Machine Learning Engineer Nanodegree

Capstone Proposal

Jürgen Mollen

Oct 4th, 2019

Proposal „Space Shuttle World“

Domain Background

I would like to examine (and solve) a theme in deep leaning robotics. The idea is to navigate a space shuttle from a source position to a target position (‘docking bay’)

This would be a continuous deep Q -Learning problem. I would like to adopt what I have learned in Q –Learning and neural networks to find a solution. To simplify the domain space, the problem shall be in a 2 dimensional world. And the shuttle shall have a varable throttle, and be able to apply it to any direction by rotating along its axis. The angular speed of the rotation shall be limited.

In practice, this is a path optimization problem and there are a lot of akin problems in real life:

- Moon landing
- Air plane approach and landing
- Navigating a ship to its habour dock
- Flying a Zeppeline
- Finding the optimal trajectoy in a Formula1 cirquit
- …

So solving this, will show also a way to solve the related problems. Of course the physics have then to be adapted (slightly) to the specific situations. In moon landing, e..g. this is basically the additional constant g-force heading downwards. An airplane has special friction and elevation forces, etc.. Rotational inertia has to be taken into account, and also adhesing friction of tyres in case of Formula2, etc.

Here are some links to further resources dealing with some of the above topics:

https://digitally.cognizant.com/evolutionary-ai-the-next-giant-leap-for-artificial-intelligence-codex4871/

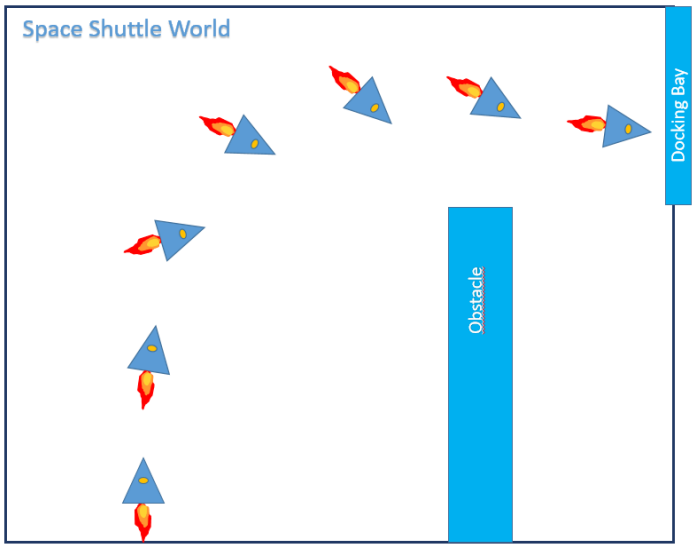
https://www.bbc.com/news/technology-48908346

https://gym.openai.com/envs/LunarLander-v2/

I have chosen this problem, because it - in its specific kind - is new and simple enough for me to write the pysics and (hopefully find a good solution), all from scratch in the provided time frame.

Problem Statement

Please see the following sketch of the 'space shuttle world' below:



The world consists of an rectangular area with an rectangular obstacle and a docking bay.

The space shuttle starts from the lower left corner heading upwards and with zero velocity. It accelerates with a variable throttle. The shuttle can rotate (with a limited angular velocity) and such steer its direction. Please note, that the velocity vector and the throttle vector usually don't have to be aligned as indicated.

The goal is that the shuttle reaches the docking bay with whatever velocity. That is, its leaves the world to the right within the vertical limitation of the docking bay.

If the shuttle leaves the world by any other means or hits the obstacle, the episode ends (too).

As simplification, the spatial dimension of the shuttle is considered as a point, when verifying its allowed position. Also the thrust does not reduce the mass of the shuttle, so its acceleration is always constant for the same throttle.

- The state vector is: positon vector, velocity vector, rotation.
- The action vector is rotate (left, right, don't rotate) , throttle

I have written a physical simulation of this world. Please see attached file 'SpaceShuttlePhysics.py'. It simulates the flight of the shuttle based on straight Newton mechanics.

It contains also a naïve heuristical steering (autopilot). One can test a flight by starting the file 'SpaceShuttleTest.py' in a python3 console. It opens a matplot, showing the trajectory of a flight under heuristic autopilot. At the end of the document I have attached a sceenshot of the 'autopilot' flight.

Datasets and Inputs

There are no datasets used. As 'Input' I will provide the simulation and a start position with velocity zero and the thrust heading down (along the Y axis), like indicated in the picture above.

Solution Statement

I consider the problem solved, when the shuttle (steered by the AI) is consistently able to find its way to the target.

This means, when the reward plot against episodes is stable for (100) episodes (stable will mean, it does not change more than a certain threshold, e.g. 5%). Then the shuttle will leave the world through the docking bay.

Benchmark Model

I will compare the time needed shuttles flight against the autopilot flight (which is already a solution to my problem) and the flight of the AI shall be quicker, i.e. it will take less time.

To achive this I will have to penalize time and thus, I will try to design the task and reward in a way such that the shuttle reaches its target position as quick as possible.

To achieve this - from my point of wiew - it would make sense, to use a reward scheme as follows:

- At each time step add a reward of -1
- If the shuttle reaches the goal, add + 200
- If the shuttle crashes, add -100

(please note that this is just an initial suggestion for the reward function to show my anticipated strategy - I might need to tweak it in detail)

Evaluation Metrics

Following the above said, the evaluation metric will be:

- The difference of the time of flight needed by the (naïve) autopilot minus the time of flight needed by the AI shall be as big as possible.

Project Design

I will use and implement this Q -Learning training algorithm (taken from Udacity's cartpole project):

Initialize the memory DD

Initialize the action-value network QQ with random weights

For episode ←1←1 **to** MM **do**

Observe s0s0

For t←0t←0 **to** T−1T−1 **do**

With probability ϵϵ select a random action atat, otherwise select $at=\text{argmax}aQ(st,a)$at=argmaxaQ(st,a)

Execute action atat in simulator and observe reward $rt+1$rt+1 and new state $st+1$st+1

Store transition $<st,at,rt+1,st+1>$<st,at,rt+1,st+1> in memory DD

Sample random mini-batch from DD: $<sj,aj,rj,s'j>$<sj,aj,rj,sj'>

Set $Qj=rj$Q^j=rj if the episode ends at $j+1$j+1, otherwise set $Qj=rj+\gamma\text{max}a'Q(s'j,a')$Q^j=rj+γmaxa′Q(sj′,a′)

Make a gradient descent step with loss $(Qj-Q(sj,aj))2$(Q^j−Q(sj,aj))2

endfor

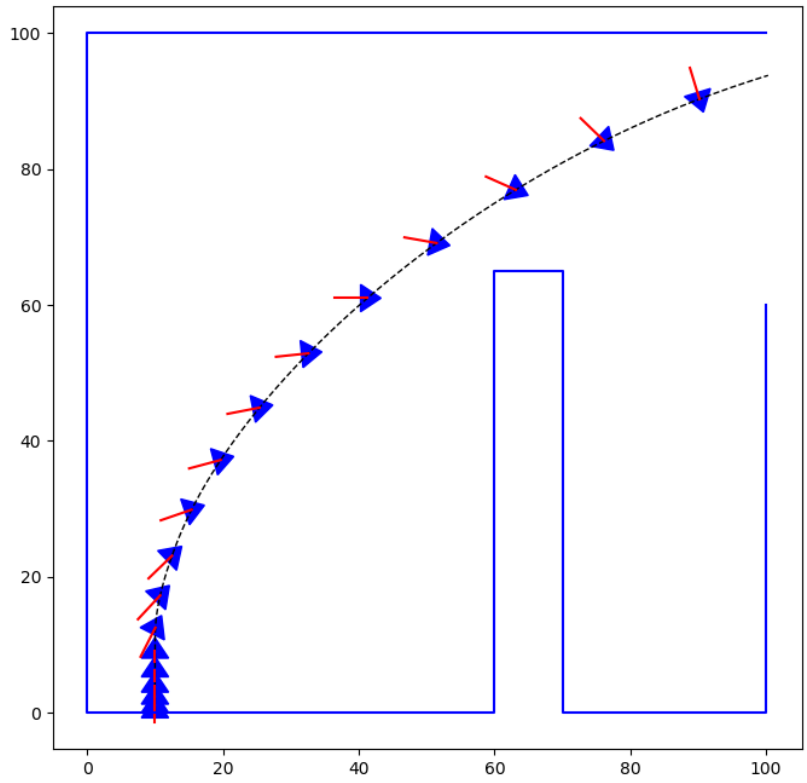
endfor

OR

Use (and adapt) an Actor Critic DDPG agent and task

Physics simulation

The following picture shows a simulated episode of the space shuttle simulator:



(It shows a flight steered by the built-in heuristic autopilot, based on a linear policy)

The physics step itself is done in the method 'next_timestep' that gets the actual action as parameter (rotation direction and throttle)

The essential step in the physics is the following:

- self.vel += self.get_trust_vector() * self.dt
- self.pos += self.vel * self.dt

This means, the velocity increases by the applied thrust (i.e. the force) and the location increases by the velocity - both linear over time. This is Hamiltons way to write Newtons law.

This is numerical equivalent with the following (naïve) view:

- The object moves over time with iths velocity (constant if no thrust).
- When the trust is applied, this movement is overlayed with a quadratic acceleration in direction of the force.
- The velocity after each time step is then 'corrected' by the formula:
    - vel(t+delta_t) = (pos(t+delta_t) –pos(t)) / delta_t (… as velocity is distance per time).

(As a side note, from the numeric standpoint, the advantage of this way to calculate the mechanics is that the simulation is numerically very accurate with respect to the kinetic energy, and thus generally too)