Comparative Study of Decision Making Algorithms for Autonomous MoonLander: Project Group 21

Aishik Pyne	Harshavardhan Abichandani	Niharika Shrivastava	Edmund Tham
A0250059E	A0250610X	A0254355A	A0250679U
E0945774	E0945792	E0954756	E0945861

1 Introduction

The pursuit of interplanetary travel has captivated humanity's imagination for generations. One formidable obstacle in this endeavour is achieving precise landings on foreign celestial bodies, demanding meticulous planning, autonomous decision-making, and adaptability under uncertain conditions. This project endeavours to systematically develop different planning algorithms, enabling a spacecraft to execute soft landings at specified destinations. Subsequently, it relaxes the assumptions on the environment, progressively making it more complex by introducing stochastic factors such as winds and turbulence; and analyzing the limitations of the algorithms based on the environment. We then provide a detailed analysis of each planning method's performance.

1.1 Problem Formulation

We used the OpenAI LunarLander-v2 Gym [OpenAI]. It provides a 2D simulator for spaceship landing with the following characteristics:

- **World**: We assume a 2D world where all forces act in the X-Y plane. The agent has an upward thruster and an alignment thruster.
- **States**: The state space is continuous. Agent has a downward-facing RGBD camera which provides the depth and lateral position of the goal w.r.t the agent's frame of reference.
- **Actions**: Action Space can be either:
  - Discrete: Thruster fired Up, Left Thruster fired, Right Thruster fired, No Op.
  - Continuous: Altitude Thruster [-1, 1] where force linearly grows from 0 to 1; Alignment Thruster [-1,1] where -1 is full left and +1 is full right.
- **Goal**: Soft-landing with the Goal state at (0, 0); Agent's horizontal and vertical velocity as well as the angle of contact should close to 0.
- **Observation Space:** Coordinates of the lander, Linear Velocities, Angular Velocity and whether the Spaceship is in contact with the ground.
- **Dynamics**: Gravity is a constant downward force which makes the environment non-stationary. The agent's state changes even without any action. Winds and turbulence are variable forces which make the environment stochastic. Winds can be modelled to an extent as they vary slowly with time, whereas turbulence is abrupt and can't be modelled at all.
- **Rewards:** After each step a reward is granted:
  - The reward for moving from the top of the screen to the landing pad with zero speed is awarded between 100 and 140 points.
  - If the lander moves away from the landing pad it loses the same reward as moving the same distance towards the pad.
  - -100 or $+100$ points for crashing or landing, respectively.
  - Grounding a leg is worth 10 points and thrusting the main engine receives -0.3 points. **An episode score of** 200 **or more is considered a solution.**

Preprint. Under review.

1.2 Planning Algorithms Overview

We implement 4 major planning algorithms to solve this problem and evaluate their performance in terms of maximum expected rewards gained, advantages, time complexity, and shortcomings. 1

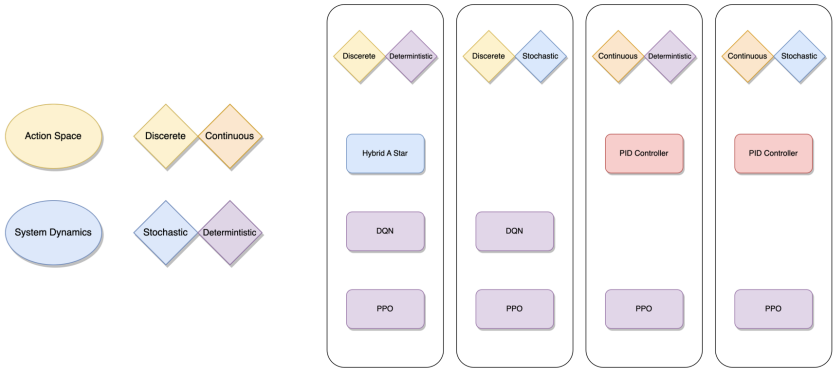


Figure 1: Algorithms used and environments where they are applicable

1.3 Work Division

- Aishik: Implement and analyse Trajectory Planning + Optimal Control; Write project proposal.
- Harshavardhan: Implement and analyse Deep Q Network (DQN); Make changes to the simulator.
- Niharika: Implement and analyse Proximal Policy Optimization (PPO); Write the final report.
- Edmund: Investigate Hybrid A Star.
- Common Work: Everyone is responsible for writing their work in the report.

2 Optimal Control

2.1 PID Controller

The Proportional-Integral-Derivative (PID) controller [William] is a fundamental and widely employed feedback control mechanism within the realm of control engineering. This controller serves as a pivotal tool in regulating various dynamic systems, such as industrial processes, autonomous vehicles, and robotic systems. Its design is rooted in a three-component architecture, encompassing proportional, integral, and derivative terms, which collectively aim to minimize the error between the desired setpoint and the actual output of the system.

The proportional term in the PID controller produces an output signal that is directly proportional to the current error. The integral term accumulates the past error values over time, thus rectifying long-term system deviations. Lastly, the derivative term provides control action based on the rate of change of the error, which assists in preventing overshooting and dampening oscillations. The tuning of these three parameters allows engineers to strike a balance between system stability and transient response, rendering the PID controller a versatile and invaluable tool for enhancing the control performance of a wide array of dynamic processes, making it an indispensable component of modern control systems.

Formally

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

In this formula:

- $u(t)$ represents the control output at time t.
- K_p is the proportional gain.
- K_i is the integral gain.
- K_d is the derivative gain.
- $e(t)$ is the error at time t.
- The integral term is calculated by integrating the error from 0 to t.
- The derivative term is calculated as the derivative of the error with respect to time t.

However, it is not necessary to use all possible components we can get away using Proportional (P) or Proportional Differential (PD) controllers only [Ni].

2.2 PD References

Setting references, or set points, represent desired values in a control system. To compute the current error, the difference between the set point and the actual system output is calculated. This error value is a key input for the control algorithm, guiding adjustments to minimize the error and bring the system closer to the set point. Accurate set point selection and error computation are essential for optimal control system performance, particularly in responding to external disturbances or changing conditions.

In our case we have two control variables, the main thruster v and the side alignment thruster ω. The main thruster is responsible for controller the altitude and the side thruster is for navigating the lander left or right.

Thus, we **define 2 PD Controllers** and define a reference for each

- Altitude Controller: $K_p^{altititude}$ $K_d^{altititude}$
- Alignment Controller: $K_p^{alignment}$ $K_d^{alignment}$

2.2.1 Altitude Controller

We model the altitude controller to match the reference of $|L_x|$. In this way the lander always tries to go down towards the center of the goal but if it is too low, it will correct it's altitude to go above the reference line.

$$e_{altitude} = L_y - |L_x|$$

Where

- L_x is the current x coordinate of the lander
- L_y is the current y coordinate of the lander

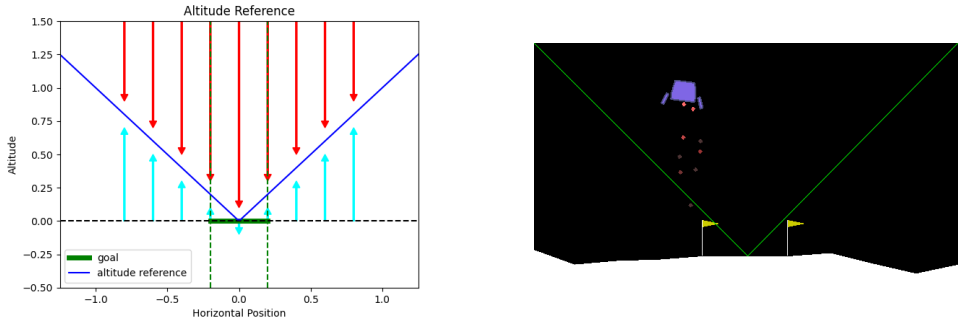


Figure 2: 1. On the left is a schematic diagram of the Altitude Reference line for the Altitude Controller to follow. Image from [William] 2. On the right is the same plot but on the GUI.

We define 2 PD controllers

2.2.2 Alignment Controller

At any given time, the spacecraft must try to minimize the cross-track error between its x coordinate and the horizontal line passing through the centre of the goal. In order to do that, the side thrusters should point the landers towards the goal line. The further the lander is from the line the steeper should be the angle of the lander towards the goal line. Moreoever it should be proportional to the speed at which

$$e_{alignment}(t) = \frac{\pi}{4}(L_x + L_{vx}) - L_\theta$$

Where

- L_x is the current x coordinate of the lander
- $L_v x$ is the current x component of the lander's velocity
- L_θ is the current orientation θ of the lander

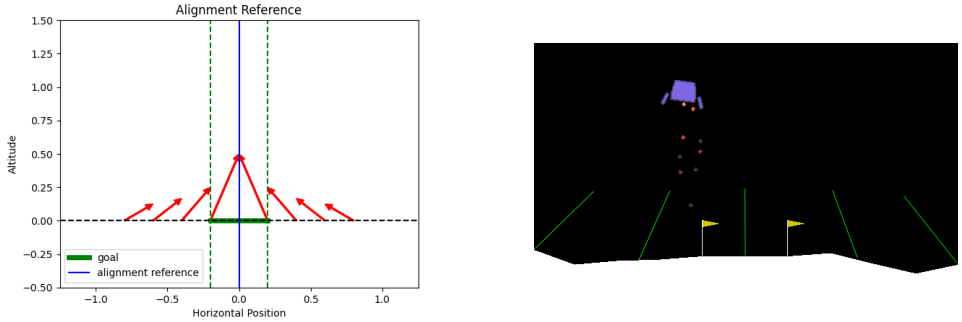
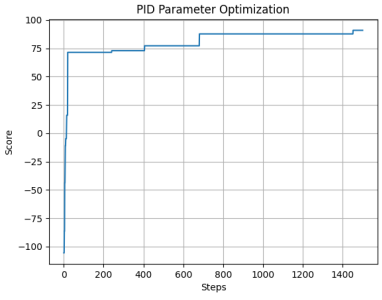
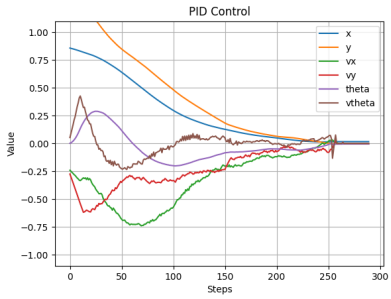


Figure 3: 1. On the left is a schematic diagram of the Alignment Reference line for the Alignment Controller to follow. 2. On the right is the same plot but on the GUI.

2.3 Optimizing PID Parameters using Randomized Hill Climbing



(a) Optimizing PID parameters using randomized hill climbing.



(b) Sample run a PID Controller

Manually tuning the gains of a Proportional-Derivative (PD) controller is challenging due to the intricate balance required between the proportional and derivative components. Achieving optimal performance often necessitates a trade-off between fast response and stability, and finding the right combination of gains can be a complex trial-and-error process. Additionally, system dynamics and environmental factors can change, making manual tuning a continual and time-consuming effort. Automated tuning methods or advanced control techniques are often preferred to address these difficulties effectively.

Thus we propose randomized hill climbing to optimize the gain parameters. We start of by setting all the parameters to zeros $K_p^{altititude}, K_d^{altititude}, K_p^{alignment}, K_d^{alignment} = [0, 0, 0, 0]$. Then we

optimize these parameters iteratively. At each optimization step we randomly add a uniform noise to each of the parameters and evaluate average score of the lander of the current set of parameters. If the new score is better than the previous, we update the parameters to the current values. Overtime we expect the parameters to reach a configuration which maximizes the evaluation score as seen in Figure 4a

2.4 Performance

Figure 4b shows a plot of the error (difference from current state to goal state) for each of the lander's parameter. We observe that the controller does a good job in converging the errors to zero over time. A more in-depth analysis of the results will be available in the results section.

3 Deep Q Network

Deeep Q Network (DQN) Mnih et al. [2013] is an off policy method that tries to learn the Q values by using a function approximator. In our case the function approximator is a simple Deep Neural Network (DNN).

The error term of the DQN comes directly from the bellman update equation.

$$Q(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a') \tag{1}$$

Here the value of the state action pair depends on the immediate reward and the max value of the next state. DQN simply reduces the difference between the two terms given below.

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} (r_t^i + \gamma \max_{a'} Q_{\bar{w}}(s_{t+1}^i, a') - Q_w(s_t^i, a_t^i))^2 \tag{2}$$

In our world setting the state space is continuous and the action space continuous and action space is discrete (for DQN). This makes it intractable to represent the Q values in a tabular format. Therefore, we need to parametrize the Q values. We represent the parametrized Q value as Q_w.

In equation 2 our goal is to update weights so that our current estimate is closer to the target network.

Algorithm of DQN is as follows.

Algorithm 1 DQN with soft update

1. Initialize policy network Q_π
2. Initialize target network Q_π^t
3. Initialize Replay Buffer B
4. Initialize Tau τ
5. For n episodes
6. s_0 get the initial state
7. while not done
8. select greedy action with ϵ probability
9. Add (s, a, s', r) in buffer B
10. Sample from B and train Q_π
11. Update Q_π^t
12. $Q_\pi^t = \tau Q_\pi^t + (1 - \tau) Q_\pi$

The traditional implementation of DQN involves updating the target network Q_π^t after a fixed number of steps. This is done to make the training of DQN stable. We utilize a different method, we soft update the target network every step. Based on Lillicrap et al. [2019], this way is better and is relatively more stable.

3.1 Performance

DQN can be only implemented for discrete action because of the $\max Q$ step. Therefore, we only evaluate our DQN model on Deterministic and Stochastic environments with discrete actions.

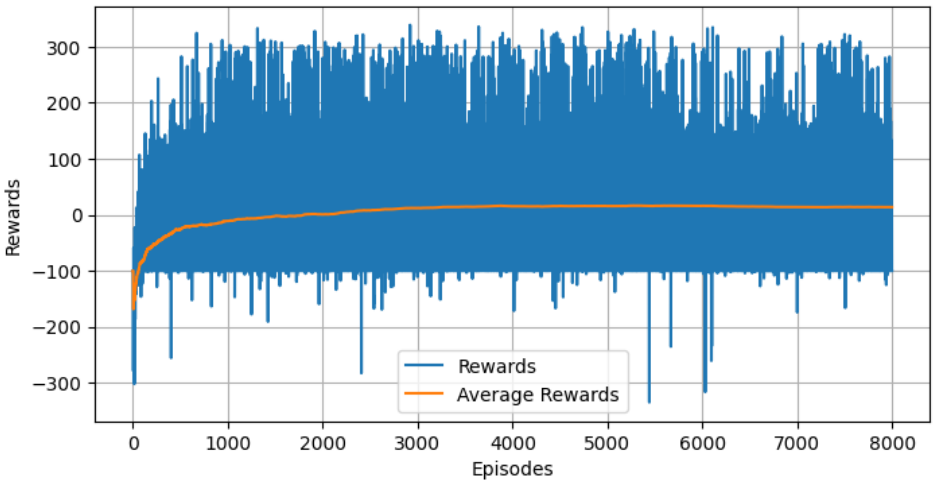


Figure 5: DQN performance on Discrete Action and Deterministic environment. It converges with an average reward of around 40

For Deterministic and Discrete Action, the DQN model converges after 7000 steps but achieves a suboptimal average reward.

In the case of Stochastic and Discrete Action, the DQN model fails to converge. We attribute this failure to the limited model size and the inadequacy of the replay buffer capacity to capture stochasticity. Despite attempts to address these issues by augmenting the model size and buffer capacity, convergence is not achieved. Therefore, we move to Proximal Policy Optimization (PPO).

4 Proximal Policy Optimization

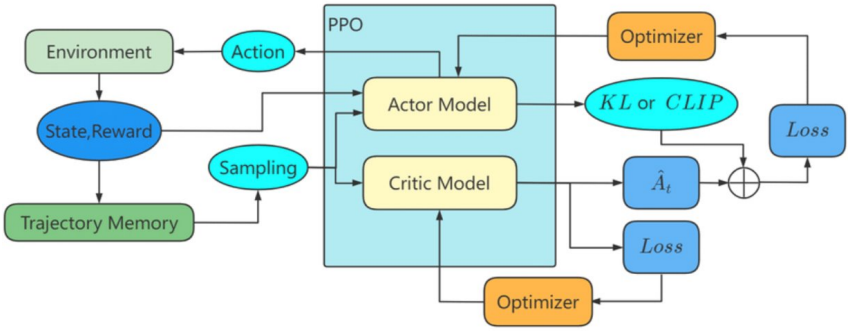
PPO [van Heeswijk [2022]] has the following properties:

1. **Online**: PPO collects data by interacting with the environment in real-time. It updates the policy based on the most recent experiences.
2. **On-policy:** It optimizes the policy used to collect data. PPO updates its policy based on the most recent interactions with the environment.
3. **Policy-gradient:** It directly optimizes the parameterized policy, instead of learning the value function. Specifically, it uses gradient ascent to iteratively update the policy parameters in the direction that increases the expected cumulative reward.

4.1 Actor-Critic Architecture

It follows the REINFORCE version of Actor-Critic setup to maximize expected rewards. The $Actor$ is responsible for taking an action and the $Critic$ evaluates the action, thereby dictating the performance of the $Actor$.

In the case of a discrete action space, the $Actor$ takes an action in the environment and receives a reward. Multiple experiences are filled into a replay buffer. The $Critic$ samples a few trajectories from this buffer and evaluates the expected reward vs gained reward for each action taken by the $Actor$. Suitably, an advantage function is computed that signifies how good a particular action is. This advantage function then dictates the behaviour of the $Actor$ by backpropagating the loss of rewards through the $Actor$ model. Every few epochs, the $Critic$ is updated as well.



For a continuous action space, the $Actor$'s head is replaced by 2 heads. They output the expected mean and standard deviation of the actions. Action selection takes place by creating a Gaussian distribution using this mean and variance and then sampling from this distribution.

The loss function of the $Actor$ is given by:

$$L_{clip}(\theta) = E[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$$

where,

- $r_t(\theta)$ = ratio of the probability that an action is better in the current policy compared to previous policies for a given state.
- A_t = Advantage function. It reinforces that the best action is used in the given state.
- $clip(...)$ = Avoids large policy updates. The bounds make sure that the updates are always in a trusted region even in the worst-case update. Moreover, since the updates are small, it leads to stable training.

Overall the loss function forms a pessimistic bound on each policy update.

4.2 Performance

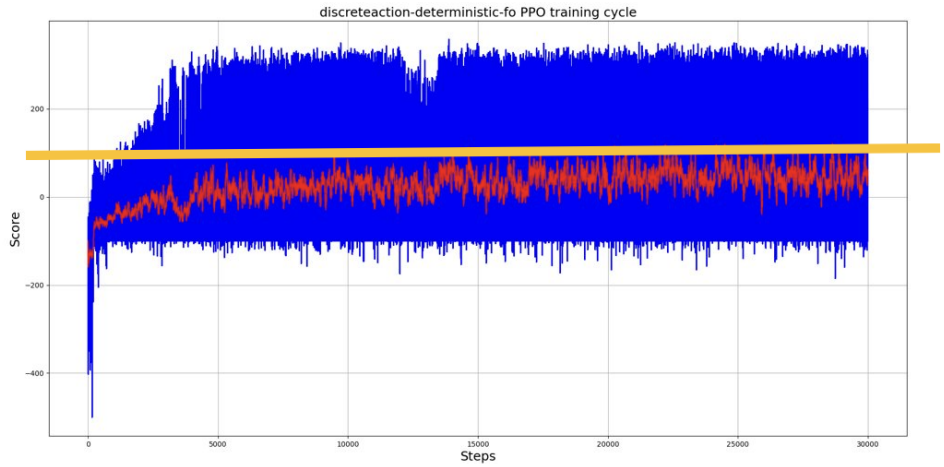


Figure 6: Unregularized PPO overfits to a local minima frequently and needs entropy regularization. That is, we try to achieve a balance between exploration and exploitation to learn important updates.

Compared with value-iteration methods such as DQN, PPO directly optimizes the parameterized policy instead of learning the value function. This has been shown to converge faster than learning the optimal Q function. Moreover, the learning process is stable compared to DQN as there are smaller policy updates in the direction of maximum expected rewards due to gradient clipping.

However, the vanilla version of PPO's training process overfits grossly on the LunarLander environment 6. Even for an increased number of episodes, we could only achieve a maximum expected reward of 10.32 in the worst-case (continuous action space, stochastic environment). Thus, we subject PPO's objective to an entropy regularization term $H(\pi() \cdot |s_t)$ [StableBaselines3] such that,

$$H(\pi(\cdot|s_t)) = -\sum_{a \in A} \pi(a|s_t) log \pi(a|s_t)$$

Another observation is that even if the sample efficiency of PPO is better than DQN due to its policy-gradient mechanism, its sample efficiency is still extremely low compared to other model-based methods. Finally, PPO was highly sensitive to hyperparameter tuning such as the clipping value ϵ and the frequency of $Actor - Critic$ updates.

5 Results

Table 1 shows the maximum expected rewards gained upon convergence. Table 2 shows the number of no. of episodes required for each algorithm to converge.

Action Space	Discrete		Continuous	
System Dynamics	Deterministic	Stochastic	Deterministic	Stochastic
Optimal Control	-	-	62.4	4.37
DQN	18.52	-47.32	-	-
PPO original	57.47	13.41	48.32	10.32
PPO Regularized	198.68	158.21	186.23	133.21

Table 1: Average Reward for three methods under various environmental conditions. When the environment is stochastic the wind power $= 15$ and turbulence $= 2$.

Action Space	Discrete		Continuous	
System Dynamics	Deterministic	Stochastic	Deterministic	Stochastic
DQN	7k	60k	-	-
PPO Regularized	25M	25M	25M	25M

Table 2: Number of iterations required to converge.

The results show that the PD controller works well for a continuous action space in a deterministic environment. However, the performance drops in a stochastic environment. DQN was able to learn a policy using a reward function for the discrete action environment. However, the regularized version of PPO outperforms every other algorithm in all environments.

The number of iterations for convergence of PPO is very high compared to that of DQN. However, even after DQN's convergence, the results were suboptimal. We believe that DQN will converge optimally with higher number of iterations ($> 25M$) due to policy-iteration methods being slower than policy-gradient methods. However, we didn't have the resources to check our theory.

We also do not have results for Hybrid A Star as it was not successfully implemented.

References

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2019.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013.

Ni. The pid controller theory explained. URL https://www.ni.com/en/shop/labview/pid-theory-explained.html.

OpenAI. Gym documentation - lunar lander. URL https://www.gymlibrary.dev/environments/box2d/lunar_lander/.

StableBaselines3. Rl baseline zoo - examples. URL https://stable-baselines3.readthedocs.io/en/master/guide/examples.html.

W. van Heeswijk. Proximal policy optimization (ppo) explained. *Medium*, 2022. URL https://towardsdatascience.com/proximal-policy-optimization-ppo-explained-abed1952457b.

F. William. Pid control of openai lunarlanderv2. URL https://github.com/wfleshman/PID_Control.