Trajectory Planning of a Cable-Based Parallel Robot using Reinforcement Learning and Soft Actor-Critic

DINH-SON VU, AHMAD ALSMADI
Mechanical Engineering Department
American University of the Middle East
Block 6, Building 1, Egaila, Kuwait
KUWAIT

Abstract: - Industry 4.0 introduces the use of modular stations and better communication between agents to improve manufacturing efficiency and to lower the downtime between the customer and its final product. Among novel mechanisms that have a high potential in this new industrial paradigm are cable-suspended parallel robot (CSPR): their payload-to-mass ratio is high compared to their serial robot counterpart and their setup is quick compared to other types of parallel robots such as Gantry system, popular in the automotive industry but difficult to set up and to adapt while the production line changes. A CSPR can cover the workspace of a manufacturing hall and providing assistance to operators before they arrive at their workstation. One challenge is to generate the desired trajectories, so that the CSPR could move to the desired area. Reinforcement Learning (RL) is a branch of Artificial Intelligence where the agent interacts with an environment to maximize a reward function. This paper proposes the use of a RL algorithm called Soft Actor-Critic (SAC) to train a two degrees-of-freedom (DOFs) CSPR to perform pick-and-place trajectory. Even though the pick-and-place trajectory based on artificial intelligence has been an active research with serial robots, this technique has yet to be applied to

parallel robots.

Key-Words: - Robotics, Parallel Mechanism, Reinforcement Learning
Received: April 23, 2020. Revised: September 10, 2020. Accepted: September 25, 2020.
Published: October 9, 2020.

1 Introduction

Cable Suspended Parallel Robot (CSPR) is a special type of parallel manipulators, in which cables are used instead of rigid links and depend on gravity to keep the cables in tension [1],[2]. CSPR has a larger workspace, are lighter and are usually cheaper than standard parallel mechanisms and thus have valuable benefits in shipping port, factory application [3], and space stations [4].

Despite their numerous benefits, CSPR are mechanisms where the cables can only pull the end-effector and are more flexible than manipulators with rigid links, which make them some sensitive to external disturbances such as wind or vibration. Thus their dynamics must be analysed carefully. Fahham et al. [5] investigated the optimal trajectory time for a redundant planar CSPR. They developed a hybrid genetic algorithm and bang-bang control approach to optimize the path that minimizes the traveling time from the initial state to the final state. Gosselin et al. [2] presented a two-DOFs CSPR. Their approach ensures that the cables are under tension for the whole trajectory with the use of parametric Cartesian trajectories in the dynamic constraints. Zi et al. [6] addressed the kinematics and graphical representation of the singularity configuration for translational three-DOFs CSPR, with the derivation of the inverse kinematics based on closed loop vector conditions and geometric methodology.

Because of the non-linearity of dynamic systems, deriving the dynamic model may be tedious, especially if the mechanisms are based on several DOFs. On the other hand, machine learning is a field of computer science where an agent learns pattern and generalization from labeled training data. In particular, deep learning and reinforcement learning have wide application in several fields, including robotics. Application such as facial recognition [7], detection of energy streams [8], modelling of surface roughness [9], and Brain MR Image Classification [10] uses deep neural network that learn a model to classify future data.

Reinforcement learning (RL) is a different paradigm in machine learning, in which an agent is trained with several interaction with its environment. In RL, the agent does not need to have prior knowledge of the mathematical model, and thus avoiding the modeling and parameter tuning process that relies on expert experience [11]. Many researchers carried out experiments to investigate the integration between reinforcement learning and robotics. In [12], deep RL is applied on cable-driven suspended robots to investigate the optimal tension distribution in it. Their simulation results show that the learning strategy was robust to certain model uncertainties. A comparative study between end-to-end deep RL and hybrid deep RL, that takes into account the kinematics of the mechanism,has been performed in [13]. In [14] they proposed a field application of RL for solving the action selection problem of a cable tracking task. To demonstrate its feasibility, they conducted real-time experiments on an underwater robot. Their results show good performance relatively rapid convergence. This integration has also been used in soft arm applications to realize the position control task [15]. Hierarchical RL for Semi-Autonomous Rescue Robots [16] has been applied to explore disaster scenes and find victims. Results showed that with the proposed Hierarchical RL, the robot could explore and react to the victim scene accurately. This paper proposed the use of RL to control a two-DOFs CSPR with RL algorithm called Soft Actor-Critic (SAC) [17]. Even though the RL has been used in several robotic applications, this paper provides the steps used in the modeling of such RL problem, from the modelling of the environment, the description of the agent, and the learning process of the agent to perform pick-and-place trajectories.

This paper is structured as followed: the dynamic model of the system and the policy gradient algorithm is presented in section II. The parameters of the simulation performed to assess the pick-and-place trajectory are described as well. Section III presents the results obtained for the Pick-and-Place trajectory and compares different reward function to achieve the desired goal. The discussion in section IV comments on the results and the conclusion introduces the future works.

2 Materials and Methods

This section presents the reinforcement learning problem, which consists of the agent, the environment, and its reward function. It also presents the parameters used, so that the agent learns to perform pick-and-place motion from interaction with the environment.

The environment corresponds to the forward dynamics of the mechanical system, which is a two-DOFs point-mass CSPR. The agent is based on a policy gradient algorithm, called Soft Actor-Critic (SAC), and its aim is to find the cable tension to move the end-effector to the desired position with the required velocity and acceleration. The reward function informs the agent about its performance while interacting with the environment. Different reward functions are presented and compared in the simulation section.

2.1 Dynamics of the CSPR

An agent learns to perform pick-and-place trajectories with reinforcement learning by interacting several times with its environment and by maximizing a reward function. In this paper, the agent is acting on the cable tensions t_1 and t_2 of the two-DOFs cable mechanism. The environment corresponds to the forward dynamics of the two-DOFS CSPR, which consists in the calculation of the Cartesian position, velocity, and acceleration of the end-effector, namely $\mathbf{p}$, $\dot{\mathbf{p}}$, and $\ddot{\mathbf{p}}$ of the point mass m given the cable tension t_1 and t_2. Even though a two-DOFs mechanism is relatively simple, the development of larger scale robot with high DOFs can be performed from the work of this paper.

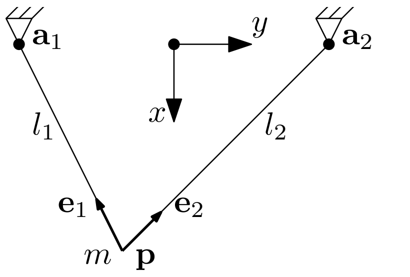


Figure 1: Two-DOFs point-mass CSPR.

The fixed attachment points $\mathbf{a}_i$ correspond to the position of the motors that are reeling the end-effector up and down, which position is given by $\mathbf{p} = [x \quad y]^T$, as shown in figure 1. The cable lengths l_1 and l_2 are the distance between the end-effector's position $\mathbf{p}$ and their respective attachment point $\mathbf{a}_1$ and $\mathbf{a}_2$. The inverse kinematics, which calculates the cable length given the Cartesian position of the end-effector, is given as:

$$l_i = \sqrt{(\mathbf{p} - \mathbf{a}_i)^T (\mathbf{p} - \mathbf{a}_i)} \quad \text{with: } i = 1, 2. \tag{1}$$

The unit vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ are the direction of the tension force acting on the point mass end-effector, which expression is given as:

$$\mathbf{e}_i = \frac{\mathbf{a}_i - \mathbf{p}}{l_i}. \tag{2}$$

The dynamics of the point mass end-effector can be obtained by applying Newton's second law, namely

$$t_1\mathbf{e}_1 + t_2\mathbf{e}_2 + m\mathbf{g} = m\ddot{\mathbf{p}} \tag{3}$$

where $\mathbf{g} = [g \quad 0]^T$, and g corresponds to the gravitational acceleration. One may obtained the forward dynamics by rearranging (3) to determine the acceleration of the end-effector given the cable tension, namely:

$$\ddot{\mathbf{p}} = \frac{1}{m}\mathbf{M}\mathbf{t} + \mathbf{g} \tag{4}$$

with:

$$\mathbf{M} = [\mathbf{e}_1 \quad \mathbf{e}_2], \quad \text{and} \quad \mathbf{t} = [t_1 \quad t_2]^T. \tag{5}$$

This derivation is similar to the one performed in [1]. The matrix **M** is the Jacobian matrix of the manipulator and is singular when one of the unit vector $\mathbf{e}_1$ or $\mathbf{e}_2$ is zero or when both unit vectors are aligned, which occurs when the end-effector is positioned between the two attachment points $\mathbf{a}_1$ and $\mathbf{a}_2$. In this paper, we will consider trajectories below the singularity lines. Dynamic trajectories outside the static workspace is an active field of research: the tension of the cable are kept under tension with the use of the inertial force generated by the mass of the end-effector, but the generation of such dynamic trajectories is challenging [18], [19]. Equation (4) corresponds to the forward dynamics of the two-DOFs CSPR and can be written in a discrete way as:

$$\ddot{\mathbf{p}}_k = f(\mathbf{t}_k, \mathbf{p}_k, \dot{\mathbf{p}}_k), \tag{6}$$

where k represents the current time step. The velocity and the position of the end-effector can be obtained with Euler's numerical integration, namely:

$$\dot{\mathbf{p}}_{k+1} = \dot{\mathbf{p}}_k + \ddot{\mathbf{p}}_k \Delta t, \tag{7}$$

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \dot{\mathbf{p}}_k \Delta t + \frac{1}{2}\ddot{\mathbf{p}}_k \Delta t^2 \tag{8}$$

where Δt represents the time step between two estimates of the Cartesian position. Euler's algorithm uses successive integration of the acceleration based on Taylor series and suffers from potential numerical divergence, especially if a simulation is run for a long period of time and if the time step Δt is relatively large.

Better integration calculation algorithm, such as Runge-Kutta, can be used to obtain the position and velocity from the Cartesian acceleration. For instance, Runge-Kutta algorithm of order 4 (RK4) is based on the weighted average of four acceleration estimation, but requires to run the forward dynamics and Euler integration—equations (6), (7), and (8)—four times each, which may be computationally expensive. The forward dynamics of the two-DOFs CSPR is not computationally expensive, so RK4 method is likely to be more accurate than the Euler's method for a given time step Δt, with little additional computational cost. However, more complicated mechanisms, such as six-DOFs serial robots, require the calculation of the inverse dynamics several times (seven times of a six-DOFs serial robot) to determine the dynamics components of the manipulator, and then to obtain an expression of the forward dynamics. This would greatly increase the computational cost of using RK4 for numerical integration. The use of a smaller time step Δt prevents the Euler's method from diverging, but also increase the number of interaction between the agent and the environment. A comparison between Euler's method and RK4 is provided in the discussion of this paper to assess the effect on the behaviour of the agent in terms of average reward.

2.2 Soft Actor-Critic Algorithm

The preceding subsection has presented the behaviour of the environment. This subsection describes the behaviour of the agent: given the observation and the reward from the environment, what should be the action of the agent. The observation is represented by the position, velocity, and acceleration $\mathbf{p}, \dot{\mathbf{p}}, \ddot{\mathbf{p}}$ of the end-effector calculated with the forward dynamics, and the action corresponds to the cable tension **t**. The actor can be seen as an operator pulling the cable to position the point-mass correctly.

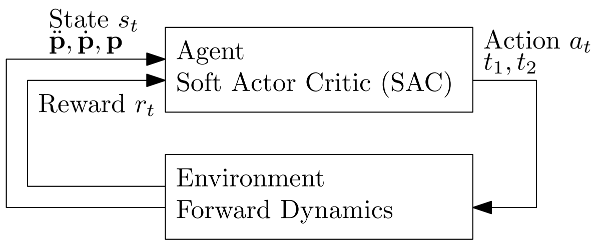


Figure 2: Reinforcement Learning Diagram. The agent outputs the cable tension, and the environment outputs the next state, namely position, velocity, and acceleration of the effector.

There are mainly two families of algorithms in reinforcement learning. One is based on the estimation of a state value V or state-action value Q that evaluates the quality of a state depending on the reward received. In the case of Pick-and-Place, being far from the target position would result in a low evaluation of this state, but being closer would lead to a higher state value. Deep Q-Learning [20] is an example of value-based RL agent that uses a neural network to estimate a value function, then a greedy policy would take the action that yields the highest reward.

The other family in RL agent is called policy gradient, which directly updates a parametric policy based on the reward received. The use of a value function is not mandatory, but is useful to guide the parameters of the policy to the optimization of the reward received during an episode. The term ”Actor-Critic” describes the relationship between the policy (Actor), which decides the action to take, and the value function (Critic), which evaluate the action taken.

The algorithm used in this paper is based on a policy gradient method called Soft Actor-Critic (SAC) [17], [21]. The ”Soft” of the SAC algorithm is a stochastic behaviour which converges to the maximization of the reward function while exploring the action space with the introduction of an entropy regularization in the cost function. This algorithm is well suited for problem with continuous action space. DDPG algorithm [22], [23] and TD3 [24] are other algorithms that function with continuous action space, but SAC uses interesting techniques, such as clipped double-Q learning that prevent overestimating the action value, and the entropy regularization, which is an elegant method to adjust the balance between exploration and exploitation compared to the added Gaussian noise used in DDPG and TD3. The key equations to understand SAC algorithm are now given, but the implementation details, such as the experience replay, the clipped double-Q learning, and the target policy, can be found in the original paper [17].

The action $\tilde{\mathbf{t}}_\theta$, which corresponds to a normalized tension cable between $[-1, 1]$, is selected from a policy π_θ, which is a squashed Gaussian distribution, namely:

$$\tilde{\mathbf{t}}_\theta \sim \pi_\theta = \tanh\left(X \sim \mathcal{N}\left(\mu_\theta(s), \sigma_\theta(s)\right)\right) \quad (9)$$

where $\mu_\theta(s)$ and $\sigma_\theta(s)$, respectively the mean and the standard deviation of the normal distribution $\mathcal{N}$, are the outputs of the actor neural network defined with the parameter θ. Using normalized tension has a tendency to reduce the variance of the cable tension. The conversion between normalized cable tension and cable tension used in the environment is performed as followed:

$$\mathbf{t} = t_{\max}\left(\tilde{\mathbf{t}}_\theta + 1\right), \quad (10)$$

where t_max is the maximum cable tension allowed. The modification of $\mu_\theta(s)$ and $\sigma_\theta(s)$ aims at maximizing the cost function J_π and the standard update rule using gradient ascent is given as:

$$\theta_{k+1} = \theta_k + \alpha_{lr}\nabla_\theta J_\pi \quad (11)$$

where α_{lr} is the learning rate, the indices k and $k+1$ correspond to the current and next values of the parameters θ of the neural network, and $\nabla_\theta J_\pi$ is the gradient of the cost function J_π to be maximized, namely the rewards obtained at each time step of an episode of length τ. The policy aims at maximizing the action value $Q_\pi(s,a)$, and the standard cost function J_π of a RL problem can be expressed as follows:

$$J_\pi = \underset{\tau\sim\pi}{\mathbb{E}}\left[R(\tau)|s,a\right] = Q_\pi(s,a), \quad (12)$$

where $\mathbb{E}[.]$ means the expected value, the underscript $\tau \sim \pi$ means following the policy π during the episode of length τ, $(R(\tau)|s,a)$ is the reward received at the end of the episode of length τ after following a sequence of state and action s, a. The novelty of the SAC algorithm is the introduction of a bonus reward proportional to the entropy of the policy, which has a tendency to increase the exploration and prevent early convergence to a local optimum. The SAC cost function is given as:

$$J_\pi = Q_\pi(s,a) - \alpha \log \pi\left(a|s\right) \quad (13)$$

with α, a hyperparameter that adjust the level of entropy bonus. The term ”$-\log \pi\left(a|s\right)$”, including the minus sign, corresponds to the entropy term, which is a measure of the ”uncertainty” of the policy. A fully deterministic policy has low entropy, whereas a random policy has high entropy. The policy is encouraged to explore neighbouring action-state thanks to the entropy bonus. Usually, the term α is decreased with as the learning step increases, since the policy has learnt which actions lead the end-effector to the target position.

The value function $Q_\pi(s,a)$ of (13) is estimated with two ”Critic” neural network, whose parameters are updated by minimizing the mean square Bellman error (MSBE). The MSBE is a measure of the neural networks output to respect the Bellman equation, which evaluates a state-action value from its successor, namely:

$$Q_\pi(s,a) = \mathbb{E}[r(s,a) + \gamma\mathbb{E}[Q_\pi(s', a')]], \quad (14)$$

where γ is the discount factor, reflecting that short-term reward has more weight that long-term reward, and s', a' are the successor state and action. The use of two critic networks is the basis of the clipped double-Q that reduces the overestimation of the value function by selecting the minimum output of the two Q-networks. After obtaining J_pi with interaction with the environment, automatic differentiation tools available with TensorFlow and PyTorch can calculate one step for the gradient ascent. The SAC algorithm used in this paper is based on the stable-baselines GitHub repository [25], which is based on TensorFlow.

2.3 The reward function

The preceding subsection have described the environment (forward dynamics) and the agent (SAC). The reward obtained by the agent at each time step, is now described.

For pick-and-place trajectories, the reward function can intuitively be expressed as the minimization of the error ε between the position of the end-effector $\mathbf{p}$ and the position of the desired position $\mathbf{p}_d$, namely:

$$r_1 = -K_1\varepsilon \quad \text{with} \quad \varepsilon = \sqrt{\left(\mathbf{p}_d - \mathbf{p}\right)^T\left(\mathbf{p}_d - \mathbf{p}\right)}, \quad (15)$$

with the negative sign to penalize the distance between the desired state and the current state of the end-effector, and with a factor K_1 to adjust the reward. Another potential reward function uses the square error instead of the error, namely:

$$r_2 = -K_2\varepsilon^2. \tag{16}$$

The reward function based on r_1 and r_2 may penalize the actor while exploring far from the desired position. Increasing the regulating factor K_1 and K_2 may improve the behaviour close to the desired position, but could result in an unstable behaviour of the agent while far from the desired position.

Another solution would use an inverse function to promote the agent to get closer to the target [26], namely:

$$r_3 = \frac{K_3}{1 + K_4\varepsilon} \tag{17}$$

with K_3 and K_4 begin two parameters to adjust this new reward function. Figure 3 shows the three proposed reward function, namely based on the error, the square error, and the inverse of the error, named r_1, r_2, and r_3 respectively.

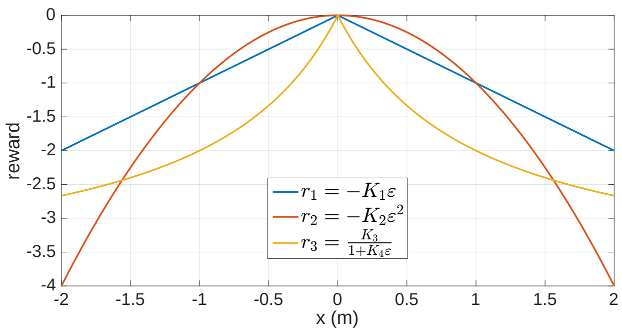


Figure 3: Reward function r_1, r_2, and r_3. The reward function r_3 has been shifted down for visual comparison.

Reward shaping in pick-and-place motion is clearly defined with the position error ε. For more abstract tasks, such as winning a game of table tennis, intermediate rewards may be designed to facilitate the end-goal, but the agent may be stuck in a local optima and favor local short-term reward instead of long-term winning return. This problem called ”reward shaping” can be assessed with sparse reward, where the agent is given a reward only on completion of the task, and with Hindsight Experience Replay (HER) [27]–[29], where the agent still learns from an episode, even if it ended with a goal different from the desired objective. This promising concept will be investigated in future research.

Parameter	Value
Learning rate α_{lr}	10^{-4}
Total training step	200k
Episode length	4000
Batch size	256
Actor neural network	$[256, 256]$
Environment time step Δt	0.005 sec
Attachment point position	$\mathbf{a}_1 = [0, -0.3]^T$
	$\mathbf{a}_2 = [0, 0.3]^T$
End-effector mass	$m = 0.5$kg
Maximal cable tension	$t_{\max} = 10$N

Table 1: Parameter for the pick-and-place task

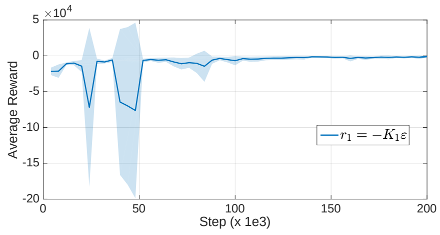
2.4 Simulation parameters

For pick-and-place trajectories, it is assumed that the velocity and the acceleration of the end-effector at the desired position is zero. Table 1 summarizes the main parameters used to launch the simulation. The simulation runs on a CPU 2.3 GHz Quad-Core i5. One training session of 200k step takes approximately 1 hour. The initial position and the final position of the end-effector are chosen randomly in the workspace of the mechanism, below the attachment point $\mathbf{a}_1$ and $\mathbf{a}_2$, between the range $[0, 1]$ in the x direction and $[-0.3, 0.3]$ in the y direction. Notice that from figure 1, the x-direction is vertical and directed downward.

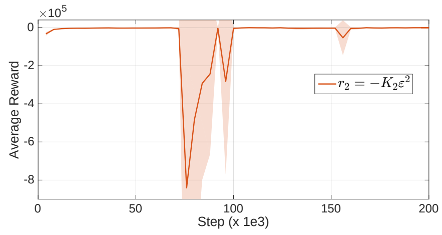
3 Results

The aim of the simulation is to compare the effect of different reward functions on the behavior of the agent. There are three environments, corresponding to the reward function r_1, r_2, and r_3. The agent is trained three times on each environment with different initial random seed, since the parameter initialization of the neural network has an influence on the agent's learning.

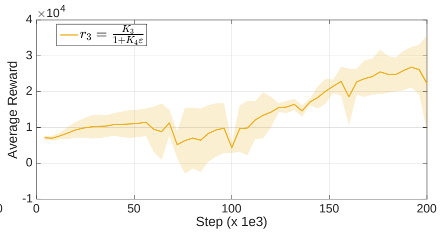
Figure 4 shows the average reward obtained during the learning phase of the agent. The average reward and the standard deviation of the three run are shown as a solid line and shaded area respectively. The convergence of the agent with the environment using the reward function r_1 takes about 150k, and the environment with the reward r_2 takes approximately 50k steps, if the drop in performance is ignored. But then, during the training with these two environments, the average reward drops drastically, which is due to the exploration of the agent. The agent is deeply penalized for exploring wrong action space. The agent interacting with the environment that outputs the reward r_3 has much less variance in terms of average reward, but after the end of the training period, it appears that additional training steps could lead to better convergence.



(a) Reward function r_1

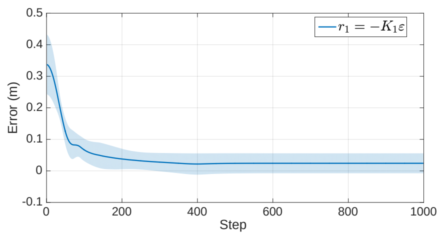


(b) Reward function r_2

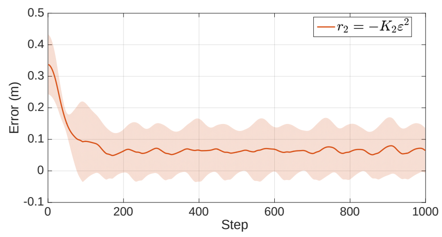


(c) Reward function r_3

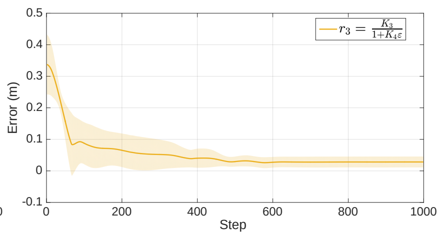
Figure 4: Average reward during the training with 200k step.



(a) Reward function r_1



(b) Reward function r_2



(c) Reward function r_3

Figure 5: Average error obtained with the trained model.

Figure 5 shows the average error obtained while using the trained agent for the three environment. During the first 200 steps, the error is high, which is expected since the agent is still at the initial position. Then the error converges toward zero for all trained agents. The environment with the reward r_2 has high variance and relatively poor convergence toward zero error. The environment with the reward r_1 and r_3 have better convergence and lower variance, even though the reward function r_1 has a slight edge in terms of convergence speed.

4 Discussion

In the simulation shown in figure 4, the agent pulls and releases the cables to position the end-effector that would maximize the reward during an episode. The standard strategy used in RL is to randomly try different cable tension until converging to a satisfactory position close to the desired one, which is an inefficient strategy that requires a lot of interaction between the agent and the environment. The inverted pendulum problem—one popular problem in the OpenAI gym (`https://gym.openai.com/`), with one continuous action space and three observation space, takes approximately 50k learning steps to converge to an optimal solution. When the dynamics of the mechanism is known, one way to improve the convergence speed and to help the learning phase of the agent is to use the inverse dynamics to calculate the required cable tension from the desired position, which gives a basis on the action to take for the agent instead of randomly guessing which action would maximize the cost function [30].

The number of learning step n depends on the step time Δt used in the simulation of the environment and the time duration of an episode τ, with the relationship given as:

$$n = \frac{\tau}{\Delta t} \tag{18}$$

The number of step can be reduced when the duration τ is reduced or if the step time Δt is increased. On a physical robot, it is often desired to have a fixed episode length. Increasing the step time Δt may lead into an unstable behaviour, especially using Euler's method to integrate the acceleration and to calculate velocity and position. Figure 6 shows the comparison between Euler's method and RK4 method for 100k training steps and $\Delta t = 0.01$ sec, with the reward function r_3, averaged on three runs with three different seeds. It can be noticed that the average reward obtained with RK4 is higher than with Euler's method: the reason is that Euler's approximation on the velocity and the position degrades as Δt gets larger. Moreover because the time step is relatively large, the position accuracy of the end-effector is worsen, which prevent the agent to converge to the desired position.

5 Conclusion

This paper presents an implementation of reinforcement learning for positioning a two-DOFs CSPR for point-to-point trajectories. The derivation of the dynamics of the mechanical system has been derived

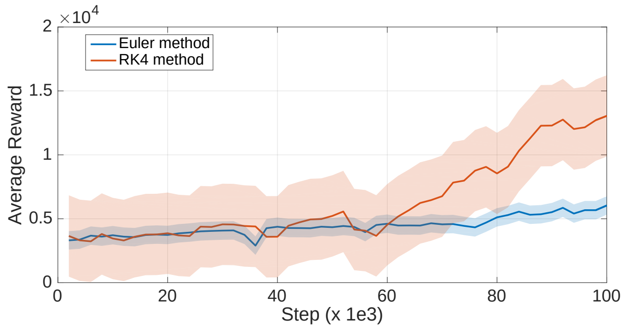


Figure 6: Comparison between Euler method and RK4.

and the reinforcement learning algorithm, namely Soft Actor-Critic, has been described. The result shows a comparison between different reward function provided by the environment and indicates that reward shaping has an influence on the behaviour of the trained agent. A comparison between Euler method and RK4 has been presented and highlight benefits of using more precise in terms of learning steps. Future research will focus on the integration of such RL technique on physical CSPR and comparison with classical control theory.

References:

[1] C. Gosselin and S. Foucault, "Dynamic point-to-point trajectory planning of a two-dof cable-suspended parallel robot," *IEEE Transactions on Robotics*, vol. 30, no. 3, pp. 728–736, 2014.

[2] C. Gosselin, P. Ren, and S. Foucault, "Dynamic trajectory planning of a two-dof cable-suspended parallel robot," in *2012 IEEE International conference on Robotics and Automation*, IEEE, 2012, pp. 1476–1481.

[3] R. Bostelman, J. Albus, N. Dagalakis, A. Jacoff, and J. Gross, "Applications of the nist robocrane," in *Proceedings of the 5th International Symposium on Robotics and Manufacturing*, vol. 5, 1994.

[4] P. D. Campbell, P. L. Swaim, and C. J. Thompson, "Charlotte™ robot technology for space and terrestrial applications," *SAE transactions*, pp. 641–648, 1995.

[5] H. R. Fahham, M. Farid, and M. Khooran, "Time optimal trajectory tracking of redundant planar cable-suspended robots considering both tension and velocity constraints," *Journal of dynamic systems, measurement, and control*, vol. 133, no. 1, 2011.

[6] B. Zi, X. Wu, J. Lin, and Z. Zhu, "Inverse kinematics and singularity analysis for a 3-dof hybrid-driven cable-suspended parallel robot," *International journal of advanced robotic systems*, vol. 9, no. 4, p. 133, 2012.

[7] J.-T. Liu, F.-Y. Wu, W.-J. Lu, and B.-L. Zhang, "Domain adaption for facial expression recognition," in *2019 International Conference on Machine Learning and Cybernetics (ICMLC)*, IEEE, 2019, pp. 1–6.

[8] J. D. Deng, "Online outlier detection of energy data streams using incremental and kernel pca algorithms," in *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, IEEE, 2016, pp. 390–397.

[9] F. Djavanroodi, "Artificial neural network modeling of surface roughness in magnetic abrasive finishing process," *Applied Sciences, Engineering and Technology*, vol. 6, no. 11, pp. 1976–1983, 2013.

[10] G. Latif, J. Alghazo, L. Alzubaidi, M. M. Naseer, and Y. Alghazo, "Deep convolutional neural network for recognition of unified multi-language handwritten numerals," in *2018 IEEE 2nd International Workshop on Arabic and Derived Script Analysis and Recognition (ASAR)*, IEEE, 2018, pp. 90–95.

[11] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[12] T. Ma, H. Xiong, L. Zhang, and X. Diao, "Control of a cable-driven parallel robot via deep reinforcement learning," in *2019 IEEE International Conference on Advanced Robotics and its Social Impacts (ARSO)*, IEEE, 2019, pp. 275–280.

[13] H. Xiong, T. Ma, L. Zhang, and X. Diao, "Comparison of end-to-end and hybrid deep reinforcement learning strategies for controlling cable-driven parallel robots," *Neurocomputing*, vol. 377, pp. 73–84, 2020.

[14] A. El-Fakdi and M. Carreras, "Policy gradient based reinforcement learning for real autonomous underwater cable tracking," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2008, pp. 3635–3640.

[15] Q. Wu, Y. Gu, Y. Li, B. Zhang, S. A. Chepinskiy, J. Wang, A. A. Zhilenkov, A. Y. Krasnov, and S. Chernyi, "Position control of cable-driven robotic soft arm based on deep reinforcement learning," *Information*, vol. 11, no. 6, p. 310, 2020.

[16] B. Doroodgar and G. Nejat, "A hierarchical reinforcement learning based control architecture for semi-autonomous rescue robots in cluttered environments," in *2010 IEEE International Conference on Automation Science and Engineering*, IEEE, 2010, pp. 948–953.

[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 1861–1870.

[18] X. Jiang and C. Gosselin, "Dynamic point-to-point trajectory planning of a three-dof cable-suspended parallel robot," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1550–1557, 2016.

[19] X. Jiang, E. Barnett, and C. Gosselin, "Periodic trajectory planning beyond the static workspace for 6-dof cable-suspended parallel robots," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1128–1140, 2018.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing atari with deep reinforcement learning*, 2013.

[21] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, *Soft actor-critic algorithms and applications*, 2018.

[22] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," 2014.

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, 2015.

[24] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, 2018.

[25] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, *Stable baselines*, `https://github.com/hill-a/stable-baselines`, 2018.

[26] L. Butyrev, T. Edelhäußer, and C. Mutschler, *Deep reinforcement learning for motion planning of mobile robots*, 2019.

[27] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, *Hindsight experience replay*, 2017.

[28] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba, *Multi-goal reinforcement learning: Challenging robotics environments and request for research*, 2018.

[29] M. Kim, D.-K. Han, J. Park, and J.-S. Kim, "Motion planning of robot manipulators for a smoother path using a twin delayed deep deterministic policy gradient with hindsight experience replay," *Applied Sciences*, vol. 10, p. 575, 2020.

[30] J. K. Gupta, K. Menda, Z. Manchester, and M. J. Kochenderfer, "A general framework for structured learning of mechanical systems,"

Contribution of individual authors to the creation of a scientific article (ghostwriting policy)

Dinh-son Vu conducted the research and investigation process. He also carried out the simulation of this work. Ahmad Alsmadi did the preparation, creation and presentation of the published work.

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)