**Question 1**: Describe the task that you specified in task.py. How did you design the reward function?

**Answer**:

My task is specified as takeoff.py which is a simple task where the goal is to lift off the ground and reach a target height of 100m.

My first attempt after coding was that no takeoff was occurring using the initially provided reward function: # original provided reward function in the get\_reward() function.

reward = 1-(0.3\*(abs(self.sim.pose[:3] - self.target\_pos))).sum()

This attempt was by using the signed difference between sim.pose & target.pose which as expected was leading to very poor score and almost no learning as the variance of the reward was very high and unbounded.

With some research, I found this article by Anne Bonner that suggests the use of the tanh function in the reward function. <https://towardsdatascience.com/how-to-train-your-quadcopter-adventures-in-machine-learning-algorithms-e6ee5033fd61>

So for my next attempt, I changed to using the non linear function 'tanh' which bounded the reward to range [-1,1] . For this to work, some scaling was done to make values meaningful for tanh function. With this function, the drone receives a reward between -1 and 1 at each step of the each episode according to the difference between its current position and target position for x, y, and z dimensions.

Initially the tanh reward function used was:

np.tanh(1 - 0.3\*(abs(self.sim.pose[:3] - self.target\_pos)).sum())

I discovered that lowering the "0.3" constant gave much better results. After much trial and error, making the 0.3 smaller and smaller and rerunning 500 episodes each, I decided to use:

reward= np.tanh(1- 0.0002\*(abs(self.sim.pose[:3] - self.target\_pos)).sum())

In addition to the improved results, the drone also learned much faster with this change.

•The last reward function I tried was scaling the current distance self.sim.pose[:3] to the agent's target position self.target\_pos between -1 and 1

reward = np.tanh(1 - 0.002\*(abs(self.sim.pose[:3] - self.target\_pos))).sum()

![Alt Text](http://mathworld.wolfram.com/images/interactive/TanhReal.gif)

Source: http://mathworld.wolfram.com/images/interactive/TanhReal.gif

During this initial testing, I had set the neural network to large dense state and action inputs, such as 400, 300, 300 with the default hyperparameters, and achieved improved score results after 200 episodes with the score/reward reaching between 500 and 593. The drone however was still not achieving liftoff to the desired 100, so I looked at making improvements to the step function in my takeoff.py.

I found that the reward function has to be coded as such to help the agent learn. Designing the reward function is one of the most important area for the agent to learn well and the penalties and reward signals have to be tailored to achieving goal and guide the quadcopter along the learning process. A training video by Ross Story, Data Scientist at Bonsai, provides a good talk about writing great reward functions in reinforcement learning at <https://www.youtube.com/watch?v=0R3PnJEisqk&list=PLAktfMEMCsOY9HUZKIuGI6yqefGBuszAV&index=4>

In this article, it is suggested that combining positional aspects and velocity could be useful  [https://medium.com/@BonsaiAI/deep-reinforcement-learning-models-tips-tricks-for-writing-reward-functions-a84fe525e8e0](https://medium.com/@BonsaiAI/deep-reinforcement-learning-models-tips-tricks-for-writing-reward-functions-a84fe525e8e0" \t "_blank)

Using these ideas, I modified the step function to reward +5 if when the z pose was above 10, then additional 10 within 50 meters of the target, and then additional 5 whenever the quadcopter was within + or – 5 meters of the 100meter z pose target, like this:

# approach to target

if self.sim.pose[2] >= 10:

reward += 5

# good approach to target

if self.sim.pose[2] >= (self.target\_pos[2]-50):

reward += 10

# when within +/- 5 of target pose (close to target)

if (self.sim.pose[2] >= self.target\_pos[2]-5) and (self.sim.pose[2] <= self.target\_pos[2] + 5):

reward += 5

if done :

reward += 10

This showed improvement after much tweaking, because I believe the copter was flying in circles during my trial and errors. Then I started lowering the neural network parameters finally ending up with state and action inputs, such as 40, 30, 30, 30 with the default hyperparameters,