6-DoF Manipulator PID Tuning with Deep Reinforcement Learning

Vineeth Vajpey
BS Computer Engineering, UMD
vvajipey@umd.edu

Vedant Ranade
M.Eng Robotics, UMD
vedantr1@umd.edu

Sameer Mustaqali
BS Computer Science, UMD
smustaqa@umd.edu

1 Introduction

In the space of robotic manipulators, motion control of a manipulator and grasping strategies are 2 major challenges faced by engineers. Developing a well performing controller and a grasping strategy for different kinds of objects is important in the satisfactory operation of a task for a manipulator. Thus in this project, we plan to find optimal PID gains of motion controllers of a 6 degree of freedom, serial manipulator robot in the simulation space, using MuJoCo as a physics engine.

6 DoF manipulators are popular in their use as a non-mobile Pick and Place/CNC machine tending/Grasping/Welding/Painting Robot while also being deployed on mobile platforms, as well as in space stations, underwater platforms etc. As the payload on the end effector of a robot changes, for eg, while it is performing a pick and place operation, and as it moves in different configurations, the gravity + friction loads felt by each joint vary a lot, throughout the expected set of motion. Tuning a PID controller for all such sets of motions and a variety of payloads,exhaustively, is almost impossible for humans to pull off. And thus, the gains often require tuning for some new applications as prior untested configurations propose slightly different challenges. Similarly, grasping of objects in a non dynamic, known environment, is done by a human programming ("teaching") the robot. To achieve autonomy in the grasping space, in dynamic environments where the object to be grasped changes, a robot has to figure out suitable grasping technique for the new type of object on the fly. Both of these tasks benefit highly from learning based approach, where deep reinforcement learning can be used to figure out robust controller gains for a variety of configurations, and suitable grasping techniques for different objects.

2 Data

As this task is reinforcement-learning based, all data used is generated by the simulated environment and outputs of the manipulator. There is no need for external training data.

3 Related Work

3.1 Stability-Preserving Automatic Tuning of PID Control with Reinforcement Learning

Our Deterministic Policy Gradient approach to training PID parameters was heavily influenced by the paper Stability-Preserving Automatic Tuning of PID Control with Reinforcement Learning by Ayub I. Lakhani1, Myisha A. Chowdhury1, and Qiugang Lu. [2] This paper addresses the issue of PID tuning of simple second order system, without any particular application and looks to automate the process using reinforcement learning. The novelty in the paper lies in the development of a tuning framework that allows for the actor and critic networks to be updated once at the end of each episode. The paper builds upon the foundations of PID tuning and reinforcement learning concepts to develop an architecture that combines both systems and implements their stability preserving pid tuning with DPG algorithm. This algorithm is what is used in our DPG implementation.

3.2 Autotuning PID control using Actor-Critic Deep Reinforcement Learning

Another paper we took influence from was Vivien van Veldhuizen's Bachelor's thesis from the University of Amsterdam titled Autotuning PID control using Actor-Critic Deep Reinforcement Learning. [4] This paper tackles the same issue of tuning PID control parameters in the application of robotic apple harvests. The paper implements the Advantage Actor Critic (A2C) algorithm on a robotic arm in simulation. This arm is functionally the same as ours, but has a different model, and while the paper utilizes the ROS framework, ours uses Mujoco. The task addressed is to move a robot to an apple in space, pick it up, and return to the original position as quickly as possible. This involves tuning the PID gains as precisely as possible. These gains are trained with the Advantage Actor Critic reinforcement learning algorithm. Veldhuizen first tested various algorithms on OpenAI Gym reinforcement learning environments modified to include PID controls, and ultimately decided on the A2C method. This decision was made because of the choice to work with a continuous action space, as compared to a discrete action space for deciding PID values. Another reason mentioned was the robustness of the A2C algorithm for future improvements and modifications. Implementing a method in which PID gains are modified within a single motion would be supported by the A2C method, but not other policy gradients.

4 Our approach

4.1 Mujoco

We began by setting up the model and environment for our 6 degree of freedom robotic manipulator. Mujoco, a physics simulator developed by Deepmind, was used to visualize and receive physics feedback data about the robot. Setup involved the installation of Mujoco 2.1.0 and converting the original manipulator .urdf model to the Mujoco supported .xml file. The manipulation of the robot's joints was implemented in python using the Mujoco API. The baselines for the tuning parameters were tuned using conventional methods using this interface. Deep reinforcement learning methods were used for tuning parameters for improved and optimal performance.

4.2 Deep Q Network (DQN)

An early approach was to employ a DQN, simply described as a neural network that approximates Q values given a state, outputting the predicted Q values for each action in the action space, prompting a selection of action whose greediness vs randomness is decided by the epsilon constant. The DQN tested had an input space of 6 joint angles, two hidden layers of size 64, and an output layer of size 54. At this stage, the task at hand was different from the final task, and the DQN was used to try to best tune the 18 gains (3 for each joint) to move to a final position from a starting position. The 54 outputs represented 3 increments (+1, 0, -1) for each of the 18 gains. These increments (inference) would be done each time step, tuning the PID gains throughout each episode in an online fashion. As the action space had to be discretized, and training brought little results, we deemed this approach unfit and moved onto to other approaches.

4.3 TD3 Deep Deterministic Policy Gradient (DDPG)

One method that we used to the task of tuning our weights, we took significant influence from the paper Stability-Preserving Automatic Tuning of PID Control with Reinforcement Learning from researchers at Texas Tech University[2] and Mathwork's[1] article on Tuning a controller using a DDPG agent.

In this approach, a TD3 based, continous action space, DDPG agent was used , per joint of the robot, to learn the PID gains, instead of learning updates of the same. The agent , was a single linear layer which takes 3 inputs and generates a action(Actor Target network is chosen to be architecturally similar to the Actor network). The PID controller can be shown as follows:

$$\begin{bmatrix} e & \int e.dt & de/dt \end{bmatrix} @ \begin{bmatrix} K_p & K_i & K_d \end{bmatrix}^T$$

where e = $\theta_{setpoint}$ - $\theta_{current}$ is the current tracking error. Thus the actor in this architecture is the same as the controller, and actor output(action) is the same as the controller's control action, the trained actor's weights are the controller gains. The agent's job is to tune the actor, so that the

following LQG cost[1] is minimized:

$$J(t) = \lim_{T\to\infty} E\left(\frac{1}{T}\int_0^T \left(\left(\theta_{set} - \theta_{Curr}\right)^2 + \alpha u(t)^2\right)\right)$$

Where u(t) is the control output/actor output of the controller/actor and α is a tune able parameter which controls contribution of the control action, on the cost function. Thus a LQG reward function[1] was chosen to model this cost.

$$r(t) = -\left(\left(\theta_{set} - \theta_{Curr}\right)^2 + \alpha u(t)^2\right)$$

Additional constant cost of 100 was added, when a control saturation was detected, penalizing the agent for reaching the limits of control efforts. The critic and critic target network architecture is as follows: In Total, the DDPG agent approximates Q-Value function(Expectation of long term reward

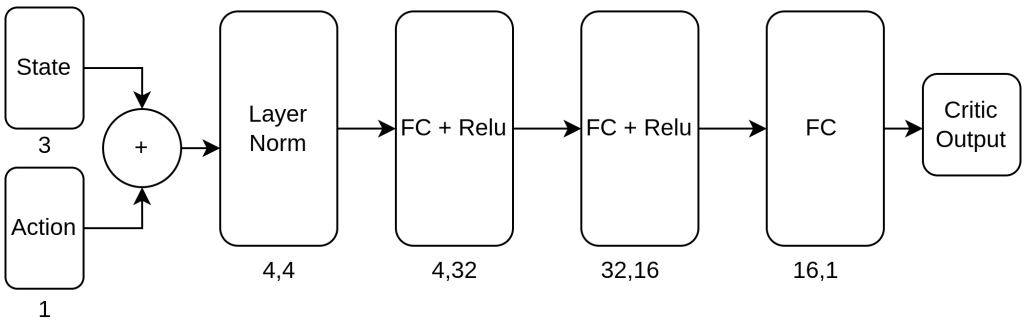


Figure 1: Critic and Critic Target Network.[3]

as a function of state and action) using 2 critic networks, and 2 critic target networks. In total, the agent has 6 deep networks , which work as function approximators. A motion trajectory of 2500 points, with a timestep of 0.002ms , was generated using velocity IK of the robot,and the controls problem is to track this trajectory accurately. The trajectory aims at drawing a circle with the robot's end effector. Link for Robot Trajectory Visualization

4.4 Advantage Actor Critic (A2C)

The decision to employ the A2C algorithm was based off the inefficiencies of the DQN. With the DQN, we were limited to a discrete action space, from which we could only take one action at a time. As the action space was limited to incrementing one gain by 1 unit each action step, there was little utility for the enormous exploration task that was tuning 18 gains at once. With the A2C approach, we inferred and trained every episode instead of every timestep, meaning that a full simulation of the task (following a circular trajectory) would be run before backprop. Each episode, the A2C networks were used to directly output a entirely new set of 18 gains, with the input being a 2700*6 length vector of joint positions throughout the whole 2700 steps of each episode. The A2C consisted of 2 networks, the actor and critic. The actor network approximates a policy function, where it would be used to choose an action given a state as an input. As this version of A2C allows for a continuous action space, the actor network would output a mean and standard deviation for each of the possible actions. This mean and standard deviation would be used to generate a distribution. An action value would be sampled from this action distribution and fed through a sigmoid to get the final continuous output. In the case of taking multiple actions simultaneously, multiple actions (18 actions for each gain) would be sampled at once from a multivariate normal distribution, generating 18 outputs. These outputs would be scaled by predetermined values for each gain. The critic network is a state-value function, approximating how good a state is, based on a loss generated from the reward (sum of PID errors for each step) and predictions from the previous episode/step. Both networks have 2 hidden layers of length 256, with inputs of length (6*2700 = 16200). The output of the actor network is length 18 and the critic network outputs a scalar representing the value of the input state.

5 Experiments and Results

5.1 Experiments with A2C

5.1.1 Crash detection

The including of a crash failsafe measure was very important. Prior to this, the simulation would not get through many episodes until MuJoCo crashed due to extreme joint velocities. These velocities likely occurred from oscillations caused by the PID controller attempting to over-correct itself once it gets off track. In order to correct these occurrences, a measure was implemented in the simulation step function that would detect joint velocities above a certain threshold and zero them out. In addition to zeroeing the velocities, the manipulators angles would be set to target angle for that step, and the PID gains would be set to the default initialization. This hand-holding would be accompanied by a hefty additional reward reward of -1000 to avoid incentivizing crashes.

Another measure was implemented to help limit crashes, that being the limit of control effort/torque applied to each joint. Even with the prior crash failsafe, the unlimited nature of the torque meant that crashes would happen so often that no successful episodes would emerge to kickstart the optimization. Limiting the control efforts to a certain value would greatly decrease the amount of crashes. There was a degree of trial and error involved; Too low thresholds would limit the joint movement and hamper its ability to follow the target. Too high thresholds would render the limit redundant. The final value was set to +/- 70, based on the high of 50 recorded in the runs with the initial gains.

5.1.2 Model hyperparameters

In order to effectively make use of the actor critic networks, a look was taken into the output of the critic networks and how well it learned the values of the states it was in. By the nature of the simulation, the majority of the first thousand iterations were all crashes with low rewards, and the critic had to quickly update to reflect this. Turning up the learning rate from the default 0.00001 to 0.0001 seemed to help. Additionally, since we were working from episode to episode, and there was little meaning in factoring in the previous episode's reward in each evaluation, so the discount rate was set to 0 to combat the exploding critic network output.

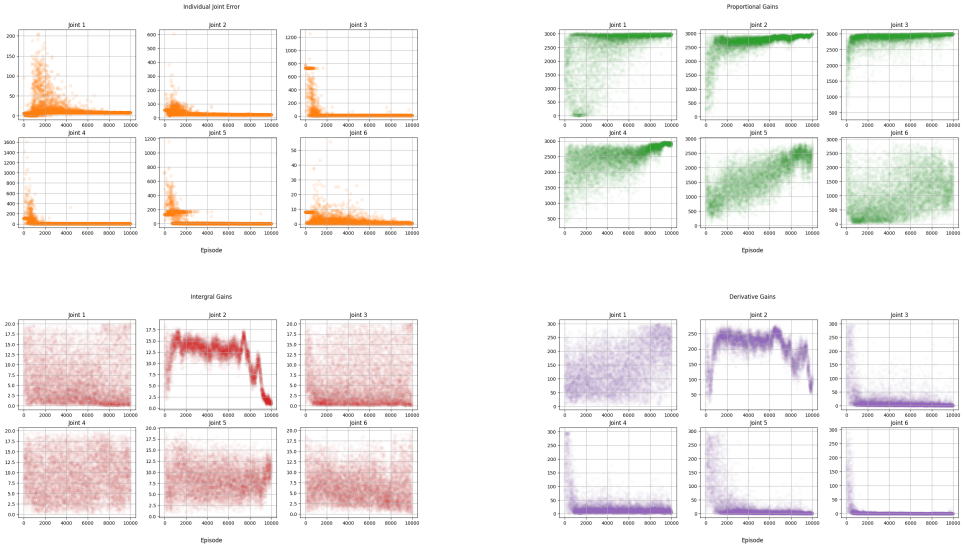
Another hyperparameter was the actor network's learning rate. This was simply increased from 0.000005 to 0.00001 until a divergence from a only crashed episodes resulted. Too low of a learning rate resulted in all episodes crashing for an indefinite time, while too high resulted in exploding gradients and saturated outputs.

5.2 A2C results

5.2.1 Training loss/reward

Model finally reached a total error/reward across all joints of -46. Note the cluster around -2000 reward; This is what most crashed/caught simulations ended up scoring.

5.2.2 Errors and gains



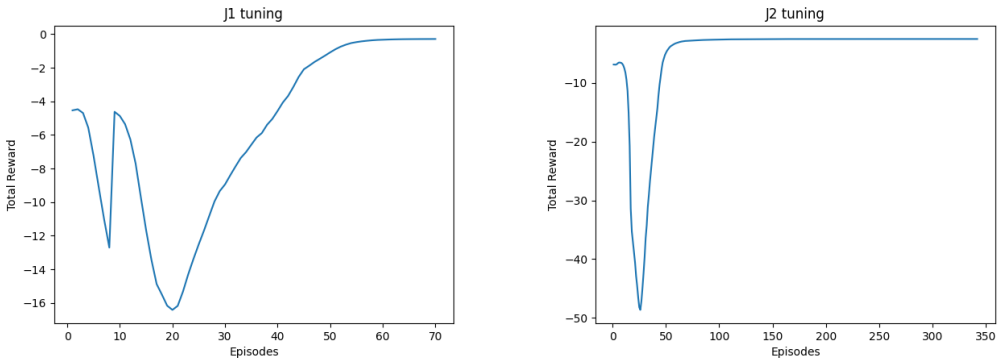
Plots for individual joint errors (top left) and optimal gains for each of proportion, integral, and derivative. Note the pattern for joint 2.

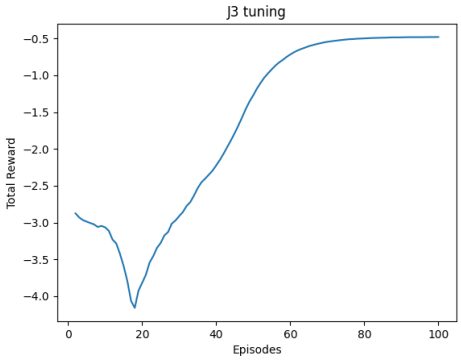
5.3 Experiments and results with TD3+DDPG

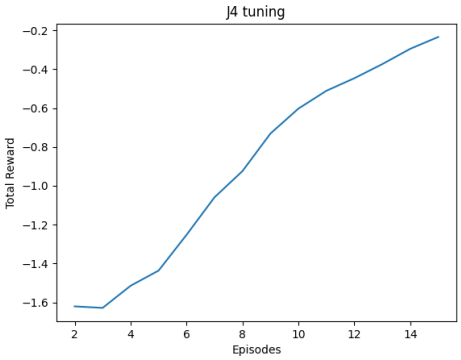
For this approach, several combinations of learning rates,discount factor were just tried, for each joint. Peculiar observation was, that Joint 3 and 5 were notoriously difficult to train. Finally, for each joint, a reward cutoff was also used, to exit the training if and when the reward is reached at the end of an episode. For each joint , a trend was observed where the model first reduces the P and D gains, and then slowly learns to increase them to an optimal value.Under certain conditions, the model also tries to reduce the cost, by purely increasing the Derivative gain, sometimes, it equalized Proportional and derivative gain, thus causing controller instability issues. Thus a penalty was added if $K_d > K_p/5$ to ensure that the robot doesnt overreacting to changing velocity targets. Following are the general hyperparameters used to train the agent:

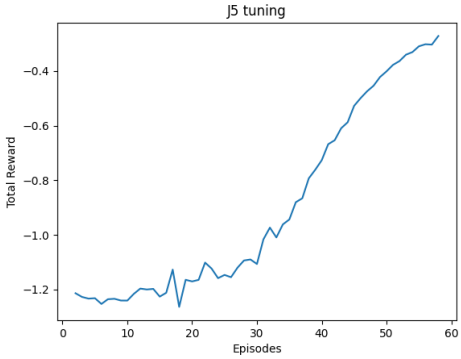
	J1	J2	J3	J4	J5
LQG cost α	1e-6	1e-6	1e-6	1e-6	1e-6
γ	0.75	0.75	0.99	0.75	0.99
Learning Rate	2e-4	2e-4	2e-4	1e-4	1e-4
Episodes	500	500	500	500	500
Mini Batch Size	100	100	100	100	100
Policy Delay	2	2	2	2	2
τ	0.005	0.005	0.005	0.005	0.005

These are the reward/training plots for each joint(J6 is not trained, as the gains are already optimal):









5.3.1 Results:

Learnt PID gains(Using A2C):

	J1	J2	J3	J4	J5
K_p	2933.78	2974.91	2985.38	2985.18	2761.19
K_i	4.57	0.11	17.43	18.34	2.42
K_d	298.07	19.78	3.08	1.65	23.51

Learnt PID gains(Using DDPG):

	J1	J2	J3	J4	J5
K_p	4859.581	24239.791	6586.265	560.602	199.86
K_i	1e-6	9.86e-7	0.00024	10	5.58961
K_d	8.67470	70.855	42.6280	11.472	28.5424

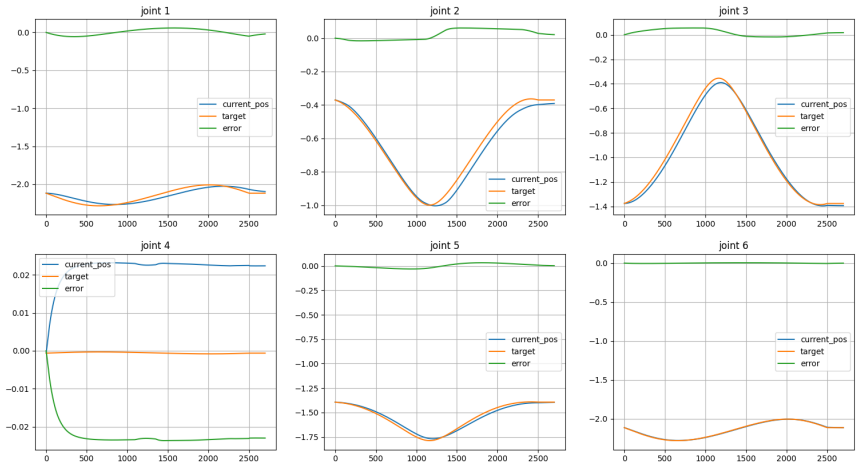


Figure 2: Performance Pre-Tuning

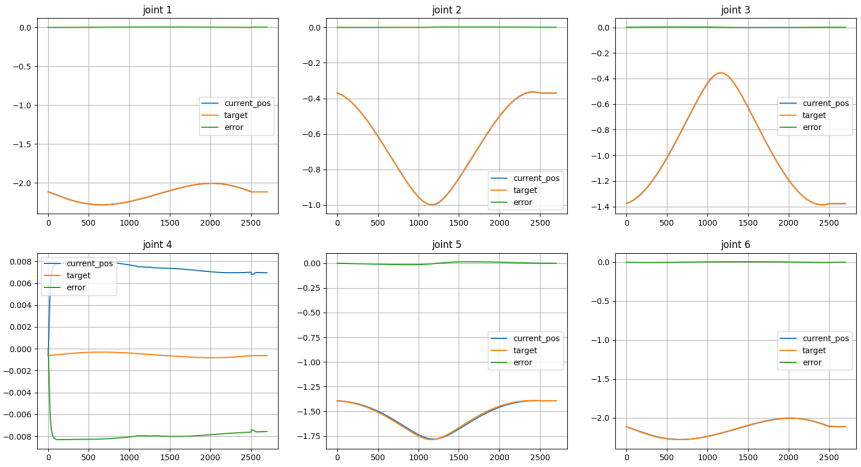


Figure 3: Performance Post Tuning

Plots for tracking error across all 6 joints pre-tuning and post-tuning. Blue represents the current position, orange is the target position, and green is the error. A tracking error for each joint, per tuning can be verified to be almost zero post tuning.

6 Conclusion and Discussion

With both A2C and the TD3 DDPG approach , we were able to achieve our objective of attaining optimal gains. The TD3 approach worked on finding the gains, while the A2C approach worked on finding update of the gains. Updating the gains , needs a heuristic , as seen in the ratio of P:I:D gain used while training the network, to ensure stability of the system. Whereas TD3 DDPG approach directly acts on the gains, and has a clear mapping between gains, action taken, and gradient descent weight update, thus not requiring a heuristic function which needs further tuning. The TD3 approach would be our go to in the future, for working on torque estimation and control as a clear mathematical

mapping , is needed in either of those tasks.Mujoco Visualization of the robot's motion , post training can be seen here : Link

7 Future work

The task of robotic manipulation, especially that of a robotic arm has immense potential. Given more time, our project could be expanded with additional functionality and applied to a variety of applications. We consider what developments could be made in the span of 2 weeks and 2 months.

7.1 In two weeks

In the shorter term of 2 Weeks we may not be able to implement new features or apply our work to another application, but we would definitely be able to make improvements in accuracy and efficiency. A major bottleneck in the process was the physics simulation itself, since inferences and training occurred after each full simulation. MuJoCo runs on a single thread and does not support GPU acceleration, so, in the 2 week, we can work on optimizing our implementations, and parallelizing the simulation runs for data collection and training.

7.2 In two months

In the next 2 months, we can implement and test the trained PID controller on a physical robot as we have only tested the PID controller in simulation so far. Implementing it on a physical robot will provide us with valuable insights into the practicality and applicability of our project and help us identify any discrepancies between the simulated and physical models and help us to close the sim-to-real gap.

8 Presentation Link

`https://docs.google.com/presentation/d/1dvyAYwHJx6FcRoqDoPU2gkwvl89-SQnp9xzMCFA_vWo/edit?usp=sharing`

9 References

References

[1] MathWorks. Tune PI Controller Using Reinforcement Learning. Available at: https://www.mathworks.com/help/reinforcement-learning/ug/tune-pi-controller-using-td3.html.

[2] Ayub I. Lakhani, Myisha A, et al. Stability-Preserving Automatic Tuning of PID Control with Reinforcement Learning. Available at: https://arxiv.org/abs/2112.15187

[3] Haider M. TD3 DDPG Agent for OpenGym Bipedal Walker. Available at: https://github.com/hmomin/TD3-Bipedal-Walker

[4] Veldhuizen, V., Autotuning PID control using Actor-Critic Deep Reinforcement Learning, 2020 `https://arxiv.org/ftp/arxiv/papers/2212/2212.00013.pdf`.