Memory-Based Reinforcement Learning for Trans-Domain Tiltrotor Robot Control

Yujia Huo$^{1,2,3,*}$, Yiping Li$^{1,2}$, Xisheng Feng$^{1,2}$

$^1$State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang, 110016, China
$^2$Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang, 110169, China
$^3$University of Chinese Academy of Sciences, Beijing, 100049, China

E-mail: huoyujia@sia.cn

Abstract. Aiming at the problems of motion control with high precision for a new type of air-water trans-domain tiltrotors, a deep reinforcement learning controller is applied to these conditions. Reinforcement learning algorithm with memory capability allows the robot to learn from dynamic information collected in the past. In this paper, the trans-domain tiltrotors are supposed operating as a quad-rotors with fixed-wing in the air. Moreover, simulation is based on ROS and Gazebo platform for training the reinforcement learning repeatedly and the results demonstrate this algorithm gets better accuracy and effectiveness compared with other non-current methods in the conditions of the tiltrotors control task.

1. Introduction

Benefiting from the continuous advances in several fields, unmanned vehicles including unmanned aerial vehicles(UAVs) and unmanned underwater vehicles(UUVs) have played irreplaceable roles in applications both in civilian and military areas. The development of a vehicle capable of both flight and underwater navigation has been of great interest for decades. A new kind of unmanned vehicle that has capabilities of flying, submersible operating and transition across the two media of air and water, incentivized the interest in a platform combining advantages of UAVs and UUVs[1].

The autonomy of these unmanned vehicles with uncertainties has been one of the most critical criteria and been studied for decades by various control theories ranging from traditional techniques to several different artificial neural network-based control architectures. Conventional approaches focus on controlling an unmanned vehicle described by one or limited and very few accurate models with hydrodynamic parameters obtained from experiments and achieve variable results. Due to the inertial, buoyancy and hydrodynamic effects, dynamics of robots are with strong nonlinearity. The trans-domain will be affected by the uncertainty surrounding the dynamic disturbances, while influenced by ocean currents and winds. The linear approximation of the dynamic model of the robot...
is not sufficient. The nonlinear model is destined to be inaccurate because some parameters are unknown or vary with unmodeled conditions.

Some traditional techniques have got achievements of controller design for AUVs as well as UAVs. However, gaps between dynamic models in the air and underwater bother scientists and engineers. There have been some approaches to satisfying such tasks. PD controllers are adopted in multi-rotors trans-domain control tasks, respectively for an octocopter [2] and quad-rotor [3]. Andrew Fabian et al. in MIT Lincoln Laboratory designed a miniature submersible UAV imitating the gannet [4]. The tasks are simple and the robots will not perform well in the condition more complex and variable.

In recent years, with the rapid development of computing hardware and artificial intelligence, agents with learning capabilities have achieved encouraging performance, especially in the planning and decision making of complex systems that are difficult to build an accurate model. Reinforcement Learning (RL) allows agents to learn actions while interacting with the environment, thereby maximizing the concept of cumulative rewards. An open-frame AUV Ictineu was trained on a simulation platform approximated to the environment and the experience of policy from simulation remained when agents continuing training in the real world [5]. Application on AUV depth control was introduced and compared with linear quadratic Gaussian integral controller and nonlinear MPC [6]. In other similar motion control applications of the UAV, including multi-rotors and helicopters [7]–[9], RL was also involved.

However, all work based on model-free reinforcement learning is in full observation [10]. Due to differences in sampling frequency and sensor noise, state obscurations, unobserved changes or functional approximations in the controlled system. The system with these problems is described as part of the observable Markov decision process of (POMDP). In [11], the proposed algorithm with RL LSTM recurrent neural networks, and to address the non-Markov task. In [12], the substituted recurrent network DQN first fully connected layer [13], and get better performance. [10] proposed an algorithm that allows the agent to resolve first consider DPG continuous state/action space in the POMDP, and the algorithm to obtain an effective and satisfactory result.

This paper aims to design a controller based on recurrent reinforcement learning for trans-domain tiltrotors position tracking tasks as a quad-rotors with fixed-wings when these problems are considered as a POMDP.

The remainder of the paper is organized as follows. In section 2, the robot dynamic system description is presented. Section 3 explains the composition of the controller combining the recurrent neural network with deterministic policy gradient algorithm. In section 4, a simulation study is provided the effectiveness of the algorithm proposed, followed by the conclusion in section 5.

2. Problem Formulation

The reinforcement learning method is to learn from repeated interactions with the environment and obtain the best control strategy. However, it takes a lot of time to physically train robots in the real world. The training task must be simulated before deployment. In this section, the dynamics model of a trans-domain quad-rotors robot is introduced and used for designing controllers and RL simulation training.

![Figure 1. The Trans-domain Quad-Tiltrotors robot](image-url)
A tiltrotors robot has working modes such as underwater navigation, water surface taking off/landing, hovering as quad-rotors or plane with fixed-wings.

In Figure 1, The robot carrier coordinate system selects the center of gravity as the origin \( G \), \( x \) points to the bow, \( y \) points to the starboard, and \( z \) is determined by the right-hand rule. The body coordinate is fixed relative to the fuselage and rotates as the aircraft rotates. The attitude angles \( \psi \), \( \theta \), \( \phi \) represent yaw, pitch and roll, respectively.

The robot is mainly affected by the force of the four rotors \( T_{i=1,2,3,4} \), gravity \( mg \), and wing force \( F_W \). Wing force \( F_W \) may be decoupled into three orthogonal the lift of wings \( F_{lift} \), side force of wings \( F_{sideforce} \) and air drag of wings \( F_{drag} \). And in the same way, the moment of the robot is obtained.

This paper mainly considers the motion control problem of the robot in the quad-rotors mode. Therefore, the tilting angle of the ducted propeller is fixed at 0, and the thrust is parallel to the \( Gz \).

### 2.1 Force and moment provided by the tilt propeller thrusts

The force supplied by thrusts in the Body-Fixed coordinates Frame can be explained in Equation(1). Where \( F_{r,x} \), \( F_{r,y} \), and \( F_{r,z} \) respectively represent the components of the resultant force \( F_r \) on the \( Gx \), \( Gy \) and \( Gz \) axes in the carrier coordinate system, \( \omega_i \) describes the rotational rate of the i-th propeller, and \( C_r \) is a constant, indicating the relationship between the thrust of the thruster \( F_{r,i} \) and the square of the rotational speed \( \omega_i^2 \), which may be derived from the propeller test measured.

\[
F_r = [F_{r,x} \ F_{r,y} \ F_{r,z}]^T
\]

Where \( F_{r,x} = 0, F_{r,y} = 0, F_{r,z} = \sum_{i=1}^{4} -F_{r,i} = \sum_{i=1}^{4} -C_r \omega_i^2 \).

At the same time, the thrust generated by the thrust of the thruster is shown in Equation(2). For the moments \( L_{x,f}, L_{y,f} \) and \( L_{z,f} \) show the rotor position vector component of the rotor in front of the robot along with the axes \( x, y, z \) respectively. And \( L_{x,b}, L_{y,b}, L_{z,b} \) in the same way for the rotors behind the robot. Moreover, the rotating propellers with gyroscopic effects can be described as \( M_{G,i} = C_q \omega_i^2 \).

\[
M_r = \begin{bmatrix} M_{r,x} \\ M_{r,y} \\ M_{r,z} \end{bmatrix} = \begin{bmatrix} T_1 L_x + T_2 L_{x-y} - T_3 L_x - T_4 L_{-x} \\ T_1 L_y + T_2 L_{y-z} - T_3 L_y - T_4 L_{-y} \\ C_q (\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{bmatrix}
\]

### 2.2 Force and moment provided by the fixed-wings

The force supplied by the fixed-wings can be decoupled into three orthogonal forces, the lift of wings \( F_{lift} \), side force of wings \( F_{sideforce} \) and air drag of wings \( F_{drag} \). The force and moment effect of the fixed wings can be rewritten in Equation(3). Dynamic pressure \( Q = \rho V^2 / 2 \), where \( \rho \) presents the air density, \( V = \sqrt{V_x^2 + V_z^2} \) represents the vector sum of \( x \), \( y \) axis velocity. In addition, \( S \) is the area of the wing. \( C_D, C_Y, C_L \) are the drag coefficient, side force coefficient and the lift coefficient, and \( C_i, C_m, C_n \) means the moment coefficient respectively.
2.3 6-DOF force and moment effect on the robot
Combining the Equation(1)-(3), in the World Frame, the force and moment effect on robot can be rewritten as follows:

\[
\begin{align*}
F_x &= F_{w,x} + F_z \sin \theta \\
F_y &= F_{w,y} + F_z \cos \theta \sin \phi \\
F_z &= F_{w,z} - F_z \cos \theta \cos \phi + mg
\end{align*}
\]

\[
\begin{align*}
M_x &= (M_x + M_w) \cos \theta \\
M_y &= (M_y + M_w) \sin \theta \sin \phi \\
M_z &= (M_z + M_w) \sin \theta \cos \phi
\end{align*}
\]

where \( m \) is the mass of the robot and \( g = 9.8 \text{m/s}^2 \).

3. Memory-based Deterministic Policy Gradient
As mentioned in the introduction, reinforcement learning has achieved some success, but all of this work is in a state of full observation. In this section, a deep reinforcement learning algorithm with current neural networks is introduced. Since the algorithm has an actor-critic structure based on DPG[14] with an actor network and a critic neural network synonyms for the policy and value function. An off-policy deterministic actor-critic method is also presented in this section.

3.1 Partially Observable Markov Decision Process
A MDP is described \((s_t, a_t, r_t, p(s_{t+1}=s'|s_t, a_t))\), where \(p(s_{t+1}=s'|s_t, a_t)\) is transition probability that action \(a\) in state \(s\) at time \(t\) will lead to state \(s'\) at time \(t+1\). When interacting with the environment, the agent receives a representation of the environmental state, \(s_t \in S\), where \(S\) is the set of possible states, and on that basis chooses an action, \(a_t \in \mathcal{A}(S_t)\), where \(\mathcal{A}(S_t)\) is the set of actions available in the state \(s_t\)[15]. At each step after the action, consequently, the agent receives a numerical reward, \(r_{t+1} \in \mathcal{R} \subseteq \mathbb{R}\), and transfer itself in new state \(s_{t+1}\).

But in the real world, the complete state of the required system is rarely provided to the agent, or even perceived by the sensor. In other words, few Markov properties are well maintained in the real world. Part of the Markov decision process (POMDP) can be observed through the clear recognition of feeling agent receives only a glimpse of the bottom part of the system state, so as to more effectively capture the dynamic environment of the real world[12].

Then, a POMDP is described as \((s, a, p, r, \Omega, O)\) with 6 components, and the \(s, a, p, r\) are described as states, actions, transitions, and rewards as before. But the agent no longer receives the states directly from the system, instead an observation \(o \in \Omega\) is received. And this observation is based on the probability distribution \(o \sim O(s)\).

3.2 Off-Policy Deterministic Actor-Critic
For a MDP, the performance can be described and evaluated as an expectation with the conditional probability density at \(\pi_\theta(s_t, a_t)\) as follows:

\[
J(\pi_\theta) = \mathbb{E}_{s' \sim \rho^\pi, a' \sim \pi_\theta} [r(s, a)]
\]

where we denote the discounted state distribution by

\[
\rho^\pi(s') = \sum_{t=1}^{\infty} \gamma^{t-1} p_\pi(s) p(s \rightarrow s', t, \pi) ds
\]

For this performance Equation(5), the basic goal is to maximize the performance function with the optimal policy. And the policy gradient is widely utilized for a continuous state/action tasks. By
updating the parameters $\theta$ with the performance gradient (6), the algorithm finds the optimal parameterized with $\theta$ and at the same time, maximizing the time cumulative reward function.

$$\nabla_{\theta} J(\pi_{\theta}) = \int \rho^\pi(s) \nabla_{\theta} \pi_{\theta}(a|s) Q^\pi(s,a) \, da \, dr = \mathbb{E}_{t \sim \rho^\pi, a \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) Q^\pi(s,a)]$$ (6)

The actor-critic has a better performance. It contains two eponymous components. The actor adjusts the stochastic policy $\pi_{\theta}(s)$ with parameters $\theta$ by stochastic gradient of the performance function (6). Meanwhile, an approximate policy evaluation algorithm such as a temporal-difference(TD) method is introduced to critic and estimate the action-value function $Q^\pi(s,a) \approx Q^\pi(s,a)$ with parameters $\omega$.

In order to improve the algorithm, the deterministic strategy replaces the stochastic strategy because the gradient of the deterministic strategy can be estimated more effectively without the problematic integration in the action space (14). It turns out that the deterministic strategy gradient is a limit case of the stochastic strategy gradient theorem.

3.3 Memory-based Deterministic Policy Gradient

As mentioned above, the states cannot be received by the agent. Instead, the agent only indirectly observes the Markov decision process through the observations. Then there involves a history $h_t = (o_t, a_t, a_2, \ldots, a_{t-1}, o_{t-1})$. Same as the goal of off-policy deterministic actor-critic algorithm, we tend to maximize the deterministic policy performance function (7) re-described for POMDP.

$$J = \mathbb{E}_{t \sim \rho} [\sum_{i=t}^w \gamma^{i-1} r(s_i, \mu(h_i))]$$ (7)

In the case of partial observability, the optimal policy and the estimated action-value function are both considering the observation-action $h_i$ with states as a sequence of history, instead of just states of one step for fully observable MDP. To solve the POMDP, feedforward networks policy and evaluation in DPG are replaced with neural networks(LSTM) in practice, which allow the agent to learn from the information preserved from past. Thus, a new policy gradient is obtain as following by replacing $\mu(s)$ and $Q(s,a)$ with $(h)$ and $Q(h,a)$.

$$\nabla_{\theta} J = \mathbb{E}_{t \sim \rho} [\sum_{i=t}^w \gamma^{i-1} \frac{\partial Q^\mu(h_i,a)}{\partial a}_{a=\mu^\theta(h_i)} \frac{\partial \mu^\theta(h_i)}{\partial \theta}]$$ (8)

Based on ideas from the DQN (13) method, a relay buffer is also introduced which can improve the data efficiency and stability. Meanwhile, target networks are appended to the algorithm. Two copies of the evaluation function $Q$ and the policy are involved, with parameters $\omega'$ and $\theta'$. $\omega'$ and $\theta'$ are updated as mirror of $\omega$ and $\theta$ with some delay. This Asynchronous update also enhances the stability of the neural networks.

**Memory-based reinforcement learning algorithm:**

- Initialize critic network $Q^\omega(a_t, h_t)$ and actor $\mu^\theta(h_t)$ with parameters $\omega$ and $\theta$.
- Initialize target network $Q^{\omega'}(a_t, h_t)$ and $\mu^{\theta'}(h_t)$ with weight $\omega' \leftarrow \omega$ and $\theta' \leftarrow \theta$.
- Initialize replay buffer $R$.
- for episode $i=1$ to max episode $M$ do:
  - Initialize empty history memory $h_0$.
  - for $j=1$ to $T$ do:
    - Observe $o_j$, Append observation and action to previous history memory $h_{t} \leftarrow h_{t-1}, a_{t-1}, o_{t-1}$.
    - Choose action $a_t = \mu(h_t|\theta)$.
    - Store the sequence $(o_j, a_j, r_j, \ldots, o_T, a_T, r_T)$ to $R$.
- end for.

---
Sample a minibatch \((o_1^n, a_1^n, r_t^n, \ldots, o_T^n, a_T^n, r_T^n)_{n=1-N}\) with shape N episodes from R

Update \(h_t^n = (o_1^n, a_1^n, \ldots, o_{t-1}^n, a_{t-1}^n, o_t^n)\)

Compute target values \((\delta_t^n, \ldots, \delta_T^n)\) of each sample episode

\[ \delta_t^n = r_t^n + \gamma Q^\pi (h_{t+1}^n, \mu^\theta (h_{t+1}^n)) - Q^\pi (h_t^n, a_t^n) \]

Calculate critic network and actor network update with BPTT \[16\]

\[
\Delta w = \frac{1}{NT} \sum_n \sum_t \delta_t^n \frac{\partial Q^\pi (h_t^n, a_t^n)}{\partial w}
\]

\[
\Delta \theta = \frac{1}{NT} \sum_n \sum_t \delta_t^n \frac{\partial Q^\pi (h_t^n, \mu^\theta (h_t^n))}{\partial \theta} \frac{\partial \mu^\theta (h_t^n)}{\partial \theta}
\]

Update actor and critic with Adam \[17\]

Update the target networks with the learning rate \(\alpha_{\text{critic}}\) and \(\alpha_{\text{actor}}\)

\[ w' \leftarrow \alpha_{\text{critic}} w + (1 - \alpha_{\text{critic}}) w', \quad \theta' \leftarrow \alpha_{\text{actor}} \theta + (1 - \alpha_{\text{actor}}) \theta' \]

end for

### 4. Simulation

The robot using a reinforcement learning method repeatedly interacts with the environment, which often costs a lot. Therefore, the simulation must be performed before the controller is physically deployed to the robot. In this section, the results of the simulation are presented. The combination of ROS and gazebo aims to implement a platform for deep reinforcement learning for the controller training. The primary parameters of the robot in the simulations are listed in table 1. And the simulation of reinforcement learning is code in Python2.7 with ROS and Python3.5 with Tensorflow1.14 on Ubuntu 16.04 system.

**Table 1 kinetics parameters**

| parameter | unit | value | parameter | unit | value |
|-----------|------|-------|-----------|------|-------|
| \(m\)     | kg   | 15    | \(L_{z,f}\)| \(m\) | 0.003 |
| \(L_{x,f}\)| \(m\) | 0.125 | \(L_{z,b}\)| \(m\) | 0.003 |
| \(L_{x,b}\)| \(m\) | 0.072 | \(S\) | \(m^2\) | 0.33 |
| \(L_{y,f}\)| \(m\) | 0.160 | \(l\) | \(m\) | 0.7 |
| \(L_{y,b}\)| \(m\) | 0.3   | \(\rho\)| \(kg \cdot m^{-3}\) | 1.293 |

Figure 2 shows the ROS & Gazebo simulation structure. The motion control algorithm publishes robot control commands to Gazebo through ROS, including the ducted propeller motor speed command, environment, and robot reset command, and consequently subscribes to the robot’s motion state information from Gazebo through ROS.

The information released and subscribed by ROS is applied to the simulation environment through a series of Gazebo Plugins, including the wing, the thrust model, the environmental model of the robot and the sensor components such as IMU and GPS, with the informations, the force and motion state of the robot are calculated. Apply_Force (), get_State () and other functions operate on the simulation environment.
In the Gazebo wing model plugin, the force provided by the fixed wing can be calculated in Equation.

\[
F_{w,x} = \frac{1}{2} \rho \left( 0.136 - 0.6737 \alpha + 5.4546 \alpha^2 - 0.3842 \beta^2 \right) \\
F_{w,y} = \frac{1}{2} \rho \left( 0.3073 \beta \right) \\
F_{w,z} = \frac{1}{2} \rho \left( 0.2127 + 10.806 \alpha - 46.8324 \alpha^2 + 60.6017 \alpha^3 \right) \\
M_{w,x} = \frac{1}{2} \rho \left( -0.0154 \beta - 0.1647 p + 0.0117 r \right) \\
M_{w,y} = \frac{1}{2} \rho \left( 0.0435 - 2.969 \alpha - 106.1541 q \right) \\
M_{w,z} = \frac{1}{2} \rho \left( 0.043\beta - 0.0827 r \right) 
\]

In comparison with the Model-based reinforcement learning, a simple deep deterministic policy gradient[18] controller is designed for simulation of quad-rotors trajectory tasks.

In the Model-based reinforcement learning algorithm, both the actor network and critic network are LSTM recurrent neural networks with 4 layers and the number of hidden layer units is 200. In the practice, we set the discount factor \( \gamma \) as 0.99, learning rate \( \alpha \) as 0.001 and the batch size as 32.

In DDPG algorithm, as the problem we aim to solve is in a continuous state/action space, it is enough that both the actor network and the critic network are fully connected. The number of the units in the hidden layers of both actor and critic networks are 100. And discount factor \( \gamma \), learning rate \( \alpha \), the batch size and the activation function of the neural networks are each set as 0.99, 0.001, 32 and Relu.

Then for reinforcement learning of the trajectory tracking tasks, the reward that agent receives at each step is set as the quadratic sum of the states and actions with different weights in Equation(10).

\[
r(t) = -c_1 \left\| \psi_f^{\text{heading}} - \psi_f^{\text{target}} \right\|_2^2 - c_2 \left\| \psi_f^{V} - \psi_f^{V} \right\|_2^2 - c_3 \left\| \psi_f^{\text{target}} \right\|_2^2 - c_4 \left\| z_t - z_t^{\text{target}} \right\|_2^2 - c_5 \left\| \phi_t \right\|_2^2 - c_6 \left\| \omega_t \right\|_2^2 - c_7 \left\| \alpha_t \right\|_2^2 
\]

(10)

In the horizontal plane, the reward function is designed using the line of sight method. However, in the vertical plane, the angle of attack always exists due to the quad-rotors dynamics model. Controls in the vertical plane use height errors instead of angular errors in the horizontal plane.
Figure 5. robot state with the Memory-based RL and DDPG

We set a maximum of 20000 episodes to train both DDPG and Memory-based RL, and the rewards in figure 3 illustrate Memory-based RL has accelerated convergence speed with better performance. Figure 4 shows the trajectory, and in Figure 5 the linear velocity(a,b), attitude(c,d), angular velocity(e,f), thrust rotation rate(g,h). The memory-based gets a small error in attitude and angular velocity while in Equation(10), the attitude and angular velocity are supposed as small as possible. Figure 5(h) shows that the actions of thrust rotation rates are smooth with less concussion when compared with actions in Figure 5(g). Simulations illustrate that a Memory-based RL controller can solve the trajectory tracking tasks with a satisfying performance which is assumed as a POMDP.

5. Conclusion
This paper solves the trajectory tracking task of the partially observable Markov decision process by recursive reinforcement learning. Extended from deterministic policy gradient, reinforcement learning with recurrent neural network are involved. This method with recurrent neural networks allow the agents to learn effectively, and is validated in simulation compared and DDPG controller.
6. References

[1] X. Yang, T. Wang, J. Liang, G. Yao, and M. Liu, “Survey on the novel hybrid aquatic–aerial amphibious aircraft: Aquatic unmanned aerial vehicle (AquaUAV),” *Prog. Aerosp. Sci.*, vol. 74, pp. 131–151, Apr. 2015.

[2] M. M. Maia, P. Soni, and F. J. Diez-Garías, “Demonstration of an Aerial and Submersible Vehicle Capable of Flight and Underwater Navigation with Seamless Air-Water Transition,” p. 9.

[3] P. L. J. Drews, A. A. Neto, and M. F. M. Campos, “Hybrid Unmanned Aerial Underwater Vehicle: Modeling and simulation,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL, USA, 2014, pp. 4637–4642.

[4] A. Fabian, Y. Feng, E. Swartz, D. Thurmer, and R. Wang, “Hybrid Aerial Underwater Vehicle (MIT Lincoln Lab),” p. 2.

[5] A. El-Fakdi and M. Carreras, “Two-step gradient-based reinforcement learning for underwater robotics behavior learning,” *Robot. Auton. Syst.*, vol. 61, no. 3, pp. 271–282, 2013.

[6] H. Wu, S. Song, K. You, and C. Wu, “Depth Control of Model-Free AUVs via Reinforcement Learning,” *IEEE Trans. Syst. Man Cybern. Syst.*, pp. 1–12, 2018.

[7] H. Bou-Ammar, H. Voos, and W. Ertel, “Controller design for quadrotor UAVs using reinforcement learning,” in *Proceedings of the IEEE International Conference on Control Applications*, 2010, pp. 2130–2135.

[8] A. Y. Ng et al., “Autonomous inverted helicopter flight via reinforcement earning,” *Springer Tracts Adv. Robot.*, vol. 21, pp. 363–372, 2006.

[9] W. Koch, R. Mancuso, R. West, and A. Bestavros, “Reinforcement Learning for UAV Attitude Control,” *ArXiv180404154 Cs*, Apr. 2018.

[10] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, “Memory-based control with recurrent neural networks,” *NIPS Deep Reinf. Learn. Workshop ArXiv151204455*, pp. 1–11, 2015.

[11] B. Bakker, “Reinforcement learning with long short-term memory,” in *In Advances in Neural Information Processing Systems 15 (NIPS-2002*, 2003, pp. 1475–1482.

[12] M. Hausknecht and P. Stone, “Deep Recurrent Q-Learning for Partially Observable MDPs,” pp. 29–37, 2015.

[13] V. Mnih, D. Silver, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *NIPS*, pp. 1–9.

[14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic Policy Gradient Algorithms,” *Proc. 31st Int. Conf. Mach. Learn. ICML-14*, pp. 387–395, 2014.

[15] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction.*, vol. 9, no. 5. 1998.

[16] P. J. Werbos, “Backpropagation Through Time: What It Does and How to Do It,” *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[17] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *ArXiv Prepr. ArXiv14126980 2014*.

[18] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” *ArXiv Prepr. ArXiv15092971 2015*, Sep.