Powered Landing Guidance Algorithms Using Reinforcement Learning Methods for Lunar Lander Case

Larasmoyo Nugroho[1], Novanna Rahma Zani[2], Nurul Qomariyah[2]
Rini Akmeliawati[3], Rika Andiarti[4], Sastra Kusuma Wijaya[5*]

[1]Rocket Technology Center, National Institute of Aeronautics and Space (LAPAN), Indonesia
[2]Electronics Department, Politeknik Elektronika Negeri Surabaya, Indonesia
[3]School of Mechanical Engineering, University of Adelaide, Australia
[4]Deputy of Aerospace Technology, National Institute of Aeronautics and Space (LAPAN), Indonesia
[*5]Physics Department, Universitas Indonesia, Depok, Indonesia

e-mail: larasmoyo.nugroho@lapan.go.id,
novannarahmazani@gmail.com, nurulnq11082000@gmail.com,
rini.akmeliawati@adelaide.edu.au, rika.andiarti@lapan.go.id,
*corresponding e-mail: skwijaya@sci.ui.ac.id

Received: 23-03-2021. Accepted: 22-04-2021. Publishied: 30-06-2021

Abstract

Any future planetary landing missions, just as demonstrated by Perseverance in 2021 Mars landing missions require advanced guidance, navigation, and control algorithms for the powered landing phase of the spacecraft to touch down a designated target with pinpoint accuracy (circular error precision < 5 m radius). This requires a landing system capable to estimate the craft's states and map them to certain thrust commands for each craft's engine. Reinforcement learning theory is used as an approach to manage the mapping guidance algorithm and translate it to engine thrust control commands. This work compares several reinforcement-learning-based approaches for a powered landing problem of a spacecraft in a two-dimensional (2-D) environment and identifies their advantages or disadvantages. Three methods in reinforcement learning, namely Q-Learning, and its extensions such as DQN and DDQN. They are benchmarked in terms of rewards and training time needed to land the Lunar Lander. It is found that the Q-Learning method also called Heuristic produced the highest efficiency. Another contribution of this paper is to show the combination usage of online weights θ between the action selection process and action evaluation process, yields a higher reward, instead of separating them, which significantly enhances their optimization performance. The simulations of the powered guidance performance in a 2-D simulation environment highlight the effectiveness of DQN to handle multiple neural networks better than DDQN.

Keywords: *planetary landing, lunar lander, Q-Learning, DQN, DDQN.*

Nomenclature

QL	=	Q-Learning Method also called as Heuristic Method
DQN	=	Deep Q-Learning Network
DDQN	=	Double Deep Q-Learning Network
MDP	=	Markov Decision Process
TD	=	Temporal Differential

1. Introduction

Nowadays, landing rockets are already part of our daily life. Back then, this feat of engineering even at the level of concept seemed very inconvenient and mind-grinding to be fulfilled. Nevertheless, constant human's eagerness and experiences gained in interplanetary missions gave a stream of new technical insights and increasing know-hows. One such case is the landing mission of Apollo's Lunar Lander Module named as the Eagle. The success of the Eagle's landing mission on the moon has become a solid reference for rocket technologies in landing on earth's surface that eventually adopted and mastered by Space-X to land Falcon rocket as reusable launch vehicle and Starship as their next RLV. (Klumpp, 1974)(Latyshev et al., 2021)

Future highly anticipated, science-driven, robotic and human missions to Mars will require a high degree of landing accuracy. Surely, the next generation of Mars landers will necessitate more advanced guidance and control capabilities to satisfy the increasingly stringent accuracy requirements driven by the desire to explore Mars regions that still necessitated to be researched further intensively. Probably the most demanding mission segments for a Mars mission is the powered descent phase, where the goal is to achieve a soft pinpoint landing, which we will define as the norm of the position error less than 5m and the magnitude of the landing velocity below 2 m/s, with the velocity vector directed primarily downward, and negligible deviation from an upright attitude and zero rotational velocity. In a powered descent phase, a few of the lander's sensors such as radar altimeter are blinded and hindered by the heat shield, therefore the lander's guidance, navigation, and control system should quickly estimate the state of the lander before drifted several km downrange and crossrange, and subsequently estimates states again to attain a soft landing at the designated position, usually under one minute after the heat shield is jettisoned. (Gaudet, Linares, & Furfaro, 2018)

Landing accuracy is important for several reasons. First, given accurate high-resolution maps of the Martian surface, the pinpoint-landing problem subsumes the hazard avoidance problem, as a hazard-free landing site can be targeted. Second, dropping a rover nearest to a specific landing zone could avert malfunctioning rover due to large diverted distance. In fact, some terrain conditions could prohibit the rover to go through unless the spacecraft lander delivered the rover with a high degree of accuracy. As a plus, reducing the distance the rover travels could relax the design requirements for the rover, especially battery in rover mass. Of course, accurate landing to just a few meters could reduce mission risk and expand the feasibility of missions. (Gaudet, Linares, & Furfaro, 2018)

2. Methodology

The Lunar Lander project is a simulation to solve the challenges of a lunar lander using reinforcement learning (RL). Open AI Gym is an open-source platform that provides an environment for applying RL to many classic 2-D engineering problems such as the Cart Pole and Lunar Lander. In this paper, we observe the performance of a few reinforcement learning (RL) algorithms in a specific environment that is OpenAI Gym's Lunar Lander-v2 continuous state space.

Generalized reinforcement learning algorithms such as Q-Learning do well in finding optimal policies for stochastic Markov Decision Problem (MDP) with discrete state and action spaces, nevertheless, problems arise when state spaces are continuous. (Von Dollen, 2017)

2.1. Related Works

The adaptation of Q-Learning that are well suited for discrete states problem into continuous states problem usually lies in the overestimation of actions values that produced high positive bias of the Q-learner in approximating the maximum expected value using the maximum action value. (Von Dollen, 2017)

Neural networks and other estimators could be used as a strategy to estimate value functions, besides binning the continuous state space into discrete variables.

Hado van Hasselt introduced the concept of Double Deep Q-Learning Network (DDQN) where two estimators are introduced to separate the concerns of the expected value from the maximization of the action values, and while this method may show an underestimation of expected values, it could help the agent to yields convergence to optimal policies faster in comparison to Deep Q-learning (DQN). (Hasselt et al., 2016)

The environment starts by sending state values to the agent, whose knowledge will then become a reference for selecting an action in response to the state. After that, the agent will update his knowledge with the reward value obtained from the environment to evaluate his last action. Terminal state value is sent by the environment when a predetermined episode is reached by the loop. In reinforcement learning, the selection of actions for each state value obtained during learning greatly affects the success of learning. Random selection of actions is usually called exploration while selecting actions based on the largest value function is called exploitation.

There is a case in the lunar lander landing motion, which is the part of the rewards obtained during training that will affect the landing lunar lander movement. This landing motion can be arranged using several methods in reinforcement learning. Reinforcement Learning assumes that there are agents located in the environment (Ermacora et al., 2016)(Brockman et al., 2016). The agents are required to continuously update their own beliefs about the world, about which actions are good and which are not in order to become an example of an efficient learner (Cini et al., 2020). Some of the reinforcement learning methods that will be discussed and implemented in the lunar lander's mission are the DQN and DDQN. The results of the training from the implementation of several of these methods will be compared to find out what method is the most optimal in carrying out the landing mission.

2.2. Problem Definition

Our problem is based on the OpenAI Gym's LunarLander-version 2 environment. OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms exerted to a physically rigid body in a 2-dimensional space environment. In particular, this project focuses on the Apollo's Eagle Module landing problem, and the goal is to teach the "lunar lander" agent to successfully land on randomly generated surfaces of the "moon". The following formally defines the state space, action space, and the reward for this RL problem, according to (Mnih et al., 2015).

State-space. The state-space for LunarLander-v2 is an eight-dimensional vector.

1. providing information of the agent's position in space, X and Y
2. horizontal (Vx) and vertical velocity (Vy)
3. pitch orientation in space (θ)
4. angular velocity ($\dot{\theta}$)
5. two flags left_leg, right_leg indicating whether the left foot/right foot is in contact with the ground.

Action space. The input actions provided by the RL agent comprised of firing a thruster to the left or the right, burning the center main engine, or shutting it off, with a total of 4 discrete input actions.

Environment. The environment has a landing pad at coordinate point (0,0) in the frame. Fuel is infinite. (Yu, 2019)

Reward. The agent receives -0.3 points for firing its main engine for each frame, and landing on the landing pad is 100-140 points, which can be lost if the agent moves away from the pad. 10 points will be rewarded when each leg contact with the ground. The episode is terminated if the agent lands or crashes, in which case receiving -100 points for a crash or +100 for a successful landing. 100 to 140 points will be rewarded when the lander moves from the top of the screen to the landing pad, and the variation ranges depending on the lander's positioning on the pad.

Goal. The Markov Decision Problem is considered solved and achieved a score of 200 points or higher on average over 100 consecutive landing attempts.

Remarks. The total maximum reward attainable is about 260 per episode. Even sometimes, 310+ reward could be reached.

Difficulty: The LunarLander-v2 continuous state must adapt in real-time the changes in the shape of the landing surface every time. The lander agent must learn to land in all situations.

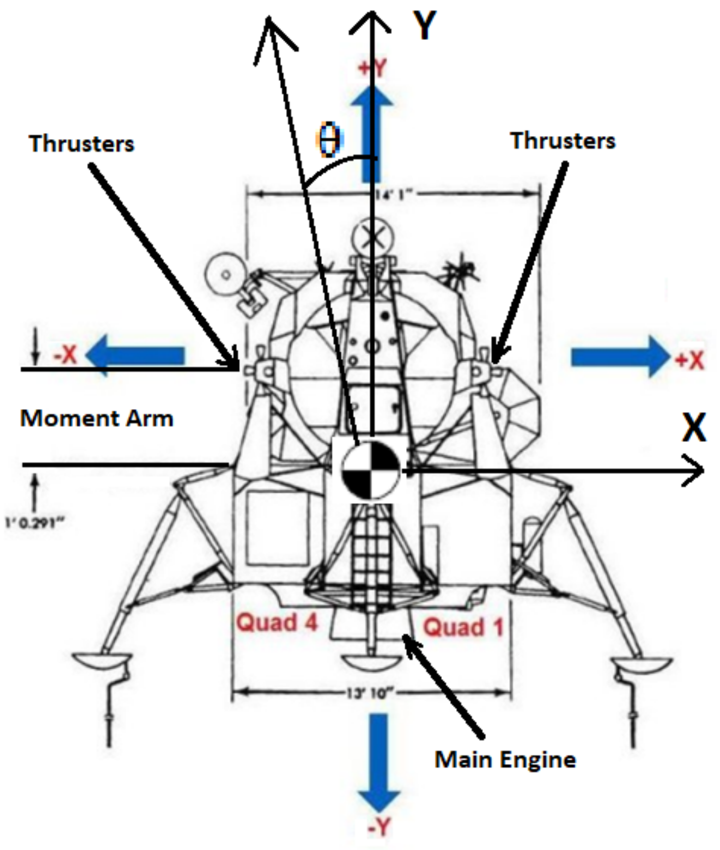

Figure 2-1: Eagle lunar lander module (Meen, 2020).

2.3. Reinforcement Learning Methods

Reinforcement Learning (RL) is an important method for solving the Markov decision process (MDP) problem (Sutton & Barto, 1998). In Reinforcement Learning, agents learn mappings from state environments to actions. Through interaction with the environment, agents obtain reward signals and change behavior selection policies. Therefore, agents can obtain the greatest rewards from the environment and maximize long-term accumulative rewards.

The RL agent interacts with the environment from time to time. At any time t, the agent receives state s_t in state space S and selects action a_t from action space A, based on mapping policy $\pi(a_t|s_t)$. This behavior namely agent behavior is mapping from state s_t to actions a_t, receiving scalar reward r_t by reward function R(s, a), and transitioning to next state s_{t+1} using state transition probability $P(s_{t+1} | s_t, a_t)$, following the environment model. Episodically, rewards are given until the agent reaches the terminal state and then this process continues restarts.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2-1}$$

is the accumulated reward with the discount factor $\gamma \in (0,1)$. The agent aims to maximize the long-term return expectations of each state. The problem is set in a discrete state and action space. It is not difficult to expand it into continuous spaces (Sutton & Barto, 1998).

2.3.1 Q-Learning

One of the most important breakthroughs in reinforcement learning is the development of the Temporal Differential/TD Off-Policy control algorithm known as Q-Learning (Watkins, 1989). Its simplest form called one-step Q-Learning is defined by:

$$\boldsymbol{Q(S_t, A_t)' = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma\, max._a\, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]} \tag{2-2}$$

with

Q'	=	*policy value at the current state*
Q	=	*policy value at the previous state*
R	=	*reward*
α	=	*Learning Rate*, $0 < \alpha \leq 1$, specifies the rate measure at which the old value will be replaced by the new value
γ	=	*Discount rate*, $0 \leq \gamma \leq 1$, determine the value of the reward in the future, the smaller the value, the agent will be more concerned with the immediate rewards, not rewards in the future
$\max_a Q(S_{t+1}, a)$	=	the maximum value of the action-value function at state t+1 for an action a.

In the Q-Learning case, the algorithm attempts to learn a state-action value Q (s, a), whose value is the maximum discounted reward that can be achieved by starting in state x, taking an action a, and following the optimal policy π^* thereafter. for each state-action pair, a separate Q(s, a) value exists when the action space is discrete. (Mitić et al., 2011)

The studied action-value function Q, directly approximates the optimal action-value function Q∗, regardless of the policy followed. This simplifies algorithmic analysis and enables early proof of convergence. The policy still affects the determination of which state-action pairs to visit and update. However, all it takes for that true convergence is that all pairs are constantly being updated. As we observed, this is a minimum requirement in the sense of what method is guaranteed to find the optimal behavior in the general case should require it. Under these assumptions and variants of the usual stochastic forecast conditions on the order of the step size parameters, Q has been shown to converge the probability of Q∗ to 1. The Q-learning algorithm is shown in the procedural form in Table 2-1.

Table 2-1: Q-Learning Pseudocode Algorithm

Q-Learning

```
Initialize Q(s,a)
Repeat:
    Initialize **s**
    Repeat:
        Choose a from s using policy derived from Q (e.g., ξ-greedy)
        Take action a, observe s', r
        $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_a Q(s',a') - Q(s,a)]$
        $Q^* \leftarrow Q$
        $s \leftarrow s'$
    until s is terminal
```

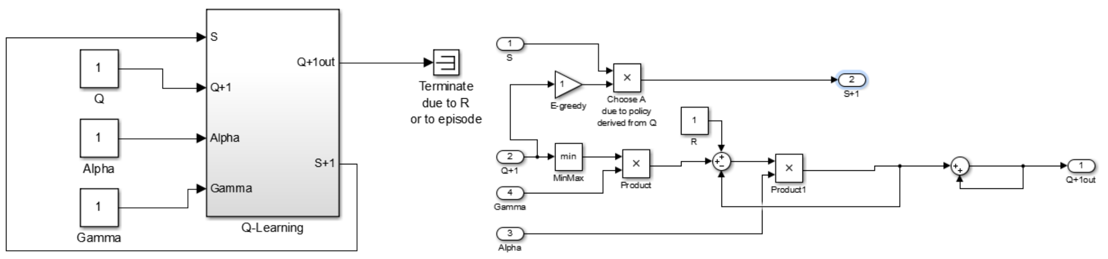

Figure 2-2: Q-Learning control diagram (left) and Bellman equation diagram (right).

2.3.2 Role of Heuristic in Powered Landing Guidance

A heuristic is a rule about the nature of the problem, which is based on a well-known environment model and its purpose is to direct the search towards the goal so that it becomes more efficient. The heuristic function h:(S→R+) given in each state s∈S is an estimate of the distance between the current state and the goal state. The value h(s), the closer s to the goal state, is getting smaller. If s is in terminal state, then h(s) = 0. Therefore Heuristic is literally Q-Learning in the deterministic environment (Basic & Snajder, 2020).

The heuristic control determines the control force, the control force generates the net force, the net force determines the acceleration, the acceleration gradually adjusts to the speed, and the landing velocity value becomes the collision speed. Therefore, it is not an easy task to manage the collision speed around the desired level. In addition, the heuristics to be used must also set the length of time needed to land at a low enough value because the long landing duration requires extensive fuel use. The two criteria, minimizing the speed of the collision and minimizing the length of time it takes to land, contradict each other, which makes the soft-landing problem a challenging task. The control heuristic, which aims to meet these two criteria, should allow the vehicle to descend to the surface rather quickly, but allow it to slow down safely to a low-speed value before landing (Tanyolaç & Yasarcan, 2012).

2.3.3 Enhancement of Q-Learning

Various achievements have been achieved in the study of RL, and some classical algorithms, such as Q-learning (Watkins, 1989), have been introduced. For MDP problems with small state space and action space, the traditional RL algorithm is an effective solution. However, in most cases where the state space or action space is very large or stochastic, it usually makes traditional RL algorithms slow down or fail to converge to optimal policies. Therefore, the introduction of a neural network and parametrization of the value function or policy function is the main solution to extend the capability of the Q-Learning method.

2.3.3.1 Deep Q-Learning Network (DQN)

Neural Network (NN) is a popular non-linear approach function. The Deep Q-Learning Network was proposed by Mnih et al. in 2015 by combining Q-Learning (QL) and Convolutional Neural Network (Bengio, 2013), which effectively blending the value function of reinforcement learning inside the deep neural networks. Two main approaches of Q-Learning enhancement are utilized, the Deep Q-Learning Network (DQN) and Double Deep Q-Learning Network (DDQN), in which the latter used two multi-layered-NN architecture parallel.

The first significant mechanism is the experience replay mechanism, which utilizes previously used samples or reinforcement learning to solve the solution convergency. In addition to this is the target network, which is used to select and evaluate actions and ensure the computational stability of the NN. DQN is well aware that deep reinforcement learning computation extends to research and application of traditional RL, and makes Deep Reinforcement Learning (DRL) a hotspot of artificial intelligence (Sutton & Barto, 1998).

By continuously updating the neural network DQN skip the Q-table to update the Q value but utilizes the neural network to update the Q value and find the optimal path of action. DQN used two neural networks: target-network to get the value of Q for evaluation, and main-network, which is used to get the value of Q with a relatively fixed parameter. (Quan et al., 2020) The mathematical description of the loss function of DQN is depicted as the Eq. (2-3), θ^- is the parameter in the target network

$$\mathrm{L}(\theta) = \mathrm{E}\left\{\left(Q_{target} - Q_{main}(s_t, a_t, \theta)\right)^2\right\} \tag{2-3}$$

$$Q_{target} = r_{t+1} + \gamma \cdot \max_{a_{t+1}} Q_{target}(s_{t+1}, a_{t+1}, \theta^-) \tag{2-4}$$

The same value is used to select and measure any action at once by the max operation in standard Q-learning and DQN. Overly optimistic value estimation may be resulted from this approach. However, the DQN paper ignited the interest of the AI community in the field of deep reinforcement learning and lead to several extensions of DQN.

2.3.3.2 Double Deep Q-Learning Network (DDQN)

In this work, DQN's extension namely Double Deep Q-Learning Network will be implemented and benchmarked to other methods to solve a landing spacecraft task as adapted from previous work by (Kersandt, 2018) for UAVs. In order to avoid overly optimistic value estimation that might occur in DQN and standard Q-Learning, DDQN came into being (Hasselt et al., 2016). This overestimation of values is also called *maximization bias*, where the maximum operator uses the same values for selecting the best action as well as an estimator for the maximum true values. Overestimation will dampen the convergence to the optimal policy (Sutton & Barto, 1998). Separating the process of selecting the optimal action from the estimation of optimal actions is a simple method to reduce the impact of overestimation as much as possible (Hasselt, 2010). Double DQN contains two networks. The policy network is used to select the action (action with best-predicted Q) in real-time. At intervals, models are stabilized to update the target networks. Changes to Q values may lead to changes in closely related states (i.e. states close to the one we are in at the time) and as the network tries to correct for errors it can become unstable and suddenly lose signficiant performance. Target networks (e.g. to assess Q) are updated only infrequently (or gradually), to avoid instability problem.

The amendment from simple DQN is to decouple training of Q for the current state and target Q derived from the next state which is expressed in the DDQN formula as shown by Eq. (2-5) and (2-6). The basis of improving DQN to DDQN is by separating the targeted Q-value,

$$Q_{target} = r_{t+1} + \gamma Q(s_{t+1}, \max_{a_{t+1}} Q(s_{t+1}, s_{t+1}, \theta); \theta^{-}) \tag{2-5}$$

where θ^{-} is the parameter in the target network, from the policy-estimated Q-value,

$$Q_{policy} = r_{t+1} + \gamma Q(s_{t+1}, \max_{a_{t+1}} Q(s_{t+1}, s_{t+1}, \theta); \theta) \tag{2-6}$$

Notice that the selection of the action, in the argmax, is still due to the online weights θ. This means that, as in Q-learning, we are still estimating the value of the greedy policy according to the current values, as defined by θ. However, we use the second set of weights θ^{-} to fairly evaluate the value of this policy. The roles of θ^{-} and θ are switched to update symmetrically the second set of weights.

To get the best action when training, the policy network produces the best-predicted Q, and periodically this Q updates the target network. So, when training, the action is selected using Q values from the policy network, but the policy network is updated to better predict the Q value of that action from the target network. The policy network is instantiated throughout the target network every n steps (e.g. 1000) (Allen, 2021).

3. Implementations and Results

This work is implemented the Lunar Lander problem in 2-Dimensional OpenAI Gym using three methods, Q-Learning or Heuristic, Deep Q-Learning Network, and Double Deep Q-Learning Network.

Lunar Lander Heuristic has a state that has been defined as the output divided into 8, namely horizontal position, vertical position, horizontal velocity, vertical velocity, angle velocity, angular velocity, left leg contact, right leg contact. Sharing in these inputs determines 4 actions, namely do nothing, fire main, fire left, fire right. Thus, the 8 outputs are called state components and the 4 actions are the agent control inputs.

3.1 Results of the Implementation of the Heuristic / Q-Learning on the Lunar Lander

The state components in the 2-D Lunar Lander problem are presented below.

Table 3-1: Heuristic State Components

Output State Components
pos = self.lander.position vel = self.lander.linearVelocity state = [(pos.x - VIEWPORT_W/SCALE/2) /(VIEWPORT_W/SCALE/2), (pos.y - (self.helipad_y+LEG_DOWN/SCALE)) / (VIEWPORT_H/SCALE/2), vel.x*(VIEWPORT_W/SCALE/2)/FPS, vel.y*(VIEWPORT_H/SCALE/2)/FPS, self.lander.angle, 20.0*self.lander.angularVelocity/FPS, 1.0 if self.legs[0].ground_contact else 0.0, 1.0 if self.legs[1].ground_contact else 0.0]

These heuristic functions below map h: S → R+ given in each state s∈S an estimate of the distance between that state and the goal state.

Table 3-2: Heuristic Functions on the Lunar Lander Problem

Reward Shaping Heuristic Functions
#Ten points for legs contact
shaping = \
- 100*np.sqrt(state[0]*state[0] + state[1]*state[1]) \
- 100*np.sqrt(state[2]*state[2] + state[3]*state[3]) \
- 100*abs(state[4]) + 10*state[6] + 10*state[7]
if self.prev_shaping is not None:
reward = shaping - self.prev_shaping
self.prev_shaping = shaping
#Less fuel spent is better, about -30 for heurisic landing
reward -= m_power*0.30
reward -= s_power*0.03
done = False
if self.game_over or abs(state[0]) >= 1.0:
done = True
reward = -100
if not self.lander.awake:
done = True
reward = +10
return np.array(state, dtype=np.float32), reward, done, {}

The results of training from the lunar lander using the PID heuristic method can be seen in the reward value in Figure 3-1.

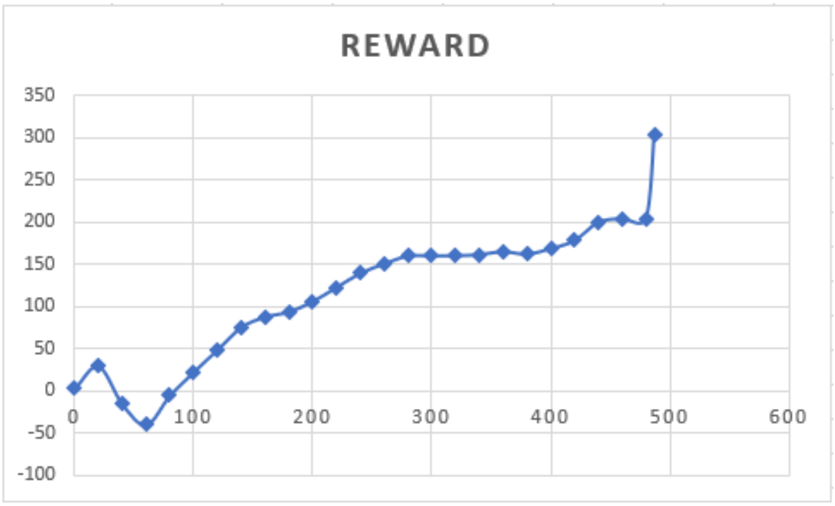

Figure 3-1: Graph of reward value on Lunar Lander with the heuristic method.

It can be seen that only in 1 episode with 435 steps, reward 200 is reached, this means the Heuristic method could find the solution in just one pass. It could be regarded as a normative behavior due to the deterministic nature of the functions. The introduction of the stochastic part to the functions will be a game-changer though and to be implemented in DQN and DDQN.

This whole research is conducted in Python running on Anaconda Navigator, using a Spyder Editor, and the following results are taken from the Lunar Lander Heuristic or Q-Learning method.

From Table 3-2 above, we can get the plotting of state data for each step as presented in Figures 3-1 to 3-5, so that we could figure out how the flight characteristics' movement of the vehicle is.

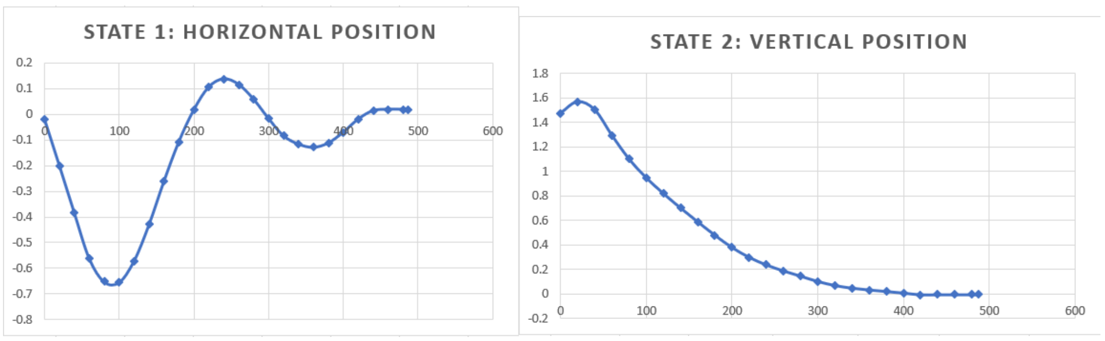

Figure 3-2: Horizontal position (m) as first state, and vertical position (m) as second state.

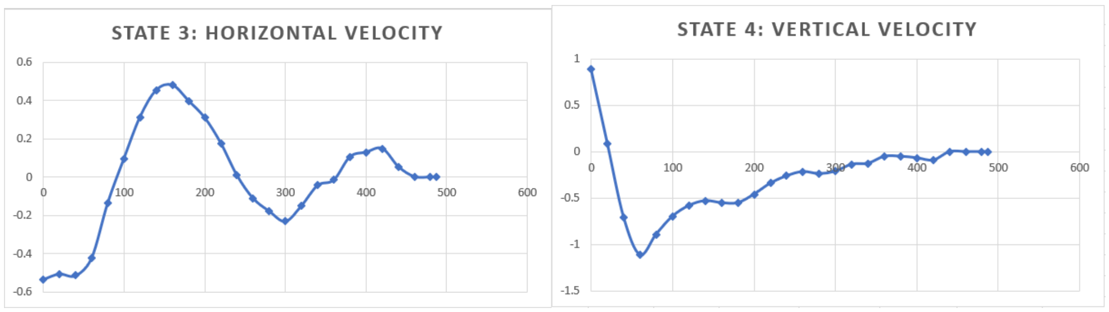

Figure 3-3: Horizontal velocity (m/s) and vertical velocity (m/s) as the third and fourth states.

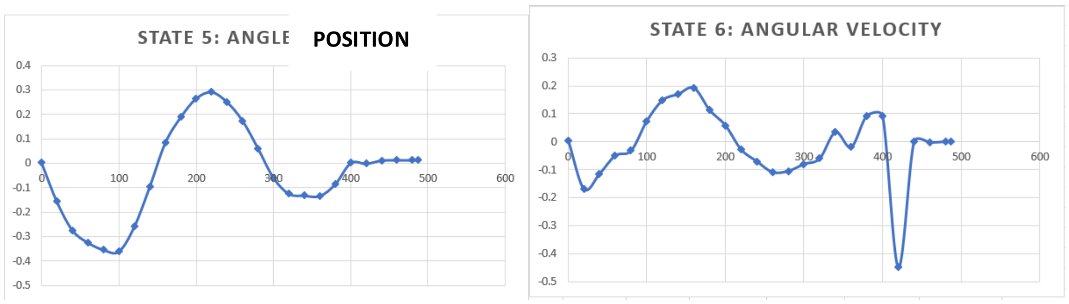

Figure 3-4: Angle position (rad) and angular velocity (rad/s) as the fifth and sixth states.

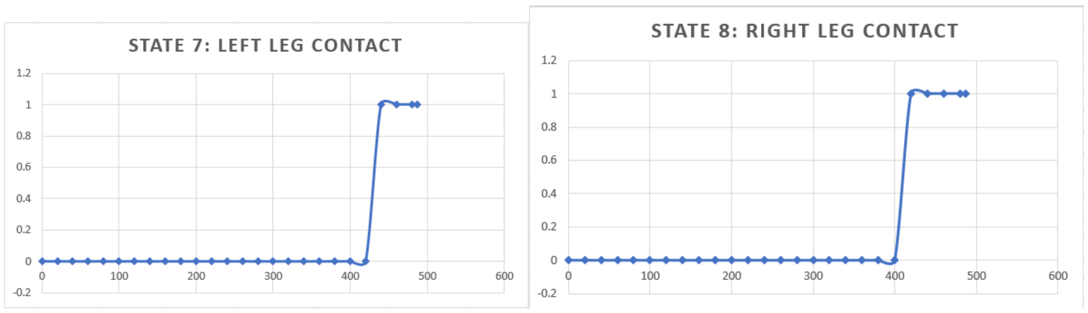

Figure 3-5: Left and right leg contact as seventh and eighth state movements.

Figure 3-6 below plots the three input commands (main engine, side thruster, and pitch controller), accompanied with the acceleration responses of the Lunar Lander, right after given the command. It can be seen that the input commands are in form of pulses.

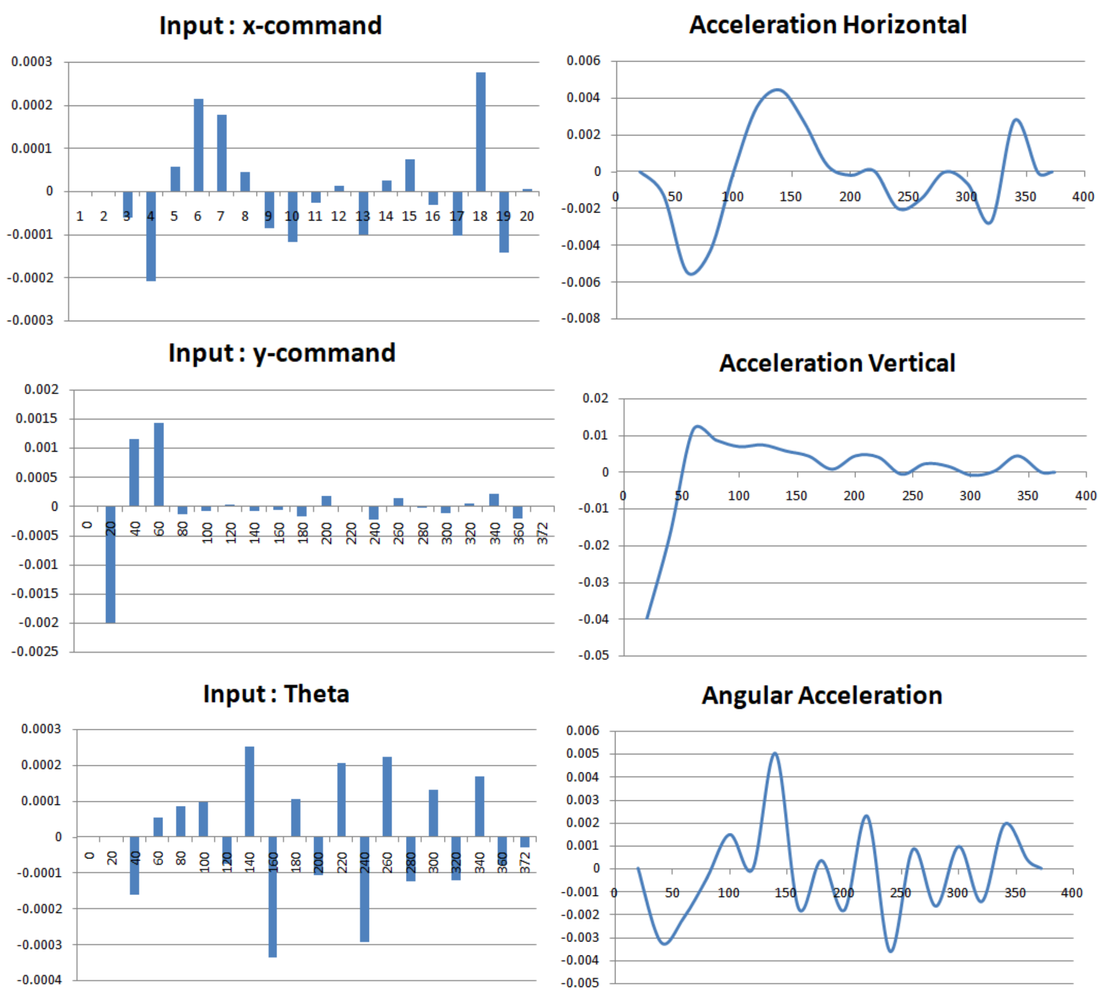

Figure 3-6: Input commands : Side Thrusters (X – N/s^3), Main Thruster (Y – N/s^3), Angular Thrusters (Theta – N/s^3) and Extra States : Horizontal acceleration (m/s^2), Vertical acceleration (m/s^2), Angular acceleration (rad/s^2).

In addition to the whole state components' plots, below is the trajectory result of the Lunar Lander landing scenario trained by the heuristic method.

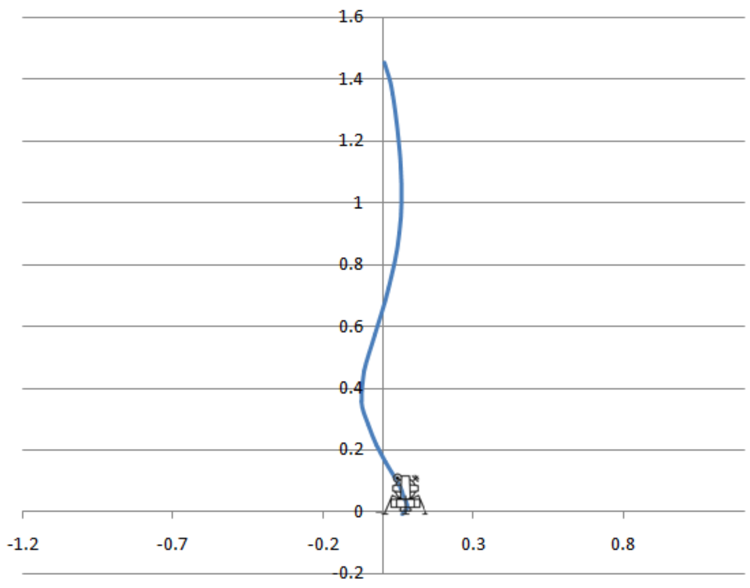

Figure 3-7: A Lunar Lander heuristic landing trajectory (m).

The heuristic or Q-Learning method yields a smooth and efficient landing maneuver. Nevertheless, why the trajectory is not simply a straight vertical line? The initial conditions dictate those undesired flight characteristics: the horizontal motion push to the right 0.6 m/s and the vertical motion pulls upright to 0.8 m/s. These make the Lunar Lander starts in a vertical jumping motion that is eventually corrected by Heuristic Powered Guidance Strategy and gravity force.

3.2 Results of the Implementation of the DQN and DDQN on the Lunar Lander

In the following implementations, the stochastic part plays important role in shaping the rewards due to the existence of the NN policy model and target model.

Table 3-3: Training Parameters in DQN and DDQN

Hyperparameters of DQN	Hyperparameters of DDQN
self.action_space = action_space	self.action_space = action_space
self.state_space = state_space	self.state_space = state_space
self.epsilon = 1.0	self.epsilon = 0.9
self.epsilon_min = 0.01	self.epsilon_min = 0.01
self.epsilon_decay = 0.996	self.epsilon_decay = 0.9993
self.gamma = 0.99	self.gamma = 0.99
self.learning_rate = 0.001	self.learning_rate = 0.0001
self.memory = deque(maxlen=1000000)	self.memory = deque(maxlen=5000)
self.batch_size = 64	self.minibatch_size = 64
self.model = self.build_model()	self.model = self.build_model()
self.target_model = self.build_model()	self.target_model = self.build_model()
	self.epochs = 1000
	self.verbose = 0

In this paper, two hidden layers of the neural network are selected as the structure of the target network and the current network. The number of hidden layer neurons is 64. The number of input and output neurons corresponds to the number of input states and actions, which are 4 and 8, respectively. The activation function is relu. According to the characteristics of relu function, the learning rate is 0.0001. To keep the training process relatively stable, the epoch is set at 1000, which means in 1 episode 1000 steps are done. The control cycle is selected as 10 ms as the cycle of actual vehicles required. In order to improve the generalization control ability of the agent, initial values are initialized in a certain range in every episode. The maximum number of training episodes is 1000. When the cost function (7) of network training is less than 10^{-6} and lasts for a period of time, the training is stopped.

$$cost = \sum_{j=1} \frac{1}{2}(y_j - Q(s_j, a_j; \theta))^2 \tag{3-1}$$

To analyze the training process, the cumulative reward of each episode and the total cumulative cost of training were recorded for analysis as planned in Eq. (2-1). The cumulative reward of each episode can indirectly reflect the change of the agent's control ability in the training process. Since the simulation time is 10 seconds and the control period is 10 milliseconds, the immediate reward is counted up to 1000 times per episode.

To proceed to our next action in the environment, even to move randomly, we do a .predict(). Three float values are shown to map the actions. By using argmax we construct the Q-value table. The updating process of the neural network is done by doing .fit() using that updated Q-value. Those 3 Q-values are fitted to make one updated Q-value. Past DQN models are used for each action, and each model used a regressor to output the Q-values.

DDQN intends to diminish overestimations by omitting the max operation in the target network in action selection and action evaluation. Updating target network periods in DDQN remain as same as DQN by copying the online network. This type of DDQN is considered the most conservative way of upgrading DQN. Besides keeping the DQN algorithm intact, this DDQN is desired to get the most benefit of Double Q-learning with the least computation.

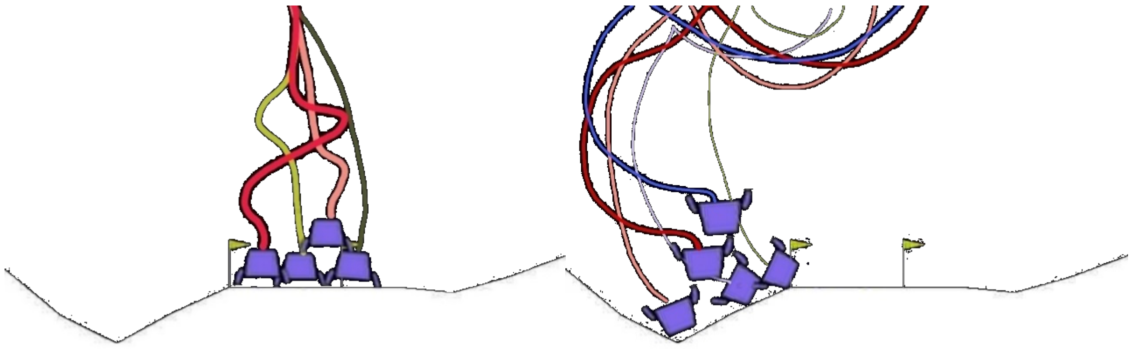

Figure 3-8: Lunar Lander' DQN (left) last 4 episodes landing and DDQN (right) la st 5 episodes landing.

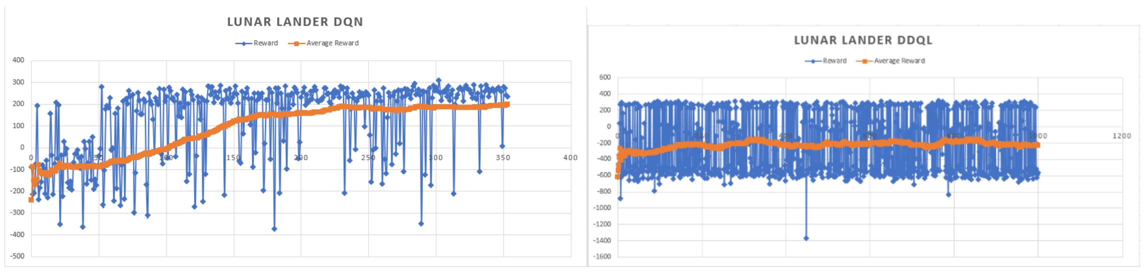

Figure 3-9: Reward gained from LL DQN (left) and LL DDQN (right) network training.

From the two figures above, the DQN method provided a convergent solution, while the DDQN failed to do so. This huge discrepancy could be explained as the result of a different strategy to approach the problem of estimation to the action values which leads to such contrasting loss values. Loss values are formulated in Eq. (2-3) which is affected dominantly by the subtraction of Q-target values and Q-realtime values, as shown in Eq. (2-4). In DQN the Q-target values follow Eq. (2-4) which are taken from max $Q_t(s, a)$, while in DDQN, those values are computed using Eq. (2-5) which are taken from Q without max. The subtraction term is visualized in Figure 3-10.

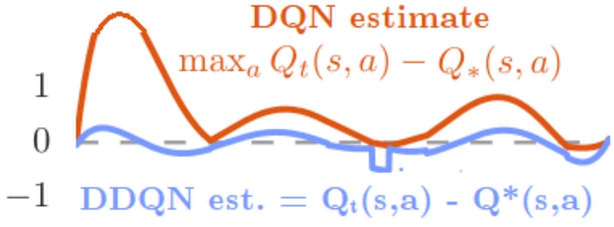

Figure 3-10: Difference in action values estimator between DQN and DDQN.

The smaller magnitude of estimation does not necessarily mean a lower degree of error or bias, on the other side, this could dampen the correcting process of action's values depending on the case. The DQN estimator could be considered overestimate, but eventually, in our case, it helps to increase training reward significantly as shown in Figure 3-9. In contrast, the DDQN training rewards couldn't go beyond negative, which explained the cause the DDQN Lunar Lander never touchdowned.

After conducting training on the Heuristic/Q-Learning, DQN, and DDQN methods, the following is a table of comparison of the training results for each method in the lunar lander.

Table 3-4: Comparison of Lunar Lander Training Results

Landing Guidance Method	Condition	Stochastic / Neural Network	Agent's Training Steps	Average Reward	Training Time	Solution Found Rate	Average Miss Distance (m)
Heuristic Search/Q-Learning	Known Environment, Known Agent, Computed Input State	No	487	117	52 sec.	95%	0.15
Deep Q-Learning Network	Unknown Environment, Known Agent, Estimated Input State	Yes	171911	86	63 min.	50%	0.2
Double Deep Q-Learning Network	Unknown Environment, Known Agent, Estimated Input State	Yes	487000	-228	70 min.	0%	Fail

4. Conclusions

From the three Lunar Lander experiments conducted, it can be analyzed that the performances of Lunar Lander's movement relate directly to the method's capability to reach a solution. Looking at a glance, the heuristic method showed an efficient method, with a very fast training time and a fairly large average reward. The heuristic movements are fairly smooth and the landing reward is never negative. The DQN method in the Lunar Lander case has a fairly good quality following the heuristic one. However, this will also vary, because the initial random factor also affects the movement of the Lunar Lander. DQN has spent quite a bit of computation on neural networks, including using experience replay and the target network. On the other hand, DDQN in the Lunar Lander case is strongly discouraged because the average reward obtained is never positive, thus still far from convergency. The proposed method of separating usage of online weight θ in the action selection process and action evaluation process in order to reduce overestimation didn't produce results as intended. Apparently, in our case, overestimation is needed to correct the negative reward. The only advantage of DDQN is its ability to produce quite diverse movements of the Lunar Lander that could be very different from the start of training, which suggests its ability to absorb more uncertainty in the environment. DDQN as an extension of DQN needs examination further in terms of hyperparameter adjustment to produce better Lunar Lander actions.

Acknowledgments

The authors would like to express gratitude to LAPAN Pustekroket, Politeknik Elektronika Negeri Surabaya (PENS), University of Adelaide, Australia, and Universitas Indonesia for their support in this research.

Contributorship Statement

LN is the main contributor of this research by developing the concept and preparing the manuscript, NRZ and NQ conduct the simulations, RAk and RAn proofread the writing, analyze the results and suggest improvements, SKW is the corresponding author and guides the whole process of research.

References

Allen, M. (2021). Double Deep Q Network (DDQN) – controlling a simple hospital bed system. Retrieved from https://pythonhealthcare.org/2020/07/05/double-deep-q-network-ddqn-controlling-a-simple-hospital-bed-system/

Bašić, B. D. & Šnajder, J. (2020). Unit 3: Heuristic search [PowerPoint slides]. Faculty of Electrical Engineering and Computing, University of Zagreb. https://www.fer.unizg.hr/_download/repository/AI-3-HeuristicSearch.pdf

Bengio, Y. (2013). Deep learning of representations: Looking forward. https://arxiv.org/pdf/1305.0445.pdf

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI gym*. http://arxiv.org/abs/1606.01540

Cini, A., Eramo, C. D., Peters, J., & Alippi, C. (2020). Deep reinforcement learning with weighted Q-Learning. https://arxiv.org/pdf/2003.09280.pdf

Ermacora, G., Rosa, S., & Toma, A. (2016). Fly4SmartCity : A cloud robotics service for smart city applications. *Journal of Ambient Intelligence and Smart Environments*, *8*(3), 347–358. https://doi.org/10.3233/AIS-160374

Gaudet, B., Linares, R., & Furfaro, R. (2018). Integrated guidance and control for pinpoint Mars landing using reinforcement learning. *Advances in the Astronautical Sciences, 167,* 3135-3154.

Hasselt, H. van. (2010). Double Q-learning. *Advances in Neural Information Processing Systems, 2010,* 2613-2621.

Hasselt, H. van., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence, 30*(1).

Kersandt, K. (2018). Deep reinforcement learning as control method for autonomous UAVs [Master's thesis, Universitat Politecnica de Catalunya]. https://upcommons.upc.edu/handle/2117/113948

Klumpp, A. R. (1974). Apollo lunar descent guidance. *Automatica*, *10*(2), 133–146. https://doi.org/10.1016/0005-1098(74)90019-3

Latyshev, K., Garzaniti, N., Golkar, A., & Crawley, E. F. (2021). Technology roadmap for future human landing systems. Conference proceedings of ASCEND 2020. https://doi.org/10.2514/6.2020-4229

Meen, V. (n.d.). Modelling the 1/32 lunar module [Image]. Retrieved from http://spacemodels.nuxit.net/1-32 LM/Orientation1.jpg

Mitić, M., Miljković, Z., & Babić, B. (2011). Empirical Control System Development for Intelligent Mobile Robot Based on the Elements of the Reinforcement Machine Learning and Axiomatic Design Theory. *FME Transactions (2011)*39*,* 1-8.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Quan, H., Li, Y., & Zhang, Y. (2020). A novel mobile robot navigation method based on deep reinforcement learning. *International Journal of Advanced Robotic Systems 17*(3), 1–11. https://doi.org/10.1177/1729881420921672

Sutton, R. & Barto, A. (1998). Reinforcement Learning: An Introduction. *MIT Press Cambridge: London, UK.*

Tanyolaç, T. & Yasarcan, H. (2012). Control heuristics for soft landing problem. Conference proceedings of the 30th International Conference of the System Dynamics Society.

Von Dollen, D. (2017). Investigating reinforcement learning agents for continuous state space environments (pp. 5–7). http://arxiv.org/abs/1708.02378

Watkins, C. (1989). Learning from delayed rewards [Doctoral dissertation, King's College (University of Cambridge)].

Yu, X. (2019). Deep Q-learning on lunar lander game. Retrieved from https://www.researchgate.net/publication/333145451_Deep_Q-Learning_on_Lunar_Lander_Game