Helicopter Control using Deep Reinforcement Learning

Aadil Hayat
hayataadil@gmail.com
Aerospace Engineering
IIT Kanpur

Autonomous helicopter flight is a very challenging control problem with high dimensional, asymmetric, noisy, nonlinear, non-minimum phase dynamics. In last few years, many advances have been made in finding good controllers for helicopters. In this project we use actor-critic, model-free algorithm based on deep deterministic policy gradient to learn a controller to perform different aerobatic trajectories. It uses simulator provided by 2014 Reinforcement Learning Competition for Helicopter domain. This approach drastically reduces the training time needed to learn controller, while achieving a performance similar to the prize winning methods.

Nomenclature

x_t	=	observation at timestep *t*
a_t	=	action taken at timestep t
r_t	=	reward received at timestep *t*
s_t	=	state at timestep *t*
R_t	=	sum of discounted future reward at timestep *t*
γ	=	discounting factor
Q^π	=	value function for π as policy function
θ	=	parameters of function approximators
L	=	loss function
μ	=	deterministic policy function mapping states to action
π	=	probability distribution mapping states to action
v_i	=	velocity of helicopter *i* direction
q_i	=	quaternion of helicopter around *i* axis

1. Introduction

One of the primary goals of the field of artificial intelligence is the development of intelligent algorithms and autonomous machines. In this project we explore and study one of most promising branch of Artificial Intelligence known as **Reinforcement Learning**.

Reinforcement learning is a branch of Artificial Intelligence which studies decision making and control, and how a decision making agent can learn to act optimally in a previously unknown environment. Reinforcement Learning is inspired by the ways humans learn, by interaction, trial-and-error and associations between what we can see in our environment and the outcomes of the actions we take.

Deep reinforcement learning studies how neural networks can be used in reinforcement learning algorithms, making it possible to learn the mapping from raw sensory inputs to raw motor outputs, removing the need to hand-engineer this pipeline. Recently, significant progress has been made in the area of deep reinforcement learning such as "Continuous control with Deep Reinforcement Learning" algorithm (Lillicrap et al.[1], 2016) that is capable of learning policies in continuous action domain whose performance are is competitive to that of algorithms with full access to the dynamics of the domain and its derivatives.

The framework of this project is set around the Reinforcement Learning Competition, annual gathering of experts and students of Reinforcement Learning that compete in a variety of problem domains. The problem domains selected for the Competition usually provide an important and challenging testbed for learning algorithms, and the Competition itself helps researchers around the globe compare and understand in more detail how their algorithms perform in different problems.

Of the three domains proposed by the Competition we focus on the problem of autonomous helicopter control. This is a well-known problem in the field of Machine Learning that has been tackled by many researchers, and that has a high number of practical applications including rescue tasks, aerial filming, access to hazardous zones and others.[2]

The helicopter problem has been part of the Competition for several years, and several groups have published their results on it.[3][4][5] This gives us an opportunity to contrast our results and know what to expect of this Competition. However, we differ in method from many of the previous attempts to solve helicopter problem within Competition. We take a different approach and choose a model-free Reinforcement Learning algorithm with continuous state and action space with specified goal of minimizing the experience needed and memory required by the training algorithm while also solving the generalized helicopter control problem without using prior knowledge.

2. Reinforcement Learning Challenge

This project is framed in the context of the 2014 Reinforcement Learning Competition (RL-C) held by the RL Community. The RL-C is aimed at RL students and researchers, and gives them opportunity to test their algorithms in well-defined problem settings, and as well as to create new specifically designed algorithms. All the documentation about past RL-Cs can be found in the website http://www.rl-competition.org/ .

2.1 Helicopter Challenge

The Helicopter challenge is based on the work of A. Ng's group at Stanford University.[6][7][8][9] The goal of the agent is to control a simulated helicopter and perform a certain task without crashing it. The task can be hovering the helicopter, flying at a constant stable speed or performing other more sophisticated aerobatics like flips and rolls. The simulator is based on a XCell Tempest helicopter, the same model used by the group at Stanford University and shown in figure 1.

Figure 1. XCell Tempest helicopter used by the Stanford University group and simulated domain in RL-C 2014

The observation or state space for this problem has 12 continuous variables, corresponding to the X,Y,Z components of the helicopter's velocity, position, angular rate and orientation. The action space has 4 continuous variables: longitudinal and latitudinal cyclic pitch and main and tail rotor collective pitch. Range of these variables is given in Table 1.

Table 1. Range of Variables in Observation and Action space.

Variable	**Range**
Position	[-20,20]
Velocity	[-5,5]
Angular rate	[-12.566,12.566]
Quaternion	[-1,1]
Action	[-1,1]

The goal of the Helicopter problem is to be able to safely control the helicopter. A large penalty is given if the helicopter moves too far from equilibrium (crashes), which should be avoided at all times. The task is run for 6000 steps, which simulates 10 minutes of real flight. The simulator provided by the Competition implements 10 different tasks of unknown content, that are identified by a (0-9) integer.

The main challenge of the Helicopter domain is its relatively high-dimensional continuous state and action space and its noisy nonlinear dynamics. Although we have the physical information needed to characterise a 3D rigid body like the helicopter, the noise in the observation and external effects like the wind makes this problem hard to model.

2.2 Software: RL-Glue

All the challenges of the RL-C domains are built on the RL-Glue software package.[10] It is a language and platform independent protocol for evaluating reinforcement learning agents with environment programs. RL-Glue separate the agent and environment development process so that each can be written in different languages and even executed over the Internet from different computers.

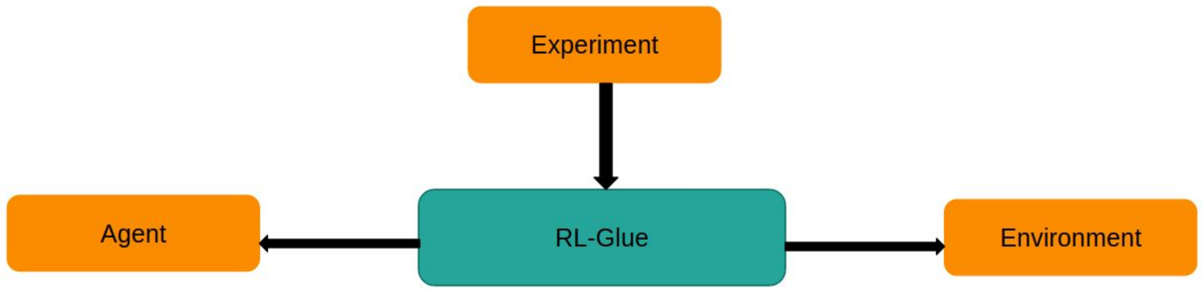


Figure 2. High level diagram of the RL-Glue architecture. The experiment call RL-Glue Core functions, and the Core calls the functions provided by the agent and the environment.

The more illustrative description of the agent, environment and experiment can be found in RL-Glue Overview Manual.[11]

3. Continuous control with Deep Reinforcement Learning

3.1 Background

An obvious approach to adapting deep reinforcement learning methods such as DQN to continuous domains is to simply discretize the action space. However, this has many limitations, most notably curse of dimensionality: the number of actions increases exponentially with the number of degrees of freedom. For example, in our case we have 4 dimensional action space with range [-1, 1]. If we discretize our action space at intervals of 0.1 units, it will lead to an discrete action space with dimensionality: $20^4 = 1.6 \times 10^5$. The situation gets worse as we want have fine control over action space and finer grained discretization leads to an explosion of the number of discrete actions.

Such large actions spaces are difficult to explore efficiently, and thus successfully training DQN-like networks in this context is likely intractable. Also, naive discretization of action spaces needlessly throws away information about the structure of the action domain, which may be essential for solving many problems. That is we used a model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high dimensional, continuous action spaces. This algorithm is based on the deterministic policy gradient (DPG) algorithm (Silver et al.[12], 2014) as described below.

3.2 Algorithm

Reinforcement learning setup consists of an agent interacting with an environment E in discrete timesteps. At each timestep t the agent receives an observation x_t, takes an action a_t and receives a scalar reward r_t. In all the environments considered here the actions as real-valued $a_t \in R^N$. We assume the environment is fully observed so $s_t = x_t$.

An agent's behaviour is defined by a policy, π , which maps states to a probability distribution over the actions $\pi : S \rightarrow P(A)$. We model it as a Markov decision process with a state space S, action space $A = R^N$, an initial state distribution $p(s_1)$, transition dynamics $p(s_{t+1}|s_t, a_t)$, and reward function $r(s_t, a_t)$.
The return from a state is defined as the sum of discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$ with a discounting factor $\gamma \in [0, 1]$. The action-value function is used in many reinforcement learning algorithms. It describes the expected return after taking an action a_t in state s_t and thereafter following policy π :

$$Q^{\pi}(s_t,\ a_t)\ = E_{r_i \geq t,\, s_i > t \sim E,\, a_i > t \sim \pi}[R_t |\ s_t,\ a_t] \tag{1}$$

Many approaches in reinforcement learning make use of the recursive relationship known as the Bellman equation:

$$Q^{\pi}(s_t,\ a_t)\ = E_{r_t,\, s_{t+1} \sim E}[r(s_t, a_t)\ +\ \gamma E_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1},\ a_{t+1})]] \tag{2}$$

If the target policy is deterministic we can describe it as a function $\mu : S \leftarrow A$ and avoid the inner expectation:

$$Q^{\mu}(s_t,\ a_t)\ = E_{r_t,\, s_{t+1} \sim E}[r(s_t, a_t)\ +\ \gamma Q^{\mu}(s_{t+1},\ \mu(s_{t+1}))] \tag{3}$$

Q-Learning uses greedy policy $\mu(s)\ =\ arg\ max_a\ Q(s, a)$. We consider function approximators parameterized by θ^Q which we optimize by minimizing the loss:

$$L\,(\theta^Q)\ = E_{s_t \sim \rho^{\beta}, a_t \sim \beta, r_t \sim E}[(Q\,(s_t, a_t |\ \theta^Q)\ -\ y_t)^2] \tag{4}$$

where

$$y_t = r(s_t, a_t)\ + \gamma Q(s_{t+1},\ \mu(s_{t+1})|\theta^Q). \tag{5}$$

It is not possible to straightforwardly apply Q-learning to continuous action spaces, because in continuous spaces finding the greedy policy requires an optimization of a_t at every timestep; this optimization is too slow to be practical with large, unconstrained function approximators and nontrivial action spaces. Instead we used an actor-critic approach based on the DPG algorithm (Silver et al.[12], 2014).

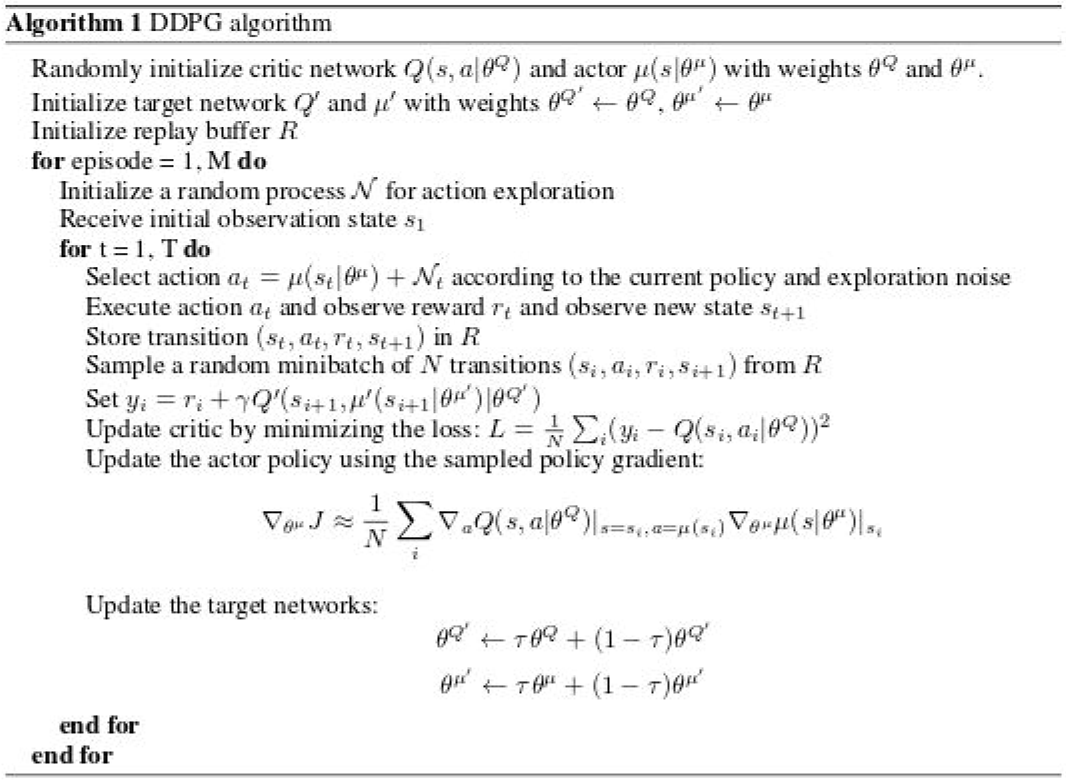
The DPG algorithm maintains a parameterized actor function $\mu(s|\theta^{\mu})$ which specifies the current policy by deterministically mapping states to a specific action. The critic $Q(s,a)$ is learned using the Bellman equation as in Q-learning. The actor is updated by following the applying the chain rule to the expected return from the start distribution J with respect to the actor parameters:

$$\begin{aligned} \nabla_{\theta^{\mu}} J &\approx E_{s_t \sim \rho^{\beta}}[\nabla_{\theta^{\mu}} Q(s, a|\theta^{Q})|_{s=s_t,\, a=\mu(s_t|\theta^{\mu})}] \\ &= E_{s_t \sim \rho^{\beta}}[\nabla_{a} Q(s, a|\theta^{Q})|_{s=s_t,\, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s=s_t}] \end{aligned} \tag{6}$$

More details about this algorithm an be found in Lillicrap et al.[1]

3.3 Pseudo Code

The above algorithm is also known as Deep Deterministic Policy Gradient (DDPG) algorithm. Its pseudo code can be found below.



Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$
for episode = 1, M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    for t = 1, T do
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
        Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:
            $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
        Update the target networks:
            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    end for
end for
```

4. Experiments and Results

In following sections we describe various experiments performed of applying reinforcement learning to the helicopter challenge. For the sake of simplicity we use average reward for describing the performance of agents, where average reward is the ratio of total reward gained to number of steps in an episode. The code for all the experiments can be found at *https://github.com/aadilh/heli-deep-q* .

4.1 Random agent

The first experiment we performed was with random non-learning agent. This agent picks a random value between range [-1,1] for each action and sends it to the trainer. As expected this agent does not perform very well and crashes helicopter. This agent crashes in 2-7 steps (0.2-0.7 seconds) resulting in very large negative reward as can be seen in figure 3 and figure 4.

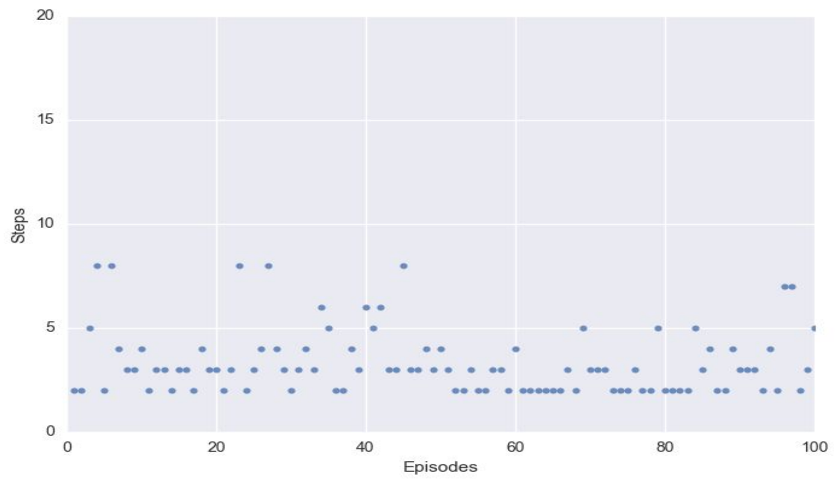


Figure 3. Number of steps completed by random agent for Task 0 before crashing.

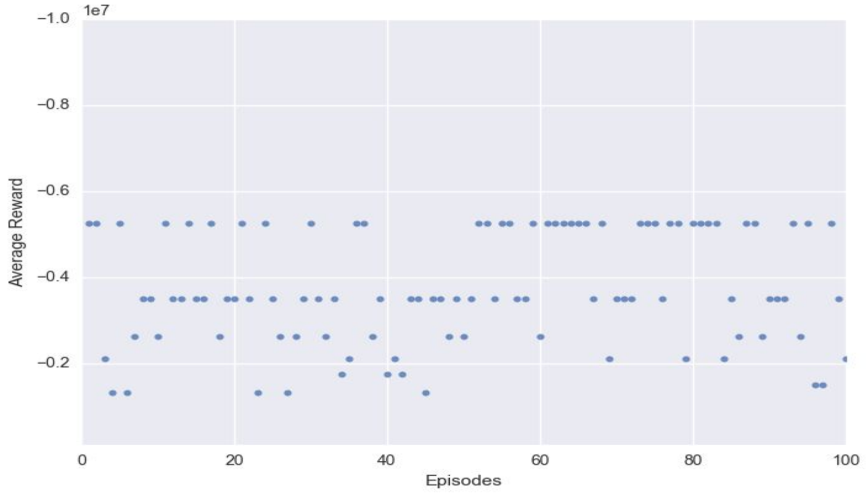


Figure 4. Average reward gained by random agent for Task 0.

4.2 Weak-baseline agent

This agent is also a non-learning agent which follows a predetermined hard-coded policy. This agent however keeps helicopter in air for entire duration of an episode. The action taken by this agent is determined by the policy described in following equations:

$$a^0 = -0.0196y - 0.0367v_x - 0.7475q_x + 0.02 \tag{7}$$
$$a^1 = -0.0185x - 0.0322v_y + 0.7904q_y \tag{8}$$
$$a^2 = -0.1969q_z \tag{9}$$
$$a^3 = 0.0513z + 0.1348v_z + 0.23 \tag{10}$$

where a^i denotes *ith* action, x, y, z are position of helicopter, v_i is velocity of helicopter in i direction and q_i is the quaternion of helicopter around i axis.

Figure 4 shows the average rewards gained from the above hard coded policy that provided with the competition software for different tasks. As expected, there is no increase in reward over time because there is no learning in this agent. It can also be seen that different tasks have different rewards. This weak-baseline agent is important because the policy of this agent is keeps the helicopter in air without crashing so it is good policy to initialize your learning algorithm as we will see later in section 4.3.

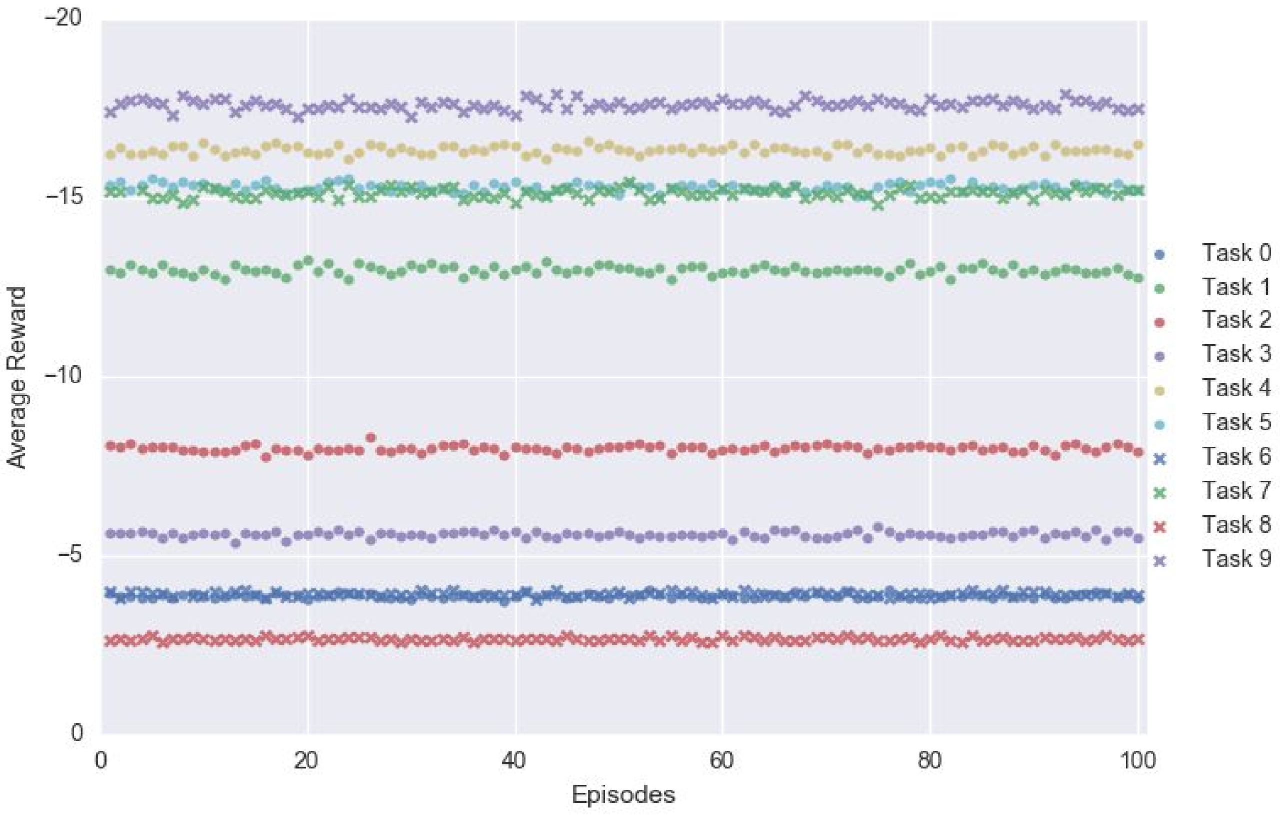


Figure 5. Average reward gained from a non-learning hard-coded policy agent for different tasks.

From figure 6 it can be seen that observation are very noisy and since actions depend linearly with observation in hard coded policy hence actions are also noisy. Hence to remove noise from observation we applied Kalman filter to observations which resulted in smoothening of actions to some extent (figure 7) but when process variance and measurement variance is increased further it results in helicopter crashing in about 90 steps (9 seconds) as can be seen in figure 8.

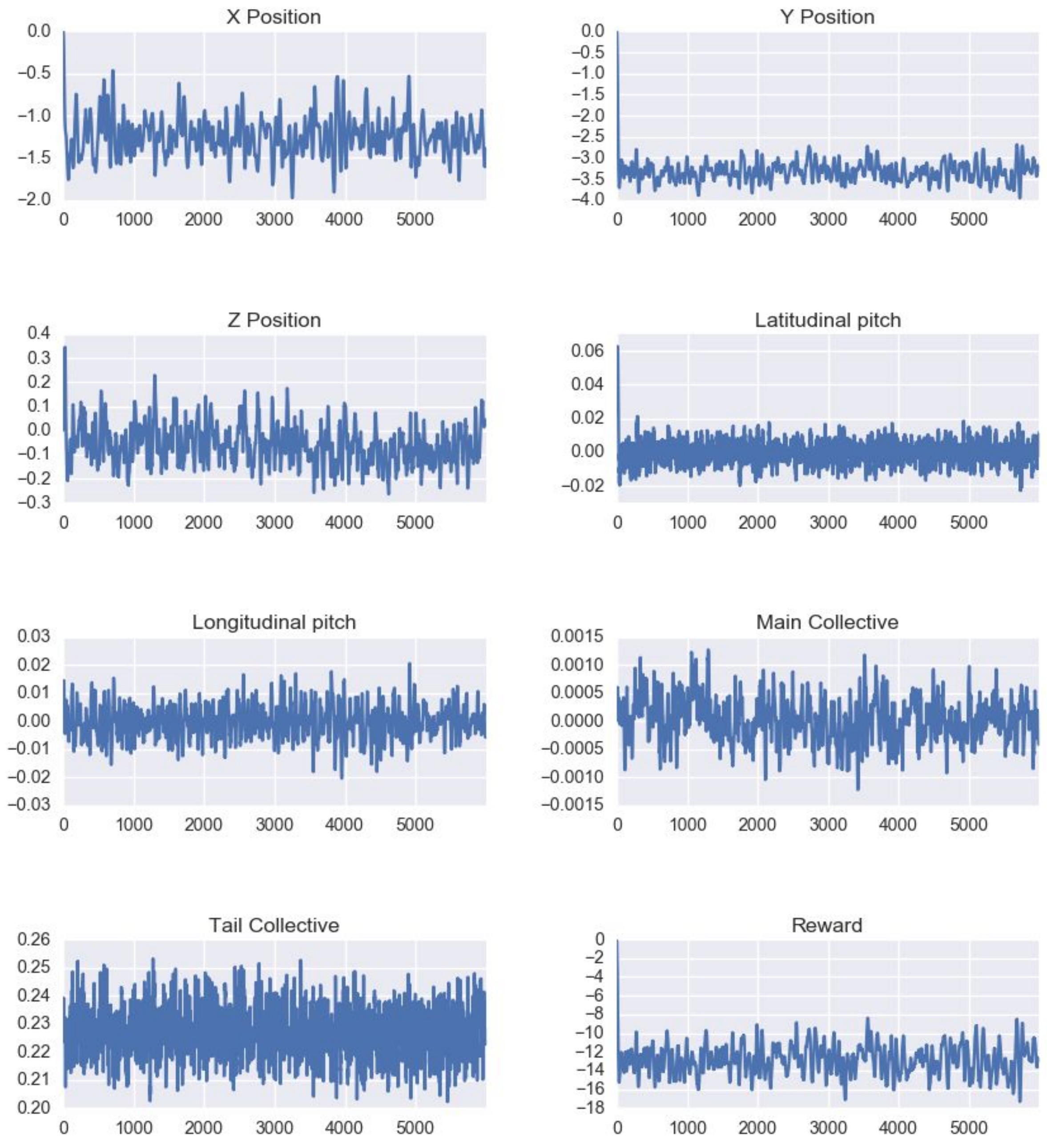


Figure 6. Plot of positions, actions and reward for Task 1 of weak-baseline agent

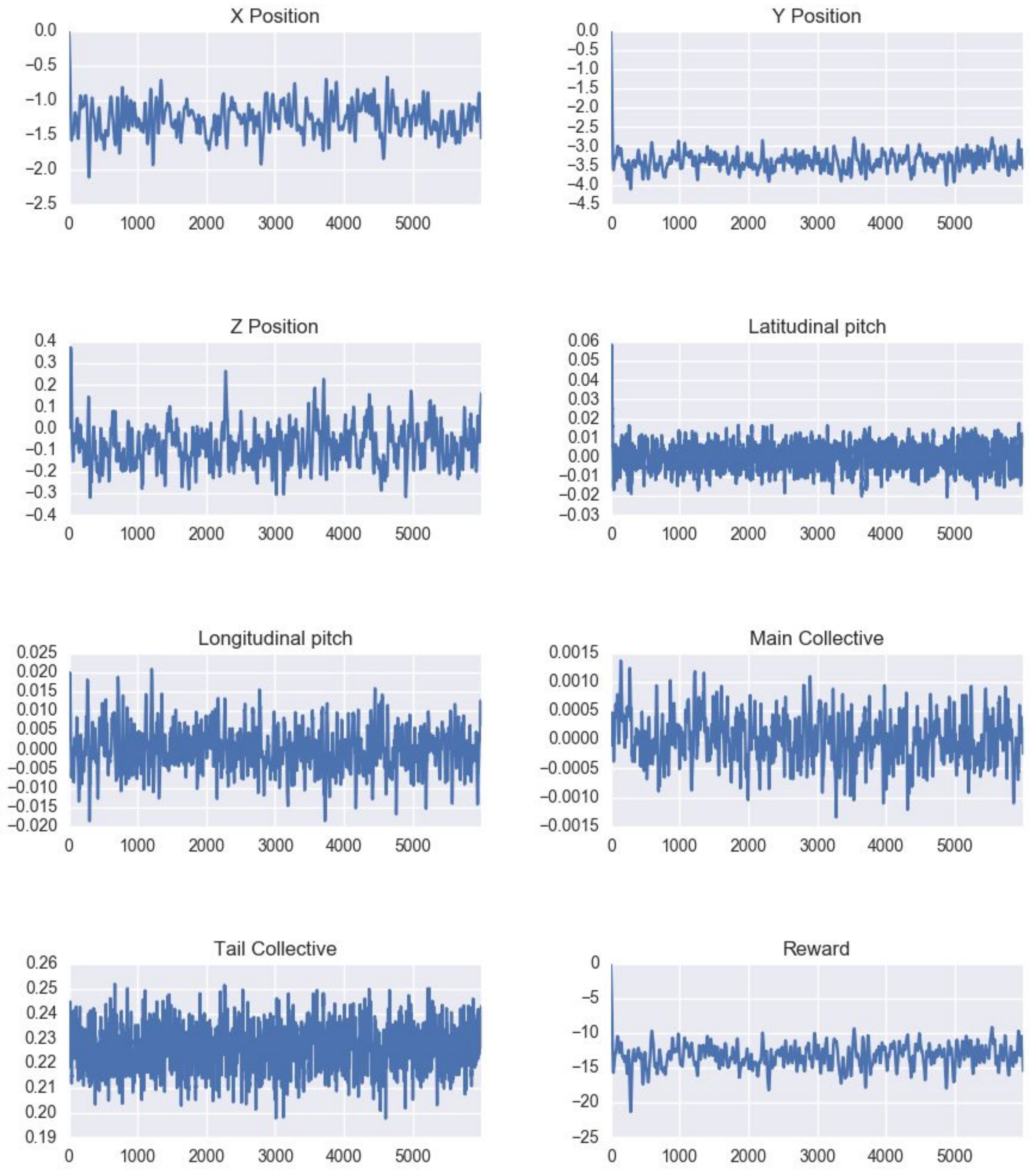


Figure 7. Plot of positions, actions and reward for Task 1 of weak-baseline agent using Kalman filter denoising with Process variance 5x10^{-3} and Measurement variance 10^{-3}

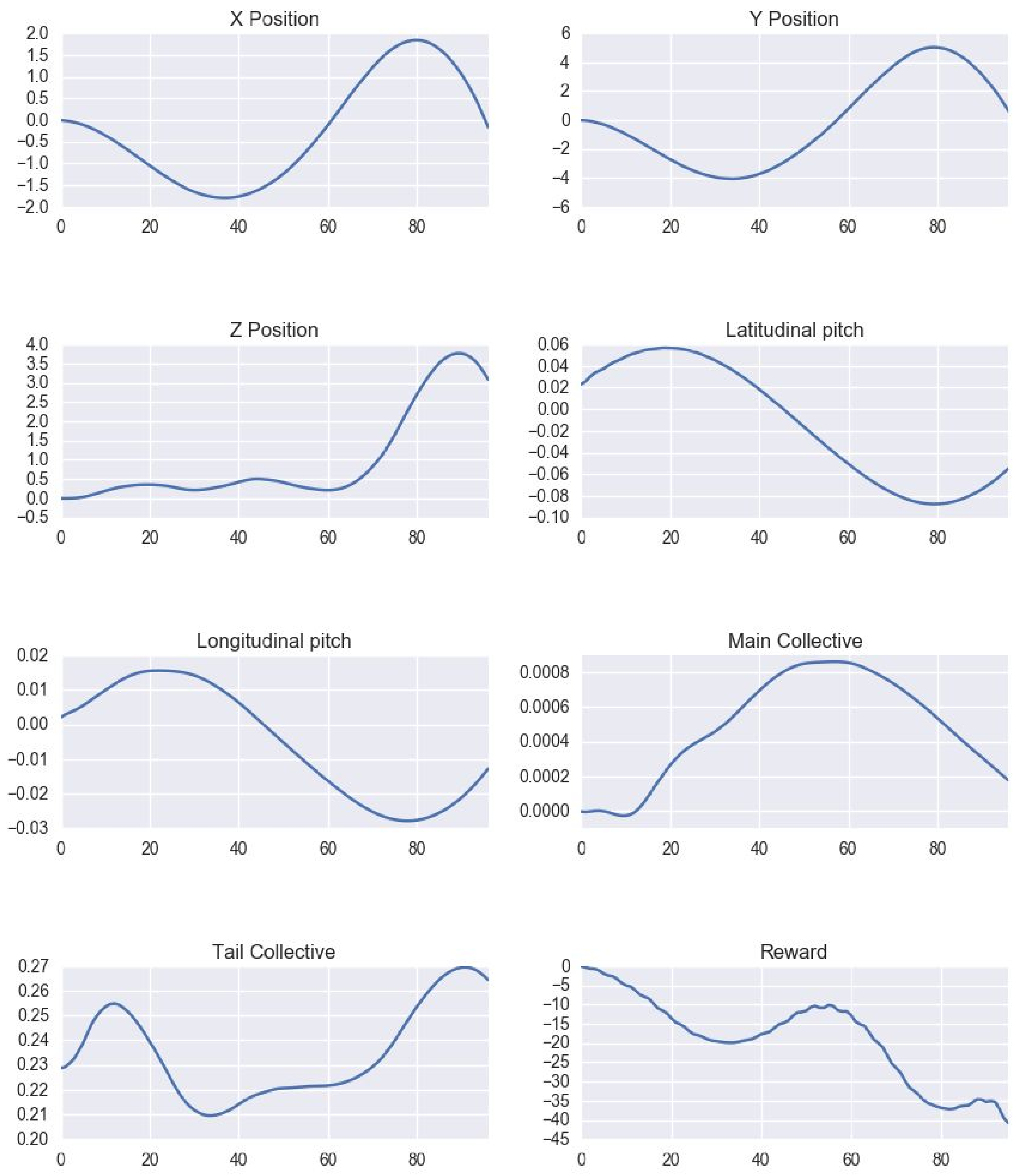


Figure 8. Plot of positions, actions and reward for Task 1 of weak-baseline agent using Kalman filter denoising with Process variance 5x10^{-11} and Measurement variance 10^{-5}

4.2 Deep Deterministic Policy Gradient agent

This agent is a learning agent based on the DDPG algorithm described in section 3. For the implementation of neural networks Keras library with Tensorflow as backend was used. All experiments were run system with i7 processor and 8GB RAM. For the sake of simplicity Ornstein-Uhlenbeck Process was not followed and hence noise was not added to the predicted actions.

Actor network consisted of 1 input layer, 2 hidden layers and 1 output layer. Input layer have 12 neurons corresponding to 12 observation states. Hidden layers have 300 and 600 neurons respectively. Output layer have 4 neurons corresponding to 4 actions. Tanh activation was used for the output layer because the range of actions was from -1 to 1.

Critic network consisted of 1 input layer, 2 hidden layers and 1 output layer. Input layer have 12 neurons corresponding to 12 observation states. Hidden layers have 300 and 600 neurons respectively. According to DDPG paper, the actions were not included until 2nd hidden layer of Q-network. Similar to actor network output layer had 4 neurons with tanh activation. Architecture of agent is depicted in figure 9.

Since learning value functions using large, non-linear function approximators is unstable, hence weights used for initialization of parameters of neural networks play a very important role in convergence. That is why actor network was pre-trained to learn weak-baseline agent's policy function using 10^6 randomly generated samples of observation states. This significantly reduced number of episodes required to train the model.

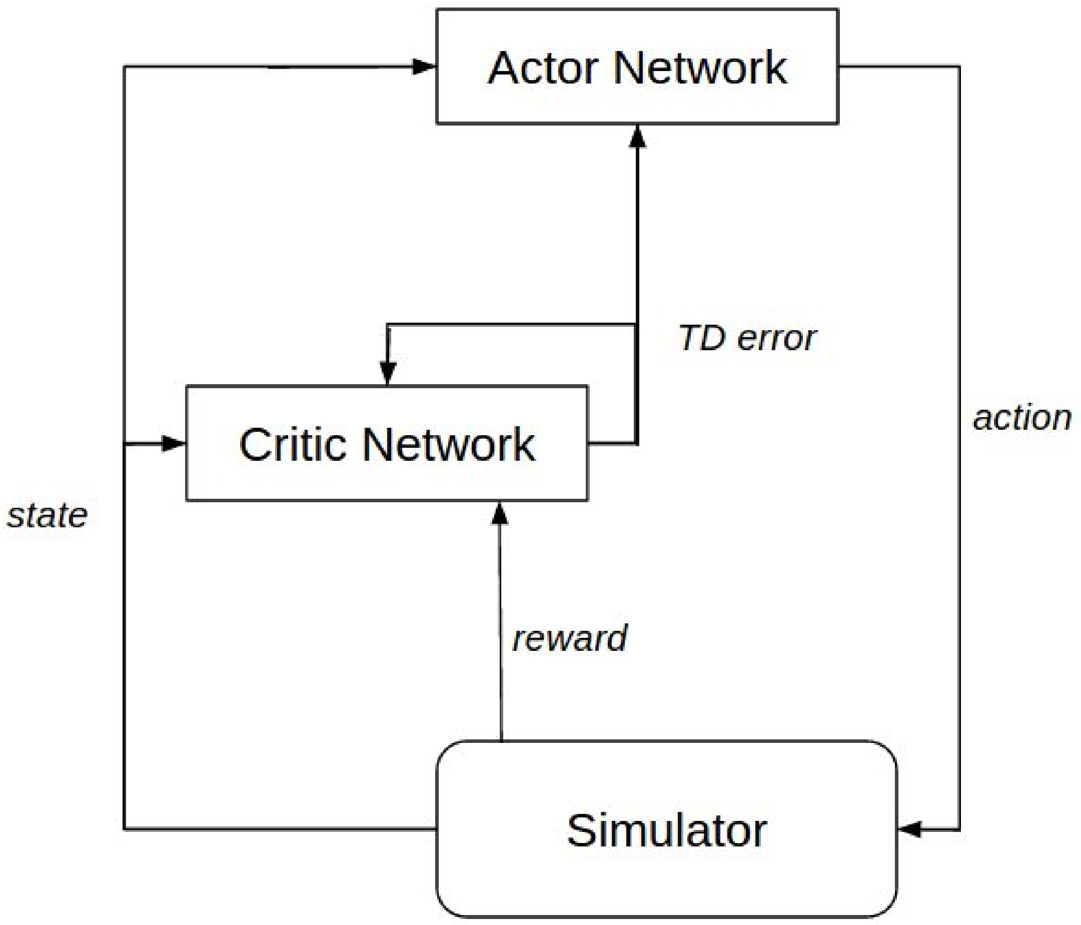


Figure 9. DDPG agent architecture

In order to obtain better results and smoother and steeper learning curve we experimented with transformed reward as shown in figure 10. The modified reward was exponential function of the given reward. The learning curve of DDPG agent for task 0 for both rewards can be seen in figure 11. As expected average reward start with average reward of weak-baseline model described and increases continuously till it reaches convergence around -0.7 in around 500 episodes. The modified reward gives better results as shown in figure 11.

$$Transformed\ reward\ (r_t^{'})\ = exp(r_t/r_0) \tag{11}$$

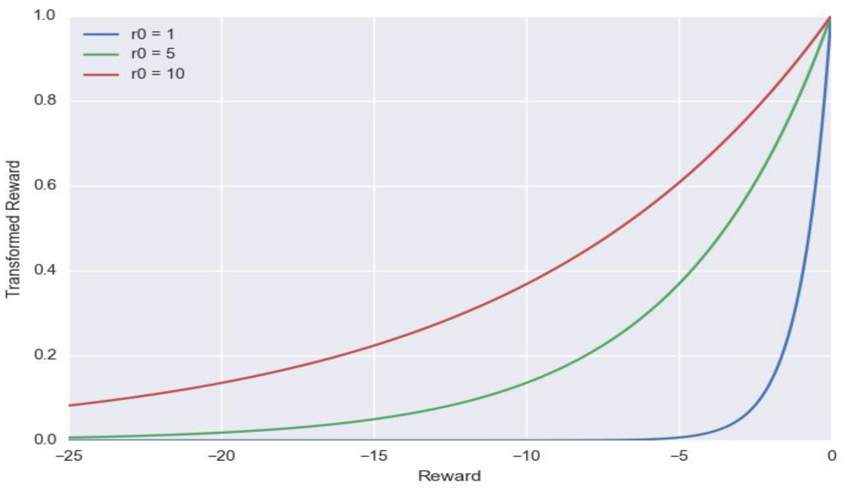


Figure 10. Transformed reward function for different values of r_0.

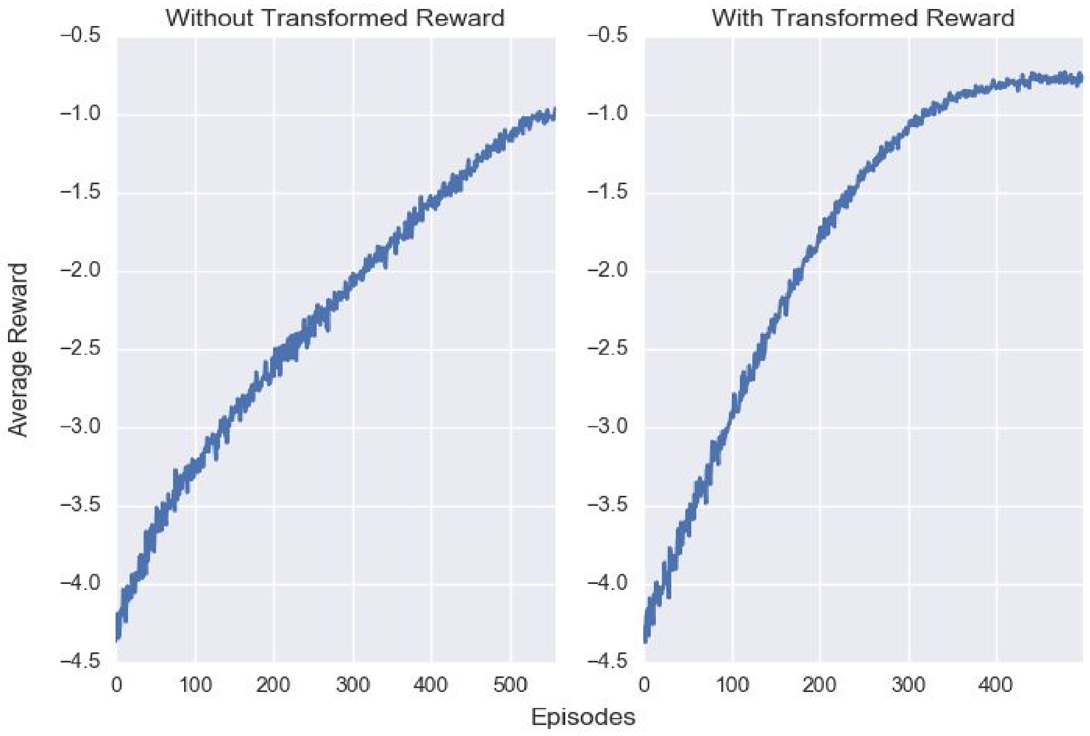


Figure 11. Learning curves of DDPG agent for Task 0.

From figure 12 it can be seen that all the observation states are almost zero and only action 4 (tail rotor collective pitch) is non zero.

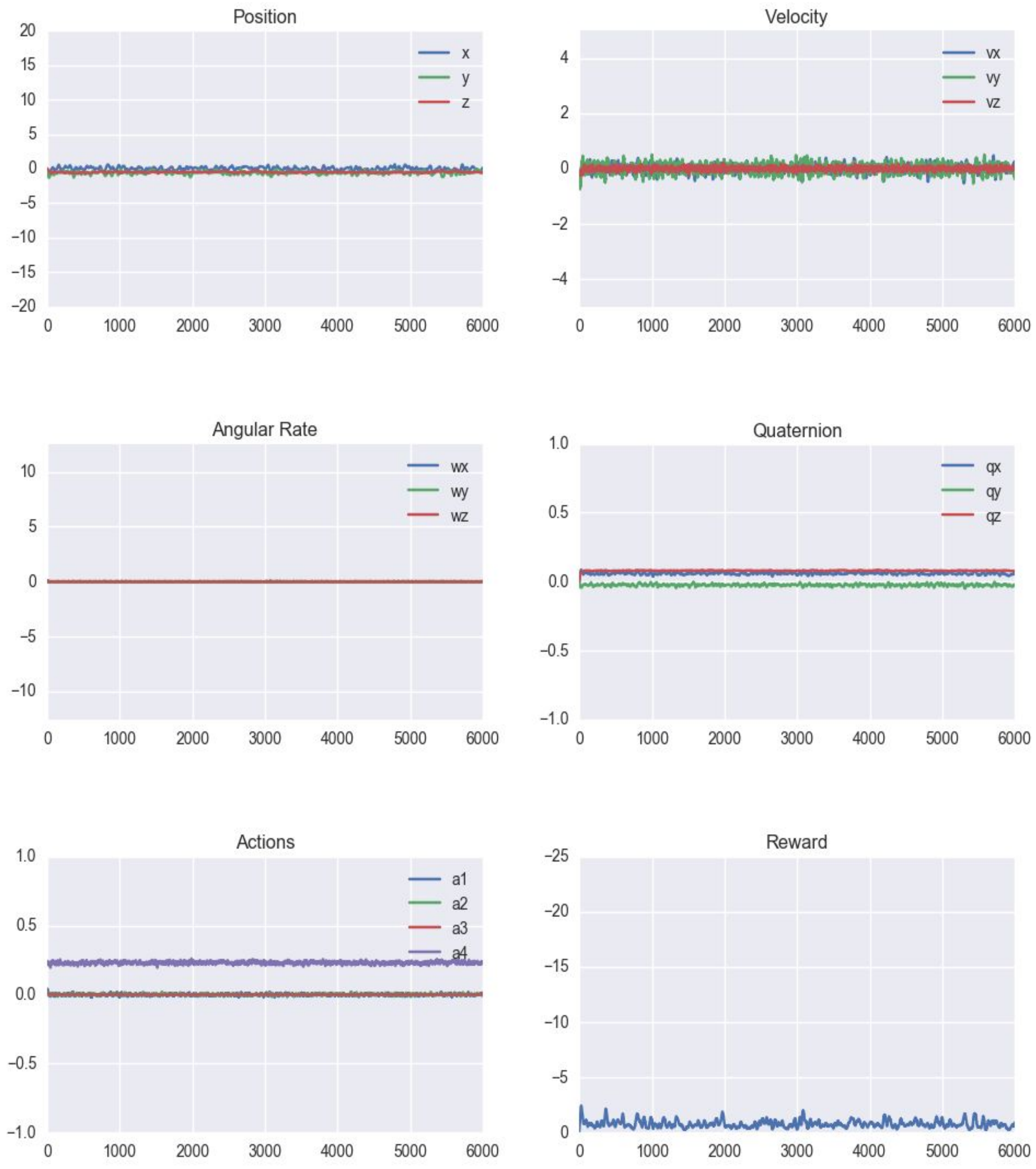


Figure 12. States, actions and reward of trained agent for task 0.

5. Conclusion and Future work

This project explored the use of deep reinforcement learning on the domain of simulated helicopters. Unlike fixed wing aircraft, helicopter control is unstable; if corrective actions are not taken the helicopter will very quickly become unstable and fall from the sky. Helicopter control is a challenging reinforcement learning problem due to multitude of factors. Firstly, the state and actions spaces are both continuous, so there are an infinite number of possible state-action combinations. Secondly, the environment is noisy and stochastic; noisy observations and random wind affect how actions are applied and states are observed. Finally, the entire environment is not observed and non-stationary; the wind effects are not known to the agent.

Future work for this project might include designing actor and critic neural networks more specific to generalized helicopter challenge such as LSTMs which can model sequential nature of sensor data.

6. References

[1] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver & Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971v5, 2016.*

[2] O. Amidi, T. Kanade and J.R. Miller. Autonomous Helicopter Research at Carnegie Mellon Robotics Institute. Proceedings of Heli Japan `98, April, 1998.

[3] R. Koppejan and S. Whiteson. Neuroevolutionary Reinforcement Learning for Generalized Helicopter Control. In GECCO 2009.

[4] A. Asbah, A. M. S. Barreto, C. Gehring, J. Pineau and D. Precup. Reinforcement Learning Competition: Helicopter Hovering with Controllability and Kernel-Based Stochastic Factorization. Proceedings of International Conference on Machine Learning (ICML), Reinforcement Learning Competition Workshop, 2013.

[5] J.A. Martin H. and J. de Lope. Learning Autonomous Helicopter Flight with Evolutionary Reinforcement Learning. EUROCAST 2009, LNCS 5717, pp. 75-82, 2009.

[6] P. Abbeel, A. Coates, T. Hunter and A.Y. Ng. Autonomous Autorotation of an RC Helicopter. In 11th International Symposium on Experimental Robotics (ISER), 2008.

[7] P. Abbeel, A. Coates, M. Quigley and A. Ng. An Application of Reinforcement Learning to Aerobatic Helicopter Flight. NIPS 2006.

[8] A.Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Autonomous Inverted Helicopter Flight via Reinforcement Learning. 11th International Symposium on Experimental Robotics (ISER), 2004.

[9] A.Y. Ng, H.J. Kim, M. Jordan and S. Sastry. Autonomous Helicopter Flight via Reinforcement Learning. NIPS 2004.

[10] B. Tanner and A. White. RL-Glue: Language-Independent Software for Reinforcement Learning Experiments. Journal of Machine Learning Research, 10(Sep):2133-2136, 2009.

[11] A. White. RL-Glue 3.04 Overview Manual. URL: http://rl-glue.googlecode.com/svn/trunk/docs/html/index.html. Accessed July 2014.

[12] Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In ICML, 2014.