Artificial Intelligence course - project report
Inverse pendulum balancing in 2 dimensions.

Gašper Kojek
Faculty of Computer and Information Science
gk9507@student.uni-lj.si

***Abstract*— I have implemented a reinforcement learning solution for two dimensional inverse pendulum balancing problem. I used ASE/ACE agent in a discretized space, which lead to the agent successfully balancing the pole for more than 15 minutes (190,000 steps).**

I. INTRODUCTION

The single dimension (cart moving only left or right) inverse pendulum balancing problem [1] is a pseudo-standard benchmark problem for testing of different controllers and artificial intelligence approaches on complex and unstable nonlinear systems. I have implemented the solution for the single dimension problem, using reinforcement learning ASE/ACE agent, which I have modified and duplicated to accommodate the two dimensional problem, in which the cart can move in 4 directions.

II. METHODS

I have implemented the reinforcement learning agent using ASE/ACE agent algorithm in discretized state space. I chose the ASE/ACE algorithm over Q-Learning because it was shown that it learns faster [2] , which is even more important in 2 dimensional space than in single dimension.

A. State space

State space consists of four continuous variable per each dimension: cart position, cart speed, pole angle and pole angular speed. For space discretization I used the *Boxes method* [3]. I have tested my agent with discretization proposed by Barto, et al., which splits continuous state into 162 discreet boxes, which I attempted to further improve by splitting the continuous state into 384 boxes - Table I. This proved to bring no visible improvements, while taking a longer time for the agent to explore all the states, which consequently lead to longer time till the agent balanced the pole for more than 300 seconds.

TABLE I
DISCRETIZATION - CONTINUOUS STATE SPLIT POINTS

	162 boxes	384 boxes
Cart position [m]	$\pm(max * 0.5)$	$\pm(max * 0.5)$, 0
Cart speed [m/s]	± 5	± 4, 0
Pole angle [deg]	$\pm(max/2)$, ± 2, 0	$\pm(max/2)$, ± 2, 0
Pole angular speed [deg/s]	± 50	± 50, 0

B. Agent algorithm

The reinforcement algorithm implemented in this research is the ASE/ACE algorithm, which consists of two neuron-like adaptive elements (ASE and ACE). The ASE (Associative Search Element) is a mapping from states to action ad implements a decision policy, while the ACE (Adaptive Critic Element) provides an evaluation of problem states and functions as a prediction of failure.

This algorithm consists of 4 weights per cart dimension for every state box. These are the weights for ACE and ASE, which persist between trial runs, and eligibility traces weights for ACE and ASE, which are reset between trial runs. ASE weights hold the current optimal strategy for each state box, ACE weights hold the failure prediction for current state box and eligibility traces are values, that represent the current frequency of state visitation. When the pole falls down and the agent gets a negative reward, that reward influences boxed states differently, depending on current frequency of state visitation - eligibility of each box state for the reward.

III. IMPLEMENTATION

In the first phase implementing and testing I implemented the agent and environment in Python. The first time I coded the physics environment [4], [1], then I used the PyGame [5] library to simulate the physics. Both of these tests were one dimensional.

In the second phase I tried to use the GazeboSim [6] robotics simulator, which I couldn't get to work, so I abandoned the GazeboSim, as I have already lost a few days working on it.

In the third phase I used the Unity Game [7] engine for physics simulation, which was also the final step in the development, as I could finally test the two dimensional problem. For this I had to re-code the code in C# and implement a different approach to persisting data between trial runs.

IV. RESULTS

Using the Unity Game Engine I modeled the two dimensional inverse pendulum balancing environment, which I used for testing the agent. After playing with different settings of the environment, the 384 box version balanced the pole for 951 seconds, which is 190,000 steps in trial 369, while the trial run 378 seemed to go to infinity (I stopped it

after 30 minutes). The 162 box version balanced the pole for a maximum of 403 seconds, which is 80.690 steps in trial 89.

The most notable changes that had the biggest impact on the performance of the agent were: time step (5 ms), α - ASE learning rate (37), β - ACE learning rate (0.4), γ - Reward discount factor (0.993) and λs - Decay rate for eligibilities (0.98).

V. CONCLUSIONS

I have learned a lot while developing, implementing and testing the reinforcement learning algorithm and its agent. I have successfully implemented an agent that is capable of balancing the pole for a prolonged time in 2 dimensional world. I regret loosing so much time trying to make Gazebo simulator work, as I would rather spend it on testing the agent in real world scenario, using an industrial robot arm instead of a cart. Considering the faculty has at least one such arm, I would really enjoy testing the agent with it, since it works so well in simulations.

REFERENCES

[1] J. Brownlee, "The pole balancing problem, a benchmark control theory problem," July 2005.

[2] T. Vodopivec, "Spodbujevalno učenje na problemu vozička s palico," 2011.

[3] C. W. A. Andrew G. Barto, Richard S. Sutton, "Neuronlike adaptive elements that can solve difficult learning problems," *IEEE transactions on systems, man, and cybernetics*, 1983.

[4] R. V. Florian, "Correct equations for the dynamics of the cart-pole," 2005.

[5] "Pygame python library," http://www.pygame.org/hifi.html, accessed: 2016-06-13.

[6] "Gazebo simulator," http://gazebosim.org/, accessed: 2016-06-13.

[7] "Unity 3d game engine," https://unity3d.com/, accessed: 2016-06-13.