International Journal of Advanced Culture Technology Vol.8 No.4 51-57 (2020)
DOI https://doi.org/10.17703/IJACT.2020.8.4.51

IJACT 20-12-7

Balancing a seesaw with reinforcement learning

[1]Ts.Tengis, [2*]L. Uurtsaikh, [3*]A. Batminkh

Mongolian University of Science and Technology,
School of Information and Communication Technology
tengis@must.edu.mn, uurtsaikh@must.edu.mn, abatmunkh@must.edu.mn

Abstract

A propeller-based seesaw system is a system that can represent one of axis of four propeller drones and its stabilization has been replaced by intelligent control system instead of often used control methods such as PID and state space. Today, robots are increasingly use machine learning methods to adapt to their environment and learn to perform the right actions. In this article, we propose a Q-learning-based approach to control the stability of a seesaw system with a propeller. From the experimental results that it is possible to fully learn the balance control of a seesaw system by correctly defining the state of the system, the actions to be performed, and the reward functions. Our proposed method solves the seesaw stabilization.

Keywords: *learning, control system, sensors, microcontroller, reward, robot*

1. Introduction

In recent years, the region of drone applications has been increasing. Following this, many ways to control it are evolving. A linear dynamic model of quadcopters is shown in [1, 2, 3]. The control methods include classical methods such as PID, and more advanced methods such as state feedback and LQR [4, 5]. Consider a case where it is difficult to model system dynamics completely mathematically, or if a linearized model leads to a non-linear dynamic model of the system inaccurate. In this case, it is difficult to manage using traditional control techniques. This can be improved through machine learning. In this study, we tested the quadrotor's single-axis system using one of machine learning methods, reinforcement learning, with the goal of learning to balance.

2. Reinforcement learning method

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. This method is more suitable for robot control [1, 13, 14, 15, 16, 17]. The structure of the reinforcement learning method is shown in Figure 1.

Manuscript received September 28, 2020 / revised: October 19, 2020 / accepted: October 26, 2020
Corresponding Author: tengis@must.edu.mn
Author's affiliation
Mongolian University of Science and Technology

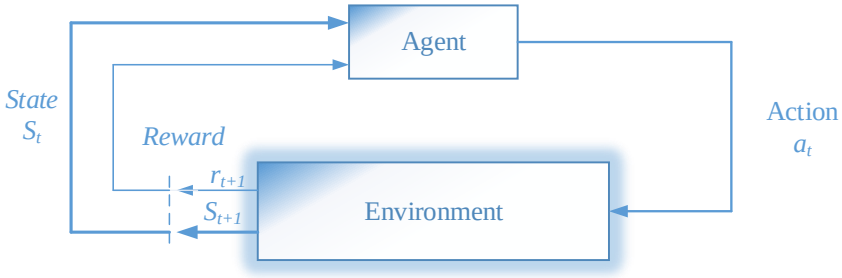


Figure 1. The reinforcement learning method

Here, a large number of *S* states represent the environment and the agent or robot selects one of the fixed operations, A, and performs a corresponding operation in discrete time steps. In other words, for every time step *t*, the agent observes the state of the environment s_t and selects the action a_t. Upon completion of the action, the agent receives a r_{t+1} reward, which indicates how well the action has been performed in a short period of time and the agent observes the new s_{t+1} state of the environment. The goal of a reinforcement learning agent is to learn a policy which maximizes the expected cumulative reward. In this study, we used the *Q*-learning method. The optimal value function of *Q*-learning is defined as follows (1).

$$Q^*(s,a) = E[R(s,a) + \gamma \max_a Q^*(s',a')] \quad (1)$$

Equation (1) represents the expected value of the transition to the state *s*', performing action *a*, in the state *s*. The parameter γ is called the discount rate ($0 < \gamma < 1$) and determines how important the future reward will be. Once the optimal *Q* function is defined as *Q*(s, a)*, it is easy to calculate the optimal policy $\pi^*(s)$ by selecting the maximum value of all operations from that state (2).

$$\pi^*(s) = \arg \max_a Q(s,a) \quad (2)$$

The *Q*-function is usually stored in a table and indexed by state *s* and action *a*. From any states we can determine the optimal *Q* function through the repeated observation of environment. Each time a robot performs an action *a*, sequence of experiments (s_t, a_t, r_{t+1}, s_{t+1}) is created. Update the cells of the table for state *s* and action *a* as follows (3).

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_a Q^*(s',a') - Q(s_t,a_t)) \quad (3)$$

This eventually converts to the optimal Q function. This learning method is sometimes described as a control method based on the reward function *R(s, a)* [7, 8, 9]. Instead of developing lower-level control calculations, we can develop and implement higher-level tasks in the form of reward functions. In most cases, the robot's tasks and rewards correspond to the physical activity of the environment. In addition, the reward function can be defined as sparse or dense. The sparse reward function is zero everywhere except for a few places, while the dense reward function provides more information after each action, but it is more difficult to construct than the sparse function [2]. The *Q*-learning method requires discrete states and corresponding actions. In this case, it is not possible to reveal the intermediate hidden states or to learn them properly enough. In some studies, states and actions are estimated to be continuous [6]. In the case of *Q*-learning, the only way to explore information about the environment is to observe the response of the environment to an action [9, 10, 11, 12]. At the beginning of the learning, the system does not have any specific information about the environment at all, so the system requires some random or forced action. After each action reward information is collected and *Q*-learning improves the value function.

3. Implementing reinforcement learning to the model

The system design of our research is shown in Figure 2. We designed this system to learn to balance at 0 degrees (horizontal overlap position). The learning software for this system is implemented on the Atmel SAM3X8E ARM 32-bit microcontroller. The angle sensor was solved using a potentiometer. There are two 2000KV propelled brushless motors with 30A ESC on the edges of the axis.

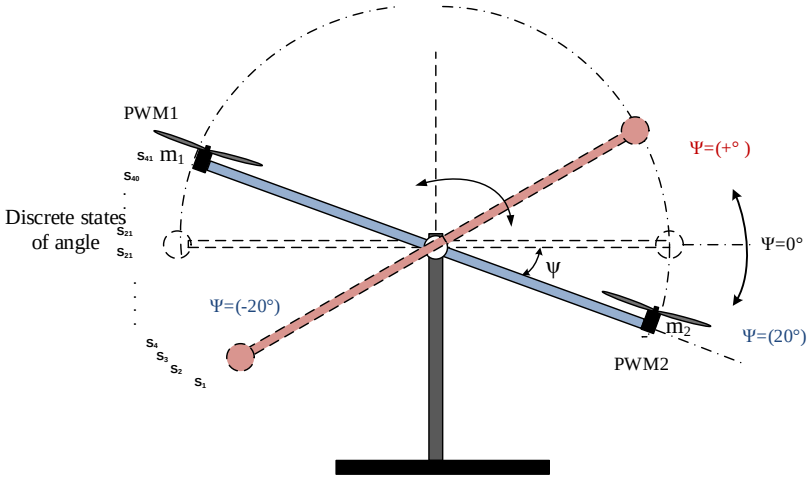


Figure 2. An unbalanced system with two propellers.

One axis of the quadcopter was used as the study model. The set of states *S* of this system is defined by the angle ψ and the angular velocity ψ '. The seesaw swings at ±20º intervals and the angular velocity varies at ±40º/sec intervals. The angle was taken with one-degree discrete step and the angular velocity is discretized at 9 levels. As a result, the *Q* table of our system will be 41x9, as shown in Table 1.

Table 1. Q table of the system

Angle/ angular speed	-20	-18	-17	....	0	1	...	19	20
-4									
...									
0									
..									
4									

Action A is defined as the set of actions corresponding to each state of the system, which is the PWM signal that changes the propeller rotation speed. A total of 9 discrete operations were selected to be able to change the state of the system, as shown in Table 2

Table 2. Discrete actions

Action	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
PWM1	800	825	850	875	900	900	900	900	900
PWM2	900	900	900	900	900	875	850	825	800

The model uses an ESC-controlled brushless DC motor, which is capable of changing the speed every 20 ms. It is possible to control the motor speed by sending a PWM signal to the ESC. The minimum value of the PWM signal is 770 microseconds and the maximum value is 2000 microseconds. In other words, we are able to change motor speed with 1 microseconds accuracy. It is seen that we have a wide choice for action A. In other words, it is possible to select 1230 operations for one motor in each state s_t, but in this case the learning process will be complicated by the huge size of the Q table. Depending on the selected action, the transition time from one state to another will change. In other words, it is necessary to observe a new state after performing any action, and we have chosen this time to be 20 milliseconds. We chose the discount rate $\gamma = 0.7$ and the learning rate $\alpha = 0.1$.

The following is our program algorithm:

1. *Load Q-table with the initial value of 0*
   while (1)
2. *Initialize counter t for random action selection*
3. *When **for (t >1)** select a random action from the state s_t **else** execute action* $a_t = {}^{max}_{a} Q(s, a)$
4. *Wait **20 milliseconds***
5. *Observe and record the new state $s_{(t+1)}$ and calculate the reward function R (s, a)*
6. *Update the Q-table* $Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma {}^{max}_{a} Q^*(s', a') - Q(s_t, a_t))$

4. Test results

The Q matrix of our system is [9x41x9] and includes 9 types of angular velocity ψ ', 41 types of angles ψ, and 9 types of A actions. Each cell in the matrix stores a 32-bit float value, which means we need at least 13 Kbytes of memory. As the number of states increases and the number of operations increases, the amount of memory increases.

This learning method is called reward-based learning and the learning process is guided by a well-defined reward function. In our experiment, several types of reward functions were considered.

Experiment 1. The experiment was performed using a sparse reward function.
if (ψ==0 && $\psi' == 0$)
Reward=1;
else
Reward=0;
In this case, the system performed each operation on one state, and the system could not exit the initial state because the value of the reward did not change. In other words, we have a one goal state, and all other states will not be rewarded.

Experiment 2. The experiment was performed using a dense reward function.
*Reward =abs(ψ) *(-1);*

In this case, the system receives the highest negative reward (penalty) at -20 degrees and +20 degrees, and 0 reward at 0 degrees. During this test, the system did not stabilize in one position, it was constantly swaying in two directions, unable to maintain balance.

Experiment 3. The experiment was performed using angular velocity into the reward function.
Reward =(ψ)+ ψ';

For this reward system, angular velocity is included. In order to find equilibrium, it is shown in the ψ = -1 and ψ = 1 states, the operation with the highest angular velocity is selected., and at ψ = 0 state is seen to be passing at high speed.

Experiment 4. The experiment was performed using a reward function with a penalty.
Reward =1-abs((ψ)+ ψ');
if (abs(ψ)==20)
Reward=-100;
if(abs(ψ)<=3)
Reward=100

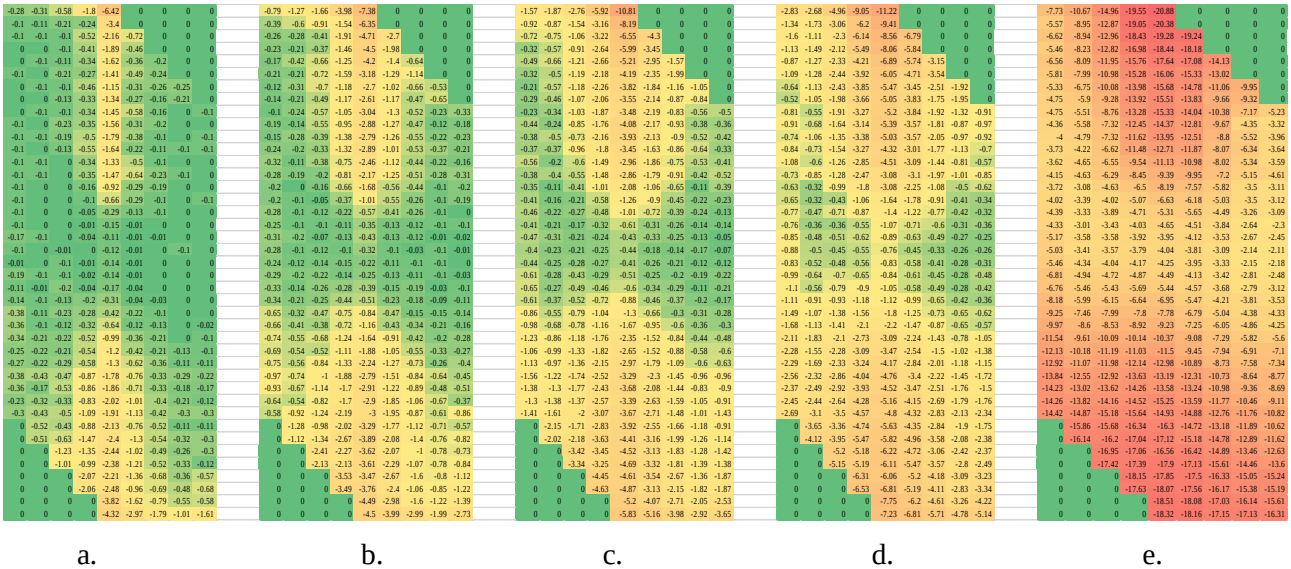
For this reward system, we have included a penalty value, and if the seesaw angle is large and the lower the angular velocity, the greater the negative reward. The larger the angle and the higher the angular velocity, the lower the negative reward. If the angle is small and the angular velocity is high, the negative reward is large.

If the angle size is small and the angular velocity is low, the negative reward will be less. In this experiment, we achieved our goal.

In addition, experiments have shown that it is necessary to take some delay after each action. From the s_t state, we select the highest value action a_t and perform it. After that, we read the new state s_{t+1}. If there is not enough time to complete the operation, the state cannot be changed, and if many operations are performed on this state, Table *Q* will be falsely updated. Since the ESC device operates at 50 Hz, we have a 20ms delay.

Table 3a shows the results of the values of the *Q* table after first 60,000 iterations. The rows in this table show the ψ state of the angle, and the column shows the value of the angular speed ψ'. Negative rewards are shown in red and positive rewards are shown in green. Table 3e shows the resulting values of the *Q*-table after 1,000,000 iterations.

Table 3. Q values of the system



a. b. c. d. e.

Table 4 shows the values of the actions corresponding to each state. The learning time of this system is at least 8 hours. In the first case, each cell in Table *Q* is filled with a value of 0. It is necessary to upgrade and improve the value of each cell in order to find the right action. In some research work suggests ways to reduce learning time [1, 2, 5, 6]. For example, first balancing operation should be done manually and fill each cell in *Q*-Table with close to the correct action value, and then by activating the learning process, the learning speed is increased [2, 5].

Table 4. Action types in each state

angle/ang. speed	-20	-19	-18	-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
-4	8	8	4	4	2	2	4	2	8	1	6	5	2	2	4	5	3	8	8	4	2	6	6	7	5	7	4	6	6	7	5	3	9	1	1	1	1	1	1	1	1
-3	9	5	7	4	7	8	7	5	8	2	3	8	5	3	7	3	6	3	3	8	8	4	1	3	6	8	4	8	4	1	3	1	3	6	3	1	1	1	1	1	1
-2	9	4	6	2	8	3	2	5	9	9	8	4	6	8	3	5	2	8	3	5	4	9	4	4	4	7	8	2	5	1	9	3	1	3	6	2	5	1	1	1	1
-1	7	6	5	4	8	3	1	5	9	3	3	7	4	7	5	8	4	7	2	5	5	9	9	7	9	1	4	9	3	1	2	6	5	8	3	6	4	1	4	1	1
0	7	3	1	7	5	5	3	5	6	8	7	1	9	4	3	6	9	1	6	7	1	9	6	2	7	7	6	3	1	3	4	1	7	5	6	3	6	4	4	2	2
1	1	1	6	1	7	4	1	1	2	8	9	8	3	8	9	1	7	7	5	1	3	8	3	9	8	1	7	2	2	1	8	6	1	3	3	4	9	3	6	4	5
2	1	1	1	1	1	5	4	1	3	8	8	1	4	5	6	7	2	2	7	3	5	1	3	8	7	4	7	1	9	6	9	4	3	9	1	2	3	4	1	1	3
3	1	1	1	1	1	1	1	6	8	5	5	5	5	8	3	8	2	2	5	6	3	5	3	3	3	9	6	2	1	4	9	1	3	2	1	9	4	4	3	2	8	3
4	1	1	1	1	1	1	1	1	6	4	2	3	1	2	9	3	9	2	2	5	7	1	4	1	2	4	6	8	3	6	9	6	3	7	1	8	1	8	5	3	1

Figure 3 shows a timeline of the achievement of the target state in order to monitor the learning process and number of learning process.

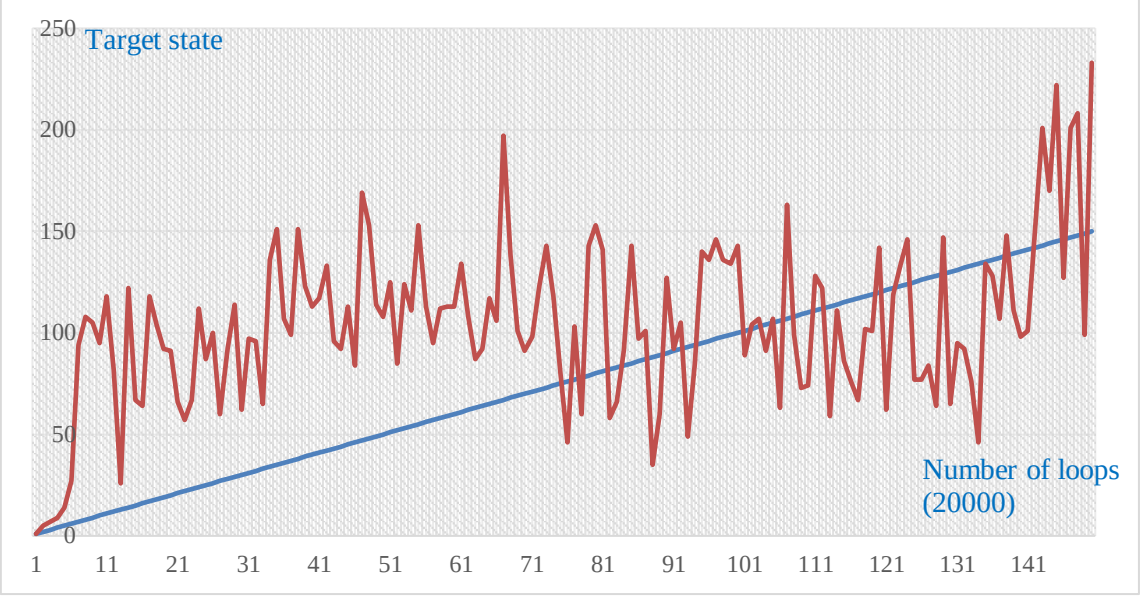


Figure 3. Number of targets is reached.

Figure 4 shows the model of the real training system.

Figure 4. Real model.

5. Conclusion

This article presents the results of the reinforcement learning method in propelled seesaw model. Without any mathematical model of system dynamics, it is possible to create a control that can balance. The learning was conducted under microcontroller-based control, which requires large amount of memory. We discretized the state and action values due to these hidden actions and states are not defined. When a discrete action is selected, it is necessary to select the correct time to delay of the action. The training was conducted using sparse and dense reward functions and the reward function needs to be very well organized. The learning outcomes vary depending on the reward function. The disadvantage of this method is long learning period.

In the future, we are working to test methods to reduce (boost) learning time.

References

[1] J Kober, JA Bagnell, J Peters, “Reinforcement learning in robotics: A survey”, The International Journal of Robotics Research Journal, pp 1238-1274, 2013

[2] Shayegan Omidshafiei, “Reinforcement learning-*b*ased quadcopter control”, 2013

[3] Amar Batmunkh, Tserendondog Tengis "*State feedback control simulation of quadcopter model*" IFOST 2016,

[4] Ц. Тэнгис, А.Батмөнх “*Disturbance Rejection Control for Unbalanced Double-Propeller System on Single Axis*” Хүрэлтогоот 2017,

[5] WD Smart, LP Kaelbling, “Effective reinforcement learning for mobile robots”, Proceedings 2002 IEEE International Conference on Robotics and Automation

[6] WD Smart, LP Kaelbling, “Practical reinforcement learning in continuous spaces”, PICML, 903-910, 2000

[7] A. D. Dubey1 , R. B. Mishra2 and A. K. Jha, “Path Planning Of Mobile Robot Using Reinforcement Based Artificial Neural Network” International Journal of Advances in Engineering & Technology, May 2013. ISSN: 2231-1963

[8] Jemin Hwangbo1 , Inkyu Sa2 , Roland Siegwart2 and Marco Hutter1, “Control of a Quadrotor with Reinforcement Learning”, IEEE robotics and automation, june, 2017

[9] W Koch, R Mancuso, R West, A Bestavros, “Reinforcement Learning for UAV Attitude Control”, ACM Transactions on Cyber-Physical Systems, 2019

[10] C Gaskett, L Fletcher, A Zelinsky, “Reinforcement learning for a vision based mobile robot”, Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots

[11] A El Assad, E Fournier-Bidoz, P Lachevre, J Sagastuy, “Inverted Pendulum on a Quadcopter: A Reinforcement Learning Approach”, CS229 - Final Report, 2017

[12] M Asada, S Noda, S Tawaratsumida, “Vision-based reinforcement learning for purposive behavior acquisition”, Proceedings of 1995 IEEE International Conference on Robotics and Automation

[13] MG Lagoudakis, “Balancing and Control of a Freely-Swinging Pendulum Using a Model-Free Reinforcement Learning Algorithm”, Duke University, 1999

[14] F Amigoni, A Bonarini, G Fontana, M Matteucci, V Schiaffonati, “Batch reinforcement learning for controlling a mobile wheeled pendulum robot”, IFIP International Conference on Artificial Intelligence in Theory and Practice, pp 156-160, 2008

[15] T Hester, M Quinlan, P Stone, “RTMBA: A real-time model-based reinforcement learning architecture for robot control”, 2012 IEEE International Conference on Robotics and Automation, pp 85-90

[16] Mohammad Abdel Kareem Jaradat, Mohammad Al-Rousan, Lara Quadan, “Reinforcement based mobile robot navigation in dynamic environment”, Robotics and Computer-Integrated Manufacturing, pp 135-149, 2011

[17] R Figueroa, A Faust, P Cruz, L Tapia, R Fierro, “Reinforcement learning for balancing a flying inverted pendulum”, Proceeding of the 11th World Congress on Intelligent Control and Automation, pp 1787-1793, 2014