ROS2Learn: a reinforcement learning framework for ROS 2

Yue Leire Erro Nuin*, Nestor Gonzalez Lopez*, Elias Barba Moral*, Lander Usategui San Juan, Alejandro Solano Rueda, Víctor Mayoral Vilches and Risto Kojcev
Acutronic Robotics, March 2019

***Abstract*—We propose a novel framework for Deep Reinforcement Learning (DRL) in modular robotics to train a robot directly from joint states, using traditional robotic tools. We use an state-of-the-art implementation of the Proximal Policy Optimization, Trust Region Policy Optimization and Actor-Critic Kronecker-Factored Trust Region algorithms to learn policies in four different Modular Articulated Robotic Arm (MARA) environments. We support this process using a framework that communicates with typical tools used in robotics, such as Gazebo and Robot Operating System 2 (ROS 2). We evaluate several algorithms in modular robots with an empirical study in simulation.**

I. Introduction

Current robot systems are designed, built and programmed by teams with multidisciplinary skills. The traditional approach to program such systems is typically referred to as the *robotics control pipeline* and requires going from observations to final low-level control commands through: a) state estimation, b) modeling and prediction, c) planning, and d) low level control translation [1]. As introduced by Zamalloa et al. [2], the entire process requires the fine tuning of every step in the pipeline, incurring in a significant complexity, where optimization at every step is critical and has a direct impact in the final result.

Artificial Intelligence methods and, particularly, neuromorphic techniques such as artificial neural networks (ANNs) are becoming more and more relevant in robotics. Starting from 2016, promising results, such as the work of Levine et al. [3], showed a path towards simplifying the construction of robot behaviours through the use of deep neural networks as a replacement of the traditional approach outlined above. The described end-to-end approach for programming robots scales nicely when compared to traditional methods.

Reinforcement Learning (RL) is a field of machine learning that is concerned with making sequences of decisions. It considers an agent situated in an environment where for each timestep the agent takes an action and receives an observation and a reward. A RL algorithm seeks to maximize the agent's total reward trough a trial and error learning process. Deep Reinforcement Learning (DRL) is the study of RL by using neural networks as function approximates. In recent years, several techniques for DRL have shown good success in learning complex behaviour skills and solving challenging control tasks in high-dimensional state-space [4], [5], [6], [7], [8]. However, many of the benchmarked environments such as Atari [9] and Mujoco [10] rarely deal with realistic or complex environments (frequent in robotics) [11], [12], or use the tools commonly used in the field such as the Robot Operating System (ROS)[13]. The research conducted in the previous work can only be translated into real world robots with a considerable amount of effort for each particular robot. Hence, the scalability of previous methods for modular robots is questionable.

Modular robots can extend their components seamlessly by just adding modules to the robotic system. This brings clear advantages for the construction of robots, however training them with current DRL methods becomes cumbersome due to the following reasons: every small change in the physical structure of the robot will require a new training; building the tools to train modular robots (such as the simulation model, virtual drivers) is a time consuming process; transferring the results to the real robot is complex given the flexibility of these systems. In this work we present a framework that utilizes the traditional tools in the robotics environment, such as Gazebo[14] and ROS 2, which simplifies the process of building modular robots and their corresponding tools. Our framework includes baseline implementations[15] for the most common DRL techniques for policy iteration methods. Using this framework we present the results obtained benchmarking DRL methods in a modular robot with 6 degrees-of-freedom (DoF).

II. Previous Work

Recent advances in the field of RL have led to the development of different approaches with neural network function approximators. Among the available techniques, the focus of this work is on model-free RL methods:

* authors contributed equally

Proximal Policy Optimization (PPO) [7] and natural gradient policy based methods, such as Trust Region Policy Optimization (TRPO) [6] and Actor Critic using Kronecker-Factored Trust Region (ACKTR) [8]. All of these are known as policy gradient methods, which perform updates at each episode to the policy parameters (on-policy).

TRPO [6] is a policy gradient method meant to solve RL problems more efficiently than "vanilla" policy gradient (VPG) [16]. The idea used is to update the weights as fast as possible without diverging. For achieving this, TRPO uses a constrain linked to the KL-Divergence [17], that gives a measure of distance between two probability distributions. TRPO can be applied both for learning non-trivial tasks in continuous control as well as for discrete control policies directly from raw pixel inputs. Thanks to the use of the natural policy gradient method, TRPO overcomes some limitations of VPG such as choosing the step-size and the low sample efficiency. Compared to other algorithms [18] [19], TRPO has proven to be a good approach for continuous control tasks. Details of the theoretical aspects of the TRPO method are given in Section III-B1.

ACKTR is an actor-critic RL method that applies trust region policy optimization using Kronecker-factored approximation (K-FAC) to the curvature. This method uses the natural policy gradient and optimizes both the actor and the critic [8]. Similarly to TRPO, ACTKR also uses the benefits from natural policy gradient and can be applied both in continuous and discrete environments. In the evaluation of Wu et. al [8], ACKTR sample and computational efficiency was evaluated in Atari and several Mujoco environments, and it was compared with the performance of Advantage Actor Critic (A2C) and TRPO. Wu et. al [8] results indicate that the performance of ACKTR surpassed the performance of A2C and TRPO. In this work, we extend the evaluation of ACKTR to a set of environments which are particular for representing different robot configurations and scenarios. Details of the theoretical aspects of the ACKTR method are given in Section III-B3.

PPO is a policy gradient method for RL which alternates between sampling data trough interaction with the environment and optimizing the "surrogate" objective using Stochastic Gradient Descent [7]. PPO differs from standard policy gradient methods by enabling multiple epochs of mini batch updates. Compared to its predecessor (TRPO), PPO uses the "surrogate" objective by clipping the policy probability ratio. In the original work, PPO was evaluated in the Atari, Mujoco and Roboschool [20] environments, where it had better performance compared to A2C, A2C + Trust region, VPG and TRPO. Details of the theoretical aspects of PPO are given in Section III-B2.

Previous works, [21], [22], [23], present partial success of transferring learned behaviour in simulation to a real robot. These works explain the importance of having scenes in simulation as similar as possible to the reality in order to simplify the process of transferring the learned behaviour to real scenarios. Yuke Zhu et. al [24] describe high-quality and realistic 3D scenes. The approach of Tobin et. al [25], randomizes the rendering in simulation, reaching enough variability. This allows for the images in the real world to be considered as just another variation in the simulator. To the best of our knowledge, the work conducted in previous approaches focuses on restricted scenarios in a controlled environment where specific algorithms for solving particular task were used. This is not the case when a robotic system needs to be deployed in realistic scenarios, specially if the robot is modular and can present a number of different configurations.

The methods presented above have all different theoretical approaches for solving RL tasks with their strengths and drawbacks, which makes it hard to determine which one is the most appropriate choice for a particular application and environment. The aim of this work is to evaluate the above mentioned RL algorithms with the focus of determining which one of them is best suited for modular robotic applications. Section III describes the theoretical aspects of the evaluated RL methods and their adaptation to applications for modular robots. Section IV presents the experimental evaluation conducted in 6DoF modular Modular Articulated Robotic Arm (MARA) robot[1]. Section V summarizes results and presents future perspective and work.

III. Methods

A. Nomenclature

The methods presented bellow will be consistent with the following nomenclature that is partially inspired on the work by Peters et al. [26]:

The three main components of a RL system for robotics include the state s (also found in literature as x), the action a (also found as u) and the reward denoted by r. We will denote the current time step by k. The stochasticity of the environment gets represented by using a probability distribution $\mathbf{s}_{k+1} \sim p\left(\mathbf{s}_{k+1} | \mathbf{s}_k, \mathbf{a}_k\right)$ as model where $\mathbf{a}_k \in \mathbb{R}^M$ denotes the current action and $\mathbf{s_k}$, $\mathbf{s}_{k+1} \in \mathbb{R}^N$ denote the current and next state, respectively. Further, we assume that most policy gradient methods have actions that are generated by a policy $\mathbf{a}_k \sim \pi_\theta\left(\mathbf{a}_k | \mathbf{s}_k\right)$ which is modeled as a probability distribution in order to incorporate exploratory actions.

The policy is assumed to be parametrized by K policy parameters $\theta \in \mathbb{R}^K$. The sequence of states and actions

[1] https://acutronicrobotics.com/products/mara/

forms a trajectory denoted by $\tau = [\mathbf{s}_{0:H}, \mathbf{a}_{0:H}]$ where H denotes the horizon which can be infinite. Often, *trajectory*, *history*, *trial* or *roll-out* are used interchangeably. At each instant of time, the learning system receives a reward $r_k = r(\mathbf{s}_k, \mathbf{a}_k) \in \mathbb{R}$.

The general goal of policy optimization is to optimize the policy parameters $\theta \in \mathbb{R}^K$ so that the expected return is optimized:

$$J(\theta) = E\left\{\sum_{k=0}^{H} \gamma \cdot r_k\right\} \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor.

For real-world applications, we require that any change to the policy parameterization has to be smooth. Otherwise, drastic changes can be hazardous for the actor, and useful initializations of the policy based on domain knowledge would vanish after a single update step. For these reasons, policy gradient methods which follow the steepest descent on the expected return are the method of choice. These methods update the policy parameters according to the gradient update rule

$$\theta_{h+1} = \theta_h + \alpha_h \nabla_\theta J|_{\theta=\theta_h}, \tag{2}$$

where $\alpha_h \in \mathbb{R}^+$ denotes the learning rate and $h \in \{0, 1, 2, \ldots\}$ the current update number.

Table I: Summary of the terms used within the article.

s	the *state* (also found in literature as x)
a	the *action* (also found as u)
r	the *reward*
k	time step
$\mathbf{s}_{k+1} \sim p(\mathbf{s}_{k+1} \mid \mathbf{s}_k, \mathbf{a}_k)$	probability distribution representing the stochasticity of the environment
γ	discount factor
τ	trajectories
$\tau \sim p_\theta(\tau) = p(\tau \mid \theta)$	roll-outs
$r(\tau) = \sum_{k=0}^{H} \gamma r_k$	return in the roll-outs

B. Benchmarked algorithms

One of the main distinctions between algorithms in RL is based on if they are value-based or policy-based. The first class, value-based, attempts to learn to assess correctly what is the reward obtained in a certain state and thus, maximize the final expected reward. The second class, policy-based, attempts to learn what action to do at each state in order to maximize the final reward. Robotics is dominated by scenarios with continuous changes in states and actions spaces, which implies that most traditional value-based off-the-shelf RL approaches are not valid for treating such situations. As pointed out by Peters et al. [26], Policy Gradient (PG) methods differ significantly from others as they do not suffer from these problems in the same way other techniques do.

One of the typical problems experienced when doing RL in robotics[2] is that uncertainty in the state might degrade the performance of the policy. PG methods suffer from this as well. However, they rely on optimization techniques for the policy that do not need to be changed when dealing with this uncertainty.

The nature of PG methods allows them to deal with continuous states and actions in exactly the same way as discrete ones. PG techniques can be used either on model-free or model-based approaches. The policy representation can be chosen in order to be meaningful for the task, and can incorporate domain knowledge. This often leads to the use of fewer parameters in the learning process. Additionally, its generic formulation shows that PG methods are valid even when the reward function is discontinuous or even unknown.

While PG techniques might seem interesting for a roboticist on a first look, they are by definition on-policy and need to forget data reasonably fast in order to avoid the introduction of a bias to the gradient estimator. In other words, they are not as good as other techniques at using the data available (their sample efficiency is low). Other typical problem with PG methods is that convergence is only guaranteed to a local maximum while in tabular representations, value function methods are guaranteed to converge to a global maximum.

1) Trust Region Policy Optimization (TRPO): Trust Region Policy optimization is an attempt of improving VPG, by choosing appropriately the magnitude of update at each iteration [6]. In order to know how much to update, TRPO uses the KL-divergence, which returns a measure of how different two probability distributions are. The formal definition of the problem is:

$$\begin{aligned} \theta_{k+1} = & arg \max_{\theta} L_\theta(\theta_k) \\ & s.t. D_{KL}(\theta \| \theta_k) \leq \delta \end{aligned} \tag{3}$$

where $L_\theta(\theta_k)$ is the surrogate advantage, representing how good a policy π_θ is with respect to an old policy

[2] provided that there is no additional state estimator (actor-critic methods)

π_{θ_k}:

$$L_\theta(\theta_k) = \underset{s,a\sim\pi_{\theta_k}}{E}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a)\right] \quad (4)$$

where $A^{\pi_{\theta_k}}$ is the advantage function. In the baselines implementation this advantage function is calculated using a value estimation (Actor-Critic structure)[15].
The analytical solution of the KL- divergence for each step is expensive, but is possible to approximate its value using a Taylor expansion of degree 2. To solve the optimization problem from Eq.3, its commonly used the Langrangian method. The obtained expression can be Taylor expanded, and the obtained result is known as the natural gradient [27]. In order to solve analytically the natural gradient, the Fisher information matrix (FIM) needs to be calculated (the Hessian of the KL-divergence), which is not trivial to compute and store. For that TRPO uses a trick, optimizing a sub-problem, and finally performing a backtracking line search.
This approach, more than a structure, is way of optimizing the search of the parameters. Therefore it can be used with an Actor-Critc structure that calculates the value function and uses it in the advantage calculation for example.

2) Proximal Policy Optimization (PPO): Proximal Policy Optimization (PPO) is an alternative to Trust Region Policy Optimization (TRPO) [6], that attains data efficiency and reliable performance of TRPO while using first order optimization. In the case of standard PG methods, the gradient update is performed per data sample. On the other hand, PPO enables multiple epochs of mini-batch updates. There are a few variants of PPO in the literature, which optimize the "surrogate" objective or use adaptive KL penalty coefficient [7]. The Clipped Surrogate Objective is given as:

$$L^{clip}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \quad (5)$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, is the probability ratio of the current policy π_θ and the previous policy $\pi_{\theta_{old}}$, $\hat{A}_t$ is an estimator of the advantage function at timestep t and ϵ is a hyperparameter, for example $\epsilon = 0.2$. The first term is the "surrogate" objective that is also used in TRPO (Eq.4). The second term, $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$, is clipping the probability ratio, r_t, to be between the interval $[1-\epsilon, 1+\epsilon]$. The $L^{clip}(\theta)$ takes the minimum of the un-clipped and clipped value, which excludes the change in the probability ratio when the objective improves and includes it when the objective is worse. The clipping prevents PPO from having a large policy update. The Adaptive KL Penalty Coefficient is an alternative to the clipped "surrogate" objective or an addition to it where the goal is to use the penalty on KL divergence and update the penalty coefficient to achieve some target KL divergence (d_{targ}) at each policy update. As described in [7], the KL Penalty Coefficient performed worse than the surrogate objective, therefore the presented pseudocode and experimental evaluation of PPO uses the clipped surrogate objective.

Algorithm 1 Trust Region Policy Optimization (TRPO)

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps
3: **for** $k = 0, 1, 2, ...$ **do**
4: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
5: Compute rewards-to-go $\hat{R}_t$.
6: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
7: Estimate policy gradient as:
$$\hat{g}_k = \frac{1}{\mathcal{D}_k}\sum_{\tau\in\mathcal{D}_k}\sum_{t=0}^{T}\nabla_\theta log\pi_\theta(a_t|s_t)|_{\theta_k}\hat{A}_t.$$
8: Use the conjugate gradient algorithm to compute:
$$\hat{x}_k \approx \hat{H}_k^{-1}\hat{g}_k,$$
where $\hat{H}_k^{-1}$ is the Hessian of the sample average KL-divergence.
9: Update the policy by backtracking line search with:
$$\theta_{k+1} = \theta_k + \alpha^j\sqrt{\frac{2\delta}{\hat{x}_k^T\hat{H}_k\hat{x}_k}}\hat{x}_k,$$
where $j \in \{0, 1, 2, ...K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.
10: Fit value function by regression on mean-squared error:
$$\phi_{k+1} = arg\ \min_\phi \frac{1}{\mathcal{D}_k T}\sum_{\tau\in\mathcal{D}_k}\sum_{t=0}^{T}(V_\phi(s_t) - \hat{R}_t)^2,$$
typically via some gradient descent algorithm.
11: **end for**

3) Actor Critic using Kronecker-Factored Trust Region (ACKTR): The idea of Actor Critic using Kronecker-Factored Trust Region (ACKTR) is to replace the Stochastic Gradient Descent (SGD), which explores the weight space inefficiently, and to optimize both the actor and the critic using Kronecker-factored approximate curvature (K-FAC) with trust region [8]. ACKTR

Algorithm 2 Proximal Policy Optimization (PPO)

Initialize the time steps (T)
Initialize the clipping value ϵ
for $i = 1, N_{iterations}$ **do**
  for $t = 1, T$ **do**
    Run MLP policy and generate action a_t
    Execute action a_t in emulator and observe reward
    Update observation (ob) based on current joint positions and end-effector position
    Estimate advantage function $\hat{A}_t$
    for $epoch = 1, N_{epochs}$ **do**
      Compute SGD of loss function:
      $L^{clip}(\theta) =$
      $\hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$
    end for
  end for
end for

replaces SGD of A2C, the synchronous version of A3C [28], and instead computes the natural gradient update. The natural gradient update is applied both to the actor and the critic.

ACKTR uses the K-FAC to compute the natural gradient update efficiently. In order to define a Fisher metric for RL policies, ACKTR uses a policy function that defines a distribution over actions given the current state, and takes the expectation over the trajectory distribution. The mathematical formulation for the Fisher metric is given by:

$$F_a = \mathbb{E}_{p(\tau)}[\nabla_\theta \log \pi(a_t|s_t)(\nabla_\theta \log \pi(a_t|s_t))^T] \quad (6)$$

where $p(\tau) = p(s_0)\prod_{t=0}^{T} \pi(a_t|s_t)p(s_{t+1}|s_t, a_t)$ is the distribution of trajectories. In practice, we approximate the intractable expectation above with trajectories collected during training. In the case of training the critic, one can think of it as a least-squares function approximation problem. In this case, the most common second-order algorithm is Gauss-Newton, which approximates the curvature as the Gauss-Newton matrix $G := \mathbb{E}[J^T J]$, where J is the Jacobian mapping from parameters to outputs [29]. The Gauss-Newton matrix is equivalent to the Fisher matrix for a Gaussian observation model, which allows to apply K-FAC to the critic as well. In more detail, the output of the critic v is defined to be a Gaussian distribution with $p(v|s_t) \sim \mathcal{N}(v; V(s_t), \sigma^2)$. Setting σ to 1 is equivalent to the vanilla Gauss-Newton method.

In the case when the actor and the critic are disjoint, it is possible to apply K-FAC updates to each of them using the same metric as defined in Equation 6. To prevent instability during training, it is important to use an architecture where the two networks both share lower-layer representations but have distinct output layers [30], [28]. The joint distribution of the policy and the value distribution can be defined by assuming independence of the two output distributions, for instance $p(a, v|s) = \pi(a|s)p(v|s)$, and constructing the Fisher metric with respect to $p(a, v|s)$. This is similar to the standard K-FAC, except that we need to sample the two networks' outputs independently. In this case, the K-FAC to approximate the Fisher matrix is:

$$F_v = \mathbb{E}_{p(\tau)}[\nabla \log p(a, v|s) \nabla \log p(a, v|s)^T] \quad (7)$$

The pseudocode presented gives an overview of the ACKTR implementation used in our evaluation.

IV. Experiments

As previously presented by Zamora et al [12], for the benchmark experiments we use an extension of the OpenAI gym which is tailored for robotics. We added four additional environments to evaluate the algorithms, which match the modular MARA 6DoF. The environments differ on how they reward the actions taken, and are described in detail in gym-gazebo2 [31]. For the training, we used the Gazebo simulator and corresponding ROS 2 packages, to convert the actions generated from each algorithm into appropriate trajectories that the robot can execute.

We set the initial position of the robot to zero for all joints and reset the robot to this initial position when the number of steps exceeds the maximum timesteps for an episode. We code this in an environment-specific variable denoted ***max_episode_steps***, which in our case is set to 2048. For these specific experiments, we located the fixed target at the coordinates $[x = -0.40028, y = 0.095615, z = 0.72466]$ with respect to the origin of the environment, which in our case is set to be the base of the 6DoF MARA robot; and the orientation at the quaternion $[w = 0., x = 0.7071068, y = 0.7071068, z = 0.]$, with respect to the table orientation. Each algorithm generates actions that are translated into the corresponding ROS 2 messages and are executed in simulation. The simulation then returns the observations (current joint positions and end-effector pose) and gives them to the algorithm. Figure 1 illustrates the experimental environment. For each environment we perform one experiment consistent in a training for 1 million steps in the environment.

Figure 2, Figure 3, Figure 4 and Figure 5 show the reward obtained in the learning process for the different algorithms. In general TRPO and PPO show ability to learn at a similar pace, particularly in non-orient environments, which is not surprising given that they both have similar formulation. The discrepancies in

Algorithm 3 Actor Critic using Kronecker-Factored Trust Region (ACKTR)

Assume shared parameter vector for the actor a and v for the critic.
Assume global shared counter $T = 0$
Initialize step counter $t \leftarrow 1$
repeat
  Reset action $a \leftarrow 0$ and state $s \leftarrow 0$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform action a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t or $t - t_{start} == t_{max}$
  $R = \begin{cases} 0, \text{for terminal } \mathrm{s_t} \\ V(s_t, \theta'_v), \text{ for non} - \text{terminal } \mathrm{s_t}. \end{cases}$
  for $i = t-1, t_{start}$ **do**
    $R \leftarrow r_i + \gamma R$
    Calculate natural gradient for the actor:
    $F_a = \mathbb{E}_{p(\tau)}[\nabla_\theta \log \pi(a_t|s_t)(\nabla_\theta \log \pi(a_t|s_t))^T]$
    $p(\tau) = p(s_0) \prod_{t=0}^{T} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$,
    with $p(\tau)$ as distribution of trajectories collected during training
    if a and v are joint **then**
      Output of the critic v is defined to be a Gaussian distribution: $p(v|s_t) \sim \mathcal{N}(v; V(s_t)\sigma^2)$
      Apply Fisher matrix for the critic
    end if
    if a and v are disjoint **then**
      Apply K-FAC to approximate Fisher matrix for the critic
      $F_v = \mathbb{E}_{p(\tau)}[\nabla \log p(a, v|s) \nabla \log p(a, v|s)^T]$
    end if
  end for
until $T > T_{max}$

orient environments might be due to the fact that those could be more dependant on the random initialization. ACKTR does not seem to be an efficient learner for this task. See more details in Section A. It can be due to the used hyperparameters, which were all the same in the three different algorithms in order to compare them.

V. Conclusion and Future Work

We have presented evaluation of different DRL techniques for modular robotics. Our setup and framework consists of tools, such as ROS 2 and Gazebo, allowing a

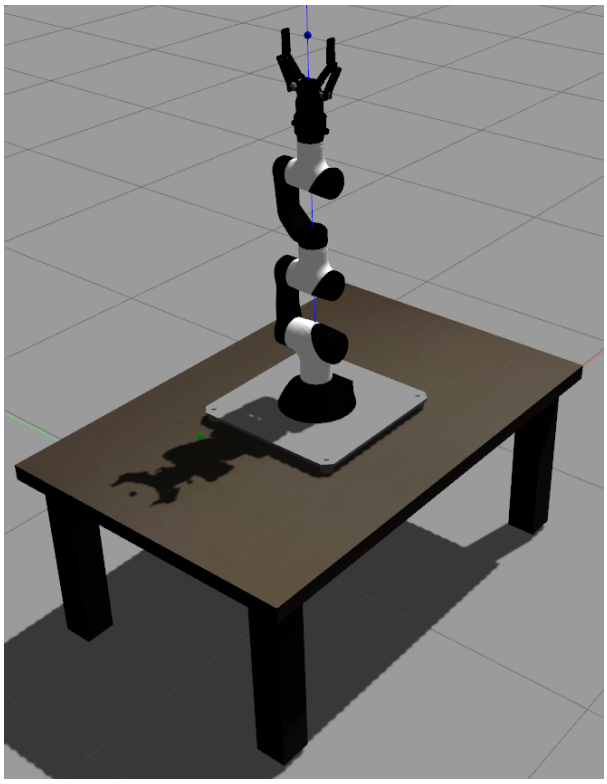

Figure 1: gym-gazebo2 MARA robot environments displayed on Gazebo gzclient simulator. All environments are included since their differences are in how the learning is rewarded and not in the model

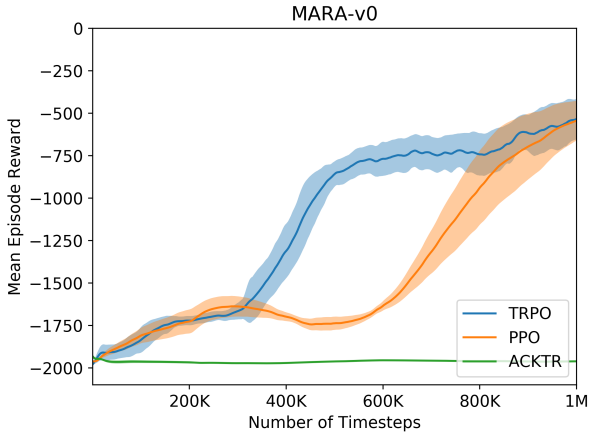

Figure 2: Performance comparisons of the tested algorithms for the $MARA - v0$ environment. The shaded region denotes the deviation with respect to the previous 100 steps. PPO and TRPO seem to achieve a similar level of performance by the end of the experiments, even though TRPO seems to learn faster at earlier stages. The reward obtained by ACKTR remains unchanged compared with the other algorithms.

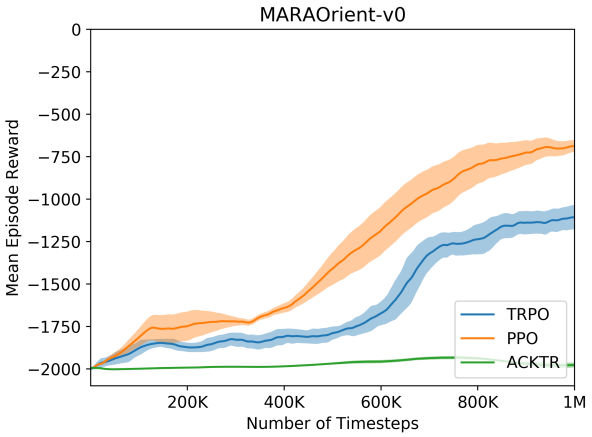

Figure 3: Performance comparisons of the tested algorithms for the $MARAOrient - v0$ environment. The shaded region denotes the deviation of the rewards with respect to the previous 100 steps. PPO is able to get a slightly better result than TRPO in this environment, while ACKTR stays flat compared to the other two.

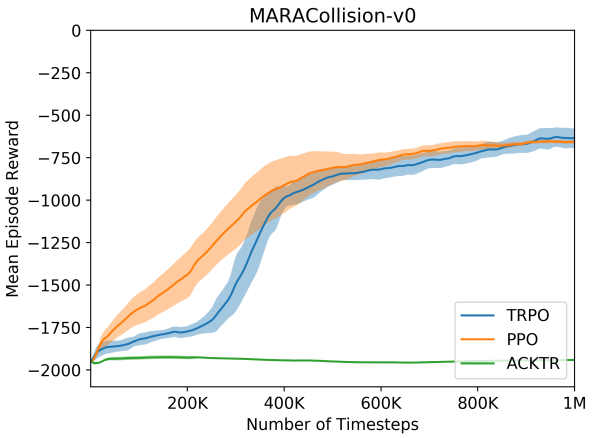

Figure 4: Performance comparisons of the tested algorithms for the $MARACollision - v0$ environment trained. The shaded region denotes the deviation of the rewards with respect to the previous 100 steps. PPO and TRPO seem to have similar performance, while the reward obtained by ACKTR remains flat in comparison.

more realistic representation of the environment. Our results show that our proposed framework is stable during training of neural networks trough RL with policy-based methods.

There still remain many challenges within the DRL field for robotics. The main problems are the long training times, the simulation-to-real robot transfer, reward shaping, sample efficiency and extending the behaviour to diverse tasks and robot configurations.

So far, our work with the modular robot MARA has

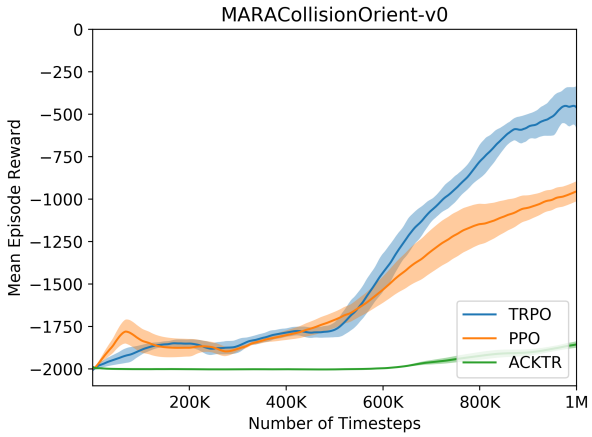

Figure 5: Performance comparisons of the tested algorithms for the $MARACollisionOrient - v0$ environment. The shaded region denotes the deviation with respect to the previous 100 steps. TRPO shows better performance towards the end of the experiment compared to PPO, while this time, ACKTR shows some learning towards the end of the experiment, though not comparable with any of the other algorithms.

focused on simple tasks such as reaching a point in space. In order to have an end-to-end training framework (from pixels to motor torques) and to perform more complex tasks, we aim to integrate additional rich sensory input such as vision. Inspired by the work of [32], [33], we intend to explore imitation learning that provides high-quality human training data through demonstrations which might be useful for the robot to learn to perform more complex tasks.

We envision the future of robotics to be modular robots where the trained network can generalize online to modifications in the robot such as change of a component or dynamic obstacle avoidance. In order to accomplish this, we aim to explore methods that allow novel training approaches of the robot for every new environment, type of robot or when the original task for which the network was trained for is changed. Inspired by [34], we aim to evaluate meta-learning and hierarchical RL methods that allow to generalize to new tasks and environments by learning sub-policies; for instance motor primitives that can be reused across different sets of tasks, and even generalizing to unseen new tasks.

APPENDIX

Figure 6, Figure 7, Figure 8 and Figure 9 show the reward obtained in the learning process with ACKTR algorithm in each MARA robot environment. The shaded region denotes the deviation with respect to the previous 100 steps.

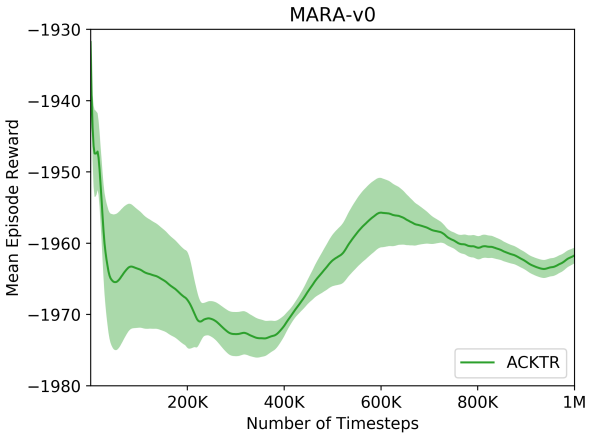

Figure 6: Performance of $MARA - v0$ environment trained with ACKTR algorithm

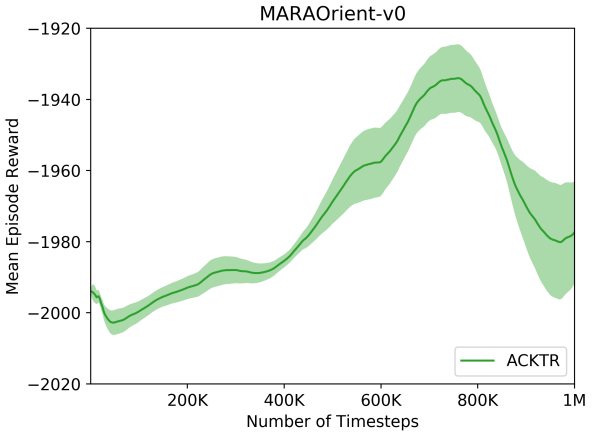

Figure 7: Performance of $MARAOrient - v0$ environment trained with ACKTR algorithm

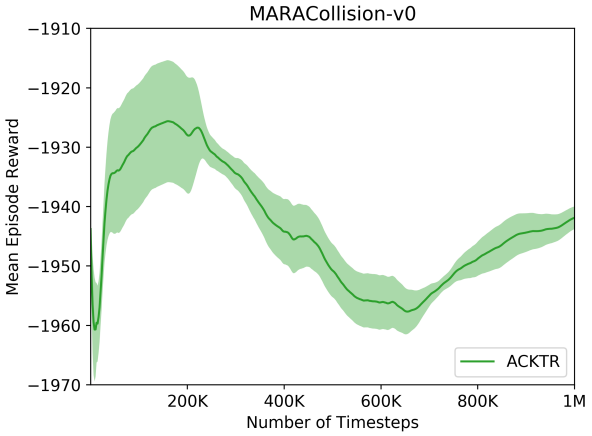

Figure 8: Performance of $MARACollision - v0$ environment trained with ACKTR algorithm

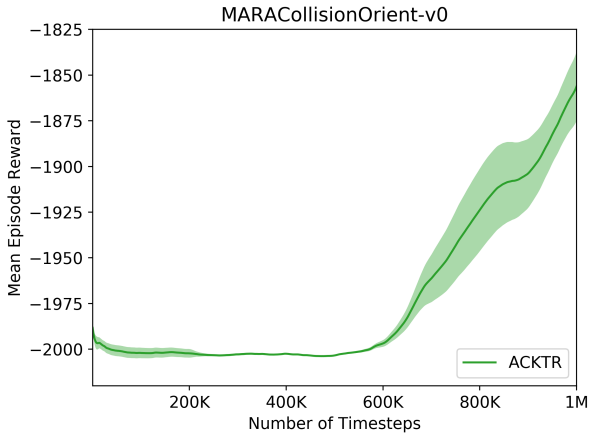

Figure 9: Performance of $MARACollisionOrient-v0$ environment trained with ACKTR algorithm

References

[1] V. Mayoral, R. Kojcev, A. Hernández, I. Zamalloa, and A. Bilbao, "Modular and self-adaptable (masa) strategy for building robots," in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE, 2018, pp. 90–95.

[2] I. Zamalloa, R. Kojcev, A. Hernández, I. Muguruza, L. Usategui, A. Bilbao, and V. Mayoral, "Dissecting robotics-historical overview and future perspectives," *arXiv preprint arXiv:1704.08617*, 2017.

[3] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *CoRR*, vol. abs/1603.02199, 2016. [Online]. Available: http://arxiv.org/abs/1603.02199

[4] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1–9.

[5] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[6] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1889–1897.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[8] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Advances in Neural Information Processing Systems*, 2017, pp. 5285–5294.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[10] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.

[11] T. Nogueira, S. Fratini, and K. Schilling, "Autonomously controlling flexible timelines: From domain-independent planning to robust execution," in *2017 IEEE Aerospace Conference*, March 2017, pp. 1–15.

[12] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo," *arXiv preprint arXiv:1608.05742*, 2016.

[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[14] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.

[15] P. Dhariwal, C. Hesse, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Openai baselines," https://github.com/openai/baselines, 2017.

[16] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, ser. NIPS'99. Cambridge, MA, USA: MIT Press, 1999, pp. 1057–1063. [Online]. Available: http://dl.acm.org/citation.cfm?id=3009657.3009806

[17] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[18] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[19] A. Ghadirzadeh, A. Maki, D. Kragic, and M. Björkman, "Deep predictive policy training using reinforcement learning," *CoRR*, vol. abs/1703.00727, 2017. [Online]. Available: http://arxiv.org/abs/1703.00727

[20] P. Dhariwal, C. Hesse, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Openai baselines," https://blog.openai.com/roboschool/, 2017.

[21] S. Barrett, M. E. Taylor, and P. Stone, "Transfer learning for reinforcement learning on a physical robot," in *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*, 2010.

[22] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," *CoRR*, vol. abs/1610.04286, 2016. [Online]. Available: http://arxiv.org/abs/1610.04286

[23] S. James and E. Johns, "3d simulation for robot arm control with deep q-learning," *CoRR*, vol. abs/1609.03759, 2016. [Online]. Available: http://arxiv.org/abs/1609.03759

[24] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3357–3364.

[25] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *arXiv preprint arXiv:1703.06907*, 2017.

[26] J. Peters, "Policy gradient methods," *Scholarpedia*, vol. 5, no. 11, p. 3698, 2010, revision #137199.

[27] S. Kakade, "A natural policy gradient," in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, ser. NIPS'01. Cambridge, MA, USA: MIT Press, 2001, pp. 1531–1538. [Online]. Available: http://dl.acm.org/citation.cfm?id=2980539.2980738

[28] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[29] S. J. Wright and J. Nocedal, "Numerical optimization," *Springer Science*, vol. 35, no. 67-68, p. 7, 1999.

[30] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[31] N. G. Lopez, Y. L. E. Nuin, E. B. Moral, L. U. S. Juan, A. S. Rueda, V. M. Vilches, and R. Kojcev, "gym-gazebo2, a toolkit for reinforcement learning using ros 2 and gazebo," 2019.

[32] Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 1087–1098. [Online]. Available: http://papers.nips.cc/paper/6709-one-shot-imitation-learning

[33] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[34] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," *CoRR*, vol. abs/1710.09767, 2017. [Online]. Available: http://arxiv.org/abs/1710.09767