Control of inverted pendulum with Reinforcement Learning

Bernardo T. Barata, Christian Garcia, Fredrik Johansson Torneus, Jonathan Jansson

***Abstract*—**

This report formulates a reinforcement learning based solution to balance an inverted 2-Degrees of freedom pendulum in the upright position. The aim is to analyze the potential of Reinforcement Learning algorithms in comparison to modern control theory. More specifically the analysis is made in comparison to a data-driven controller. The report illustrates the advantages of Reinforcement Learning in comparison, e.g. that it can be simulated with a non-linear model. The results show a promising behaviour for the reinforcement learning based controller in comparison to the data-driven controller.

I. Introduction

The area of Reinforcement Learning (RL) is showing potential for new applications every day. A recent example is how Google Deepmind developed the reinforcement based program AlphaGo which managed to beat one of the best human players at the game Go [1]. This report aims to investigate the potential of implementing reinforcement learning algorithms in the area of control theory.

The setting for the project is a model of a 2 Degrees of Freedom (2DoF) Inverted pendulum, for which tuned state-of-the art analytic controllers already exist. Therefore there are measurements of performance with which an RL solution of the model can compare.

A. Related Work

Previous work on 1DoF pendulums has been studied and is what inspired this project.

The paper in [2] uses a 1DoF Inverted Pendulum and trains directly on a hardware setup with visual inputs only. From this one can draw the conclusion that training on hardware is time-consuming and that it is seemingly reasonable to initially train networks in simulation even for implementations for physical systems.

In [3] a Q-Learning, an Actor-critic policy gradient method and a Temporal Difference (TD) with Value Function Approximation method is trained to balance a 1DoF inverted pendulum. These are then compared to an LQR controller which solves the same problem. Their results show a significant difference in performance between the algorithms with TD with Value Function Approximation outperforming both the other algorithms significantly and reaching oscillations between 0.3 and -0.4 degrees after only 19 episodes of training. Moreover, their performance comparison to LQR shows that the RL implementation reaches similar performance to the control theory based solution.

II. Background

This section will describe the basics of RL, introduce the system model of the Quanser 2DoF Inverted Pendulum rig used for the project and describe the Runge-Kutta of order two (RK2) method used for simulating the motion of the pendulum.

A. Reinforcement learning

"Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal." [4, p.1].

This can be translated into saying that an agent needs to learn how to respond to different states, or observations, in the best possible way. The goal of the agent is to maximize the accumulated reward gained. The actions of the agent are therefore heavily influenced by a defined reward function. Without a properly defined reward function, which reflects the desired behaviour, a satisfactory result will be difficult to obtain.

The reward is fed along with an observation to the agent which decides the next action. One problem that is always present is the decision to either explore or exploit. Exploration is connected to taking actions leading to parts of the state space that have not been visited before, this can lead to finding paths through the state space which give more reward than previously obtained. On the other hand, when an action for a specific state - which gives a good reward - is found, it should be taken. This is called exploiting i.e. to use the existing knowledge of what actions are good to take [4].

B. System model

For this project a model of the Quanser 2DoF Inverted pendulum rig was used for simulation. The rig is illustrated in Fig. 1. The action space, or model inputs, to the rig consists of the voltages that drive the motors x and y. The motors supply a torque that rotates a rod connected to it in the horizontal plane by an angle θ_x and θ_y respectively. This angular movement of both rods dictates the position of the pendulum in both the X and Y direction.

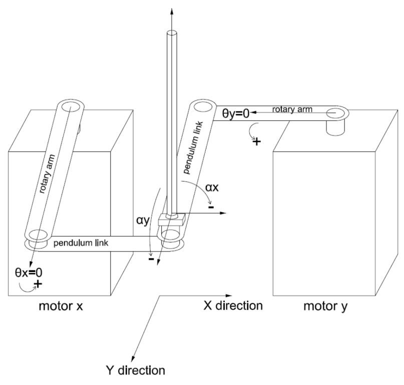

Fig. 1. Quanser 2DoF Inverted Pendulum

The pendulum itself can tilt in a combination of the defined X and Y directions, the angle it forms from the upright position to either direction is referred to as α_x and α_y. The model outputs, which is also called the observation space is made up of θ_x, θ_y, α_x and α_y.

Since there isn't much deviation in position, considering the area of space that the pendulum can move within, then the coupling in the dynamics of two directions is negligible in the model [5, p.4]. As such, the pendulum is simply modelled as two 1DoF pendulums - one moving in the X direction and the other in Y. The equations of motion for the 1DoF pendulum are, therefore [5]:

$$\begin{aligned}\ddot{\boldsymbol{\theta}}(t) = &\frac{-\frac{1}{2}((4M_pL_p^2\alpha(t)\dot{\theta}(t)\dot{\alpha}(t)-}{((4J_r+4M_pL_r^2)J_p+M_pL_p^2J_r)}\\ &\frac{8C_oV_m(t)+8D_r\dot{\theta}(t))J_p+M_p^2L_p^4\alpha(t)\dot{\theta}(t)\dot{\alpha}(t)}{((4J_r+4M_pL_r^2)J_p+M_pL_p^2J_r)}\\ &-\frac{-\frac{1}{2}((M_p^2L_p^3L_r\dot{\theta}^2(t)+20M_p^2L_p^2L_rg)\alpha(t)-}{((4J_r+4M_pL_r^2)J_p+M_pL_p^2J_r)}\\ &\frac{2M_pL_p^2D_r\dot{\theta}(t)+2M_pL_p^2C_oV_m(t))}{((4J_r+4M_pL_r^2)J_p+M_pL_p^2J_r)}\end{aligned} \tag{1}$$

$$\begin{aligned}\ddot{\boldsymbol{\alpha}}(t) = &\frac{((M_p^2L_p^2L_r^2+M_pL_p^2J_r)\dot{\theta}^2(t)+}{((4J_r+4M_pL_r^2)J_p+M_pL_p^2J_r)}\\ &\frac{20J_rM_pL_pg+20M_p^2L_r^2L_pg)\alpha(t)}{((4J_r+4M_pL_r^2)J_p+M_pL_p^2J_r)}\\ &+\frac{2M_pL_rL_pC_oV_m(t)-2M_pL_rL_pD_r\dot{\theta}(t)-}{((4J_r+4M_pL_r^2)J_p+M_pL_p^2J_r)}\\ &\frac{M_p^2L_p^3L_r\alpha(t)\dot{\theta}(t)\dot{\alpha}(t)}{((4J_r+4M_pL_r^2)J_p+M_pL_p^2J_r)}.\end{aligned} \tag{2}$$

Since equations (1) and (2) are identical to both 1DoF pendulums (and together represent the 2DoF pendulum), then the system states are defined as

$$x = [\theta_x, \alpha_x, \dot{\theta_x}, \dot{\alpha_x}, \theta_y, \alpha_y, \dot{\theta_y}, \dot{\alpha_y}] \tag{3}$$

The constants M_p, L_r, L_p, J_r, J_p, D_r, C_o and g are defined in Table I and V_m is the input voltage.

TABLE I
TABLE OF CONSTANTS - EQUATIONS OF MOTION [5, SEC. 3.1]

Parameters	**Description**	**Value**
M_p	Pendulum mass with T-fitting	0.1270 kg
L_r	Length of rotary arm	0.1270 m
L_p	Length of the pendulum (/w T-fitting)	0.3111 m
J_r	Equivalent inertia with the 4-bar linkage	0.0083 $kg.m^2$
J_p	Pendulum inertia around CoG	0.0012 $kg.m^2$
D_r	Arm viscous damping coefficient	0.0690 $N.m.s/rad$
C_o	Voltage convert coefficient	0.1285 $N.m.s/rad$
g	Gravitational constant	0.981 $kg.s^2$

One of the big advantages of utilizing Reinforcement Learning is that it does not require a linear system model to operate. Thus, the simulation of the pendulum was performed directly with the the non-linear model.

C. Runge-Kutta simulation

In order to increase the accuracy of the simulation of motion of the 1Dof and 2Dof inverted pendulum further the RK2 method was used. The steps for calculating the new state can be explained by equations (4).

$$x_{i+1} = x_i + (A_1 \cdot k_1 + A_2 \cdot k2)\, h \tag{4a}$$

$$k_1 = h \cdot f(x_i, t_i) \tag{4b}$$

$$k_2 = hf\,(t_i + Bh,\, x_i + Qk_1h) \tag{4c}$$

For this project the midpoint method has been used and as such $A_1 = 0$, $A_2 = 1$, $B = \frac{1}{2}$ and $Q = \frac{1}{2}$, [6].

III. Method

A. Learning Algorithms

This report intends to investigate two different reinforcement learning algorithms for training an agent to control the pendulum. The first algorithm is Double Deep Q-Network (DDQN), which is based on Dynamic Programming. The second algorithm is Proximal Policy Optimization (PPO), which is a Policy Optimization method. The two algorithms differ in a few aspects. First of all in their approach on how to solve the optimization problem, and secondly that DDQN needs to discretize the actions space. Hence it is interesting to investigate the difference in performance for the application of control.

Q-Learning is a common reinforcement learning algorithm which focuses on optimizing the action value function $Q(s, a)$ in reference to all actions a and states s by using the bellman equation [7]. In order to do this it uses an ϵ-greedy policy which chooses the best action a with ϵ probability and otherwise it explores the state space by taking random actions.

Given a large and/or continuous state space and/or action space when training with Q-Learning one might never explore the complete state space and thus might never find the optimal solution. DDQN is one of many neural network based reinforcement learning algorithms and approaches the problem with large state/action spaces by estimating the action value function Q with a neural network [8].

DDQN updates the network parameters θ, whose purpose is to capture the properties of the otherwise vast state space, based on the Loss function L defined as

$$L(\theta_t) = \mathbb{E}[(Y_t^{DDQN} - Q(s, a; \theta_t))^2] \tag{5}$$

, where Y_t^{DDQN} is the target and θ_t is the online network parameters [9, Sec. 2.1].

The target is computed according to

$$\begin{aligned} Y_t^{DDQN} &= R_{t+1} + \\ \gamma Q(s_{t+1}, &\arg\max_a Q(s_{t+1}, a; \theta_t); \theta_t^-) \end{aligned} \tag{6}$$

, where θ_t^- is the target network parameters and R$_{t+1}$ is the direct reward for reaching a specific state. Meanwhile, γ is the discount factor which weighs in the expected future reward from the action value function Q. During training the online network is updated continuously while the target network is updated with the parameters of the online network periodically [8, p. 4]. This aims to reduce the chance of overestimating a certain path through the state space.

PPO, on the other hand, is a policy optimization method and instead of the action value function it replaces its policy π with a neural network. It estimates the advantage $\hat{A}$, which is the accumulated and discounted state values $V(s_t)$ to give an estimate of how good an action was [10, Sec. 2.2]. PPO calculates its policy gradients based on a loss function L based on the updated and prior policy network parameters θ. The loss function is defined as

$$\begin{aligned} L_t^{CLIP+VF+S}(\theta) &= \\ \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) &+ c_2 S[\pi_\theta](s_t)] \end{aligned} \tag{7}$$

, which is the empirical entropy of a weighted sum of three different components where $c_1,\ c_2$ represent the weights and the third component S denotes an entropy bonus.

The two other components can be more explicitly described as in

$$\begin{aligned} L^{CLIP}(\theta) &= \\ \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, &clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \\ L_t^{VF} &= (V_\theta(s_t) - V_t^{targ})^2 \end{aligned} \tag{8}$$

and

$$r_t(\theta) = \frac{\pi_\theta(a_t, s_t)}{\pi_{\theta_{old}}(a_t, s_t)} \tag{9}$$

, where $r_t(\theta)$ is the ratio of probability between the action probability before and after the policy network is updated [10, Sec. 3].

One important characteristic of PPO is that the Loss function L^{CLIP} is estimated on the clipped ratio of probability as seen in equation (7) [10, Sec. 3]. This clips the partial loss based on the ϵ, thus keeping the gradients fairly low and restricts the network from changing very much from one update to another.

1) Terminal states: While training the model a set of terminal states is defined, mostly to preserve safety in the real pendulum environment. During training, reaching these states acts as a termination of an episode while exploring so as to not gather training data in unreasonable areas of the state space, e.g. when the pendulum has fallen. These states are defined as:

$$\theta_{Tr} = \pm 20^\circ,\ \alpha_{Tr} = \pm 20^\circ,$$
$$\dot{\theta}_{Tr} = \pm 100^\circ/s,\ \dot{\alpha}_{Tr} = \pm 100^\circ/s$$

B. Reward function

The reward function was chosen to be

$$R_i = \sum_{j=1}^{8} q_j \left(1 - \left(\frac{x_{j,i}}{x_{j,max}}\right)^2\right) + \sum_{j=1}^{2} r_j \left(1 - \left(\frac{u_{j,i}}{u_{j,max}}\right)^2\right) \tag{10}$$

, where $x_{j,i}$ and $u_{j,i}$ are the state and action x_j, u_j at time instant i and q_j and r_j are sets of scalars chosen to weight the importance of the corresponding state/action.

1) Performance Evaluation: H. Liang shows great performance in controlling the 2DoF Inverted pendulum rig produced by Quanser in his report *Design of a Robust Mean-square Stabilizing Data-Driven Controller* [11] and was chosen as a baseline of performance in the analysis. H. Liang computes the total energy of the behaviour of his controller by Parseval's theorem. In order to compare the performance of the RL algorithms to the Data-Driven controller the total energy of the outputs for both controllers was calculated by Parseval's theorem as [12]:

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2 \tag{11}$$

, where $X[k]$ is the discrete Fourier transform of $x[n]$.

C. Deep-training strategies

During training the algorithms were restricted to running a single simulation for a maximum of 15 simulated seconds, after that the model was reset. This was to opt for more diverse exploration through the state space. It further forces the algorithm to optimize the reward during the constrained time sequence rather than reaching infinite reward simply by avoiding a terminal state.

Fast results have been a determining factor for which algorithm to be used in the final model. Short and equivalent training sessions were performed to compare performance between modifications. As this favours steep initial learning curves it might neglect some more robust solutions which require longer training sessions to show potential.

Furthermore, when good settings were found for all parameters, the model was trained for longer training sessions while using learning rate decay by continuously lowering the learning rate throughout training. This enables a fast initial learning curve, while allowing for an easier convergence towards an optimum at the later stages.

IV. Setup and Results

This section describes the chosen parameters in the setup and all results based upon them.

A. Hyper-parameters

The two different algorithms were set up with a set of initial hyper parameters, most of which as the default parameters from the different libraries used and others tuned based on behaviour during training.

For DDQN, the hyper-parameters were setup as

Hyper-parameter	**Value**
Discount factor γ	0.99
Batch size	32
Policy	Boltzmann Q [13] with exploration rate=1
Network structure	Multi layered perceptron with 3 layers of 16 neurons

, and for PPO the hyper-parameters were setup as

Hyper-parameter	**Value**
Discount factor γ	0.99
ϵ	0.2
Maximum Gradient Clipping	0.5
c_1	0.5
c_2	0.01
Parallel actors	12
Network structure	Multi layered perceptron with 2 layers of 64 neurons

B. Reward function

The reward function was tuned for the two algorithms independently by training for short session with fixed hyper-parameters. An initial reward function for both algorithm was defined based on inspiration of the performance vector ζ from Liangs paper [11] by defining q- and r-values as

$$q_\theta = 10,\ q_\alpha = 100,\ q_{\dot{\theta}} = 0,\ q_{\dot{\alpha}} = 0,\ r = 0. \tag{12}$$

Both algorithms resulted in similar behaviour with quite stable pendulum angles $\alpha_{x,y}$, but with a continuous drift in θ until reaching a terminal state. It further introduced very high angular velocities for all angles resulting in high frequency oscillations. The drift is assumed to be due to a lack of importance in reward to stabilize the pendulum position at equilibrium, so a higher importance to the motor angles in relation to the pendulum angles is given. Furthermore, a reward for keeping the angular velocities low was introduced to reduce jerk behavior.

After tuning the reward functions with this in mind the best performing reward functions differed slightly between the two algorithms. For PPO the best performing settings were found to be

$$q_\theta = 40,\ q_\alpha = 60,\ q_{\dot\theta} = 20,\ q_{\dot\alpha} = 10,\ r = 0 \tag{13}$$

, while the best performing settings for DDQN were found to be

$$q_\theta = 50,\ q_\alpha = 90,\ q_{\dot\theta} = 5,\ q_{\dot\alpha} = 5,\ r = 0. \tag{14}$$

C. Performance of PPO and DDQN

With the tuned reward functions both algorithms were trained for longer sessions with learning rate decay to investigate their difference in performance. Both proved to yield reasonable but slightly different performance. The energy of the two resulting behaviours were calculated by applying Parsevals theorem as in equation (11) to α and θ resulting in

$$E[\theta_x]_{PPO} = 1.0551$$
$$E[\theta_x]_{DDQN} = 1.1020$$
$$E[\alpha_x]_{PPO} = 0.007837$$
$$E[\alpha_x]_{DDQN} = 0.008912$$

, which showed that PPO slightly outperformed DDQN in terms of total energy. Looking at the behaviour of the two final models pointed towards the same conclusion as DDQN seemed to struggle in keeping the θ values centered around 0 in comparison to PPO, while showing similar behaviour in terms of α.

D. Final model

As PPO demonstrated greater potential performance-wise than DDQN it was chosen to further investigate how far one could go with this algorithm. The procedure was to train for short sessions of about $2 * 10^6$ time steps while decreasing the learning rate between sessions.

The resulting model achieved a total energy of

$$E[\theta_x] = 0.067424$$
$$E[\alpha_x] = 0.0053545$$

, which is a significant performance increase in comparison to the shorter trained model. The resulting behaviour as presented in Fig. 2 shows great performance in stabilizing both the pendulum and motor angles.

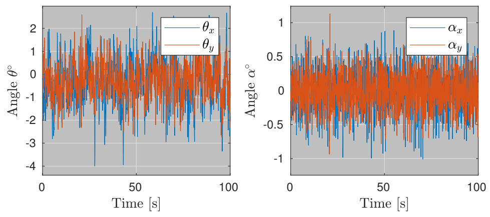

Fig. 2. A sample of the resulting behaviour from the final model after training with a decaying learning rate.

The state sequence is also presented in the form of a histogram in Fig. 3 where it becomes apparent that θ is still not exactly zero-centered even during the longer training session.

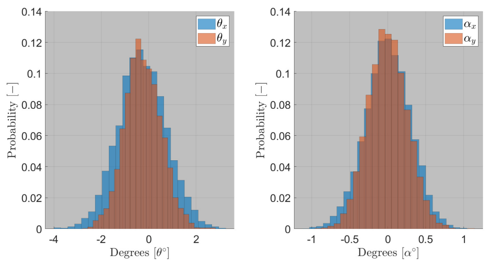

Fig. 3. The probability distribution for the final model of θ_x and θ_y in the left figure, and of α_x and α_y. The standard deviations of the respective angles are $\sigma_{\theta_x} = 1.0148$, $\sigma_{\theta_y} = 0.76615$, $\sigma_{\alpha_x} = 0.29645$ and $\sigma_{\alpha_y} = 0.25672$,

V. Analysis & Discussion

The final model resulted in reasonable stability of both the pendulum and motor angles α and θ. However, the angles still oscillate more than ideally desirable. There might be several reasons for this, first of all it might be a problem that is simply solved through longer training and further learning rate decay. It is also possible that this is due to the tuning of the reward function and perhaps a more refined one could be necessary.

A. Learning Algorithms

Both algorithms demonstrated promising behaviour even if the results show a difference in performance between them. However, the most significant difference was in tuning the two algorithms. The experience of tuning PPO was fairly straightforward and one could clearly see trends in behaviour based on modifications. Meanwhile, this was not the case for DDQN where a slight change could completely change the result in an arbitrary direction.

Many of the hyper-parameters for both algorithms were not tuned. It would be interesting to investigate the difference in behaviour when tuning these parameters as well, as it might for example lead to a more stable performance curve for DDQN. Additionally, the difference in performance makes it interesting to evaluate other types of algorithms for the specific application as well, as there might be others that are more suitable in capturing the problem.

B. Comparison to data-driven controller

The data-driven controller implemented by Liang in [11] is shown to have a total energy for the first 10^4 samples of the angles θ and α as:

$$E[\theta_x] = 0.14877$$
$$E[\alpha_x] = 0.000841$$

One can observe that the energy regarding α is significantly lower than the final obtained model, while the energy related to θ is higher. It would seem then that the data-driven controller oscillates more in θ and, in turn, manages to have a more stable α when compared to the final model. This could be mainly due to the chosen q-values in the reward function not penalizing high values of α that much more than penalizing high values of θ and its derivative as well, as ideally it would be desirable to minimize both α and θ variations as much as possible. Perhaps with more training time the variations in α of the final model could be even further minimized.

Due to the fact that the simulated environment is based on a non-linear model, it would be interesting to see the final solution applied to the real pendulum environment. One could assume that the loss in performance from simulation to practice would not be as significant in this scenario and any further non-linearities that were not accounted for could possibly be overcome without as much difficulty.

VI. Conclusion

The results of the project prove that RL can be used for control and that both investigated algorithms can be used to stabilize a 2DoF inverted pendulum. However, the question remains if it can reach or surpass the performance of modern control theory. Furthermore, based on the experience during training of the algorithms one finds that the procedure for tuning the algorithms is quite similar as that of control theory. Thus, there is likely a lot from the vast amount of research in control theory which can be adapted in to the area of Reinforcement Learning in order to develop and tune algorithms meant for control tasks.

Since the time span of this project was quite short, we would also like to encourage further studies based on the results of this report. It would especially be interesting to verify and assess the developed solution's performance on the real pendulum environment and further tune it accordingly.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, and et al., "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, 2016.

[2] J. Verlie, "Control of an inverted pendulum with deep reinforcement learning," *Ghent University Department of Electronics and Information Systems*, 2017. [Online]. Available: https://lib.ugent.be/catalog/rug01:002367558

[3] R. U. Savinay Nagendra, Nikhil Podila and K. George, "Comparison of reinforcement learning algorithms applied to the cart-pole problem," *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2018. [Online]. Available: https://arxiv.org/abs/1810.01940

[4] R. S. Sutton and A. G. Barto, "Reinforcement learning:an introduction," *The MIT press*, vol. second edition, 2017.

[5] P. Gong and B. Kulcsár, "Manual for 2-dof inverted pendulum," *Department of Signals and Systems,Chalmers University of Technology*, 2016.

[6] C. Runge, "Ueber die numerische auflösung von differentialgleichungen." [Online]. Available: https://doi.org/10.1007/BF01446807

[7] C. J. C. H. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, 1992.

[8] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[9] M. H. H. v. H. M. L. Ziyu Wang, Tom Schaul and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2016. [Online]. Available: https://arxiv.org/abs/1511.06581

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[11] H. Liang, "Design of a robust mean-square stabilizing data-driven controller - with implementation to a 2 dof inverted pendulum," Master's thesis, 2017.

[12] G. H. Hardy and J. E. Littlewood, "Notes on the theory of series (v) : On parsevals theorem," *Proceedings of the London Mathematical Society*, vol. s2-26, no. 1, p. 287–294, 1927.

[13] C.J.Watkins, "Models of delayed reinforcement learning," Ph.D. dissertation, 1989.