Teaching Real Robots: The Right Way to Set up Learning-Based Systems for Your Robot

Ali Younes[1] and Arkady Yuschenko[2(✉)]

[1] Technische Universität Darmstadt, Darmstadt, Germany
ali@robot-learning.de
[2] Bauman Moscow State Technical University, Moscow, Russia

Abstract. The typical robotics system consists of perception, planning, and control modules. Each module is built upon information about its components, where modeling each part of the system plays an essential role in the design process. In practice, working with the non-linear models in robotics systems involves a lot of approximations which hinders reaching the optimal behavior for the goal task. Alongside the difficulties in redeploying the system to solve other similar tasks. Learning-based methods provide a promising approach for robotic systems. In the last decade, the interest in incorporating machine learning into robotics systems has been evolving rapidly. The benefit of using learning is the possibility to design systems that are independent of the dynamical model of the robot, with the flexibility to adopt new tasks and learn to excel in performance over time. The theory behind designing a learning-based system is still under development, ranges from end-to-end systems to hybrid systems that use inaccurate approximate models. In this paper, we are proposing the results of our research in learning-based systems, presenting our view for the right way to set up learning systems for robotics. The results are a whole learning-based framework for robotics applications, works efficiently (1 h of training – 10 min robot movement) with minimum human intervention (user has to provide video demonstrations only).

Keywords: Robotics · Reinforcement learning · Self-supervised learning · Contrastive learning

1 Introduction

The ingredients to design robotics systems start from understanding the needed components to be included in it. The standard structure of a robotic system consists of perception, planning, and control modules. The choice of each one of these modules depends on the application, the desired performance, and some design criteria.

A perception system is responsible for interpreting the data from the sensors to compatible state signals. The type of the output of the perception system depends on the set value's type in a simple system (e.g. voltage and current). In a more advanced system, the perception system (state estimation block) has to infer informative states from the sensor data e.g. localization of the robot [1], or the position and velocity of

A. Yuschenko (Ed.): MPoR 2020, CCIS 1426, pp. 1–13, 2021.
https://doi.org/10.1007/978-3-030-88458-1_6

objects. Designing a perception system is a challenging task [2], in robots lots of research done with states given by positioning systems. Recently, the interest shifted to extract informative states from more accessible sensors (systems), like images from cameras, depth information from laser sensors, and torque from motors [3].

We argue that using machine learning to learn representation is the future for perception modules in robotics systems. The positional and movement information of the object doesn't describe the correct step of the progress of the task, a higher latent representation could be more beneficial [4]. Using self-supervised learning to learn an embedding representation [5] for each set of sensor data in the latent space helps in forming a suitable perception system for each task.

The task of planning is the process of building a trajectory for the robot to follow. Giving the fact that the robot is moving from a state to state (in a state-space), possible trajectories are all combinations of actions that lead to getting the robot moved from an initial state to the desired goal state. The most common uses of planning [6] are planning over grids; where each node of the grid corresponds to the coordinates of a point in a map. Another common type is planning over graphs, where we represent possible states of the robot as nodes of the graph, and the edges are the possible transition. The goal of the planning over grids is to reach the target with minimum transitions, the same for unweighted graphs. For weighted graphs, we have to minimize the sum of the weights over the path, from the source state to the goal.

We are interested in planning in the Markov decision process (MDP) [7]. The environment can be represented as a Markov decision process, by defining the state and action states (possible states and actions), a transition model (the conditional probability of going from a state to another by executing an action), and a reward\cost function (how likely this transition will get us closer to the goal state). The goal of the planning in the Markov decision process is to maximize the reward, which likely leads to find the best trajectory from the current state to the goal. Planning is used in reinforcement learning [8] explicitly when we have the transition model of the environment (in model-based reinforcement learning [9, 21]). In model-free reinforcement learning [10], trajectory optimization is used instead of direct planning.

Model predictive control [11] proposes planning over a finite horizon instead of planning to the goal, while it couldn't guarantee the best trajectory from the source to the target states, but it provides a practical online planning algorithm to work with physical systems. Another important property of the model predictive control is natively considering the constraints (safety or desired conditions), as it formed based on the optimal control paradigm. The result of planning is a sequence of actions to follow, this sequence represents a trajectory that the robot has to follow.

The control part of the robot system is the process of interpreting a trajectory of action to actuators' commands. It depends on the type of states and actions in the planned trajectory, i.e. for rotation, velocity, or voltage state of the motors we can use simple low-level controllers for the motors (e.g. PID controllers [12]). If the states are the position of the robot, and the actions are movements of the center of the mass of the robot; then we have to use the model of the robot to find the corresponding motor commands. In the case of serial robots (e.g. manipulators) [13], the inverse kinematic and the dynamic model of the robot are used, which suffer from approximations and linearization when forming controllers, as they are nonlinear models.

Having a general view over the robotics systems, we are proposing a learning-based approach for controlling robots. We aim to have our method to (1) be independent of the robot's dynamical model (applicable to teach tasks to any robot) (2) achieve tasks effectively (works with small datasets), (3) learn fast (to make the robot able to adapt to unforeseen cases, with low computational cost), (4) work with real robot directly (little time of interaction on the real robot) and (5) need minimum human intervention to set-up a new task.

The structure of the paper starts by presenting the proposed learning-based system, starting by introducing its architecture, the Markov decision process representation, perception system, and the reinforcement learning system. We will discuss our experiments and results in each system, to conclude with high-level results and the possible future direction building on our work.

2 The Proposed Learning-Based Robotics System

2.1 The General Architecture

The proposed learning-based robotics system consists of the following.

1. The environment (green block in Fig. 1):
   It is the part that is responsible to interact with the physical system or the system in the simulation. For robotics systems, the environment contains the robot and the sensors, it receives an action from the control system, executes it on the control object (robot in our case), then collects the data from the sensors, and sends it to the perception system to form states.
2. The perception system (red block in Fig. 1):
   The perception system receives sensor data from the environment and uses it to form informative states. In our proposed system, we use the output of the perception system to compute the reward, by computing the distance to a goal image stored in the environment (white circle Fig. 1).
3. The reinforcement learning system (blue block in Fig. 1):
   The reinforcement learning system performs the planning and control part in our system, where instead of having the output of the reinforcement learning system in the end-effector's coordinate space, the output in our case is in the joints' space (direct commands to the motors). This idea helps in making our system independent of the robot dynamical model and can be used to any robotic arm with minimum effort. The goal of the reinforcement learning system is to maximize the expected long-term reward.

2.2 The Proposed System as a Markov Decision Process

A Markov Decision Process (MDP) is defined for the proposed system. This MDP describes the transitions between states. To make a connection between the architecture of our system, and the MDP in Fig. 2. We will define the following terminology:

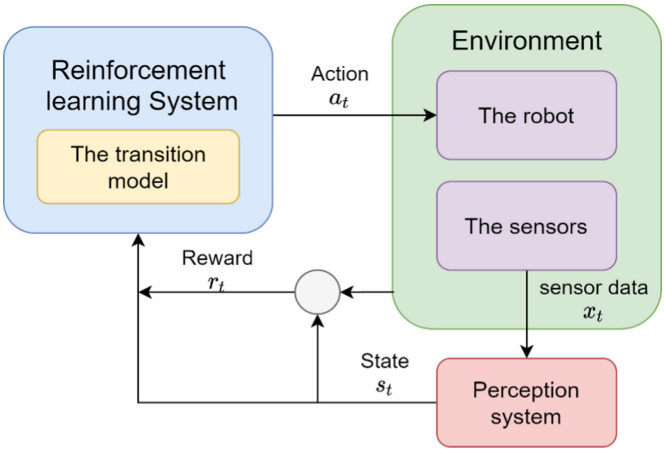

Fig. 1. The general architecture of the proposed learning-based robotics system (Color figure online)

1. The policy (of the Reinforcement learning systems in Fig. 1).
   The policy $\pi_\theta(s_t)$ is a function of the state, to be optimized in the reinforcement learning procedure to give the action corresponds with the highest predicted long-term return.
2. Execution and data acquisition (Environment block in Fig. 1).
   Given an action a_t from the policy, the action to be transformed into primitive commands, by executing these commands the state of the environment will change, sensors will be used to collect data x_t (data acquisition).
3. The embedding model (the model of the perception system in Fig. 1).
   By feeding the data x_t to the embedding model, we get the state of the system $s_t = y_e(x_t)$, the embedding model should be trained beforehand, and states will be used to compute rewards (or state_cost).
4. The transition model (to be learned in the reinforcement learning system).
   Given the current state of the system s_t and the chosen action a_t, the transition model predicts the probability of the next state $p(s_{t+1}|s_t, a_t)$, the transition model is learned over the training process (using the transition dataset $D = \{s_t, a_t, s_{t+1}\}$).

2.3 The Environment

The environment contains the robot and the sensors, it is defined following the rules of OpenAI gym [14] environments. The gym environment is defined as a class with the main methods; initialization, reset, and step methods (Fig. 3).

The initialization method sets up the components of the environment, by checking the connection to the simulator or the real robot, connection to the sensors, define the initial state of task-related objects, and move the robot to its initial position, it may also ask the user to provide some hyperparameters or to set the goal state. In our system, we

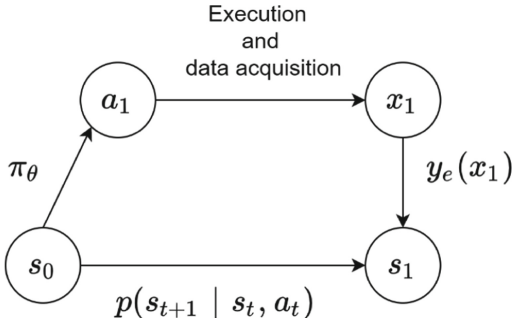

Fig. 2. The Markov decision process of the proposed system

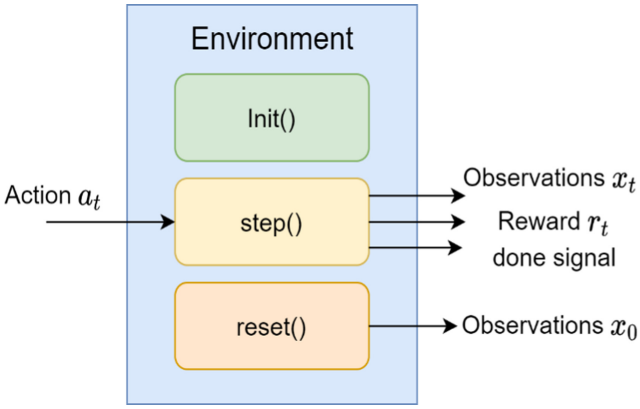

Fig. 3. The environment class

propose to load a pre-trained embedding model to be used to get states out of sensors' data.

As noted earlier, the environment receives an action a_t, the execution of this action is done in the step method. Where first the action is checked for its validity (e.g. doesn't exceed the limits of the motors), then given the current joints' angles, the action vector is scaled to an angle offset change, and the resulted angles are sent to the motors directly while moving the robot we have to check the lockdown cases (we have used a limit on the current used by each motor). The control of the robot done using RestAPI request, we prefer using low-level API control as it gives better flexibility than high-level APIs. When the robot finishes its movement, the data acquisition step is performed by collecting the output of the sensors, the raw data is called observations x_t, which will be forwarded to the perception system. Reward value is computed using the observation of using the states after using the perception system. In our system, we studied the use of low-cost RGB cameras and sensors equipped with the robot (encoders, current sensors). A done signal is returned whether the task was achieved or the robot fell into a lockdown case.

When receiving a done signal, the reset function returns the environment to its initial state, and it may ask the user to change the position of some objects, or do this randomly if possible (depends on the task properties).

Technically, we need a helper class to control the robot using RestAPI requests, also when working with cameras, we need a buffer-less video capture class (instead of an ordinary video capture class from OpenCV).

In our proposed learning-based system, the user doesn't have to care about the environment or change anything to train the robot on a new task, he just wants to collect some video demonstrations to train the perception system and provide one of them to the learning system, this one will be used to define the goal state, the reward will be the distance rot he goal state in the latent space.

2.4 The Perception System

The Model

Consider we have an input image $x \in \mathbb{R}^{W \times H \times 3}$. The output of the model is an embedding of the image in a latent space $y_e \in \mathbb{R}^E$. W, H are the width and the height of the image, E is the size of the embedding vector. The base CNN network (blue block in Fig. 4) is taken from the inception model [15] pre-trained on ImageNet [16], the parameters of this part are frozen and won't be re-trained. We add two additional convolutional layers, followed by a spatial softmax layer. A fully connected layer is added to get the embedding of the input image $y_e = f_\theta(x)$. The parameter to train embedding model θ is the training parameters of the model (weights and biases of the 2CNN layers and the feed-forward layer – green blocks in Fig. 4).

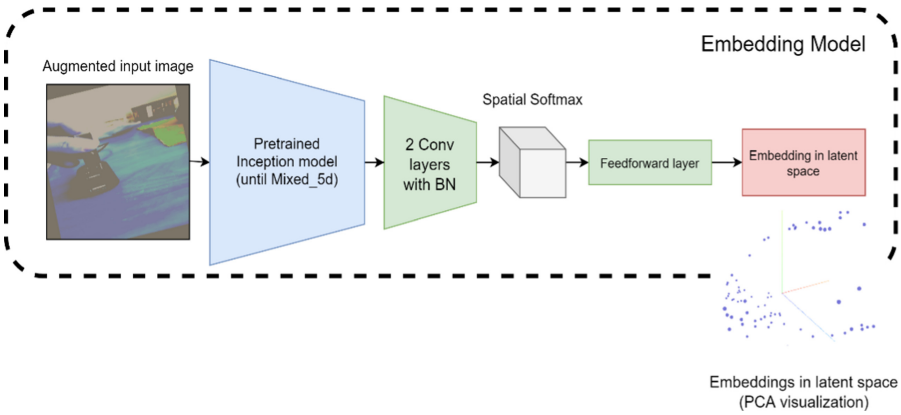

Fig. 4. The architecture of the embedding model (Color figure online)

The Dataset Formulation

The dataset is formed by extracting images from video files and form triplets for learning. In contrastive learning, the process starts by sampling an image from a video, this image

is called anchor, a positive range is defined as all the images that far from the anchor by less than a positive margin, the negative margin is the rest of the images. One positive and one negative image is sampled from the positive and negative ranges respectively. The triplet consists of an anchor x_i^a, a positive image x_i^p, and a negative image x_i^n.

All images normalized and augmented before feeding them to the network. The normalization is needed because the pre-trained part trained on normalized images. The augmentation was done by using a color jitter (random coloring) and random rotation. The augmentation is useful for transfer learning, and to avoid overfitting to the training dataset.

Time Contrastive Training
Given a triplet of images; an anchor x_i^a, a positive image x_i^p, and a negative image x_i^n, the time-contrastive loss is [17] the loss tries to ensure that the anchor and the positive images are closer to each other in the latent space than the negative image. i.e. the aim is to learn an embedding such that:

$$\left\|y_e\left(x_i^a\right) - y_e\left(x_i^p\right)\right\|_2^2 + \alpha \leq \left\|y_e\left(x_i^a\right) - y_e\left(x_i^n\right)\right\|_2^2$$

The time contrastive loss is defined as:

$$L = \min\left(0, \alpha + \left\|y_e\left(x_i^a\right) - y_e\left(x_i^p\right)\right\|_2^2 - \left\|y_e\left(x_i^a\right) - y_e\left(x_i^n\right)\right\|_2^2\right)$$

Training of the embedding model starts by sampling a triplet of augmented images from the dataset, feeding them to the model, and use the output to compute the sequential time contrastive loss, the loss then is used to update the parameters of the model. The margins are updated every while.

Sample Results After Training
The embedding model was trained for 100 epochs (10000 triplets), Alongside the visualization of the latent space (embeddings) in Fig. 4, we have plotted a reward function depends on the distance to the target image in latent space:

$$r(x_i) = -\left\|y_e(x_i) - y_e\left(x_{target}\right)\right\|_2^2$$

For each video file, the reward function should be monotonically increasing to zero, smoother function means better performance. To judge the benefit of using time contrastive learning, we have plotted the latent space and reward before (Fig. 5) and after training (Fig. 6). The reward/cost function corresponds to each latent space. Before training, random distribution led to a bad reward function. After training, better monotonically increasing function, could be used in RL and control.

2.5 Reinforcement Learning System

We propose using a model-based reinforcement learning system (Fig. 7), with some key features to fulfill our requirements. Reinforcement learning will (1) interact with the environment (which is formed as a Markov Decision Process). (2) Learn the transition

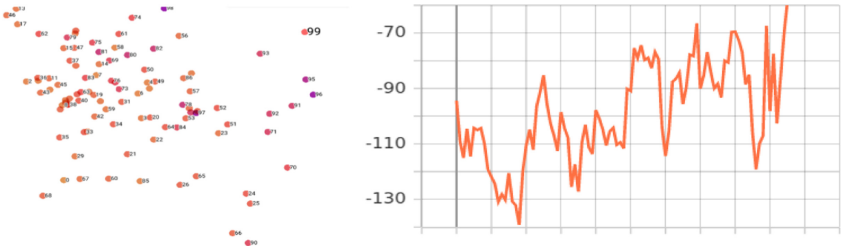

Fig. 5. Latent space and reward function before the training

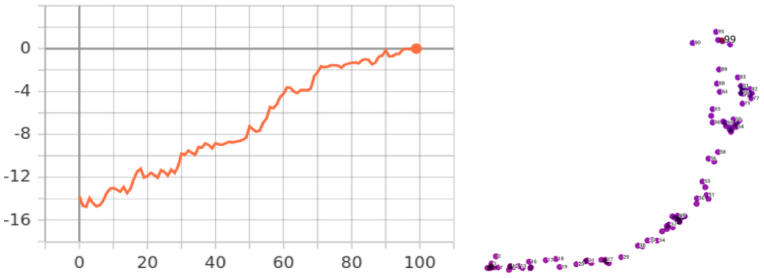

Fig. 6. Latent space and reward function after the training

model, which has to handle uncertainty, we used Ensemble neural network to learn the model. (3) Plan using the learned model, we are following the model predictive control in planning. (4) In the planning, we have to use an optimizer, that fits our system, we used Cross-Entropy Maximization optimizer.

The Transition Model – Ensemble Neural Network

The goal of learning the transition model of the environment is to predict the dynamics of the task. Learning the model is an essential part of the model-based reinforcement learning as it will be used in the planning.

The model should work efficiently in the cases of small and large datasets; i.e. learns good dynamics at the early stages of learning, and enhance the performance further with time. This is done by choosing a model that can handle uncertainty (to learn from small datasets), and deep enough to have the capacity to infer from large datasets.

We have chosen Ensemble neural networks [18], as it can handle the uncertainty by using the prediction from several deep neural networks, and have the needed capacity because of the deep structure in its components. Alongside the low computational cost when parallelized on the graphical processing units.

The Optimizer – Cross-Entropy Maximizer

The cross-entropy maximization (CEM) [19] is a gradient-free stochastic optimization algorithm, based on Monte-Carlo methods. Given a state cost function (negative reward from Sect. 2.3), and a learned transition model, CEM chooses the optimal series of actions (with length equal to the planning horizon) corresponds to the biggest predicted cost.

In our system, we use a Random-shooting method in CEM, this is done by sampling series of actions from a predefined distribution, evaluating them, and separating ones with the highest reward (we call them elites):

$$A_i = \left\{a_0^i, a_1^i, a_2^i, \ldots a_{H-1}^i\right\}; \quad a_t^i \sim N\left(\mu_t^m, \Sigma_t^m\right)$$

$$A_{elites} = sort(A_i)[:J]$$

The mean and variance of the sampling distribution updated according to the mean and variance of the elite sets.

$$\mu_t^{m+1} = \alpha * mean(A_{elites}) + (1-\alpha) * \mu_t^m$$

$$\Sigma_t^{m+1} = \alpha * var(A_{elites}) + (1-\alpha) * \Sigma_t^m$$

After m iterations, the optimal series of actions is the mean of the elites.

Planning – Model Predictive Control for the System
The planning is done by using the optimizer over the learned model. In our case we have to use the ensemble neural network to find the predicted states after executing each action because the output of the ENN is a distribution, we used particles of the initial state, and propagate them using the ENN model:

$$s_{t+1}^p \sim f_\theta\left(s_t^p, a_t\right); \quad s_{t=0}^p = s_0$$

Then we use state cost function to find the cost of the series and continue the optimization.

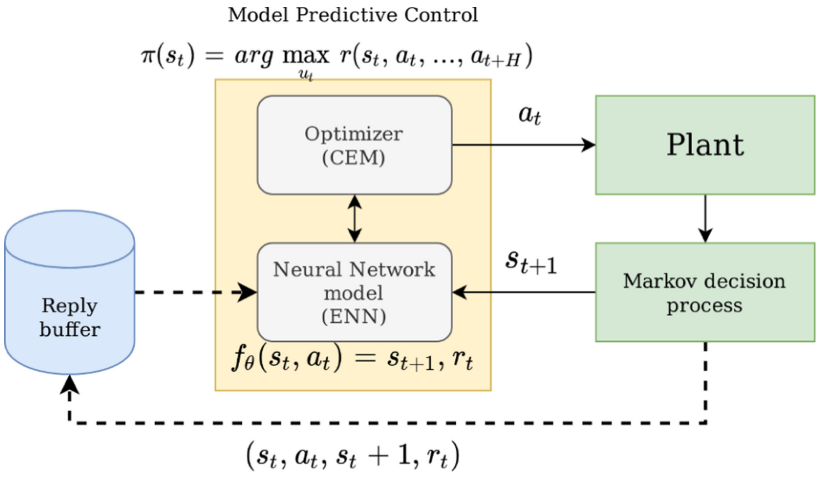

Fig. 7. Model-based reinforcement learning with ensemble neural networks

Sample Results for Experiments

To evaluate the Reinforcement learning system, we build an environment in simulation (VREP simulator [20]), contains a KUKA manipulator, a USB flash desk was attached to its end effector, and a USB socket placed on the ground in its workspace.

The state consists of the position of the object (flash desk and USB socket) and joints' angles. The control in the joint's space (actions are commands to motors). The transition model in Fig. 8.

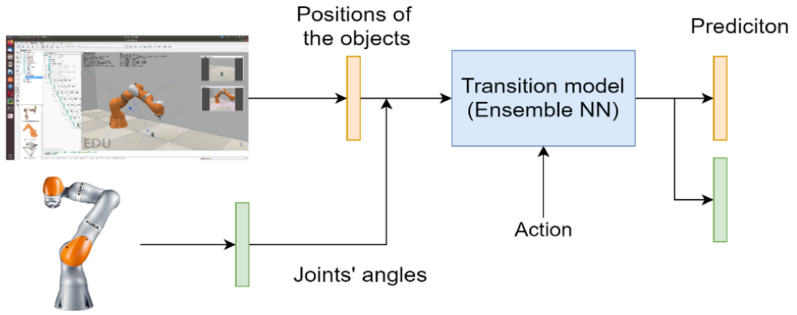

Fig. 8. Transition model for the experiment in simulation

In the simulation, we have tested the reinforcement learning system to reach a specified point in its workspace, similar to the step response in evaluating control systems.

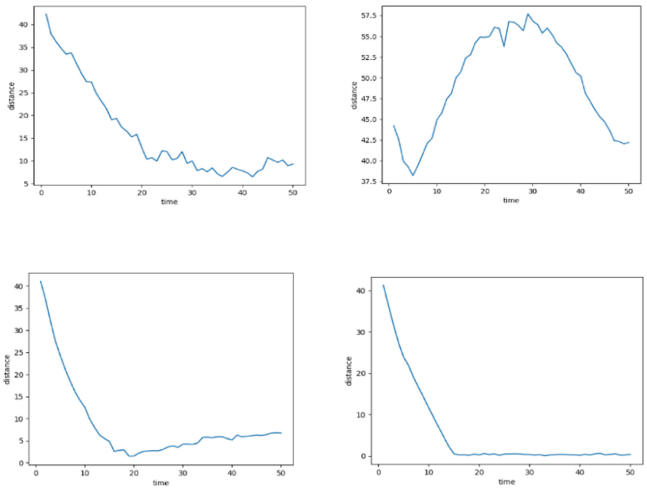

Fig. 9. The results for reaching a point in the workspace (step response analysis), from upper-left to lower right: iterations 1, 5, 8, 50

We have plotted the distance between the end effector over the training process (Fig. 9), we can notice that the system started to learn a correction from early iteration

(iteration 8), and by further training perfected its response (iteration 50), without any overshoot and holding the goal after reaching it. These results match our goal of learning efficiently from small datasets and improve the performance when larger datasets are available.

To test the whole system, we have to use the learned embedding model (trained according to Sect. 2.4), to extract image embeddings from RGB images, and replace the states in the last experiment (Fig. 10).

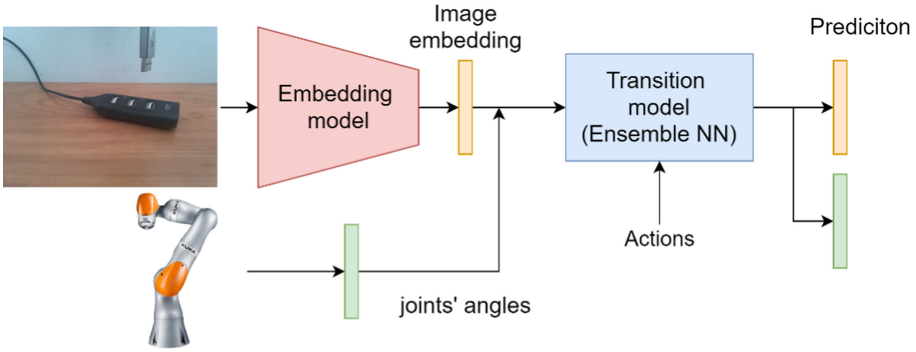

Fig. 10. The transition model of an experiment with the whole system

The reward (and state cost) depends on the distance in the latent space between the embedding of the current image and the target image (see Sect. 2.4 for details). We can judge the performance of the system on this task by drawing the reward over the training time (Fig. 11).

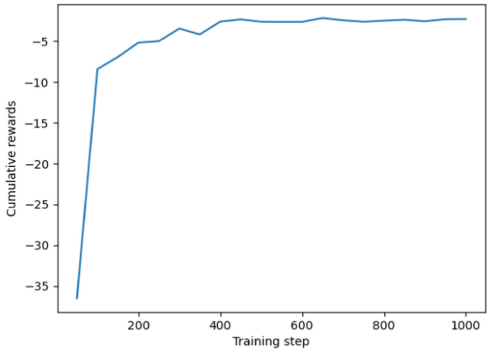

Fig. 11. The reward function over time, bigger is better.

We can notice the system can optimize its performance over the training time, and learns a good policy after 800 timesteps (30 min on the robot).

3 Conclusion

We have proposed a learning-based framework for learning robotics manipulation tasks. The framework uses self-supervised representation learning for perception, and reinforcement learning for planning and control. The framework doesn't depend on the dynamical model of the robot, data-efficient, learns tasks fatly, can be run the training process on the real robot directly, and works with minimum human intervention.

Perception systems have to be pre-trained before run the training process for the reinforcement part. The user has just to provide the system with some videos demonstrating the task, and then the learning process runs in a self-supervised way. The learning process takes around 30 min on a PC with single GPU (low computational cost, and short training time), the result is an embedding model to infer a representation in the latent space for an image on its input.

After that, we should define an environment as a Markov decision process, then we can run the reinforcement learning part. By using the trained perception system, the user can use our predefined environment, by just providing a goal image (the user doesn't have to change anything in the environment).

The reinforcement learning system will be trained lastly, in a fully automated way, its goal is to minimize the Euclidian distance between the current state and the goal image. The reinforcement learning system uses a model-based approach, with a probabilistic model, and the training time in our experiment between 30–45 min.

The whole framework provides a promising way to run robotics experiments, the user have just to collect some videos (possibly with a smartphone) demonstrating the task and run the learning process for the perception system, then provides a goal image and run the reinforcement learning part, without the need to any coding or engineering experience.

Open-source code is made available for reproducibility and validation:
https://github.com/Alonso94/Self-supervised-RL.

Future direction for our research is to make the extend the framework for lifelong learning, where the robot learns skills instead of learning just tasks, this needs an extension for our perception system, and an algorithm for planning over skills, not just states, but we believe this would be essential to have fully intelligent robotic systems.

References

1. Mihajlov, B.B., Nazarova, A.V., Yushchenko, A.S.: Avtonomnye mobil'-nye roboty-navigaciya i upravlenie. Izvestiya YUzhnogo federal'nogo universiteta. Tekhnicheskie nauki **2**(175) (2016)
2. Bagutdinov, R.A.: Princip razrabotki algoritmicheskogo obespecheniya si-stemy tekhnicheskogo zreniya robotov. Naukoemkie tekhnologii v kosmicheskih issledovaniyah Zemli **9**(5) (2017)
3. Lee, M.A., et al.: Making sense of vision and touch: self-supervised learning of multimodal representations for contact-rich tasks. :In 2019 International Conference on Robotics and Automation (ICRA), pp. 8943–8950. IEEE (2019, May)
4. Zhu, H., et al.: The ingredients of real-world robotic reinforcement learning. arXiv preprint arXiv:2004.12570 (2020)

5. Sermanet, P., et al.: Time-contrastive networks: self-supervised learning from video. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 1134–1141. IEEE (2018, May)
6. Носков, В.П., Рубцов, И.В.: Ключевые вопросы создания интеллектуальных мобильных роботов. Инженерный журнал: наука и инновации **3**(15) (2013)
7. Bellman, R.: A Markovian decision process. Indiana Univ. Math. J. **6**(4), 679–684 (1957). https://doi.org/10.1512/iumj.1957.6.56038
8. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, vol. 1. MIT Press, Cambridge (1998)
9. Chatzilygeroudis, K., Vassiliades, V., Stulp, F., Calinon, S., Mouret, J.B.: A survey on policy search algorithms for learning robot controllers in a handful of trials. arXiv preprint arXiv:1807.02303 (2018)
10. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning, pp. 1889–1897 (2015, June)
11. Camacho, E.F., Alba, C.B.: Model Predictive Control. Springer Science & Business Media (2013)
12. Рогач, В.Я: Расчет настройки реальных ПИД-регуляторов. Теплоэнергетика **10**, 31–35 (1993)
13. Jazar, R.N.: Theory of Applied Robotics: Kinematics, Dynamics, and Control. Springer Science & Business Media (2010)
14. Brockman, G., et al.: Openai gym. arXiv Preprint arXiv:1606.01540 (2016)
15. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet, and the impact of residual connections on learning. In: the Thirty-First AAAI Conference on Artificial Intelligence (2017, February)
16. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: a large-scale hierarchical image database. In: 2009 IEEE Conference On Computer Vision and Pattern Recognition, pp. 248–255. IEEE (2009, June)
17. Sohn, K.: Improved deep metric learning with a multi-class n-pair loss objective. Adv. Neural Inform. Process. Syst. 1857–1865 (2016)
18. Pearce, T., Zaki, M., Brintrup, A., Neel, A.: Uncertainty in neural networks: Bayesian ensembling. arXiv Preprint arXiv:1810.05546 (2018)
19. De Boer, P.T., Kroese, D.P., Mannor, S., Rubinstein, R.Y.: A tutorial on the cross-entropy method. Ann. Oper. Res. **134**(1), 19–67 (2005)
20. Rohmer, E., Singh, S.P., Freese, M.: V-REP: a versatile and scalable robot simulation framework. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1321–1326. IEEE (2013, November)
21. Chua, K., Calandra, R., McAllister, R., Levine, S.: Deep reinforcement learning in a handful of trials using probabilistic dynamics models. Adv. Neural Inform. Process. Syst. 4754–4765 (2018)