JOURNAL OF
EMERGING INVESTIGATORS

Article

Reinforcement learning in 2-D space with varying gravitational fields

Timothy Rousseau[1], David Jose Florez Rodriguez[2]
[1] Menlo-Atherton High School, Menlo Park, California
[2] Stanford University, Stanford, California

SUMMARY

Given the challenges humans encounter when navigating through unpredictable environments in interstellar space, machine and reinforcement learning could be useful tools for efficiently and safely navigating a predefined space containing other objects. In this work, we explored whether reinforcement learning could be used to navigate a plane in space with a defined gravitational field set by two randomly placed planets. We used a reinforcement technique called a Q-learning algorithm which employs accumulated learning. Three parameters were tested: how the number of times the Q-learning algorithm ran through the data affected percent improvement, how the number of simulations we ran affected percent improvement, and how the location of the planets affected percent improvement. We sought to show that the tests using the data based on the Q-learning algorithm performed better than tests using random actions to navigate and identified any new navigation methods that machine learning can develop, such as a slingshot method. Our results suggest that more iterations of Q-learning did not have a significant effect on performance, the number of simulations ran had a slight negative correlation with performance, and the location of planets played a highly significant role in Q-learning performance. We also determined that machine and reinforcement learning can produce novel methods to effectively arrive at the reward. In this work, we investigated the basics of reinforcement learning and demonstrated its use as a solid tool to navigate any given setting.

INTRODUCTION

Reinforcement learning is a subtopic of machine learning that includes learning through trial and error and can be an incredibly useful tool in decision-making and optimization in many fields. One such industry is autonomous vehicles, which use reinforcement learning to make the next best action, showing how this topic has further applications than only space travel (1). However, in space, it is difficult for a human to perform precise modifications at the right moment and efficiently navigate around objects. However, with the use of reinforcement learning, making small adjustments to the trajectory at the right moments ensures that a spacecraft arrives at the correct destination while avoiding all obstacles (2). This approach could greatly increase the efficiency and reliability of space travel and make it much easier to calculate an effective flight plan.

Reinforcement learning will likely be incredibly important for future endeavors, such as sending the first humans to Mars and are already used today in vehicles like self-driving cars (1). With reinforcement learning, new kinds of missions could be possible due to its high level of accuracy and reliability, fulfilling the demands for more advanced navigation systems (2). Already, there have been several studies on the application of reinforcement learning to navigating a given space. The effectiveness of deep reinforcement learning in mapping a robot throughout a given plane has been demonstrated (3). Similarly, Brandonisio et al. examines various reinforcement learning techniques (algorithms) to optimize the path of agents through space (4). They found that reinforcement learning, using algorithms such as Q-learning and A2C, is more effective at navigating space than randomly assigned actions (4). Our paper expands upon this previous body of work, exploring the applications of deep reinforcement learning in navigating a two-dimensional (2D) space with varying gravitational forces.

We hypothesized that using the Q-learning algorithm will optimize the navigation of a spacecraft and that making decisions based on past decisions will yield better results than making random decisions. We expected to see a higher reward value for simulations run on the training done through reinforcement learning than simulations without (the spacecraft takes random actions at each location). More specifically, we predicted that as the number of simulations the algorithm trains on increases, the more optimized the pathing will become, and the more times we run the algorithm on past simulations, the better it will perform. In addition, we sought to present evidence of new navigation methods, like the slingshot strategy, developed by the computer. The novelty in this paper is training the agent on both random simulations through the space and on the paths of previously trained agents. This iterative training with Q-learning may prove more useful than plain Q-learning.

Our research focuses on the practicality and optimization of reinforcement learning in navigating a plane in space. This topic will strengthen current and future navigation in unexplored spaces. By letting a computer determine the optimal path through a field of obstacles, space navigation could become much simpler. Some applications of this research are for unmanned vehicles which can be programmed to travel a certain way based on the results of the reinforcement learning algorithm (2). Additionally, it can be useful for navigating dangerous and unknown environments, as the computer can develop different strategies to effectively travel through space without the risk of loss of life or resources using realistic simulations.

https://doi.org/10.59720/23-277

RESULTS

In this study, we aimed to show that using reinforcement learning to determine the best action at each point in space yields better results in navigating a 2D plane than performing random actions at each point. More specifically, we ran many random simulations, ran a Q-learning algorithm on them to learn the best way to reach the reward, and then calculated how actions determined by the Q-learning algorithm performed compared to random actions. We tested the impact of how many simulations the algorithm tested on (*NumSims*), how many iterations of the algorithm we ran on one simulation (*qruns*), and the location of the planets.

We showcased one example of the computer using a slingshot method to reach the reward (**Figure 1**). Given this trajectory, beginning in the lower left, over many simulations employing the Q-learning algorithm, the total reward would have been 20. This is over 500% higher than the average reward granted by random simulations under the same conditions, which was 3.2.

We found that the largest percent improvement was 96.36% over the random simulations (**Figure 2**). This means that simulations ran with the Q-learning algorithm performed 96.36% better over random simulations with this pair of planets. Additionally, the location of the planets can make a difference in how effective the spacecraft is at arriving at the reward area (**Figure 2**).

We calculated this by using a standardized scale between 0 and 1 for the percent improvement. is the trend line for percent improvement as *qruns* increased. Using a simple linear regression model, we found that the formula for percent improvement as *qruns* increased (**Figure 3**) is:

$$(Avg\ \%\ improvement) = -0.00925(qruns) + 0.343 \qquad \textbf{(Equation 1)}$$

From the graph, we can infer that there is a slightly negative linear relationship between percent improvement and *qruns*. The slope was -0.00925 which indicates that the percent improvement decreased by about 0.925% per integer increase in *qruns*. The novelty in our employment of Q-learning, training the Q-learning algorithm on the paths of previously trained agents, did not achieve higher results than plain Q-learning and may in fact be slightly harmful.

Next, we found a trend line for percent improvement as the number of simulations (denoted as *NumSims*) ran increased (**Figure 4**). Using a simple linear regression model, we found that the formula for this trend is:

$$(Avg\ \%\ improvement) = -0.00000886(NumSims) + 0.334 \qquad \textbf{(Equation 2)}$$

This graph indicates that there is a slight negative linear relationship between percent improvement and *NumSims*. The average improvement decreased by about -8.886 x 10^{-3} % per additional 1000 simulations.

DISCUSSION

Our data shows that increasing the number of *qruns* and the number of *NumSims* made the simulations less effective at reaching the reward as we increased the amount. This suggests that lower values for *qruns* yield higher percent improvement of the Q-learning algorithm when compared to random simulations. Additionally, our data demonstrate that there is a higher percent improvement of the Q-learning algorithm over random simulations when fewer simulations are run.

Overall, the evaluated Q-learning model reached the reward zone more often (produced higher reward values) than the random simulations, meaning that the computer model was able to properly calculate more optimal actions to take to reach the reward. One prediction as to why our hypothesis isn't supported by the data is random error at smaller values for *NumSims* and *qruns,* which might have led to some simulations that were outliers which skewed the data. With more computing power and time to explore higher values of *NumSims* and *qruns*, a clearer effect from either parameter may have surfaced. It is also possible that another reward strategy could have improved the model, such as using a different formula to calculate the reward value at each point in space.

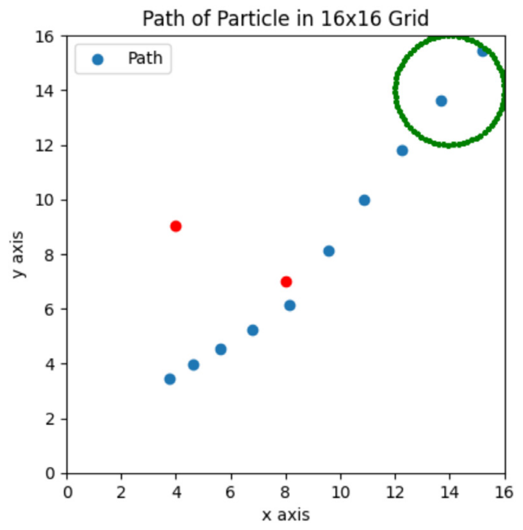

Figure 1. Sample path of agent employing a slingshot technique. Sample path of the particle (blue) around the plane with two planets (red). The green region is where the agent earns a reward. There are no units because gravity was calculated based on the distance from any point in the plane to a planet, which used units defined by locations in the grid.

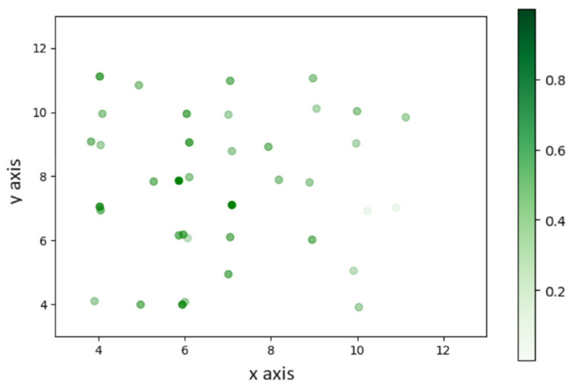

Figure 2. Planetary locations and learning algorithm performance. Position of planets we used in our simulations. The gradient of the green is a normalized scale demonstrating the effectiveness of each planet combination - the darker the green, the higher average reward. The graph only displays coordinates from (4,4) to (12,12) instead of the full 16 x 16 grid because this is the space the planets were confined to when we generated their locations.

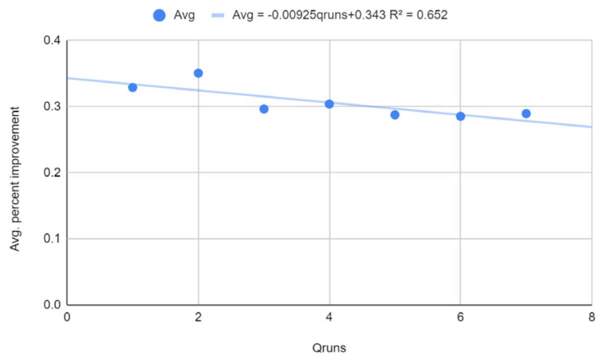

Figure 3. Trend in Q-learning's improvement over random simulations as qruns vary. There was a negative correlation between the average percent improvement and qruns. The R2 value for this relation is 0.652 and the R value is -0.802, suggesting a moderately strong negative correlation. Note that all trials yielded improvement over random simulations, indicating the effectiveness of our algorithm.

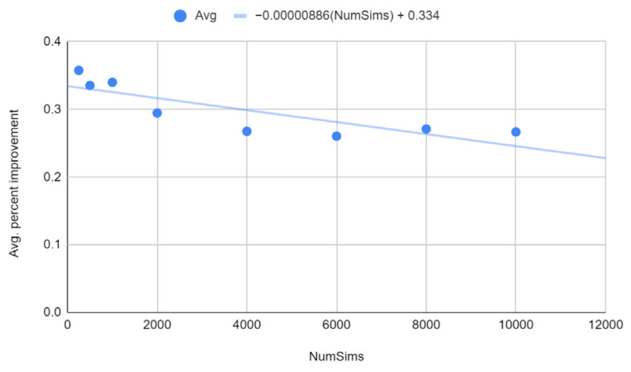

Figure 4. Trend in Q-learning's improvement over random simulations as the number of simulations vary. There was an inverse correlation between average percent improvement and NumSims. The R2 value for this relationship is 0.695. Note that all trials yielded improvement over random simulations, indicating the effectiveness of our algorithm.

Additionally, the location of the planets played a significant role in how effective machine learning was. Our results suggest that there is a slight benefit of having planets closer to the initial starting point of the particle rather than closer to the reward. Additionally, the computer did better when the planets were between the starting position and the reward zone, suggesting that the computer used the planets' gravity to its advantage, similar to a slingshot strategy.

The slight benefit of having planets closer to the initial starting point of the rocket, rather than near the reward, could be attributed to the smaller 'gravitational' acceleration near the reward, making actions chosen by the computer have a greater effect. Additionally, the computer tended to use the slingshot strategy (employing the gravitational force of the planets to optimize the path to the reward) when the planets were in between the starting position and the reward zone. Since there tends to be a higher percent improvement in this scenario, we can conclude that a slingshot strategy can benefit travel in a 2D space with varying gravitational fields.

The way gravity was calculated could be a source of error due to its lack of accuracy. For simplicity, we calculated the force of gravity as a vector according to Newton's law of gravity at each integer. Then, we rounded the location to the nearest integer and used the calculated gravity at that point (for example, if the ship was located at (4.1, 8.9), we would round it and use the gravity that was initially calculated at (4, 9)). In future simulations, it would be best to use interpolation, finding the gravitational force based on the ship's relative location to the four points surrounding it and taking a weighted average. This would make the gravitational field continuous and could reduce inaccuracies. However, since we had overall better results using reinforcement learning to navigate the plane than using random actions, our results demonstrate that our machine learning algorithm was nonetheless effective and useful.

Another limitation was the computing power available for this project. Since we only had access to one computer, we had to choose a low-computing power program to run our code and we weren't able to test extremes. To complete the project in a reasonable amount of time, we could only run a maximum of 10,000 simulations, 7 *qruns,* and 20 different planet pair locations. With more computing power, we could increase these values to test extremes and run more tests to generate more accurate results. Parallel computing would be a good option for this.

Our project creates an initial foundation for reinforcement learning, exposing the reader to some of its basics that can be expanded on in the future. Already, artificial intelligence is becoming more prevalent and its continued development can help us solve important problems, such as complicated navigational decisions. Looking at various strategies developed by the computer during learning, such as the slingshot method described in this paper, can be another useful application of reinforcement learning.

MATERIALS AND METHODS

Computing and Q-learning

For this project, we modeled reinforcement learning in Google Colab, a Python environment with low computing power. To get around computing limitations such as running the simulations on only one computer processor, we used a relatively simple algorithm, called a Q-learning algorithm (5). This algorithm provides access to reasonable and accurate results even with low computing power. Q-learning is one of the simplest and most commonly-used methods for reinforcement learning because it can converge on the optimal policy given a large amount of test simulation (6). This means that it is effective at finding the best action given a specific initial state. Additionally, unlike other reinforcement learning methods, Q-learning is considered "off-policy," meaning that sequential actions do not rely solely on the initial state. This means that it can continually learn from past actions and does not depend on some initial state to process the most optimal action. The computer can learn along the way to the reward, allowing for many more configurations and better pathing. Although other reinforcement learning algorithms do exist and have been tested by others, we decided to use Q-learning because of the aforementioned aspects that fit our model.

This approach employs a formula that enables the model to learn from past simulations and improve over time by taking the correct actions. For each iteration of the algorithm, the model has an initial state, chooses a random action, and then calculates the reward for that specific action. It then calculates the new initial state and repeats the process (2). For our purposes, we terminated the algorithm when it reached the desired area in the top right of the plane or after 30 iterations.

Basic definitions for the simulations

The number of simulations run (*NumSims)* was the number of random simulations that were run to train an initial model. The Q-learning algorithm was run on an equal number of simulations as *NumSims*. The number of iterations that the Q-learning algorithm was run (*qruns)* tests the rigor of the Q-learning. The higher the value of *qruns*, the more times the algorithm was applied to the data. To test how the location of the planets affected the effectiveness of our model (measured by the value of the reward variable), we generated random locations for two planets in the 16 by 16 two-dimensional grid and ran the simulations on a gravity field based on those locations. To create the gravity field, we calculated gravity using Newton's law of gravity equation (shown below) at each integer point in the 16 by 16 grid.

$$F_g = G\frac{mM}{r^2}$$

Where, F_g is the force of gravity, G the gravitational constant, m the mass of the spacecraft, M the mass of the planet, and r the distance between the spacecraft and planet. Then, the program pulled the value for the force of gravity at each location and used it to determine the direction of movement.

To decrease the simulations' required computing, we only calculated gravity at each integer on our coordinate grid. However, we changed the location of the ship by decimal increments, meaning that the ship could be between any four lattice points. In this case, the location was rounded to the nearest integer when calculating gravity. To calculate gravity, we used Newton's Law of Gravity. Since we don't have defined measurements for mass, the planet and ship each had a set mass of 1 kg. We also omitted the gravity constant because it is just a multiplier. These slight changes to Newton's gravitation equation made the force of gravity experienced by the rocket solely dependent on the distance it was away from the planet. All code is publicly available on our repository (7).

Simulation and evaluation process

For the program to improve based on previous actions, we needed to define a reward. We investigated whether the model could find a path to end up in a reward area and defined it to be a circle of radius 2 in the top right of the plane. We then determined the value of the reward based on how close the spacecraft made it to the center of this reward area which is considered a successful flight path. This ensured that the model could know whether it had successfully navigated to the designated reward area in each iteration, and its success could be quantified. A dictionary was generated, holding the initial location, initial action, the reward value, and the following location. For each simulation, the previously mentioned information was input in one row of the dictionary, representing the data at the location of the spacecraft at a specific moment. This was repeated until the simulation had terminated, which is when we inserted an entry containing -1, which would let us determine the termination of a simulation later when we ran the Q-learning algorithm. This dictionary lets us use the Q-learning algorithm and measure the effectiveness of the algorithm through the value of the reward, where 0 represents not reaching the reward zone and 10 represents the rocket arriving at the reward area.

Random simulations were provided to the model, as test data for the model to train with. We placed the rocket at random locations in the bottom left quadrant and initiated it with a positive vector velocity (where both vector entries were positive). Additionally, although the planet locations were random, we limited the area that they could be to only the center regions of the grid (4 to 12 in both x and y directions) to decrease the average distance between the spacecraft and the planets and therefore increase their gravitational effect on the spacecraft. During the simulation, the model had 30 opportunities to choose and perform a random action (choosing to increase velocity up, left, down, right, or stay still) that would move the rocket around in the grid. After 30 choices, if the rocket did not reach the reward, the simulation was terminated. This way, we could eliminate processing time with possibly thousands of iterations of decisions while still letting the rocket still have the possibility of ending up at the reward. The computer model could reach the reward solely based on motion from the force of gravity, from completing actions, or a combination of both (representative of a ship adjusting its course) (8). After generating a dictionary based on these random simulations, the computer would run the Q-learning algorithm, calculating whether the action taken at a specific point was the "correct" one, or the action that got it closer to the reward (9). After running the Q-learning algorithm on all the simulation data, the computer produced estimates for which action was best at each location and stores it in a new dictionary (we will call it Qfx for simplicity). The simulations were then rerun using the same number of *NumSims* as the random simulations. Now, since we had the theoretical "best" decisions to make at different locations in the grid (located in the Qfx), when we reran the simulations, the computer would look into Qfx, find the optimal action at the location the spacecraft is at, and use that action. We then recorded whether the resulting simulations were better or worse than the initial random simulations.

To determine how much better the machine learning algorithm was at navigating a plane than random simulations, we utilized a separate simulation code to evaluate it. This code measured how effective random simulations were at getting to the reward, how effective a machine learning algorithm was at getting to the reward, and the percent difference between these two values (9).

Received: October 17, 2023
Accepted: August 21, 2024
Published: June 7, 2025

REFERENCES

1. Chen, Xuemei, et al. "Research on Vertical Strategy for Left Turn at Signal-Free T-Shaped Intersections Based on Multi-Layer Reinforcement Learning Methods." *Green Energy and Intelligent Transportation*, 1 Jan. 2025, pp. 100261–100261, https://doi.org/10.1016/j.geits.2025.100261.

2. Tipaldi, Massimo, et al. “Reinforcement Learning in Spacecraft Control Applications: Advances, Prospects, and Challenges.” *Annual Reviews in Control*, vol. 54, 2022, pp. 1-23. https://doi.org/10.1016/j.arcontrol.2022.07.004.
3. Tai, L., Paolo, G., and Liu, M., “Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation,” IEEE International Conference on Intelligent Robots and Systems, Vancouver, Canada, 2017, pp. 31-36. https://doi.org/10.1109/IROS.2017.8202134.
4. Brandonisio, Andrea., Lavagna, Michèle., and Guzzetti, Davide., “Reinforcement Learning for Uncooperative Space Objects Smart Imaging Path-Planning,” *The Journal of Astronautical Sciences*, vol. 68, 2021, https://doi.org/10.1007/s40295-021-00288-7.
5. Kochenderfer, Mykel J, et al. *Algorithms for Decision Making*. Cambridge Massachusetts Institute Of Technology, 2022.
6. Woergoetter, Florentin, and Bernd Porr. “Reinforcement Learning.” *Scholarpedia*, 2008, http://dx.doi.org/10.4249/scholarpedia.2705. Accessed 26 Jan. 2025
7. Rousseau, T. et al. *Extra-Planetary Reinforcement Learning.* Github. https://github.com/ThePapayaInstitute/ExtraPlanetaryReinforcementLearning/tree/main. Accessed 26 Jan. 2025
8. Sonter, Mark Joseph, *The technical and economic feasibility of mining the near-earth asteroids*, Master of Science (Hons.) thesis, Department of Physics, University of Wollongong, 1996. https://ro.uow.edu.au/theses/2862. Accessed 26 Jan. 2025
9. Dayal, Aveen, et al. “Reward Criteria Impact on the Performance of Reinforcement Learning Agent for Autonomous Navigation.” *Applied Soft Computing*, vol. 126, Sept. 2022, p. 109241, https://doi.org/10.1016/j.asoc.2022.109241.
10. Bolles, Dana. “Basics of Spaceflight: A Gravity Assist Primer.” *NASA*, January 2024, https://science.nasa.gov/learn/basics-of-space-flight/primer/. Accessed 26 Jan. 2025
11. Randall Dewey Knight. *Physics for Scientists and Engineers: A Strategic Approach*. Boston, Pearson, 2017.