Spacecraft maneuvering near small bodies using reinforcement learning

Team 10[1]

[1] *Department of Physics; Department of Mathematics and Informatics, St. Kliment Ohridski University of Sofia, 5 James Bourchier Blvd, 1164, Sofia, Bulgaria*

(Dated: January 23, 2021)

We approach the idea of an unmanned spacecraft trying to land on an asteroid from a reinforcement learning perspective. We model a rotating ellipsoidal body and try to teach a reinforcement learning agent to hover above a specific landmark on it. The agent is guided by a Double Dueling Deep Q-Network which approximates the action value functions of the model by independently calculating the state value and state-dependent advantage functions. As any deep Q-learning algorithm, our approach poses the danger of instability with divergence. We explore various strategies in order to improve our algorithm and make it more reliable.

I. MOTIVATION

Spacecraft maneuvering is a difficult task. In the case of a controlled spacecraft we have two instances - either it is manned or controlled from afar. In the former there is danger for the man aboard and the latter is limited because the signal from the spacecraft has to travel to the control room and back, therefore the spacecraft's reaction time is at least $2c * \mathrm{distance-to-control-room}$. So, in a dynamic environment, the idea of a spacecraft that can test its surroundings and then act accordingly with a relatively short reaction time, comes natural. In this project we humor the idea of a spacecraft hovering above a small dynamic celestial body, loosely following the approach in [1].

To aid the reinforcement learning agent in its quest to a fixed target point near the ellipsoid, a Double Dueling Deep Q-Network (D3QN) is used to generate approximate functions for the action values (Q-values) of the environment. These Q-values are used to construct an ϵ-greedy policy, which by definition is biased towards choosing the action which would lead to highest immediate reward only most of the time, thus achieving balance between exploration and exploitation.

The reinforcement learning environment is characterized by a continuous state space and a discretized action space. The agent picks an action according to the policy derived from the D3QN, and receives a state-dependent reward. The new state is then determined by the physical equations of motion of the spacecraft.

This paper is organised as follows: in Section II we go over the physical description of the system and derive the equations of motion for the spacecraft, as well as describe how those translate into a reinforcement learning environment. Section III deals with the implementation of the D3QN, and the convergence issues that may arise from the instability of a DQN algorithm. A brief summary and conclusions are presented in Section IV.

II. DESCRIPTION OF THE PROBLEM

A. Building the environment

1. **Description of the spacecraft and the asteroid**
   A small celestial body is modelled by an ellipsoid with uniform density ρ and smooth surface. Assuming the asteroid is tumbling with principal moments of inertia I_x, I_y, I_z and has dimensions a, b, c along a non-inertial reference frame $\mathcal{F}_S$ with an origin at its center of mass.
   The equations of motion EoM of a spacecraft around said body according to [1] and are as follows

$$\begin{aligned}\ddot{\mathbf{r}} =& -2\omega \times \dot{\mathbf{r}} - \dot{\omega} \times \mathbf{r} - \omega \times (\omega \times \mathbf{r}) \\ &+ \mathbf{a}_{\mathrm{g}} + \mathbf{a}_{\mathrm{SPR}} + \frac{\mathbf{T}}{m}, \qquad (1)\\ \dot{m} =& -\frac{T}{I_{\mathrm{SP}} g_0}, \qquad (2)\end{aligned}$$

where with boldfaced characters are denoted vector quantities; the non-boldfaced characters are the quantities' magnitudes. The first term is the Coriolis acceleration, the second is the Euler acceleration, the third - the centrifugal acceleration, $\mathbf{a}_{\mathrm{g}}$ is the gravitational acceleration due to the small body's gravitational pull, $\mathbf{a}_{\mathrm{SPR}}$ is the acceleration due to solar pressure radiation and other unknown perturbations, T is the thrust of the engines and m is the mass of the spacecraft. Said spacecraft's mass changes with time due to the expelled propellant with a derivative of $\dot{m}$ proportional to the thrust and inversely proportional to the propulsion system specific impulse I_{SP} and $g_0 = 9.8 m/s^2$.

We adopt a model where the asteroid is spinning uniformly, hence there is no Euler acceleration in (1). We also forgo the Rocket equation (2) which can be justified for spacecrafts with smaller engines. The acceleration due to solar pressure radiation is also taken out of the equations of motion in our model. The now simplified equations of motion are

$$\ddot{\mathbf{r}} = -2\omega \times \dot{\mathbf{r}} - \omega \times (\omega \times \mathbf{r}) + \mathbf{a}_{\mathrm{g}} + \frac{\mathbf{T}}{m}. \tag{3}$$

We evaluate each coordinate of the spacecraft's net acceleration $\ddot{\mathbf{r}}$ at each time step and add it to its velocity $\dot{\mathbf{r}}$.

In our model the gravitational acceleration is Newtonian and equal to

$$\mathbf{a}_{\mathrm{g}} = G\frac{M}{r^3}\mathbf{r}, \quad M = \frac{4}{3}abc\rho, \tag{4}$$

where G is Newton's constant and M is the asteroid's mass.

The angular velocity of the rotating asteroid ω is overall unknown, but we can set an upper limit to it. Given that asteroids are mostly piles of boulder with no other force but gravity keeping them together, we set the condition that the centrifugal force should not exceed the gravitational pull. Therefore

$$\omega^2 r \leq \frac{GM}{r^2} = \frac{G\frac{4}{3}abc\rho}{r^2},$$

$$\Rightarrow \omega \leq \sqrt{\frac{G\frac{4}{3}abc\rho}{r^3}}\Bigg|_{r=a} = \sqrt{\frac{G\frac{4}{3}bc\rho}{a^2}}. \tag{5}$$

The spacecraft is described by its mass m and its maximum thrust T. We assume that the spacecraft knows its exact location relative to the landmark at all times via optical sensors. The problem could be further developed by introducing random noise σ_{SN} into the agent's sensory input as well as some other random noise related to the engine's specific impulse $\sigma_{I_{\mathrm{SP}}}$ in (2) - currently not used.

In FIG. 1 we can see the agent interacting with the environment.

2. **Positioning of the spacecraft**
The spacecraft's initial position and initial velocity are randomly sampled from a uniform distribution, similarly to [2]. Their magnitude is also similar to that in [2].

The hoovering position $\mathbf{p}$ is chosen to be right above one of the poles. Further randomness could be induced by obtaining $\mathbf{p}$ by an algorithm similar to sampling on a sphere [1].

3. **States, Actions and Reward functions**

The state space is six dimensional $S = [\mathbf{r}, \dot{\mathbf{r}}]$. Here $\mathbf{r}$ is the current position and $\dot{\mathbf{r}}$ is the velocity, both calculated at each step from the equations of motion (3).

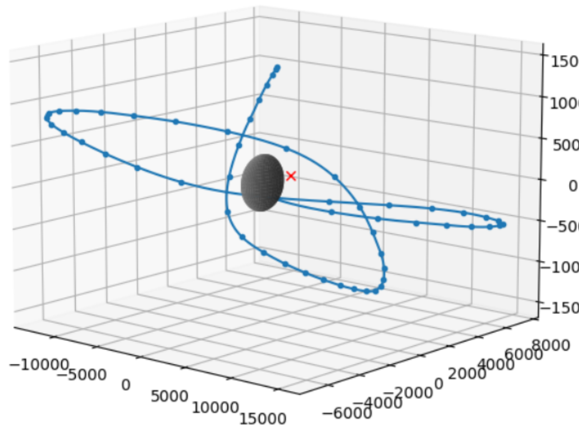

FIG. 1: The spacecraft's position as affected by the equations of motion. Here the agent is taking random actions which make it stray from planar motion. The agent eventually collides with the asteroid as expected due to gravity.

Each of the spacecraft's controllers corresponds to a thruster along the same axis and is associated with a one dimensional action space, with the action being the the thrust along the axis. In the model at its current state there is only one action along each axis that the agent could take. The end goal is to have the agent take continuous actions along each axis varying in magnitude from 0 to the maximum thrust T.

Detailed description of how actions are chosen, through a certain policy, is given in the next subsection.

We tried different rewards where the reward function is a negative function of the relative offset of the spacecraft to the desired hovering position

$$\mathcal{R}(S) = -|\mathbf{r} - \mathbf{p}|, \tag{6}$$

a negative quadratic of the distance

$$\mathcal{R}(S) = -|\mathbf{r} - \mathbf{p}|^2, \tag{7}$$

or a logarithm

$$\mathcal{R}(S) = -\log(\mathbf{r} - \mathbf{p}). \tag{8}$$

The functions gave varying results depending on the parameters of the system but neither proved to be better than the other.

III. DEEP REINFORCEMENT LEARNING FOR SPACECRAFT MANEUVERING

A. Algorithm and Network Structure

We approach the problem of a continuous action space by discretizing the possible action that the agent can

choose in all dimension. This forms a problem that can be solved by Deep Q-learning, an algorithm for deriving an optimal ϵ-greedy policy by approximating the action value functions associated with each state.
We use a Double Dueling Deep Q-Network (D3QN) estimator as an implementation of our reinforcement learning agent. The idea behind a Dueling DQN [3] is that the agent no longer approximates the Q-function of a state-action pair, but instead it learns the state value function, and the *advantage function* of taking any action from that state independently. This approach has proven to be advantageous in the cases of many similarly-valued actions and an environment which is not affected by the actions in a way that's relevant to the agent, both of which are applicable to our problem. The network architecture consists of a 128-neuron fully connected input layer, a 64-neuron hidden layer, which decomposes into two parallel 32-neuron hidden layers with outputs of 1, and (number of actions), the first of which approximates the value function of the state, while the second one gives us the state-action pairs dependent advantage function. We use the ReLU (Rectified Linear Unit) activation function in between every two layers. Then we calculate the Q values from the inputs as follows

$$Q(s_i, a_i) = V(s_i) + A(s_i, a_i) - \frac{1}{|\mathcal{A}|} \sum_{j=0}^{|\mathcal{A}|} A(s'_j, a'_j), \quad (9)$$

where dependence of the Dueling DQN's paramaters ϕ is implied. Here, (s_i, a_i) denotes the possible state-action pairs, Q is the action value function, V - the state value function, while the last term averages the advantage function over all possible actions for a fixed state.

These Q-values are then fit to the targets

$$y_i(s_i, a_i) = r_i(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s'_j, a'_j), \quad (10)$$

where $r_i(s_i, a_i)$ is the reward obtained for observing a given state-action pair, $Q_{\phi'}$ are calculated by a semi-stationary network of the same structure as the main (base) network ϕ which updates only once per a fixed number of updates for ϕ. This additional structure is introduced as a solution to the issue of a single DQN regressing on self-dependent running target. Thus we implement both the double DQN and dueling DQN algorithms in order to try to achieve convergence.

Our algorithm also utilizes a memory replay buffer of a fixed size which stores sequences of data sets of the type (state, action, reward, terminal state bool) that have been observed by the agent either by following a random policy (prefilling) or during learning. The buffer is updated periodically (*ring buffer*), as random of data sets that are to be used for learning are sampled each time the network is updated. This technique reduces correlation between sub-sequential data sets, however, it could introduce a problem of over-fitting sometimes referred to as *catastrophic forgetting* where after a certain point, the network has learned an optimal policy and thus refills the buffer only with samples of that policy. In this way it can "forget" about the non-optimality of some actions, therefore re-learning a policy that is too optimistic, and approximating the value functions incorrectly. In order to avoid that problem, we do not update all of the buffer memory as the agent learns, but keep 10% of it filled with randomly sampled non-optimal actions [4].

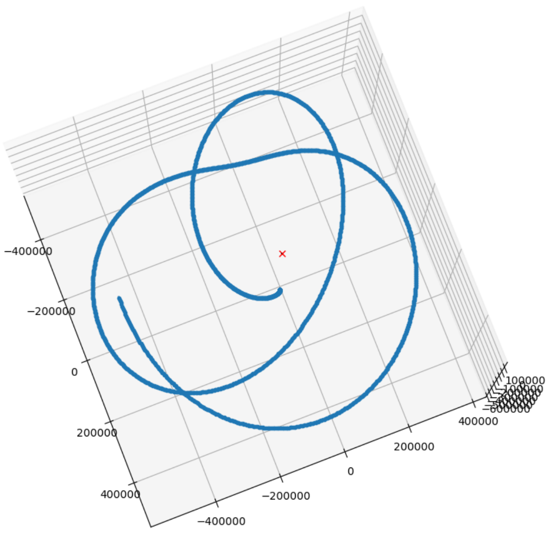

FIG. 2: The spacecraft trying to reach the target point, denoted by the red mark. This trial achieved fairly good convergence, however it took too long, and the episode was terminated by a preset cap on the allowed number of steps.

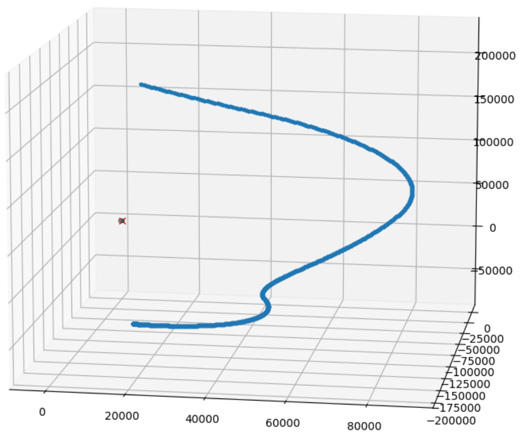

FIG. 3: The spacecraft is fooled by the incorrectly optimistic approximations for the value functions, and so picks non-optimal actions, driving itself away from the target (red mark). Episode was terminated, because the distance to the target got larger than allowed.

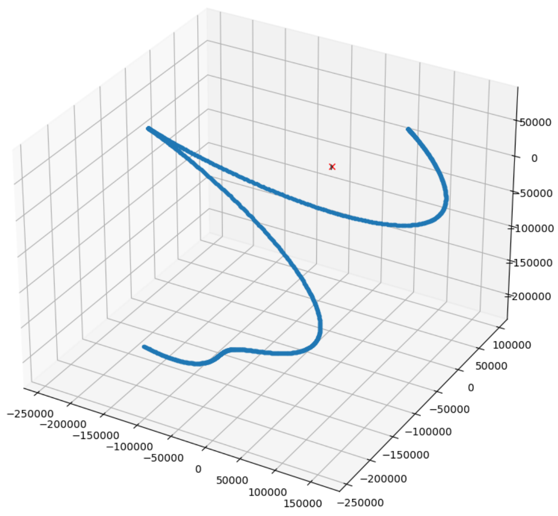

FIG. 4: Even when fine-tuning the hyper-parameters, action values and rewards, the spacecraft still doesn't manage to reach the target, despite having a somewhat good starting trajectory.

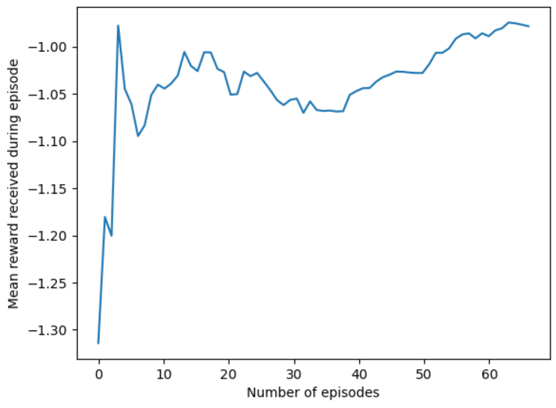

FIG. 5: Mean reward achieved during training episodes. Rewards are scaled as $-|\mathbf{r} - \mathbf{p}|/100000$.

B. Divergence issues and outlook

Sutton and Barto refer to the combination of function approximation, bootstrapping (using existing estimates to update targets) and off-policy training as *the deadly triad* due to the immensely high probability of instability and divergence occurring when all of the three methods are combined into a single algorithm [5] . DQN is exactly this type of algorithm and we were certainly faced with the problem of dealing with the deadly triad. Fine-tuning the hyper-parameters and introducing an exponential learning rate schedule certainly helped with both, as did relying on a more exploration heavy approach and improving on the replay buffer size. We also tried to utilize the technique of limiting (clipping) the values of both the gradients of the loss function and the targets (to prevent divergence), but that too didn't provide satisfactory results. After implementing these improvements, the previously exponential divergence that we observed has improved tremendously, but unfortunately we have yet to be able to get rid of these problems completely, as illustrated in FIG. 2 , 3 and 4. Furthermore, as we can observe from FIG. 5 that it seems as if the approximated policy imposes an upper limit of around -1 for the maximum achieved reward, which prevents us from achieving the desired reward of 0.

An alternate approach of using a policy evaluation algorithm could be taken to see if it would perform better. Further fine-tuning might aid in the battle with the deadly triad as well, however, we believe that had we strictly followed the approach originally outlined in [1] of using a different network architecture with a generational particle swarm optimizer, the task of achieving convergence would have been much more manageable, but unfortunately, both the understanding and implementation of such algorithms are beyond the scope of this project and course.

IV. CONCLUSION

We tried to approach the idea of a deep reinforcement learning agent navigating near a small celestial body, guided by a deep Q-network approximated ϵ-greedy policy. As discussed above, this idea turned out to be unfruitful, partially because of the instability issues often associated with deep Q-networks, as well the possibility that a Q value approximator function based solution is simply unsuitable for this problem. Further testing is required in order to try to implement a more effective neural network structure such as Deep Deterministic Policy Gradient with a continuous action space, or a Feed Forward neural net with a generational swarm particle optimiser.

V. ACKNOWLEDGEMENTS

The code that makes up our project is available for viewing in the following GitHub repository: `https://github.com/rl-team-10/rl_project` .

[1] S. Willis, D. Izzo, and D. Hennes, AAS/AIAA Space Flight Mechanics Meeting (2016).

[2] B. Gaudet and R. Furfaro, AIAA/AAS Astrodynamics Specialist Conference , 5072 (2012).

[3] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, in *Proceedings of The 33rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 48, edited by M. F. Balcan and K. Q. Weinberger (PMLR, New York, New York, USA, 2016) pp. 1995–2003.

[4] T. de Bruin, J. Kober, K. Tuyls, and R. Babuska (2015).

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, Cambridge, MA, 2017).