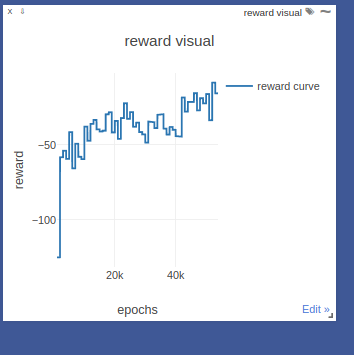
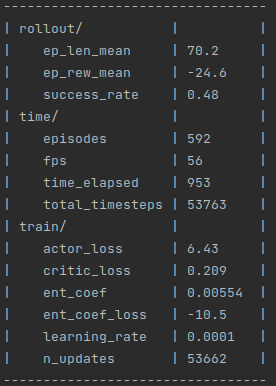
14/12/2022

model = SAC(MultiInputPolicy, learning\_rate=1e-4, gamma=0.95, env=env, verbose=1)

the above parameter is the same for all training here.

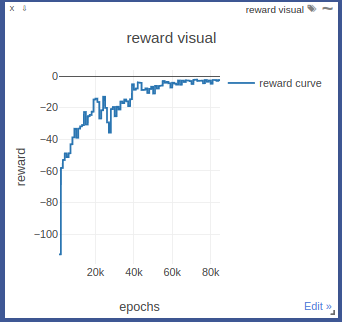
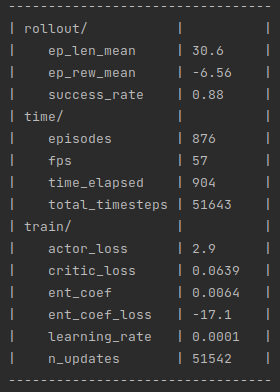
trying to train the robot with the simplest seeting, reaching without regulation. The action is set to +-pi max, but limit to 0.3 sec every step. Success rate is about 0.5 after 50000 epochs. The training processed 3956 steps in 60 seconds.



First we’ll try turn off the rendering and see if that increases the processing speed.

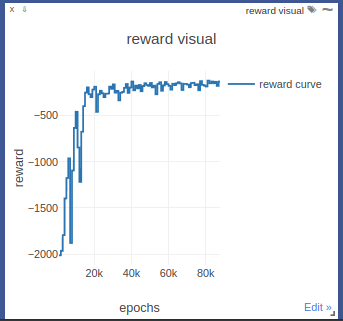
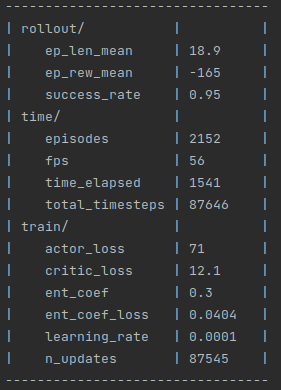
2719 steps processed in 60sec after turning off the render, not improving the training speed.

After changing the action limit to 0.1 sec per step, the training perfomace improved significantly. All other parameters are same with the previous one.



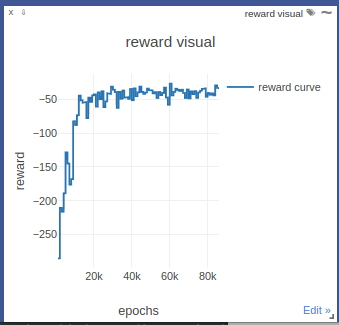
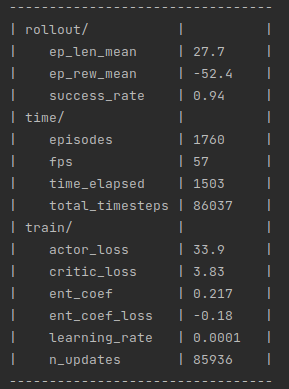
Reach + joint regulation:

Using the ? Function for distance reward, joint regulation coefficient -1, distance coefficient -200, no collision detection.



next we will try to change the distance coefficient to -20.

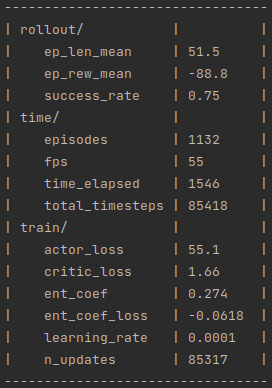
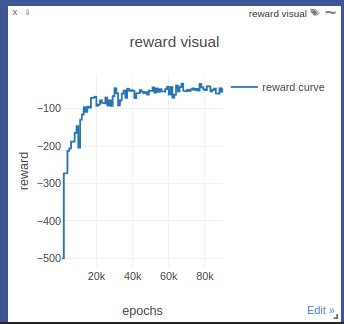
Can’t spot a big difference between this trial and the previous one



Now we try to change the regulation weight on each joint to encourage less motion on the big joint

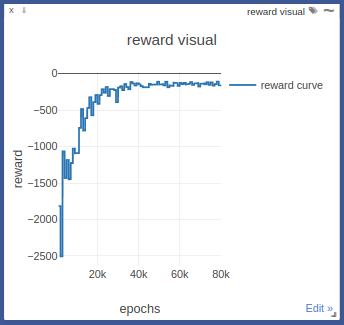
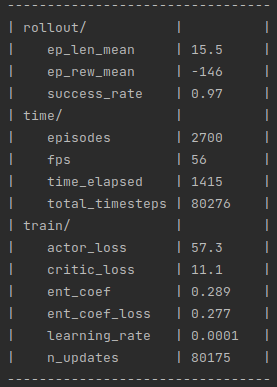
np.sum(np.dot(np.square(self.robot.get\_action()), [2.0, 1.8, 1.6, 1.4, 1.2, 1.0])) \* self.action\_weight

the peformance is not improved by doing this



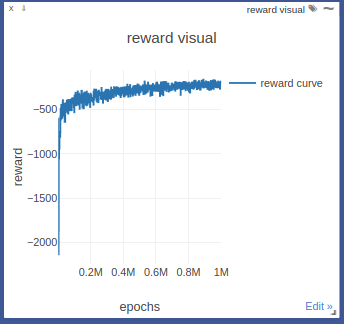
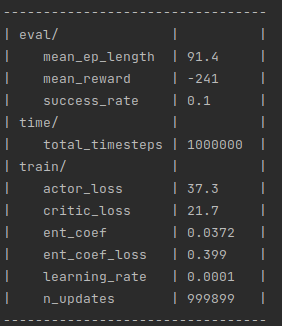
Now we try to add collision detection to the reach + regulation training, once the robot collide with table or track, -200 reward will be applied.

Performance improved after adding collision detection



15/12/2022

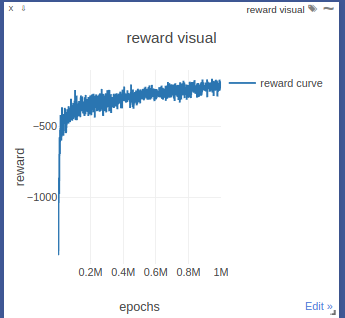
Try to train the robot with orientation matching, using the collision weight -200, translational distance weight -200, orientational distance weight -4. The model seems can still improve after 1 million steps. However, the success rate is only 10%. I suspect there are orientations cannot be achieved.



wrote a program that controls the robotic arm using inverse kinematic only, after 100 exxperiments and 100 trials per experiment, we get a successful rate 14.25%. This kind of proves that reach 10% success rate with reinformcement learning is almost hitting the plato and the reward curve tend to flat. However, a 6DoF robot should be able to any point in the 3D space, we need to confirm the success rate with something else, for example, traditional algorithms like RRT connect.



Do another training with the translational distance weight -160, orientational distance weight -4. (the optimal ratio we carried out from 10 experiments). The result is not that good, not good as the previous one



found some problem with the previous robot model, after changing the robot model. Success rate for inverse kinematics solver become to 24.37%, which is still pretty low. Maybe need to look into why the joints cannot achieve the desired pose.

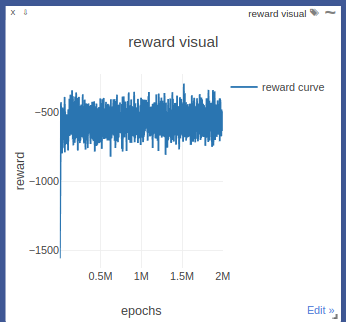


16/12/2022

Found some problem with the pybullet inverse kinematics solver, which makes the success rate of inverse kinematics very low, trying to use the ik\_fask package to solve inverse kinematics. There are some problem with the quaternion rotation system on the new ur5e urdf file, still working on that.

Perfrom another training using the new ur5e model with 2000000 steps, hope we can see some better result.

The reward curve doesn’t even converge in this training, need to find out why



19/12/2022

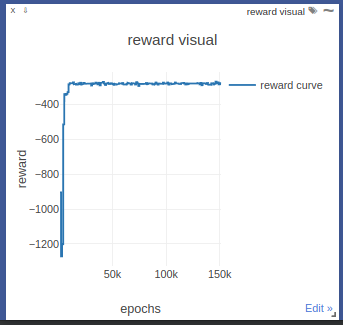
Finally solved the inverse kinematics issue, now if we run the code for 10000 times, the solving success rate reachs 92%. However, we also noticed that 8% failaure rate remains and the no solution found situation occured 224 time. Which means not all the end point configurations can be achieved by the robotic arm.



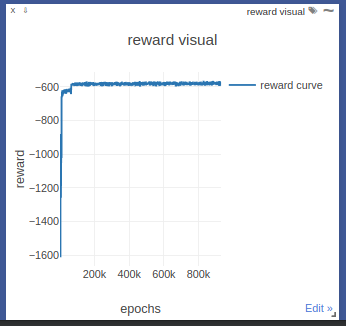
For the next step. We can use the inverse kinematics solver to guide the reinforcement learning training process. When a random goal is generated, the goal will first be verified by the inverse kinematics solver to make sure it is a achievable one.

We will put the algorithm for another 2000000 epochs of training and see the result.

The robot is keeping knocking itself on the table, probably because the collision punishment is not big enough.



Enlarge the collision punishment to -500 and try again



something was wrong about the quaternion calculation before so none of the result is valid.

To make it clear, in python:

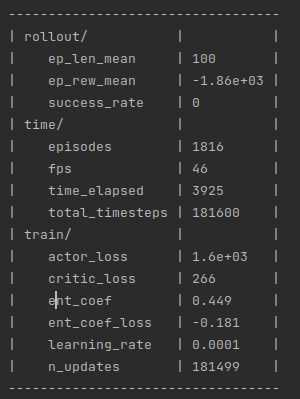
The quaternion package uses default quaternion representation as (r, x, y, z) real number first

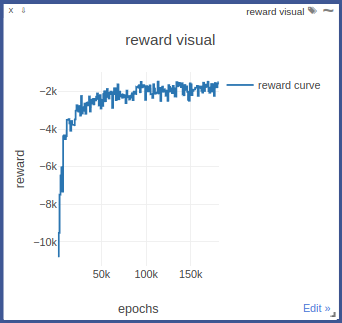
The pybullet package uses default quaternion representation as (x, y, z, r) real number last

The ur\_kinematics package uses default quaternion representation as (x, y, z, r) real number last

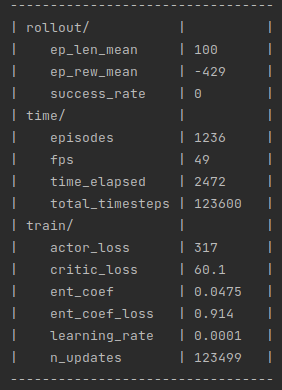
all format will unified to (x, y, z, r) for simplicity from now on.

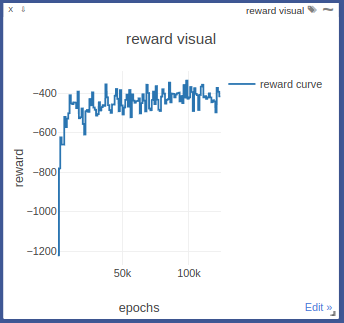
Another training without collision checking and joint regulation, see if the robot can find the pose in the most simple setting

stopped training since it seems not learning anything, the position is roughly correct but not the orientation. This is expected since the reward on orientation is very low

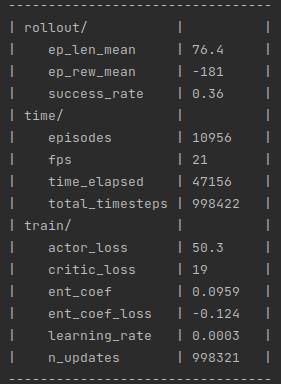
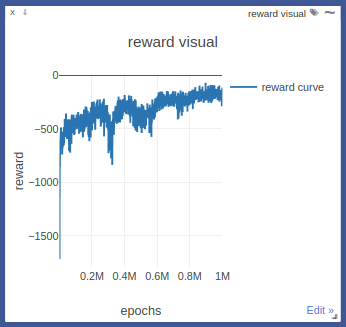


Stopped this training early as well, cannot see a good trend, distance weight -16, orientation weight -4.





HER is added to get better result, we use SAC to train for 1M epochs with -16 distance weight and -4 orientation weight. The reward reached -150. success rate rreached around 36%. (highest 50%)



we will try again with exactly the same parameters but using DDPG.

In the previous experiment, we have determined that -16 and -4 is the best ratio for distance and orientation reward. But since the code has changed so much, it is worth to do the experiment again.

Other things we can try:

increase the HER buffer size

use a linear learning rate rather than the constant value

increase the number of training epochs

use the inverse kinematics result in reward calculation (this is kinda cheating)