INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078

(Volume 7, Issue 4 - V7I4-1884)
Available online at: https://www.ijariit.com

Training of a robot manipulator model by three reinforcement learning algorithms (TRPO, PPO2, and ACKTR), use cases, and comparative studies

Narilala Daniel Ranaivoson
narilala@gmail.com
ED-STII, Antananarivo, Madagascar

Soloniaina Rakotomiraho
rakmiraho@gmail.com
ED-STII, Antananarivo, Madagascar

Rivo Randriamaroson
rivomahandrisoa@gmail.com
ED-STII, Antananarivo, Madagascar

ABSTRACT

A model of the manipulator robot is available. It presents properties necessary for its integration in a reinforcement learning loop. This is the subject of the article [1].Three reinforcement learning algorithms are used, TRPO, PPO2 and ACKTR. The first two algorithms learn well, but for the third one the reward curve fails to find an increasing variation. Rather an erratic variation is observed. Experiments or use cases of the policy functions in controlling the robot were carried out. It is ok for the first two. The robot moves and reaches the targets presented after some iteration. For ACKTR there is no complete convergence but an oscillation around a fixed position.

Keyword : *Robot Modeling, Robot Manipulator, Ros, Reinforcement Learning, Trust Region Policy Optimization, Trust Region Policy Optimization, Actor Critic Using Kronecker-Factored Trust Region, Robot Control Reinforcement Learning, Robot-Reinforcement Learning Interface, Training, Use Case, Robot Environment, Task Environment, Reward Curve, Joint Values Curve, Error Curve.*

1. INTRODUCTION

Robot's model of the paper [1] is prerequisites for what follows. Here its use is focused on control using reinforcement learning approaches. Three algorithms are implemented for training the robot. The observations recorded during the training are reported in this paper. Afterwards, use cases are simulated and the evolutions of the values are recorded as well as the curves of variations on each joint of the robot.

2. ROBOT MODEL

The modeling is done with ROS. Its description is written in XML file in URDF format. An overview of the robot in Rviz is given by Fig -1.

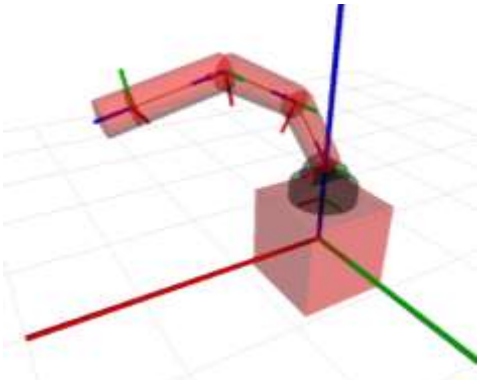

Fig -1: Visualization of the robot model

There are quantities variables needed in order to implement the model. Variables and values of the model implement cinematic, dynamic and geometric properties. Table -1 repertories link's variables and their respective values.

Table -1: Link's variables and values

Link #i	Mass	Inertial tensor	shape	Dimensions
0	1024	$I = \begin{bmatrix} 170.667 & 0 & 0 \\ 0 & 170.667 & 0 \\ 0 & 0 & 170.667 \end{bmatrix}$	cubic	edge=1
1	9.655	$I = \begin{bmatrix} 13.235 & 0 & 0 \\ 0 & 13.235 & 0 \\ 0 & 0 & 9.655 \end{bmatrix}$		
2	57.906	$I = \begin{bmatrix} 12.679 & 0 & 0 \\ 0 & 12.679 & 0 \\ 0 & 0 & 0.651 \end{bmatrix}$	cylindrical	radius=0.15 height=0.8
3	57.906	$I = \begin{bmatrix} 12.679 & 0 & 0 \\ 0 & 12.679 & 0 \\ 0 & 0 & 0.651 \end{bmatrix}$	cylindrical	radius=0.15 height=0.8
4	57.906	$I = \begin{bmatrix} 12.679 & 0 & 0 \\ 0 & 12.679 & 0 \\ 0 & 0 & 0.651 \end{bmatrix}$	cylindrical	radius=0.15 height=0.8
5	18.056	$I = \begin{bmatrix} 0.479 & 0 & 0 \\ 0 & 0.479 & 0 \\ 0 & 0 & 0.204 \end{bmatrix}$	cylindrical	radius=0.15 height=0.25

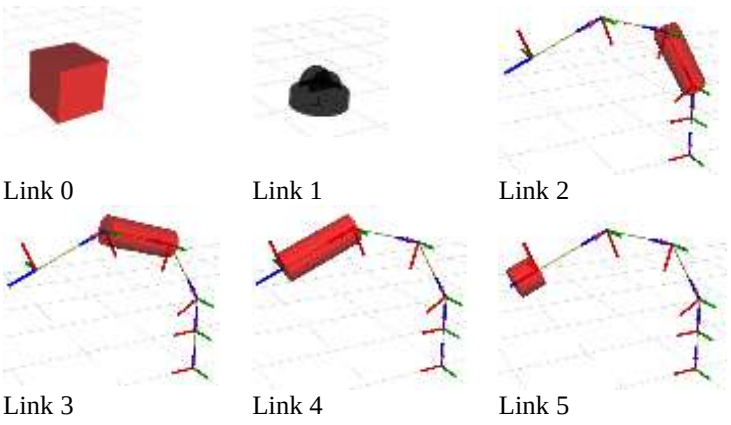

Fig -2: Visualization of individual link shape

The images of each articulation are shown in Fig -2. Local frames and relative position vectors are represented to illustrate relative joint positions. The articulations number 2, 3 and 4 are geometrically identical. Table -5 shows that their dynamic properties are also identical. i.e. they have the same inertia tensor, the same mass, the same dimensions and shape.

PID gains and the types of controllers and the joints on which they work must be specified in a YAML file. Table -2 is summarizes this information:

Table -2: PID implementation on the model

YAML code	PID parameters
	Joint #**1** $K_p, K_i, K_d = [2000 \quad 100 \quad 500]$
	Joint #**2** $K_p, K_i, K_d = [50000 \quad 100 \quad 2000]$
	Joint #**3** $K_p, K_i, K_d = [20000 \quad 50 \quad 1000]$
	Joint #**4** $K_p, K_i, K_d = [2000 \quad 50 \quad 200]$

`# Publish all joint states ----------`
`joint_state_controller:`
`  type: joint_state_controller/JointStateController`
`  publish_rate: 50`
 	
`# Position Controllers ----------`
`joint1_position_controller:`
`  type: effort_controllers/JointPositionController`
`  joint: base_link__link_01`
`  pid: {p: 2000.0, i: 100, d: 500.0}`
`joint2_position_controller:`
`  type: effort_controllers/JointPositionController`
`  joint: link_01__link_02`
`  pid: {p: 50000.0, i: 100, d: 2000.0}`
`joint3_position_controller:`
`  type: effort_controllers/JointPositionController`
`  joint: link_02__link_03`
`  pid: {p: 20000.0, i: 50, d: 1000.0}`
`joint4_position_controller:`
`  type: effort_controllers/JointPositionController`
`  joint: link_03__link_04`
`  pid: {p: 2000.0, i: 50, d: 200.0}`
`joint5_position_controller:`
`  type: effort_controllers/JointPositionController`
`  joint: link_04__link_05`
`  pid: {p: 700.0, i: 50, d: 70.0}`
	Joint #**5**
	$K_p, K_i, K_d = [700 \quad 50 \quad 70]$

ROS works with various module of program. The module called "JointStateController" returns information from joint positions. And the "JointPositionController" module receives the input controls for the robot's model.

The conclusions of the paper [1] are:
The robot model is functional and has fairly realistic kinematic and dynamic behaviors. The controllers work correctly. We are rather confident about its use in the rest of our work. This model will then be used to study commands using reinforcement learning algorithms. We think that this paper presents a way to model a robot, which will not necessarily be a manipulator.

3. TRAINING

The robot model is integrated in a specific reinforcement learning environment. The algorithm and the robot are linked through standardized interfaces. Thus the algorithms are interchangeable at will. Three algorithms are used:
- TRPO
- PPO
- ACKTR

3.1 Training steps

Inside the training loop the following steps are carried out:
1) Perform an action
2) Take an observation from the environment
3) Check if a collision has occurred or not
4) Calculate a reward
5) Return a status informing if the current episode ends or not
6) Reset the agent according to the status of step 5

3.2 Programming

In short, it is about writing the training script and coding the training environment. The training script is the file where the reinforcement learning algorithm is specified and then initialized. For our purposes, we will use the three files in Fig -2.

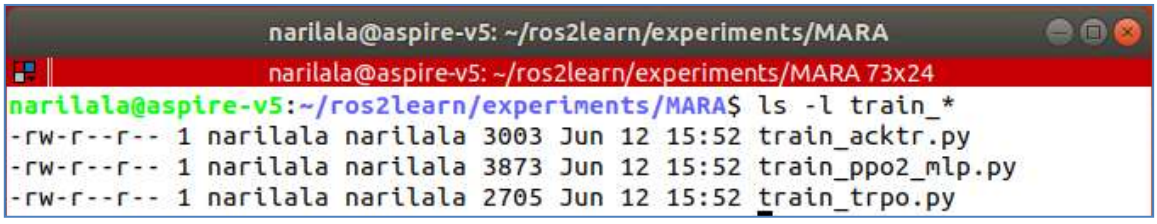

Fig -2: List of training scripts

The three files correspond to the three algorithms TRPO, PPO and ACKTR used for the studies of the application of reinforcement learning of the manipulator robot.

3.3 Training environment

First the most basic environment is modeled. It is an abstraction of the basic concepts of reinforcement learning. From this basic object, the other types of environment are derived by inheritance as shown in Fig -3.

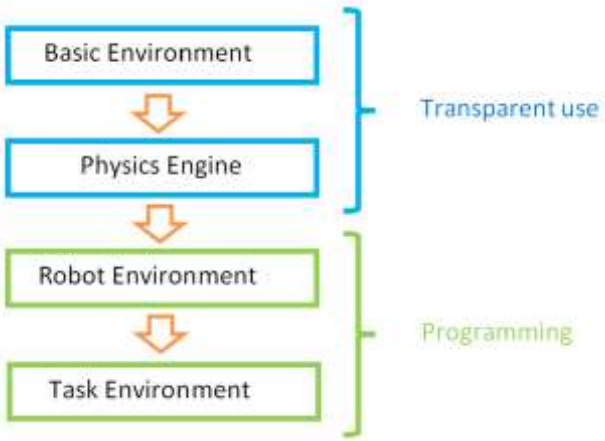

Fig -3: Hierarchy of environment modeling classes

3.4 Basic environment

This is a model of the general characteristics common to reinforcement learning. Fig -4 is an illustration of these general characteristics.

Fig -4: Agent-environment interaction

This class encapsulates models corresponding to functionalities. An object named env corresponds to the environment in Fig -4. Then *env* implements a function *env.step()*. This function returns the following objects :
. *observation* representing the observations on the environment,
. *reward* : a real number representing the reward obtained,
. *done* a boolean variable indicating the end of an episode,
. *info* a dictionary of information for debugging.

The *action_space* and *observation_space* objects are containers. They contain all possible actions in the case of discrete sets. In the continuous domain, they are functions that return an action depending on the observation. In our case, it is a neural network.

3.5 Physics engine

It is an object that has transparent functionality to the user. It inherits the basic environment. Then, it adds functionalities allowing a simulation according to the laws of classical mechanics, thus offering a realistic environment similar to the working conditions of a robot.

3.6 Robot environment

Inherited from the physics engine environment, the Robot environment implements the specificities of the robot object of reinforcement learning. It contains all the functionalities that allow controlling the robot from a command signal. Each joint of the robot can be solicited. Each possible combination of commands constitutes an action that the robot and its environment perform.

3.7 Task environment

The task environment contains the contexts of the specific task that the robot has to learn. It depends on the task and the robot. There are then two cases:
For the same robot but with a different task. It is in the task environment that the modifications will be made. The training can be done without modification in the robot environment.

For the same task but with a different robot. Modifications may be made on this class if the new robot does not have the same interfaces as the old robot.

3.8 Training with TRPO

The goal of training is to produce the policy function that would be the most optimal for the task at hand. This policy function is realized with a neural network. The characteristics of the machine used are as follows
Processor: Intel Core i3, 4th generation

RAM: 4GB
Hard disk: 500Go

Training time: 1 day, 1 hour, 15 minutes and 25 seconds.

The neural network used to model the policy function has the following characteristics. The number of internal layers is given by the parameter *num_layers*. Here its value is 2. And the parameter *num_hidden* has the value 64. So we have a neural network made of 2 internal layers each containing 64 neurons.

Fig -5 below gives the evolution of the rewards obtained during the whole learning process. A progressive increase in reward is observed. The algorithm is stable and the robot evolves positively and acquires a better and better policy over time.

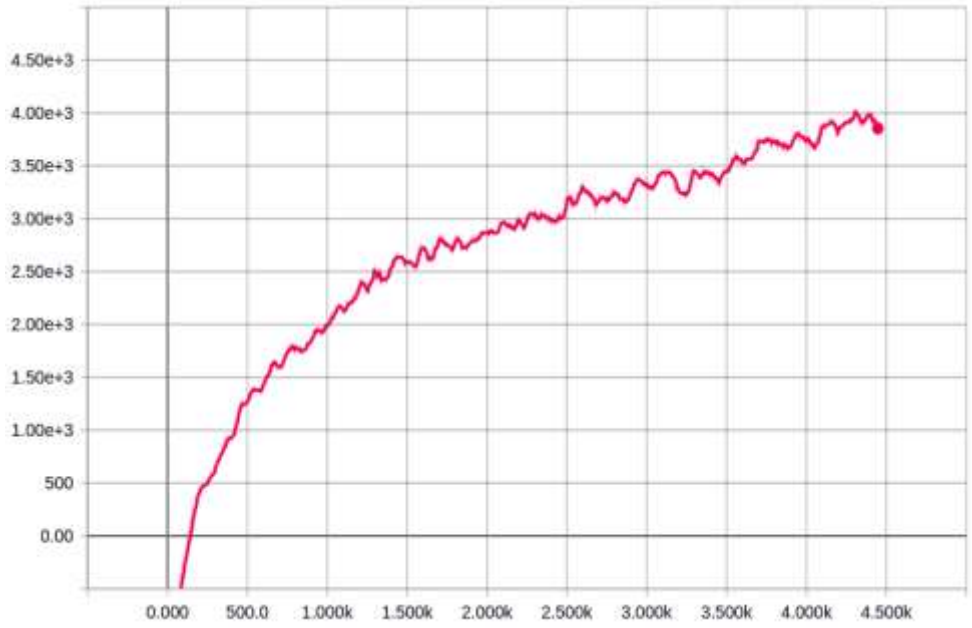

Fig -5: Evolution curve of the reward returned by the environment for TRPO

3.9 Training with PPO2

Training time: 23 hours, 27 minutes and 27 seconds
The neural network modeling the policy is similar to the one used with the TRPO algorithm. The neural network used to model the policy function has the following characteristics:
num_layers = 2
num_hidden = 16.
It is a network with two internal layers with 16 neurons each.
A curve similar to the previous algorithm is obtained as shown in Fig -6 below.

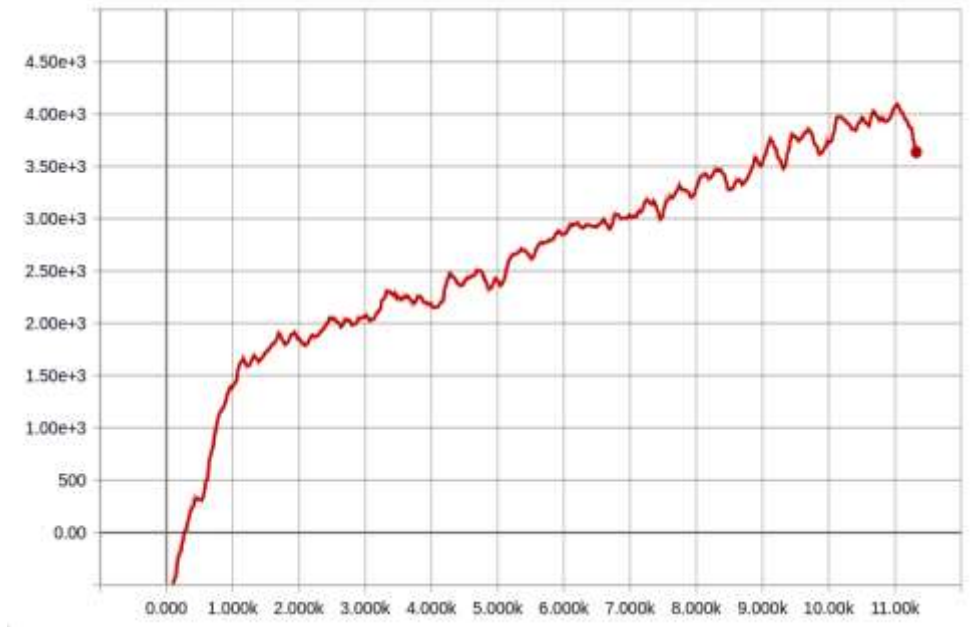

Fig -6: Evolution curve of the reward returned by the environment for PPO2

3.10 Training with ACKTR

What distinguishes this algorithm is that it uses a technique called Kronecker-factored Approximate Curevature. This is a method that allows, among other things, to design neural networks that more efficiently model the dynamic gradient descent learning algorithm.

Learning time: 23 hours, 36 minutes and 18 seconds

Hyperparameters are: number of layers 2, number of hidden neurons 64.

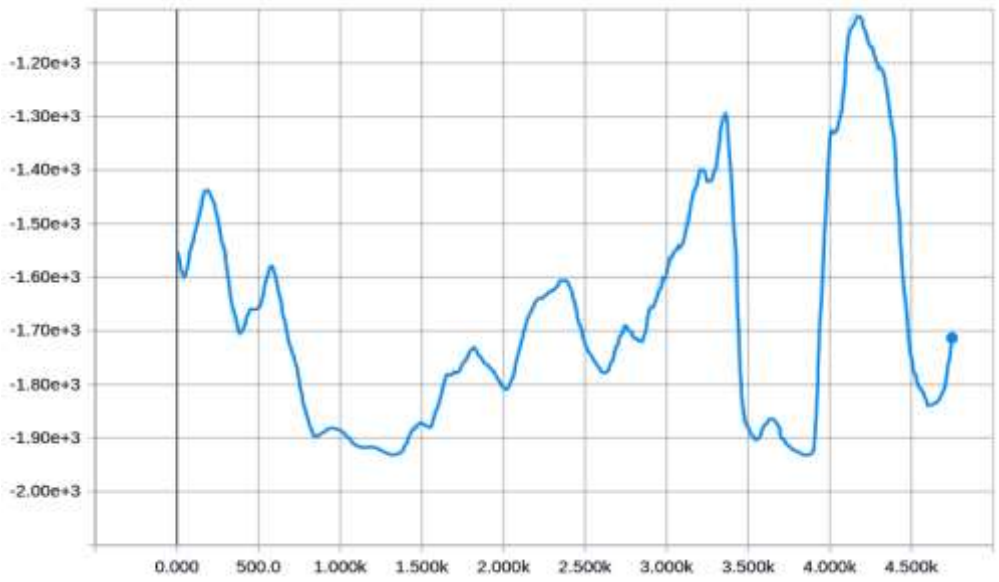

Fig -7: Evolution curve of the reward returned by the environment for ACKTR

The learning has failed. The reward variation curve is always below zero. It does not show any positive value. Its variation is quite random for a long learning time, while this curve is recorded for more than 4500 iterations. The policy never found the right direction of increasing reward to accumulate positive values. The robot will not reach its goal.

3.11 Comparisons and discussions

Characteristic points:

To compare the results of the three training, characteristic information of the reward variation curves was noted. They are shown in Table -3 below.

Table -3: Three characteristic points of the three experiments

	TRPO	PPO2	ACKTR
Simulation time (minutes)	1515	1407	1416
Number of iterations	4446	11320	4749
Maximum value	3830	3571	-1697

Algorithm Speeds:

From the numerical values in Table -3, the relative speeds of each algorithm during learning can be deduced. The simulations were run on a single computer to allow performance comparisons. Table -4 compares these relative speeds. PPO2 is the fastest.

Table -4: Relative learning speeds

Algorithm	Number of cycles per minute	rank
TRPO	2.93465347	3
PPO2	8.04548685	1
ACKTR	3.35381356	2

Convergences:

The algorithm converges well when it accumulates positive values for each iteration. The larger the numerical value, the better the performance of the algorithm. On Table -5 we observe that TRPO and PPO2 converge well with similar values. TRPO is slightly ahead. But for ACKTR there is no convergence at all.

Table -5: Relative Convergences

Algorithm	Final value of the reward	Converge
TRPO	3830	YES
PPO2	3571	YES
ACKTR	-1697	NO

4. USE CASES

After the training stage, we move on to the use stage. We have two points of view. Observations with respect to the agent and the one with respect to the robot.

4.1 Data structure

The agent evolves under the direction of his political function. Fig -8 shows the flow of information during execution. The policy function is here already well fixed during learning. The policy presents an action to be executed by the environment. In turn, the latter returns the reward and a new observation. The reward is a scalar, the observation is a vector. The latter represents the current state in which the agent is. The new observation is used by the policy to move the agent and the robot in a new configuration getting closer and closer to the goal.

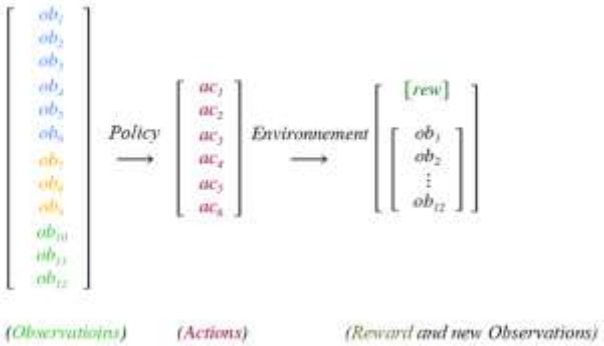

Fig -8 : Information flow at the agent level during the execution

The observation vector is illustrated in Fig -9. It is formed by two types of information. This information is provided by the environment and describes the state of the system.

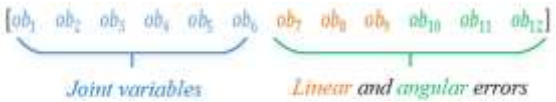

Fig -9 : Constitution of the observation vector

The action vector is formed by the six instructions to be presented at the six joints of the robot. The policy function is a neural network that recommends an action according to the state of the system. This function works in two continuous spaces. These are the spaces of states and actions. The environment is constituted by the model of the robot itself and the laws of mechanics. The behavior of the robot is a resultant of internal and external influences.

4.2 Evolution of the reward during use cases

For the two algorithms that converge (Fig -10 and Fig -11) the curves show a relatively sustained increase.

The robot reaches its final position after a few dozen intermediate positions. The number of iterations is 50 for TRPO and 70 for PPO. TRPO arrives faster than PPO but it goes through a local minimal during its evolution. This is the risk run by this algorithm. To mitigate this risk PPO limits the updates of the policy parameters during training.

The number of iterations is higher for PPO because it is an algorithm that limits excessive variations on the parameters of its policy function. But, thanks to this, its curve has a sustained increase. This way the learning process is more secure and confident even if it is longer.

For the ACKTR algorithm, the curve has only negative values. Moreover, it shows erratic variations. The curve in Fig -12 was recorded for an excessively long time for one run. After a number of iterations of 700, the agent still does not recover satisfactory rewards. There is no convergence and the learning has not been successful. The policy function is not exploitable.

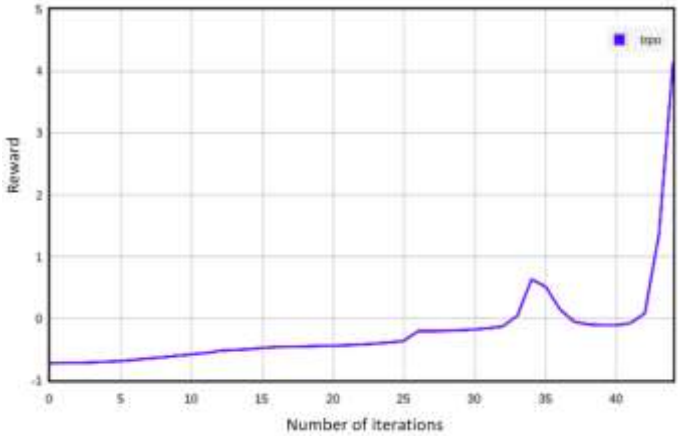

Fig -10 : Evolution curve of the reward returned by the environment for TRPO

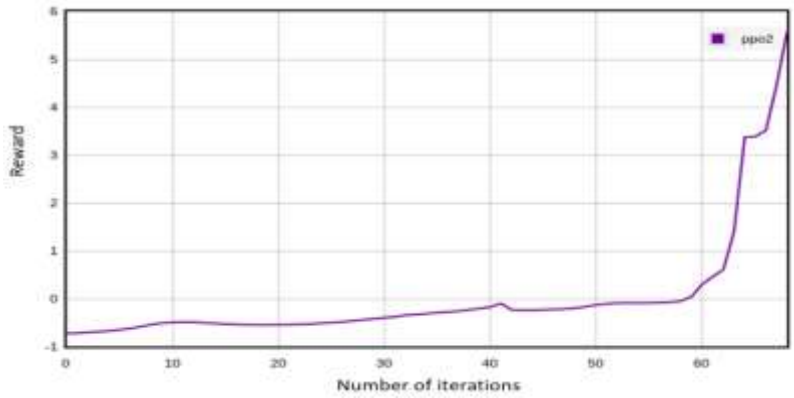

Fig -11: PPO reward curve for execution

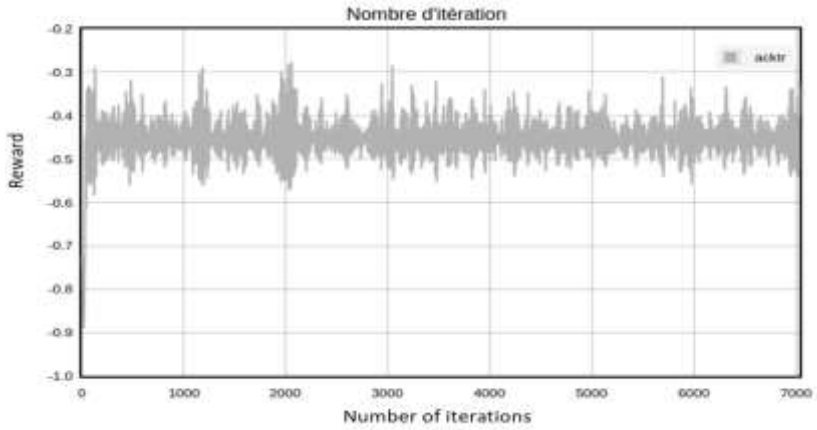

Fig -12: ACKTR reward curve for execution

4.3 Data collected from the robot

At the robot level, each action computed by the neural network of the policy function translates into an instruction for the controller at each joint. The policy function seeks to obtain a positive reward at each iteration. In doing so, it makes the robot move, so as to decrease the distance between the objective point and the current point where the robot is located. The evolution of this error allows to appreciate the evolution of the system in its search for a solution. At the same time as the monitoring of the error, it is also possible to follow directly on the robot the variations at the joints. Thus, the two types of observation made on the robot are :

- Positional error
- Joint values

Positional error:

For the two algorithms that converge, we have the curves of Fig -13 and 14. The values for each x, y, and z axis vary gradually to finally reach zero values. The Cartesian distance $\sqrt{ex^2 + ey^2 + ez^2}$ decreases and cancels at the end. The evolution of this distance is given by the black colored curves. For the ACKTR algorithm, the errors on the axes as well as the Cartesian distance do not tend towards zero, even in the long run. The curves of Fig -15 give the evolution of these errors until about 7000 iterations. We can say that the curves do not converge to zero but oscillate around the fixed values. Fig -16 shows the same curves but in the shorter term. On this last one, we have the errors until the end of 250 iterations. From around the 50th iteration the errors rotate around the values:

$$[ex \quad ey \quad ez] = [0.2 \quad 0 \quad 0.05]$$

This observation would lead to the conclusion that the algorithm is not at fault, but the problem could be in its adaptation to the robot.

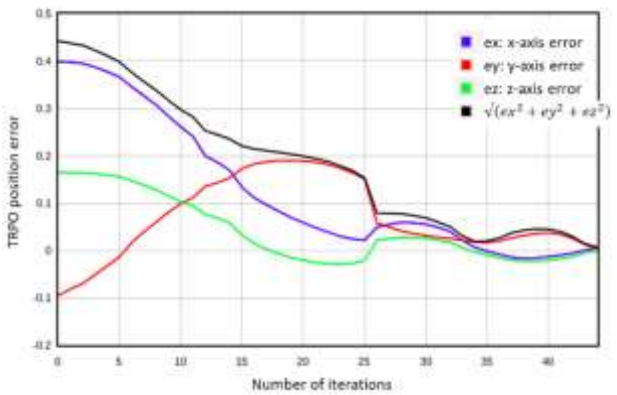

Fig -13: Evolution of positon errors with TRPO for the execution

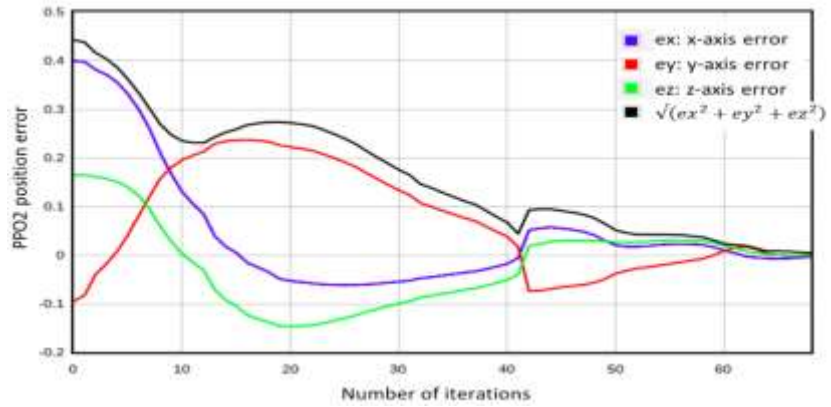

Fig -14 : Evolution of position errors with PPO2 for the execution

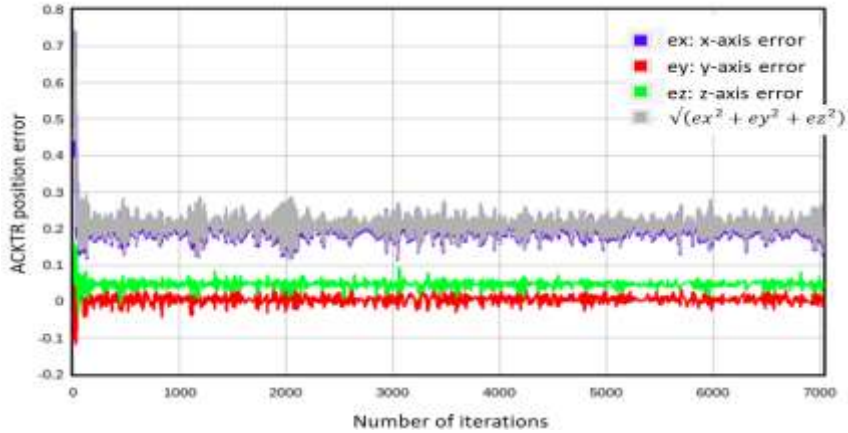

Fig -15 : Evolution of the position errors with ACKTR for the execution

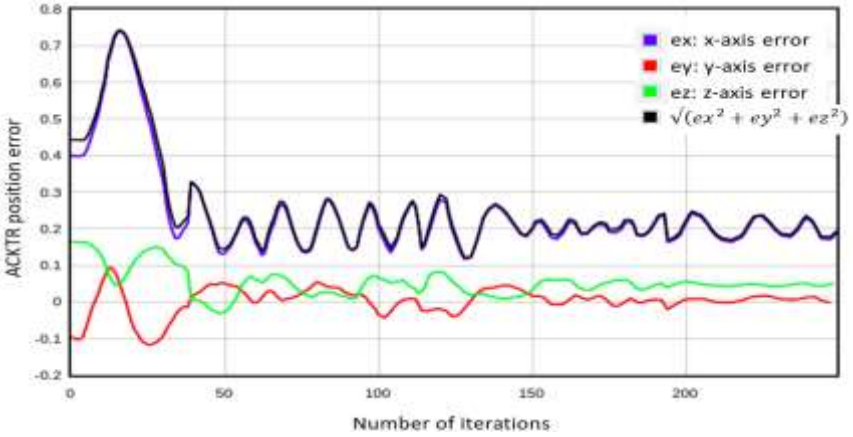

Fig -16 : Evolution of the position errors with ACKTR for the execution

Joint values:
The second class of observations on the robot focuses directly on the rotation angles of the joints.

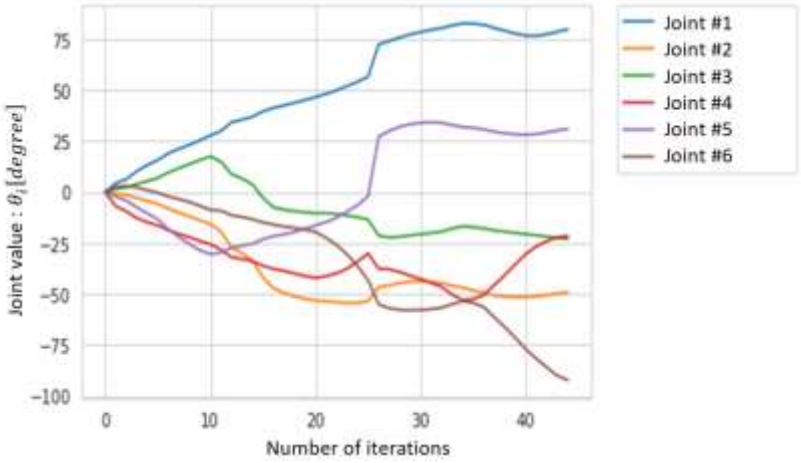

Fig -17: Evolution of joint angles with TRPO

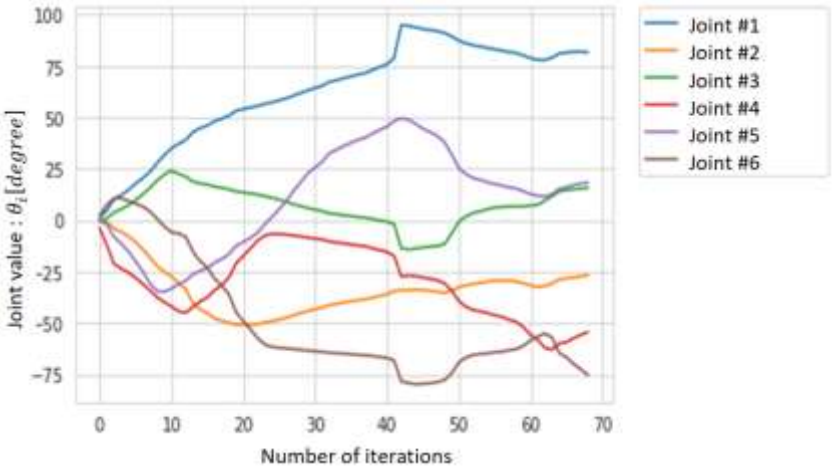

Fig -18: Evolution of joint angles with PPO

At each iteration, each joint changes its angle around its axis of rotation. The instructions come from the policy function. At each iteration, it returns a vector with six components. Each component of the action vector corresponds to a joint on the robot. Thus the robot moves in the space of possible states. The environment returns a reward and a new observation. Fig -17 and Fig -18 show the variation curves for TRPO and PPO.

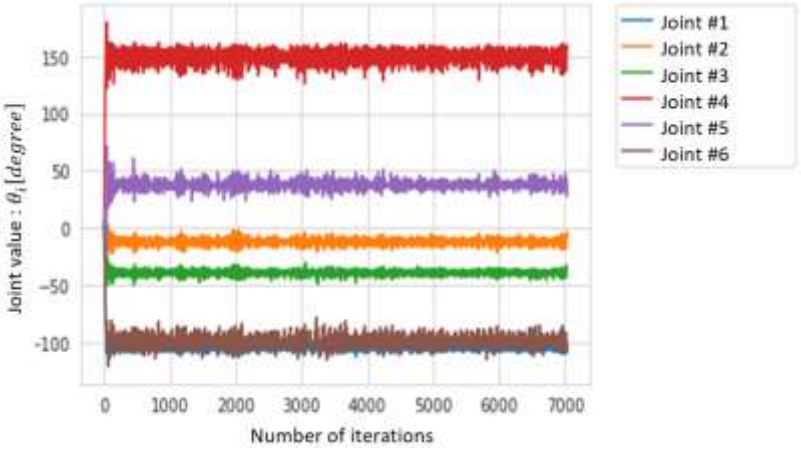

Fig -19: Evolution of joint angles with ACKTR

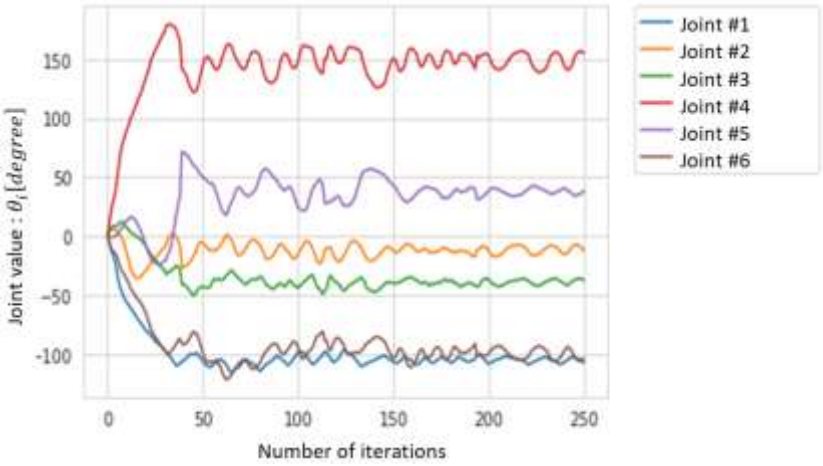

Fig -20: Evolution of joint angles with ACKTR for 250 iterations

For ACKTR, the joints never stabilize. But in the long run, we observe that each joint moves back and forth around a fixed position.

4.4 Comparison for use cases

For the use cases, the characteristic data on the reward curves are:

Table -6: Characteristic points on the reward curves

	TRPO	PPO2	ACKTR
Number of iterations	45	69	7031
Maximum value	4.1565	5.5703	-0.2770

TRPO converges faster with 45 iterations but at a lower maximum value than PPO2. The latter has the merit of presenting a sustained increasing curve despite a relatively low speed compared to TRPO. The ACKTR algorithm does not converge. The first two algorithms have relatively short execution times and stop because they have reached their objectives. But, the third one is caught in an endless loop since the output conditions are never fulfilled. The robot oscillates without finding a final resting position.

5. CONCLUSIONS

TRPO, PPO2 and ACKTR algorithms were taken one by one to train the agent. The first two produced stable and convergent policy functions. The third one produces a reward curve that fails to exceed the zero thresholds. Apparently the algorithm does not work, but the execution on the robot gives indications. Indeed it seems that it is not the algorithm that does not work but the coupling with the robot that destabilizes the system. This observation opens the way to another hypothetical research topic. Use of commands, i.e. trained political functions allows to draw conclusion. The concept of control by reinforcement learning a robot works. It seems that we have something concrete that works. The results obtained are comparable to inverse kinematic controls. For a new approach of Machine Learning type control, we find that it is a start that can lead to a new way of controlling a robot

6. REFERENCES

[1] Narilala Daniel Ranaivoson, Soloniaina Rakotomiraho, “A Modeling approach of manipulator robot to use in reinforcement learning control”, https://www.ijariit.com/, paper id 130800, November 2020.
[2] Dhariwal, Prafulla, Hesse, Christopher, Klimov, Oleg, Nichol, Alex, Plappert, Matthias a,Radford, Alec ,Schulman, John, Sidor, Szymon, Wu, Yuhuai, Zhokhov, Peter, “OpenAI Baselines”, GitHub, https://github.com/openai/baselines, 2017.
[3] Zamora, Iker, Lopez, N. Gonzalez, Vilches, V. Mayoral, Cordero, A. Hernandez, “Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo”, 2016.
[4] N. G. Lopez, Y. L. E. Nuin, E. B. Moral, L. U. S. Juan, A. S. Rueda, V. M. Vilche, R. Kojcev, ”gym-gazebo2, a toolkit for reinforcement learning using ROS 2 and Gazebo”, arXiv:1903.06278, 2019.
[5] Y. L. E. Nuin, N. G. Lopez, E. B. Moral, L. U. San Juan, A. S. Rueda, V. M. Vilches, R. Kojcev, “ROS2Learn: a reinforcement learning framework for ROS 2”, arXiv:1903.06282, 2019.
[6] J. C. Rocholl, F. Xicluna, I. Lee, “pep8 documentation , Release 1.7.1.dev0”, buildmedia.readthedocs.org, 2016.
[7] Baseline team, “stable-baselines”, https://github.com/hill-a/stable-baselines, 2019.
[8] AcutronicRobotics , “AcutronicRobotics/baselines”, https://github.com/AcutronicRobotics/baselines, 2019.
[9] J. Schulman, S. Levine, P. Moritz, M. Jordan, P. Abbeel, “Trust Region Policy Optimization”, https://arxiv.org/pdf/1502.05477.pdf, 2017.
[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, “Proximal Policy Optimization Algorithms”, https://arxiv.org/pdf/1707.06347.pdf, 2017.
[11] Y. Wu, S. Liao, J. Ba, R. Grosse, E. Mansimov, “Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation”, University of Toronto Vector Institute, New York University, https://arxiv.org/pdf/1708.05144.pdf, 2017.
[12] Kullback, Leibler, «Kullback–Leibler divergence», wikipedia, https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence, 2019.
[13] Scholarpedia,“Policy gradient methods”, http://www.scholarpedia.org/article/Policy_gradient_methods,scholarpedia, 2019.
[14] J. Achiam, “Advanced Policy Gradient Methods”, UC Berkeley, http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_13_advanced_pg.pdf, 2017.
[15] S. Ruder, “An overview of gradient descent optimization Algorithms”, Insight Centre for Data Analytics, NUI Galway Aylien Ltd., Dublin, https://arxiv.org/abs/1609.04747, 2017.
[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, “OpenAI Gym”, OpenAI, http://arxiv.org/abs/1606.01540, 2016.
[17] Acutronic Robotics Team, “Acutronic Robotics documentation”, Acutronic Robotics Docs, https://acutronicrobotics.com/docs/technology/h-ros/api/level2/algorithms, 2019.

