer vision algorithm
in order to extract the important features and feed them to the agent. The vision based method consisted in 4 different computer vision algorithms to experiment. One is a simple neural network with 2 convolutional layers. 2 experiments are focused around the SSD algorithm: one detects only the \((x, y)\) pixel coordinates of the ball and the other computes the distance in the picture between the head and the target. In order to adapt the algorithm to our problem, we created and annotated our own data-set of robot head and target. We then used an SSD model pre-trained on the COCO data-set and fine tuned it on ours. The last is a contrastive learning algorithm that learns high-level features from the images then, like the others vision methods, feeds it to the RL agent.

IV. BACKGROUND

Our method is based on the continuous reinforcement learning mechanism, using the SAC as the basic framework. Hence, we first introduce the background of RL and SAC, followed by an explanation on the vision algorithm we used. Then, the overall approach and pipeline is presented in detail.

A. Deep Reinforcement Learning

Object reaching is a control problem in a stochastic environment where the robot acts by choosing actions sequentially over sequences of time steps \(t\), and the goal is to maximize a cumulative reward. We can formalize this RL reaching problem as a Markov decision process (MDP). This process can be modeled as a tuple \((S, A, P, R, \gamma)\), where \(S\) and \(A\) denote the state and action space respectively, \(P\) is the transition probability matrix. \(P := P[s_{t+1} = s' | s_t = s, a_t = a]\) representing the probability distribution to reach a state, given the current state and taking a particular action, \(R\) is the state-action associated reward space and can be pictured as \(r_t\), and \(\gamma \in [0, 1]\) is the discounted factor.

The goal of the robot is to learn a policy \(\pi_\theta\) by interacting with the environment, the policy (i.e., the robot controller) maps \(S \rightarrow A\) and is used to select actions in the MDP. It is the “deep” part in DRL as it is commonly parametrized as a Deep Neural Network where \(\theta\) is the general parameter storing all the network’s weights and biases. In the case of SAC which is an actor-critic algorithm, 3 neural networks are used. One for the policy and two for the critic. The policy can be represented as \(\pi(a_t | s_t)\), which guides the agent to choose action \(a_t\) under state \(s_t\) at a given time step \(t\) to maximize the future \(\gamma\)-discounted expected return \(R_t = \mathbb{E} \left[ \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \right]\).

We performed this learning through the Soft Actor Critic algorithm.

Soft Actor Critic

We started using the TD3 [29] algorithm, but we failed to achieve any sign of learning with it, so we switched to the SAC [10] algorithm, because it is currently one of the most reliable reinforcement algorithm.

Instead of solely seeking to maximize the lifetime rewards, SAC [10] seeks to additionally maximize the entropy of the policy. It brings several advantages: first, it explicitly encourages exploration of the state space and it allows the policy to capture multiple modes of good policies, preventing premature convergence to bad local optima.

It changes the RL problem to :

\[
\arg \max_{\pi} \mathbb{E}_{\pi} \left[ \sum_t \gamma^t (r_t, a_t) - \alpha \log(\pi(a_t | s_t)) \right]
\]

where \(\alpha\) is the non-negative temperature parameter.

B. Computer Vision

In order to rely only on observations from the cameras, a Computer Vision algorithm extracts the features from the RGB pixels. We first tried with a simple 2 layer Convolutional Neural Network (CNN) however this was not sufficient. We then implemented two Deep Learning algorithms: a Single Shot Detector and a contrastive learning algorithm.

Single Shot Detector

The Single Shot Detector (SSD) is a Deep Learning algorithm based on a feed forward convolutional network that produces bounding boxes around areas of interest and then applies a non max suppression to eliminate the ones that have a low confidence score [19]. As seen on Fig. 5, the backbone of our used SSD is a MobileNet architecture [20]. This has the advantage of being quick and efficient while still achieving high performance.

The SSD algorithm was chosen over the YOLO algorithm [30] for its capacity to be more precise especially for small objects. This was a requirement for our research since we had to detect a small ball in front of the robot. This increased precision of the SSD is explained by its architecture. Indeed, since the SSD applies more convolutional layers to the feature maps of the backbone and uses the outputs of all layers; the early layers bearing a smaller receptive field are more capable to detect smaller objects of interest [31]. This difference with the YOLO architecture is a needed advantage for us.

The loss used for training is the original loss from the paper which is a is a weighted sum of the localization loss (loc)
and the confidence loss (conf) and N the number of matched default boxes [19]:

$$L(x, c, l, g) = \frac{1}{N}(L_{\text{conf}}(x, c) + \alpha L_{\text{loc}}(x, l, g))$$ (2)

The confidence loss is a soft max loss of classes of confidence (c) and the localization loss is a L1 loss between the ground truth boxes (g) and the predicted ones (l). Finally (x) is the indicator for matching the ground truth box to the default box.

Unsupervised Representation learning: Contrastive Learning

In the case of computer vision, contrastive learning is a self-supervised method in order to maximize the agreement between two similar images and to minimize it between two different images by learning features. It is an approach to find similar and dissimilar data for a Machine Learning model. Contrastive learning in aid of model-free RL has experienced a recent resurgence since researchers have achieved state of the art results by incorporating it to the RL model. Contrastive learning relies on queries (q) and keys (k). Given a query q and a key k ∈ K = \{k_1, k_2, ...\}, if k is a positive k_+(similar to q), the goal is to maximize a similarity function \(f_{\text{sim}}\), if k is a negative k_−(different than q), \(f_{\text{sim}}\) needs to be minimized. In our case, \(f_{\text{sim}}\) is the dot product \(q^T k\). Other similarity functions exist such as the bi-linear product \(q^T W k\) or euclidean distances. The loss function used to learn embeddings is the InfoNCE loss [24]:

$$L_q = \frac{\exp(q^T W k_+)}{\exp(q^T W k_+) + \sum_{p=0}^{K-1} \exp(q^T W k_p)}$$ (3)

State of the art RL from pixels methods use what is called instance discrimination [22]. Instance discrimination generates keys and queries as two data augmentations of the same image, negative observations come from an other image. In this paper we use random crop data augmentation, where a random square patch is cropped from the original rendering as shown on Fig. 6, since it has achieved impressive results in computer vision.

C. The overall pipeline

The overall approach combines the Vision Algorithm and the Reinforcement Algorithm SAC.

Here is an overview of the different pipeline used during our vision based experiments.

- **The first pipeline** is based on the use of our SSD network. Each five steps, the RGB pixels from the front camera are passed through the SSD which predicts two bounding boxes, respectively for the robot’s head and the ball-target. If those two boxes are successfully predicted, the normalized distance between the center of the two boxes is computed and added to a buffer containing the 5 most recent distances. If they are not predicted, nothing is added to the buffer in order not to confuse the agent. This buffer is concatenated with the current angular positions of the robot’s joints to form an observation vector with 11 coordinates, these observations are the states \(s_t\) in the MDP, we can therefore write the state space \(S\) as \([0,1]^5 \times [-\pi, \pi]^6\).

- **The second pipeline** is very similar to the first since it also uses our SSD network. However, it only predicts the \((x,y)\) pixel coordinates of the ball on the front camera. Therefore, the observation buffer is a concatenation of the pixel coordinates of the ball and the joints’ positions of the robot. These coordinates are predicted at the start of each episodes, once they are known, we stop to feed forward the images in the computer vision algorithm. Indeed, since the camera is still and the ball does not move, the coordinates are the same through the episode. In this case, \(S = [0,1]^2 \times [-\pi, \pi]^6\).

- **The third pipeline** uses a simple 4 layers neural network to extract features from the front camera and the tip camera. The network consists of 2 convolutional layers and then 2 dense layers. The 2 images are concatenated by channels which results in a \((64, 64, 6)\) image. The output of this network is a vector of \(\mathbb{R}^{3000}\) and will be
concatenated with the proprioception observations of the robot to become the new state in input of the agent.

- **The fourth pipeline** uses a contrastive learning agent to extract high level features from the RGB pixels of the tip and front cameras. It is a 3 layers network: 2 convolutional layers and 1 dense layer with an output of dimension equals to 50.

As the precedent experiments, the feature vector is then fed into the agent with its proprioception observations. The action $a_t$ represents the velocities at each joints as we are controlling the robot through it, we can therefore write $A = \mathbb{R}^6$. All the pipelines are shown on Fig. 7.

V. EXPERIMENTS

In this section, we experiment different combinations of possible pipelines in order to find the most efficient one. We first start by only studying the reward inside the simulation with the ball’s and agent’s coordinates. This "cheating" method allows us to choose the best reward for our future experiments with a vision based method. Afterwards, we explain the vision based experiments with 3 main ideas based only on the camera stream with a Deep Learning algorithm that analyses this stream. The first uses the predicted bounding box around the head and the ball. The second one uses the $(x, y)$ pixel coordinates of the ball still with the SSD network at the start of an episode. The third one uses the 4 layers network to extract features from the pixels then feeds to the agent. Finally, we also studied the impact of imitation learning on the agent’s training. We then elaborated a final benchmark of all the methods that we experimented in order to have the most sample efficient pipeline. Then we try to use contrastive learning to make our agent learn.

Baseline for Benchmark: Every experiments were done with the UR3 6 DOF robot within the RLbench simulator. The metric used is the success rate over the past episodes. It is important to understand that the first experiment to study the reward was done using the ball’s and agent’s coordinates but all the others were done only with a vision based method.

A. Experimental Setup

To perform our experiments, we set up an RLbench environment for the reaching task. As said earlier, the robot used is the UR3 6 DOF robot. In the virtual scene, we defined two cameras, one attached to the robot’s gripper and the other to the front of the scene (Fig. 8), they will be used during the vision-based method. Each one of these cameras return a $64 \times 64$ image.

On the algorithm side, we fine tuned the hyperparameters as shown in the following table

| Hyperparameters          | Symbol | Value  |
|--------------------------|--------|--------|
| Discount factor          | $\gamma$ | 0.95   |
| Temperature              | $\alpha$ | 0.2    |
| Temperature learning rate| $\mu_{\alpha}$ | $10^{-4}$ |
| Target update ratio      | $\tau$ | 0.005  |
| Actor learning rate      | $\mu_Q$ | 0.005  |
| Critic learning rate     | $\mu_\pi$ | 0.005  |
| Memory buffer size       | $B$    | $10^3$ |
| Batch size               | $N$    | 32     |
| Number of episodes       | $M$    | $3 \times 10^4$ |
| Steps per episode        | $T$    | 100    |

B. Reward Benchmark

The 1\textsuperscript{st} experiment was to study the impact of different rewards on the learning speed. We studied mainly 2 rewards:

- A **sparse reward** that is close to the "natural" reward we would use for humans. It consists of giving a reward only at the end of an episode, 1 if the agent touched the target and 0 if not. Here, $\epsilon$ is the radius of the ball, $s$ the agent’s position and $s_g$ the target’s position:

$$r(s, a) = \begin{cases} 
1, & \text{if} \|s - s_g\|_2 < \epsilon \\
0, & \text{else} \end{cases}$$

- A **continuous reward** that returns to the agent a reward at each step instead of each episode. We give him the exponential of minus his distance to the target with a
minus in front in order to make him maximize it. If he touches the ball, a bonus of 100 is given:

\[
r(s, a) = \begin{cases} 
  \exp(-\|s - s_g\|_2^2) + 100, & \text{if } \|s - s_g\|_2 < \epsilon \\
  \exp(-\|s - s_g\|_2^2), & \text{else}
\end{cases}
\]

We access the distance between the two objects within the virtual environment, nevertheless, it’s not a cheating configuration because the reward function is only used during training, and we intend to carry out the training only in simulation.

After more than 18k episodes, the robot hadn’t learned anything with the sparse reward and couldn’t reach the ball. However with the continuous reward, he achieved great results of more than 99.5% of success rate as shown on Fig. 9.

**C. Computer Vision experiment**

Now that we know which reward is the most efficient one, we can proceed by studying which vision algorithm works better. The reward used for the next experiments is the continuous reward like for all other experiments. An important precision is to understand that giving this continuous reward is necessary and does not affect at all the agent if implemented in real life. Indeed, because of material constraints, we cannot train with the real robot. It is why we decided to train in simulation and to compute a working policy that is then put into the real robot. Therefore, during the training process in simulation we can use the coordinates as a **reward, not as an observation** to find the most efficient policy without cheating like in the previous reward benchmark. The 2\textsuperscript{nd} experiment used the predicted bounding boxes from the SSD algorithm to compute the pixel distance between the target and the agent. This observation was concatenated into a vector with the proprioception of the robot which are his joints’ positions. This vector was the input of the DRL agent. The results are given by Fig. 10. We can see that the agent is learning but at a very slow rate. Indeed, this took 7 days of training which is too long to be practical.

![Success rate by episodes for the continuous reward using the ball’s and agent’s coordinates.](image1.png)

**Fig. 9.** Success rate by episodes for the continuous reward using the ball’s and agent’s coordinates.

![Success rate by episodes for bounding box method.](image2.png)

**Fig. 10.** Success rate by episodes for bounding box method.

Since the precedent experience takes too much computational power and is not practical, we decided to try a 3\textsuperscript{rd} experiment by locating the \((x, y)\) coordinates of the ball on the image and feed it to the agent with its proprioception information. We can see on Fig. 11 that the results are much better: the agent learns better and faster which is convenient.

![Success rate by episodes for pixel coordinates method.](image3.png)

**Fig. 11.** Success rate by episodes for pixel coordinates method.

Then we have the 3\textsuperscript{rd} experiment using a simple convolutional neural network upstream of the agent to extract features and give them as inputs. At first, the results were showing no learning pattern and after more than 8k episodes the agent did not seem to learn. It is why we decided to help him by adding **Imitation Learning** to the process. Imitation Learning intuitively improves the learning since the robot has demos to take example from. To put demos in the agent’s buffer is very useful especially for tasks we know perfectly how to execute such as reaching a target. Fig. 12 illustrates the improvements brought by imitation learning, the agent learns very quickly.
with a success rate of more than 80% after 30k episodes compared to the experiment without any imitation.

Since Imitation Learning seems to work very well, we also decided to add it to the 2\textsuperscript{nd} experiment using the ball’s \((x, y)\) pixel coordinates. We can clearly understand with Fig. 13 the impact of imitation learning and more precisely the importance of it at the start of the training since it accelerates the training for the first episodes.

Finally, here is our final benchmark of the first 2 experiments using vision based methods, illustrated by Fig. 14.

![Fig. 12](image12.png)  
Fig. 12. In orange is the success rate by episodes for the training without imitation learning and in blue is the same training with imitation learning.

![Fig. 13](image13.png)  
Fig. 13. In orange is the success rate by episodes for the training without imitation learning and in blue is the same training with imitation learning.

The 2 most sample efficient are the one using the pixel coordinates of the ball and the one using a simple convolutional neural network. Both are using imitation learning to accelerate the training process.

Table 1 offers a wider view of all the performances of all the experiments studied during our research, with the spatial coordinates and with vision only.

![Fig. 14](image14.png)  
Fig. 14. In orange is the success rate by episodes for the training using the bounding box method, in red and blue is the training using the pixel coordinates of the ball respectively without and with imitation and the green plot is the training using the neural network.

Our 4\textsuperscript{th} and final experiment was to try the contrastive learning method which reduces the input space and facilitates the upstream vision part since it generalize this method to all use case, not only reach or grabbing objects but also opening doors... We tried with and without imitation learning. The experiment without imitation learning did not work and the results are not worth analyzing. Nevertheless, the one with imitation learning showed impressive improvements compared to the others. In order to illustrates the results of the CURL experiment, we only plotted it with experiment 3. Fig. 15 illustrates that CURL achieves better results with less training episodes. This method is clearly the best method of the whole benchmark.

![Fig. 15](image15.png)  
Fig. 15. In orange is the results of experiment n°3, in blue is the success rate by episodes for the training using CURL as the encoder.
| Overview of successes rates for several configurations |
|-----------------------------------------------------|
| Configuration \ Number of episodes | 10k | 20k | 30k | 10k | 20k | 30k |
| No vision | Cheating ball coordinates (Sparse reward) | 3.5% | 5.2% | 6.6% | 98.2% | 99.9% | 99.9% |
| | Cheating ball coordinates (Continuous reward) | 98.2% | 99.9% | 99.9% | 31.3% | 42.7% | 55.4% |
| Vision | Distance between two boxes | 45.2% | 78.1% | 82% | 73.1% | 82.2% | 85.2% |
| | Only the ball’s box coordinates | 6.1% | 7.1% | 6.4% | 70.6% | 79.9% | 81.3% |
| | Raw image | 1.4% | 0.7% | 2.1% | 91.2% | 97.6% | 98.3% |
| | Raw image + CURL |

VI. CONCLUSION

In this work we proposed to establish a sample efficient pipeline to train a deep reinforcement agent in vision based robotics. The challenge was to find the most efficient way to train an agent based only on the camera input and the proprioception sensors of the robot. Our final pipeline can be easily deployed to real life robots and can be generalized to many different tasks. We hope that this enables future real life usage of RL robots.

VII. ACKNOWLEDGMENT

We would like to especially thank Jesús BUJALANCE MARTIN for his wise advice and his help during our 2 months of research. We also thank Fabien MOUTARDE and MINES PARIS for making this project possible and supervising it.

REFERENCES

[1] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control. arXiv:1511.03791 [cs], November 2015. arXiv: 1511.03791.
[2] Sebastian B. Thrun. Efficient exploration in reinforcement learning. Technical report, 1992.
[3] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. Foundations and Trends® in Machine Learning, 11(3-4):219–354, 2018.
[4] Konstantia Xenou, Georgios Chalkiadakis, and Stergos Afantitos. Deep Reinforcement Learning in Strategic Board Game Environments. In Marija Slavokvíc, editor, Multi-Agent Systems, volume 11450, pages 233–248. Springer International Publishing, Cham, 2019. Series Title: Lecture Notes in Computer Science.
[5] Kapil Katyal, I-Jeng Wang, and Philippe Burlina. Leveraging deep reinforcement learning for reaching robotic tasks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, July 2017.
[6] Steven James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. Rbench: The robot learning benchmarking environment. IEEE Robotics and Automation Letters, 5(2):3019–3026, 2020.
[7] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving, 2017.
[8] Noam Brown and Tuomas Sandholm. Safe and nested subgame solving for imperfect-information games, 2017.
[9] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games, 2019.
[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv:1801.01290 [cs, stat], August 2018. arXiv: 1801.01290.
[11] Andrew S. Morgan, Daljeet Nandha, Georgia Chalvatzaki, Carlo D’Eramo, Aaron M. Dollar, and Jan Peters. Model predictive actor-critic: Accelerating robot skill acquisition with deep reinforcement learning, 2021.
[12] Thomas Degris, Martha White, and Richard S. Sutton. Off-policy actor-critic, 2013.
[13] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. page 23.
[14] Albert Zhan, Philip Zhao, Lerrel Pinto, Pieter Abbeel, and Michael Laskin. A Framework for Efficient Robotic Manipulation. arXiv:2012.07975 [cs], December 2020. arXiv: 2012.07975.
[15] Manuel Lopes, Francisco S. Melo, and Luis Montesano. Affordance-based imitation learning in robots. In 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1015–1021, 2007.
[16] Allen Z. Ren, Sushant Veer, and Anirudha Majumdar. Generalization guarantees for imitation learning, 2020.
[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.
[18] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016.
[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. arXiv:1512.02325 [cs], 9905:21–37, 2016. arXiv: 1512.02325.
[20] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalchenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.
[21] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.
[22] Sheikh Muhammad Saiful Islam, Md Mahedi Hasan, and Sohaib Abdullah. Deep learning based early detection and grading of diabetic retinopathy using retinal fundus images, 2018.
[23] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners, 2020.
[24] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019.
[25] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In Proc. of The International Conference on Intelligent Robots and Systems (IROS), 2013.
[26] Stephen James, Marc Freese, and Andrew J. Davison. Pyrep: Bringing v-rep to deep robot learning, 2019.
[27] Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. Optlayer - practical constrained optimization for deep reinforcement learning in the real world. CoRR, abs/1709.07643, 2017.

[28] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps, 2014.

[29] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. arXiv:1802.09477 [cs, stat], October 2018. arXiv: 1802.09477.

[30] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[31] Songtao Liu, Di Huang, and Yunhong Wang. Receptive field block net for accurate and fast object detection. In Proceedings of the European Conference on Computer Vision (ECCV), September 2018.

[32] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning, 2020.