Deep Reinforcement Learning for Process Control

A B.Tech Project Report Submitted
in Partial Fulfilment of the Requirements
for the Degree of
Bachelor of Technology

By
Ritam Majumdar
(170107030)

to the
DEPARTMENT OF CHEMICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

This is to certify that the work contained in this thesis entitled “Deep Reinforcement Learning for Process Control” is a bonafide work of **Ritam Majumdar** (Roll No. **170107030**), carried out in the Department of Chemical Engineering, Indian Institute of Technology Guwahati under our supervision and that it has not been submitted elsewhere for a degree.

Supervisor:
Dr. Resmi Suresh,
Assistant Professor,
Department of Chemical Engineering,
Indian Institute of Technology Guwahati

Deep Reinforcement Learning for Process Control

Abstract:

In this Bachelor Thesis Project, we explore the applications of Deep Learning and Reinforcement Learning for Process Control Applications and make a comparative analysis with the existing PID and MPC Controllers on various industrial use cases. In this report, we briefly introduce PID, MPC and DRL Controllers, their governing equations, and their use cases in SISO and MIMO Control Systems.

Introduction:

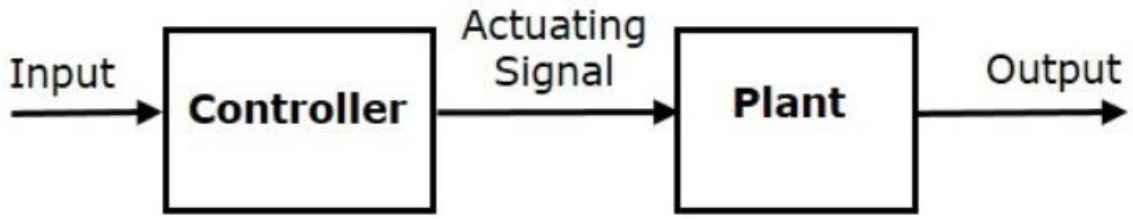
A Control system is a system, which provides the desired response by controlling the output. Classical controller design procedure involves careful analysis of the process dynamics, development of an abstract mathematical model, and finally, derivation of a control law that meets certain design criteria. Terminologies associated with Control Systems

Continuous time control systems: All the signals are continuous in time.

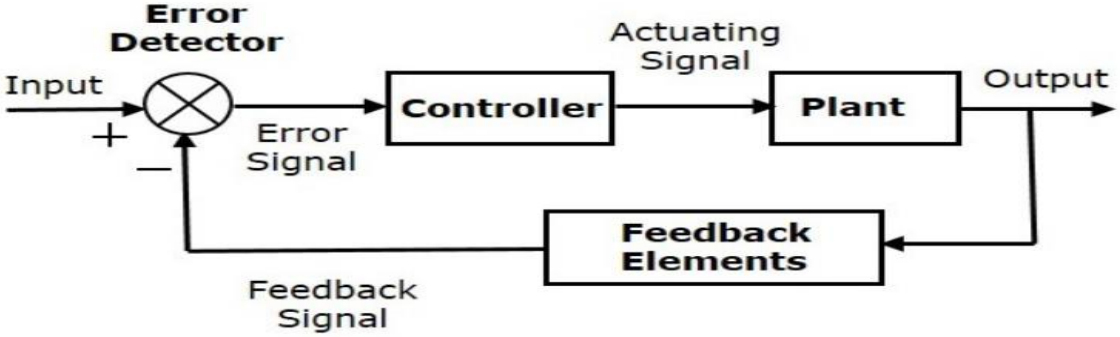
Discrete time control systems: There exists one or more discrete time signals.

SISO (Single Input and Single Output) Systems: They have one input and one output. **MIMO** (Multiple Inputs and Multiple Outputs): They have more than one input and more than one output.

Open loop control systems: Output is not fed-back to the input. So, the control action is independent of the desired output.



Closed loop control systems: Output is fed back to the input. So, the control action is dependent on the desired output.

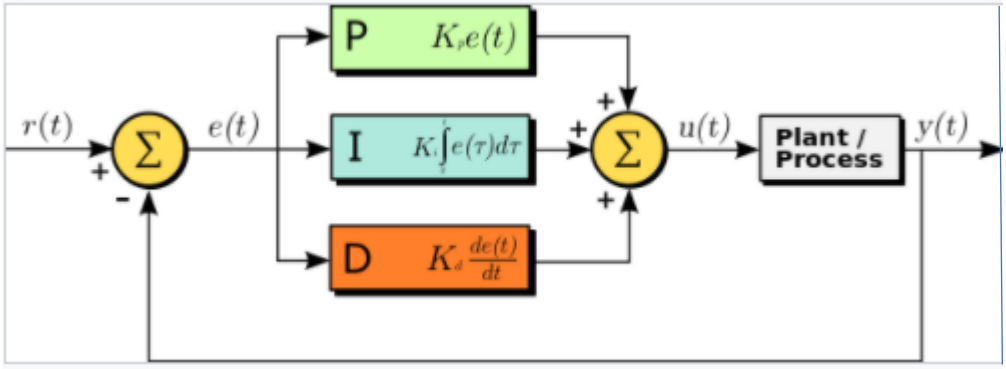


Review of PID and MPC Controller

PID Controller:

Known as Proportional-Integral-Derivative Controller, it is a control loop mechanism which employs feedback.

Block Diagram of PID Controller:



A PID Controller calculates an error value as the difference between the Setpoint (SP) and Process Variable (PV), and makes corrections based on Proportional (P), Integral (I) and Derivative (D) Terms.

Here, Setpoint (SP) is given by r(t), Process Variable (PV) is given by y(t), and Error term e(t) is given by r(t) – y(t). Inside the model:

Term **P**: It is proportional to the current value of the error e(t). For example, if the error is large and positive, the control output will be proportionately large and positive, taking into account the gain factor "K"

Term **I**: It accounts for past values of the error e(t) and integrates them over time to produce the **I** term.

Term **D:** It is a best estimate of the future trend of the error e(t), based on its current rate of change and is sometimes called Anticipatory Control. The more rapid the change, the greater the controlling or damping effect.

Mathematical form of the controller:

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(t')\,dt' + K_\mathrm{d} \frac{de(t)}{dt}$$

Where, K_p K_i K_d denote the coefficients for Proportional, Integral and Derivative Control.

Standard Form:

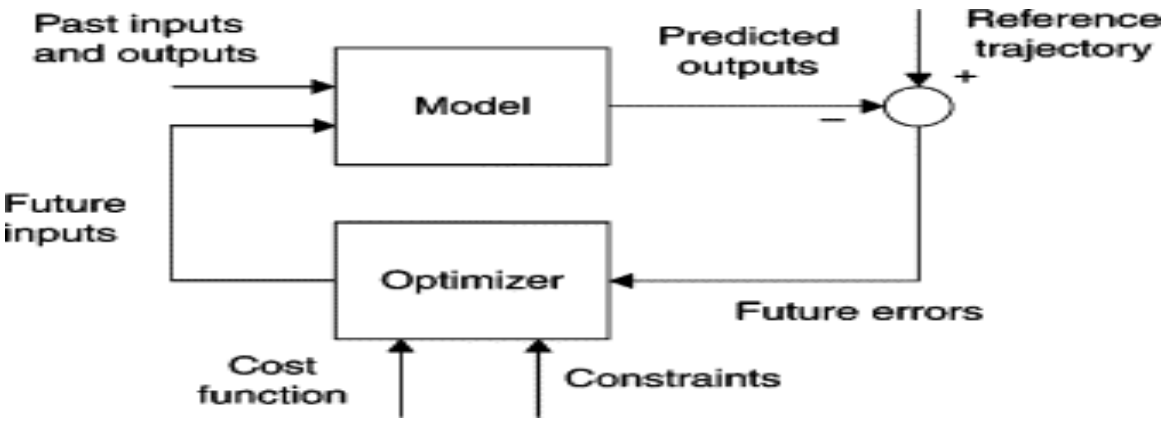
$$u(t) = K_\mathrm{p} \left(e(t) + \frac{1}{T_\mathrm{i}} \int_0^t e(t')\,dt' + T_\mathrm{d} \frac{de(t)}{dt} \right)$$

Here, Ki is replaced by $\frac{Kp}{Ti}$ and Kd is replaced by Kp*Td, where Ti and Td are called Integral Time and Derivative Time respectively.

MPC Controller:

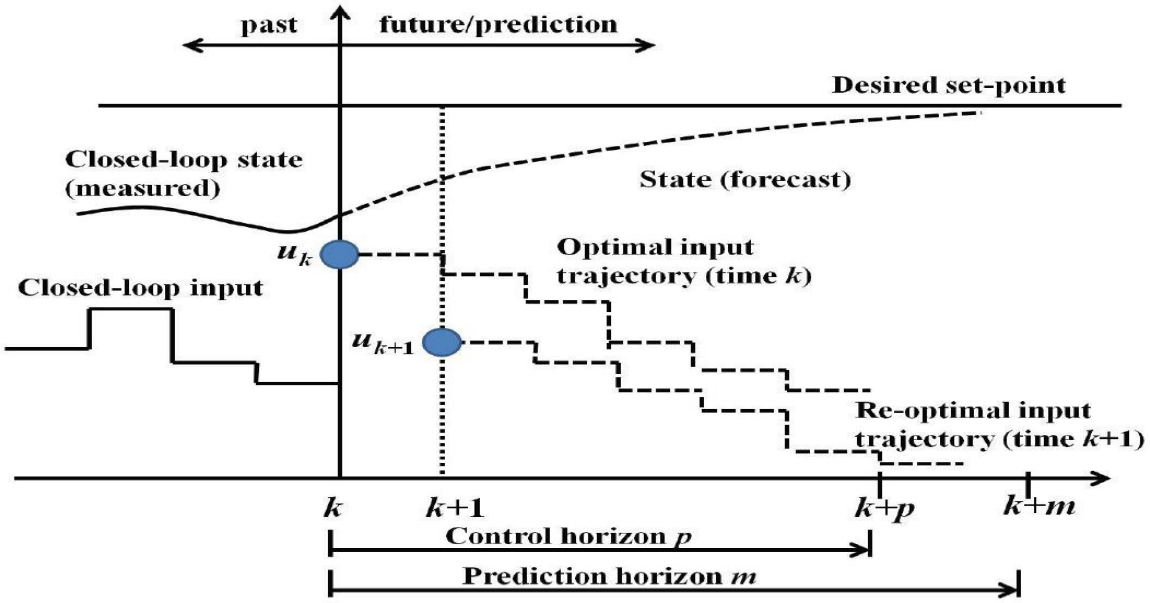
MPC, also known as Model Predictive Controller, is used to control a process by satisfying a set of constraints. MPC is based on iterative, finite-horizon optimization of a plant model.

At time t, the current state is sampled, and a cost minimizing control strategy is computed using an Optimization Algorithm for a relatively short time horizon in the future. This step explores the possible state trajectories that start from the current state and find a cost-minimizing control strategy until next time step.



However, we only implement the first step of the control strategy, then the plant state is sampled and the calculations are repeated starting from the new current state, yielding a new control and new predicted state path. The prediction horizon keeps being shifted forward and hence, MPC is also called **receding horizon control**.

Horizon Control Description:



Mathematical Details of Model Predictive Control:

Model Predictive Control (MPC) is a multivariable control algorithm that uses:

- an internal dynamic model of the process
- a cost function *J* over the receding horizon
- an optimization algorithm minimizing the cost function *J* using the control input *u.*

An example of a cost function is given by:

$$J = \sum_{i=1}^{N} w_{x_i} (r_i - x_i)^2 + \sum_{i=1}^{N} w_{u_i} \Delta u_i{}^2$$

Where,

X_i is the i^{th} control variable
R_i is the i^{th} reference variable
U_i is the i^{th} manipulated variable
$W(x_i)$ Weight coefficient of respective x_is
$W(u_i)$ Weight coefficient to penalize big changes in u_i.

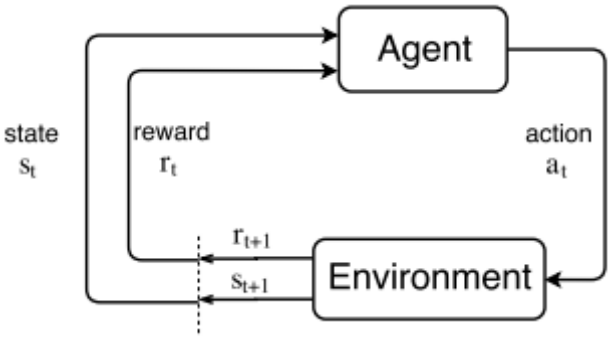
These Optimization algorithms can be solved by using mathematical programming techniques like Linear Programming or Non-Linear Programming, or by metaheuristic techniques like Genetic Algorithms, etc.

Reinforcement Learning

Background:

Reinforcement learning is a sequential decision making problem in which an agent takes an action at time step t and learns as it performs that action at each and every time step, with the goal of maximizing the cumulative reward. The decision maker is the agent and the place the agent interacts, or acts is the environment. The agent receives observations x_t (or states) from the environment and takes an action a_t. Once the action is performed, the environment then provides a reward signal r_t (scalar) that indicates how well the agent has performed.

Block Diagram of Reinforcement Learning:



Terminologies:

Agent: The person who will execute the action.

Environment: The world where Agents interact with the System

Rewards: R(s, a) defines the reward collected by taking the action a at state s

Policy: Policy defines the behaviour of an agent. An agent takes an action using policy, π, which is a distribution over action given states. Π: S -> P(A)

Episode: Playing out the whole sequence of state and action until reaching the terminate state or reaching a predefined length of actions.

Return: The return from a state is defined as the sum of discounted future reward.

$$R_t = r(s_1, a_1) + \gamma r(s_2, a_2) + \gamma^2 r(s_3, a_3) .. = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$$

Where γ ∈ [0, 1] is a discounting factor. If γ is smaller only rewards in immediate present matters and if γ is larger rewards in the distant future matters as well.

Value function: The value function $V_\pi(s)$ gives a long-term value of state s. The state value function is the expected return starting from state s,

$$V\pi(s) = E[Rt|St = s]$$

The objective is to learn a policy, π that maximizes the expected return from the start distribution, $J = E_{ri,si\sim E,ai\sim\pi}[Rt]$.

Action value function: In value function only states are considered into account In action value, we also consider the variance of Actions. The action value function is the expected return starting from state s, taking action a, and then following policy π is given by, $Q\pi(s, a) = E[R_t|S_t = s, At = a]$.

Bellman Equation:
Bellman Equation is used to solve the objective function recursively, and determine the value of V and Q.

$$\begin{aligned} V(s) &= \mathbb{E}[R_t|S_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3}..|S_t = s \\ &= \mathbb{E}[r_{t+1} + \gamma R_{t+1}|S_t = s] \\ &= \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})|S_t = s] \end{aligned}$$

The Bellman equation for action value function is given as,

$$Q_\pi(s_t,a_t) = E_{ri,st+1\sim E}[r(s_t,a_t)+\gamma E_{at+1\sim\pi}[Q_\pi(s_{t+1}, a_{t+1})]]$$

If the target policy we are considering is deterministic (Discrete Action Space) , we can describe it as a function μ : S → A and the above equation is simplified to:

$$Q_\mu(s_t,a_t) = E_{ri,st+1\sim E}[r(s_t,a_t)+\gamma E_{at+1\sim\pi}[Q_\mu\ (s_{t+1}, a_{t+1})]]$$

Here, μ is the Behavioural policy

Q-learning:

In general in reinforcement learning, two steps are involved to learn an agent,
• Policy evaluation i.e., Estimating the value function
• Policy improvement.
Q-learning uses the greedy policy $\mu(s) = arg\ max_a Q(s,a)$ for policy improvement.

Q-learning Update Equation:

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{temporal difference}} \underbrace{- Q(s_t,a_t)}_{\text{old value}}\Big)$$
$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

However, when the number of states and action pairs are high, the action value of those many states cannot be stored individually, hence function approximators parameterized by θ^Q, are used to learn action value function.

Θ^Q is obtained by minimizing the loss function:

$$E_{st\sim\rho\beta,at\sim\beta,rt\sim E}[(Q(s_t,a_t|\theta^Q)-y_t)^2],$$

where $y_t = r(s_t,a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta^Q))$.

Actor- Critic Approach:

Drawbacks with Q-Learning:

1. The process control application we are considering in this work has continuous states and actions.
2. Q-learning cannot be directly applied to continuous action spaces, because the greedy policy in Q-learning requires computing max over actions.
3. Hence finding the greedy policy requires an optimization of at at every timestep; this optimization step is slow with unconstrained, large function approximators with large number of action variables.
4. Hence, actor-critic approach is used to tackle this challenge.

In actor-critic, the critic learns the action value function and the actor learns the policy. The actor-critic framework maintains two set of parameters:

• Critic Updates action-value function parameters Wc
• Actor Updates policy parameters Wa, in the direction suggested by critic

In deterministic policy gradient, the policy is represented by a function approximator with weights Wa given by $a = \pi(s, Wa)$.

The objective function is defined as total discounted reward as follows:
$J(W_a) = E[r_1 + \gamma r_2 + \gamma^2 r_3 + ..]$

Critic estimates the value of current policy by Q-learning. The parameter Wc of the critic is updated using the gradient derived from Q-learning as follows:

$$\frac{\partial L(Wc)}{\partial Wc} = E[(r+\gamma Q(s^0,\pi(s^0),W_c)-Q(s,a,W_c))\frac{\partial Q(s,a,Wc)}{\partial Wc}]$$

The actor updates policy in direction that improves Q, hence the gradient with which the actor is updated is given as,

$$\frac{\partial J(Wa)}{\partial Wa} = E\left[\frac{\partial Q(s,a)}{\partial Wa}\right] = E\left[\frac{\partial Q(s,a,Wc)}{\partial a}\frac{\partial \pi(s,Wa)}{\partial Wa}\right]$$

The goal of learning is to adjust parameters Wa and Wc to achieve more reward. To approximate the Actor and Critic functions, we use Actor and Critic Neural Network a

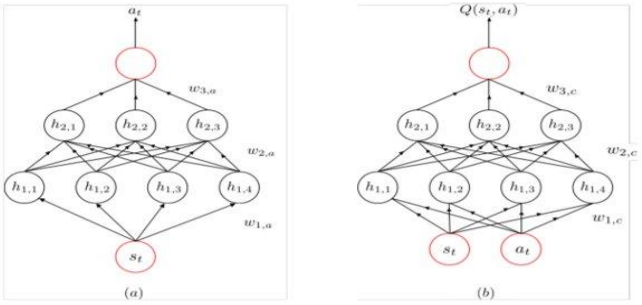


Figure 3: A deep neural network representation of (a) the actor, and (b) the critic. The red circles represent the input and output layers and the black circles represent the hidden layers of the network.

After introducing the terminologies in Reinforcement Learning, we are going to connect them with Process Control. Now, we will pose all the terminologies as a Process Control Problem.

States: A state s consists of features describing the current state of the plant. Since the controller needs both the current output of the plant and the required setpoint to take necessary action to get closer to the setpoint, the state must contain information about current plant output and the required setpoint. For instance it can be a combination of current output and deviation from setpoint < y, (y − ysetpoint) > or just current output and setpoint < y, ysetpoint >.

Actions: The action, a is the means through which RL agent interacts with environment. The controller input to the plant is the action in process control.

Reward: The reward signal r is a scalar feedback signal that indicates how well an RL agent is doing at step t. For process control task, the objective is to take the output closer to the set point, while meeting certain constraints. This specific objective can be fed to RL agent (controller) by means of a reward function. Thus, the reward function serves as a function similar to an objective function formulation in Model Predictive Control.

$$r(s,a,s') = \begin{cases} c, & \text{if } |y^i - y^i_{set}| \leq \varepsilon, \forall i \\ -\sum_i^n |y^i - y^i_{set}|, & \text{otherwise} \end{cases}$$

Where i represents ith output of n outputs in MIMO systems and c > 0, is a constant. A high value of c leads to larger value of r when the ouputs are closer to setpoint by ε and results results in quicker tracking of setpoint.

Policy and Value Function Representation:

The policy π is represented by an actor using deep feed forward neural network parameterized by weights Wa. Thus, an actor is represented as π(s, Wa). . The action value function is represented by a critic using another deep neural network parameterized by weights Wc. Thus, critic is represented as Q(s, a, Wc). The critic network predicts the Q-values for each states and action and actor network proposes an action for the given state. The goal is to learn the actor and critic neural network parameters by interacting with the plant.

Deep Reinforcement Controller:

As outlined in Algorithm 5, first we randomly initialize the parameters of the actor and critic networks (Step 2).

We then create a copy of the network parameters and initialize the parameters of the target networks (Step 3).

The final initialization step is to supply the RM with a large enough collection of tuples (s, a, s0 , r) on which to begin training the RL controller (Step 4).

Each episode is preceded by two steps.

First, we ensure the state definition st in (25) is defined for the first dy steps by initializing a sequence of actions ha−1, . . . , a−ni along with the corresponding sequence of outputs hy0, . . . , y−n+1i where n ∈ N is sufficiently large.

A set-point is then fixed for the ensuing episode (Steps 6-7). For a given user-defined setpoint, the actor, μ(s, Wa) is queried, and a control action is implemented on the process (Steps 8–10).

Implementing μ(s, Wa) on the process generates a new output yt+1.

The controller then receives a reward r, which is the last piece of information needed to store the updated tuple (s, a, r, s0) in the RM (Steps 9–12).

M uniformly sampled tuples are generated from the RM to update the actor and critic networks using batch gradient methods (Steps 14– 23).

Finally, for a fixed τ , we update the target actor and target critic networks in Steps 24–25. The above steps are then repeated until the end of the episode.

Block Diagram:

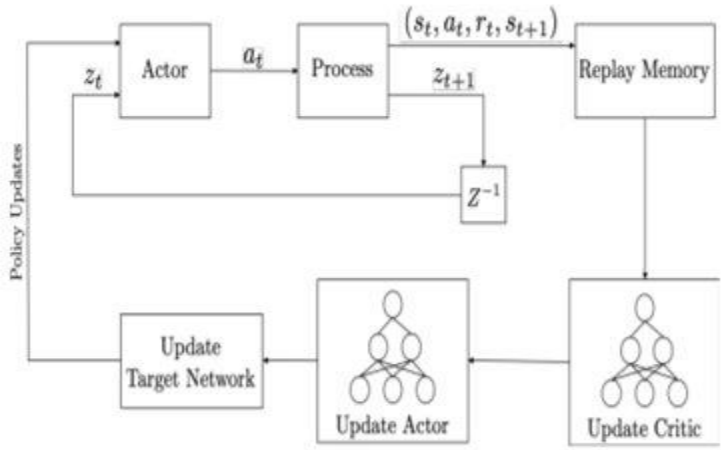


Figure 4: A schematic of the proposed DRL controller architecture for set-point tracking problems.

Algorithm 5 Deep Reinforcement Learning Controller

Output: Optimal policy $\mu(s, W_a)$
Initialize: W_a, W_c to random initial weights
Initialize: $W'_a \leftarrow W_a$ and $W'_c \leftarrow W_c$
Initialize: Replay memory with random policies
for each episode **do**
  Initialize an output history $\langle y_0, \ldots, y_{-n+1} \rangle$ from an action history $\langle a_{-1}, \ldots, a_{-n} \rangle$
  Set $y_{\text{sp}} \leftarrow$ set-point from the user
  for each step t of episode $0, 1, \ldots T-1$ **do**
    Set $s \leftarrow \langle y_t, \ldots, y_{t-d_y}, a_{t-1}, \ldots, a_{t-d_a}, (y_t - y_{\text{sp}}) \rangle$
    Set $a_t \leftarrow \mu(s, W_a) + \mathcal{N}$
    Take action a_t, observe y_{t+1} and r
    Set $s' \leftarrow \langle y_{t+1}, \ldots, y_{t+1-d_y}, a_t, \ldots, a_{t+1-d_a}, (y_{t+1} - y_{\text{sp}}) \rangle$
    Store tuple (s, a_t, s', r) in RM
    Uniformly sample M tuples from RM
    for $i = 1$ to M **do**
      Set $\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}(s'^{(i)}, \mu(s'^{(i)}, W'_a), W'_c)$
    end for
    Set $W_c \leftarrow W_c + \frac{\alpha_c}{M} \sum_{i=1}^{M} (\tilde{y}^{(i)} - Q^{\mu}(s^{(i)}, a^{(i)}, W_c)) \nabla_{W_c} Q^{\mu}(s^{(i)}, a^{(i)}, W_c)$
    for $i = 1$ to M **do**
      Calculate $\nabla_a Q^{\mu}(s^{(i)}, a, W_c)|_{a=a^{(i)}}$
      Clip $\nabla_a Q^{\mu}(s^{(i)}, a, W_c)|_{a=a^{(i)}}$ using (33)
    end for
    Set $W_a \leftarrow W_a + \frac{\alpha_a}{M} \sum_{i=1}^{M} \nabla_{W_a} \mu(s^{(i)}, W_a) \nabla_a Q^{\mu}(s^{(i)}, a, W_c)|_{a=a^{(i)}}$
    Set $W'_a \leftarrow \tau W_a + (1-\tau) W'_a$
    Set $W'_c \leftarrow \tau W_c + (1-\tau) W'_c$
  end for

Algorithm 4 Deterministic Off-policy Actor-Critic Method

Output: Optimal policy $\mu_\theta(s)$
Initialize: Arbitrarily set policy parameters θ and Q-function weights w
for true **do**
  initialize s, the first state of the episode
  for s is not terminal **do**
    $a \sim \beta(\cdot|s)$
    take action a, observe s' and r
    $\tilde{y} \leftarrow r + \gamma Q^{\mu}(s', \mu_\theta(s'), w)$
    $w \leftarrow w + \alpha_w (\tilde{y} - Q^{\mu}(s, \mu_\theta(s), w)) \nabla_w Q^{\mu}(s, a, w)$
    $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu}(s, a, w)|_{a=\mu_\theta(s)}$
    $s \leftarrow s'$
  end for
end for

8th Semester BTP Summary:

- Got DRL controller to function, and was able to track setpoints efficiently.
- Got a comparative performance between PID, MPC, DRL.
- DRL was based on DDPG Algorithm, which was Noisy in nature.
- Explored TD3 and Soft Actor Critic Architecture, which served as replacement to the DDPG Framework.
- Was able to generate results using TD3, which performed better than DRL Algorithm.
- Starting with a detailed explanation of DDPG and TD3 algorithms, this report goes on to explain the core heuristics, followed by the results and the corresponding plots associated with the algorithms.

Deep Deterministic Policy Gradient (DDPG):

Deep Deterministic Policy Gradient (DDPG) is a reinforcement learning technique that combines both Q-learning and Policy gradients. DDPG being an **actor-critic technique** consists of two models: Actor and Critic. The actor is a policy network that takes the state as input and outputs the exact action (continuous), instead of a probability distribution over actions. The critic is a Q-value network that takes in state and action as input and outputs the Q-value. DDPG is used in the continuous action setting and the “deterministic” in DDPG refers to the fact that the actor computes the action directly instead of a probability distribution over actions.

Theory:

In DQN the optimal action is taken by taking argmax over the Q-values of all actions. In DDPG the actor is a policy network that does exactly this. It outputs the action directly (action can be continuous), bypassing the argmax.

Pseudo Code for the algorithm:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

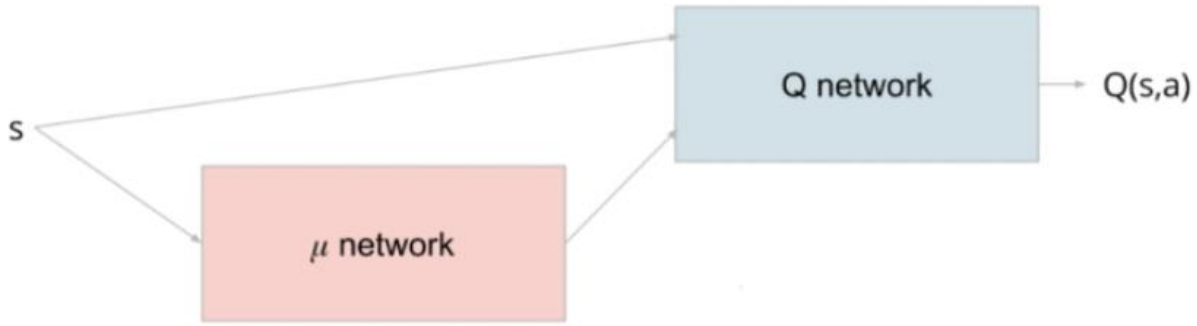
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Model Architecture:

The policy is deterministic since it directly outputs the action. In order to promote exploration some Gaussian noise is added to the action determined by the policy. To calculate the Q-value of a state, the actor output is fed into the Q-network to calculate the Q-value. To stabilize learning we create target networks for both critic and actor.

These target networks will have “soft”-updates based on main networks. We will discuss these updates later.



Replay Buffer:

As used in Deep Q learning (and many other RL algorithms), DDPG also uses a replay buffer to sample experience to update neural network parameters. During each trajectory roll-out, we save all the experience tuples (state, action, reward, next_state) and store them in a finite-sized cache — a “replay buffer.” Then, we sample random mini-batches of experience from the replay buffer when we update the value and policy networks. The reason why we use experience replay is because, in optimization asks, we want the data to be independently distributed. This fails to be the case when we optimize a sequential decision process in an on-policy way, because the data then would not be independent of each other. When we store them in a replay buffer and take random batches for training, we overcome this issue.

Actor (Policy) & Critic (Value) Network Updates:

The value network is updated similarly as is done in Q-learning. The updated Q value is obtained by the Bellman equation:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

However, in DDPG, the **next-state Q values are calculated with the target value network and target policy network**. Then, we minimize the mean-squared loss between the updated Q value and the original Q value:

$$Loss = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

For the policy function, our objective is to maximize the expected return:

$$J(\theta) = \mathbb{E}[Q(s, a)|_{s=s_t, a_t=\mu(s_t)}]$$

To calculate the policy loss, we take the derivative of the objective function with respect to the policy parameter. Keep in mind that the actor (policy) function is differentiable, so we have to apply the chain rule.

$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_a Q(s, a) \nabla_{\theta^\mu} \mu(s|\theta^\mu)$$

But since we are updating the policy in an off-policy way with batches of experience, we take the mean of the sum of gradients calculated from the mini-batch:

$$\nabla_{\theta^\mu} J(\theta) \approx \frac{1}{N} \sum_i [\nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_i}]$$

Target Network Updates

We make a copy of the target network parameters and have them slowly track those of the learned networks via “soft updates,” as illustrated below:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$$

Exploration

In Reinforcement learning for discrete action spaces, exploration is done via probabilistically selecting a random action (such as epsilon-greedy or Boltzmann exploration). For continuous action spaces, exploration is done via adding noise to the action itself:

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$$

Drawbacks of DDPG Algorithm:

1. DDPG can be unstable and heavily reliant on finding the correct hyper parameters for the current task.
2. This is caused by the algorithm continuously over estimating the Q values of the critic (value) network.

3. These estimation errors build up over time and can lead to the agent falling into a local optima or experience catastrophic forgetting.

Twin Deep Deterministic Delay (TD3) Algorithm:

TD3 is the successor to the Deep Deterministic Policy Gradient (DDPG). TD3 addresses this issue faced in DDPG Algorithm by focusing on reducing the overestimation bias seen in previous algorithms. This is done with the addition of 3 key features:

1. Using a pair of critic networks.
2. Delayed updates of the actor.
3. Action noise regularisation.

Twin Critic Networks:

The first feature added to TD3 is the use of two critic networks. This was inspired by the technique seen in Deep Reinforcement Learning with Double Q-learning, which involved estimating the current Q value using a separate target value function, thus reducing the bias.

However, the technique doesn't work perfectly for actor critic methods. This is because the policy and target networks are updated so slowly that they look very similar, which brings bias back into the picture. Instead, an older implementation seen in Double Q Learning is used. TD3 uses clipped double Q learning where it takes the smallest value of the two critic networks.

This method favours underestimation of Q values. This underestimation bias isn't a problem as the low values will not be propagated through the algorithm, unlike overestimate values. This provides a more stable approximation, thus improving the stability of the entire algorithm.

Delayed Updates

Target networks are a great tool for introducing stability to an agents training, however in the case of actor critic methods there are some issues to this technique. This is caused by the interaction between the policy (actor) and critic (value) networks. The training of the agent diverges when a poor policy is overestimated. Our agent's policy will then continue to get worse as it is updating on states with a lot of error.

In order to fix this, we need to carry out updates of the policy network less frequently than the value network. This allows the value network to become more stable and reduce errors before it is used to update the policy network. In practice, the policy network is updated after a fixed period of time steps, while the value network continues to update after each time step. These less frequent policy updates will have value estimate with lower variance and therefore should result in a better policy.

Noise Regularisation

The final portion of TD3 looks at smoothing the target policy. Deterministic policy methods have a tendency to produce target values with high variance when updating the critic. This is caused by overfitting to spikes in the value estimate.

In order to reduce this variance, TD3 uses a regularisation technique known as target policy smoothing. Ideally there would be no variance between target values, with similar actions receiving similar values. TD3 reduces this variance by adding a small amount of random noise to the target and averaging over mini batches. The range of noise is clipped in order to keep the target value close to the original action. By adding this additional noise to the value estimate, policies tend to be more stable as the target value is returning a higher value for actions that are more robust to noise and interference.

To Summarize:

1. TD3 uses two separate critic networks, using the smallest value of the two when forming its targets.
2. TD3 uses a delayed update of the actor network, only updating it every 2 timesteps instead of after each time step, resulting in more stable and efficient training.
3. Clipped noise is added to the selected action when calculating the targets. This prefers higher values for actions that are more robust.

Pseudo-Code of the Algorithm:

Algorithm 1 TD3

1 Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$

2 Initialize replay buffer $\mathcal{B}$

for $t = 1$ **to** T **do**

3 Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'

4 Store transition tuple (s, a, r, s') in $\mathcal{B}$

5 Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
$\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
$y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$

6 **if** t mod d **then**
Update ϕ by the deterministic policy gradient:
$\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$

7 Update target networks:
$\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
$\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$

end if

end for

Algorithm Steps: (Except for the update steps as mentioned above, the remaining aspects are similar to DDPG).

1. Initialise networks
2. Initialise replay buffer
3. Select and carry out action with exploration noise
4. Store transitions
5. Update critic, actor
6. Update target networks
7. Repeat until sentient.

Results and Experiments:

For this section, we are sticking to SISO Systems only, of 1st and 2nd order. For the 1st order, the Gain is 3 and the Time Constant is 5. For the 2nd order system, we consider 3 cases, each of them with Gain as 2 and Time constant as 2. We have 3 damping factors, with 2 being for Overdamped scenario, 1 being for critically damped scenario, and 0.5 being for underdamped scenario.

We set the metric as sum of the absolute difference between the setpoint and the tracked output at every timestep, for DRL and TD3, we consider the best episode with Maximum reward. For each scenario, we have 300 timesteps, with 11 different scenarios of setpoint, 5 of them being step changes, 5 being just 1 value to track, whereas other is multiple setpoint changes.

		1st Order		
Setpoints	PID	MPC	DRL	TD3
1	36.61	6.85	51.55	78.88
5	183.08	34.83	143.06	98.69
10	366.16	58.1752	122.49	129.07
50	1830.8	290.01	424.39	384.68
100	3602.69	732.48	699.82	711.679
0-1	28.5	5.63	25.69	66.01
0-5	142.543	24.26	75.56	76.86
0-10	285.08	54.68	125.1	90.96
0-50	1425.432	244.624	201.477	167.605
0-100	2850.86	514.37	450.96	270.86
370105	621.24	150.282	325.169	290.15

Setpoints	2nd Order-OD MPC	DRL	TD3	2nd Order-CD MPC	DRL	TD3	2nd Order-UD MPC	DRL	TD3
1	6.19	41.44	46.78	6.19	28.052	50.631	6.19	108.65	50.64
5	30.9	63.81	72.08	30.9	81.49	63.05	30.9	62.07	62.44
10	61.811	81.99	96.23	61.811	68.51	79.125	61.811	90.48	72.12
50	309.02	394.88	317.6	309.02	240.4	221.27	309.02	193.32	205.9
100	618.02	8562.71	588.53	618.02	11355.5	393.59	618.02	2257.84	367.65
0-1	5.189	41.94	45.21	5.189	29.398	54.693	5.189	39.7	49.58
0-5	25.9	51.25	56.91	25.9	65.61	54.3	25.9	73.44	55.232
0-10	51.82	62.34	65.71	51.82	106.32	70.015	51.82	48.06	64.22
0-50	259.02	173.01	146.27	259.02	202.05	158.45	259.02	144.27	216.48
0-100	518.02	360.14	251.06	518.04	7315.82	258.29	518.04	8940.82	284.299
370105	153.54	167.14	171.15	153.54	172.57	188.51	153.54	274.302	290.15

Discussions:

For both 1st and 2nd order systems, for most lower setpoint scenarios, MPC has lower error as compared to PID, DRL and TD3, but it comes with a cost of high training time, approximately 2.5 times more as compared to DRL and TD3. We are using the Prediction and Control Horizon values to be 25 and 5 respectively, which was tested to be most efficient after multiple experiments.

For 2nd order experiments, MPC seems to perform significantly better than DRL and TD3, in each of the Overdamped, Critically Damped and Underdamped scenarios, some of which can be identified by better manual model selection, although comes with a cost of time taken.

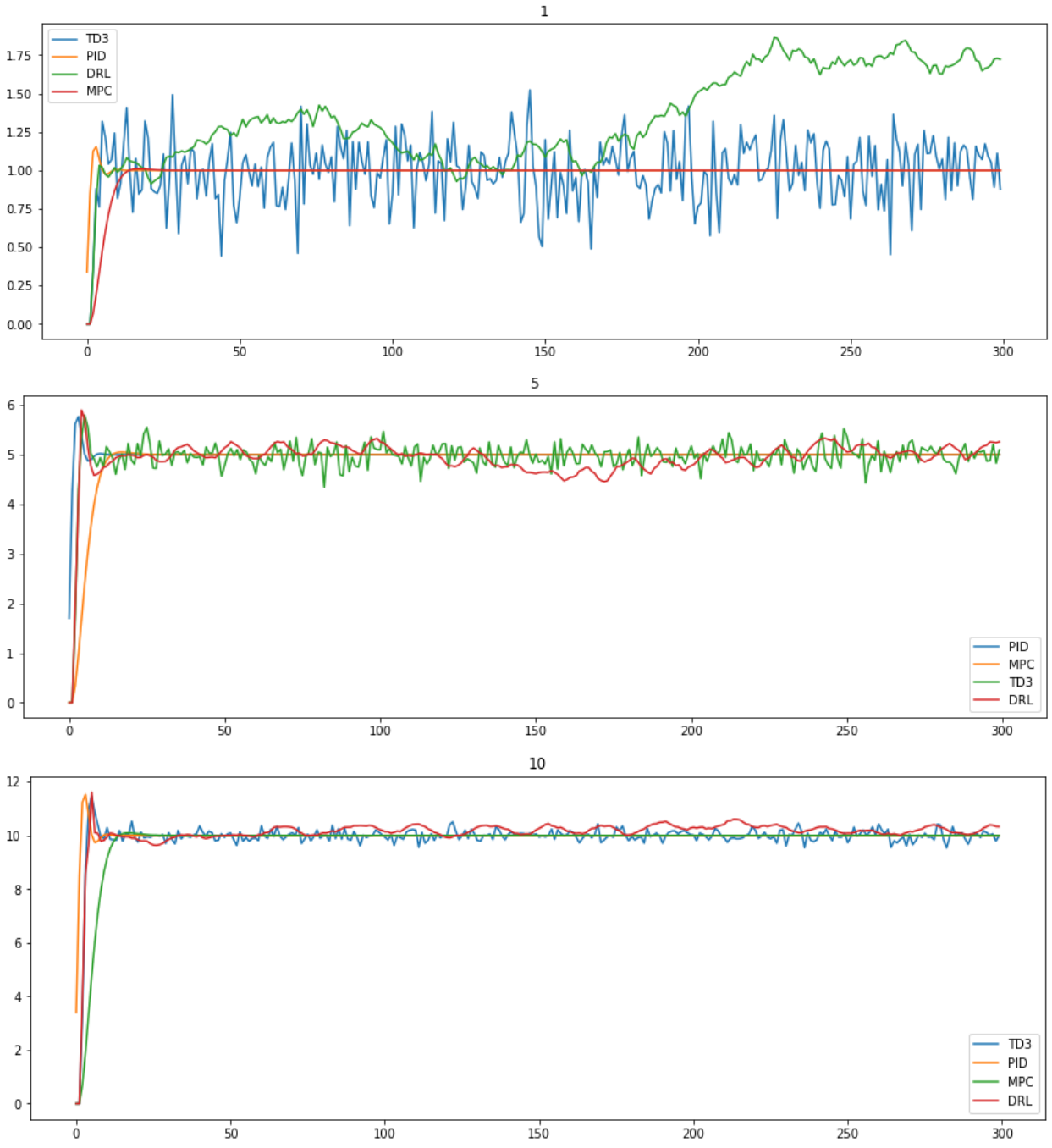
DRL and TD3 were excellent in being able to track setpoints and are able to gradually learn with increasing number of episodes (we fixed it to 100), and can perform better with higher number of episodes. DRL has a drawback of being noisy in nature, and struggles to converge even after hovering around the setpoint.

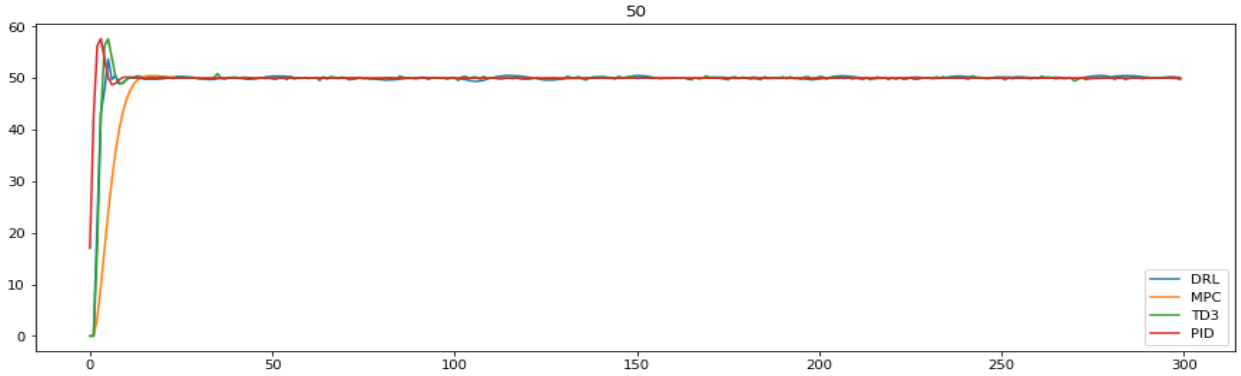
TD3 was aimed to address this issue, as it showed comparatively less noise as compared to DRL, and was better able to converge because of reduction of overestimation Bias by using 2 Critics, and taking the minimum of the 2 Critic values.

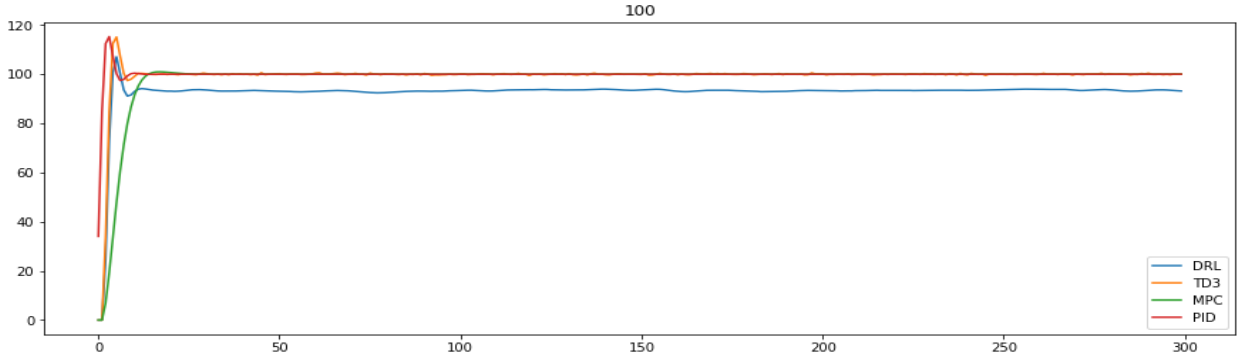
In 85% of the experiments, TD3 showed better performance than DRL, across both 1st and 2nd order processes, marking it superior for our use case scenarios.

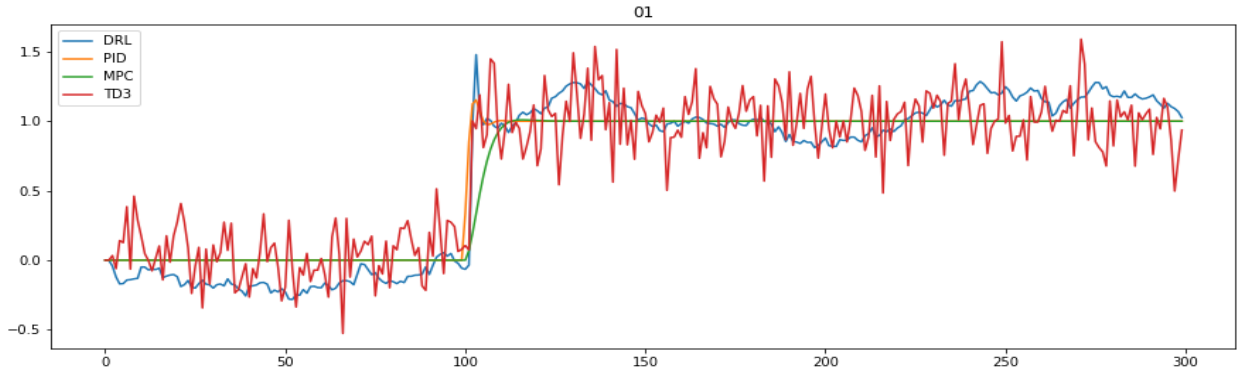
Plots

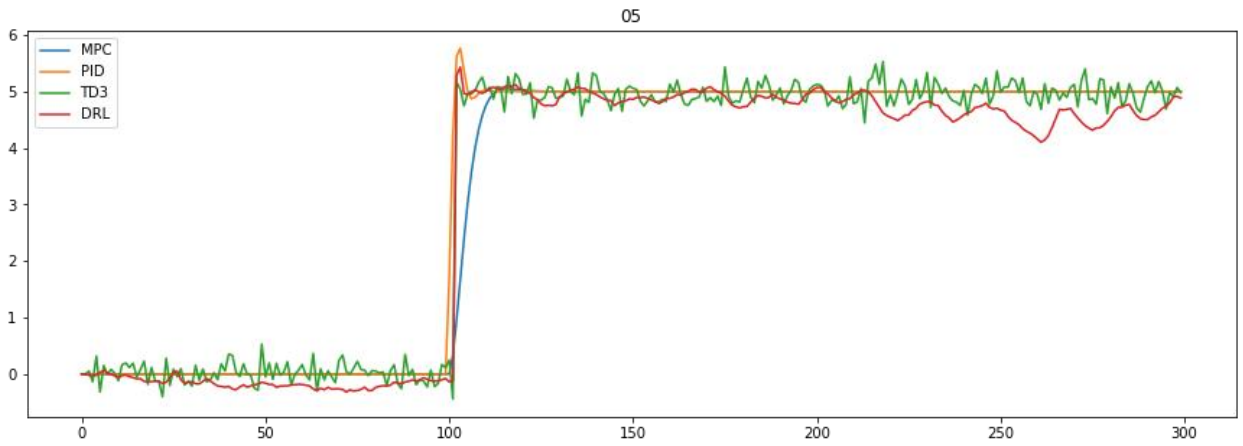
1st Order

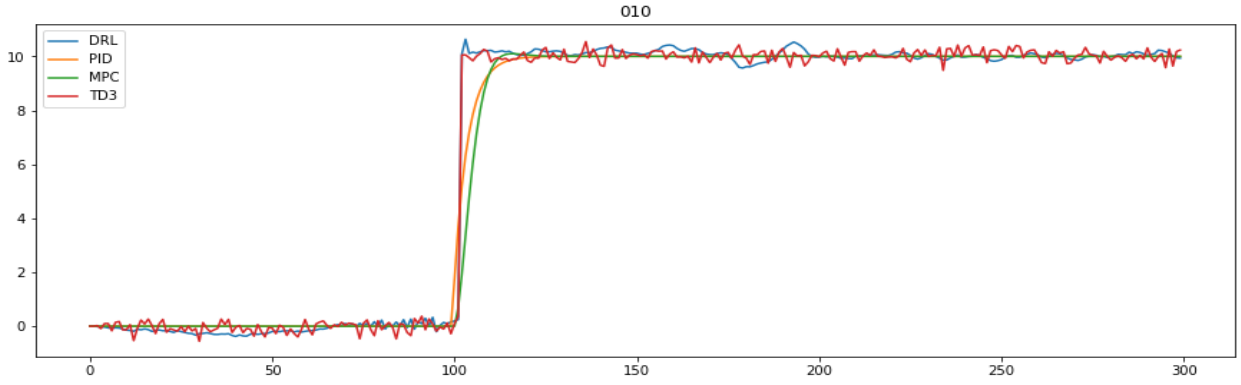


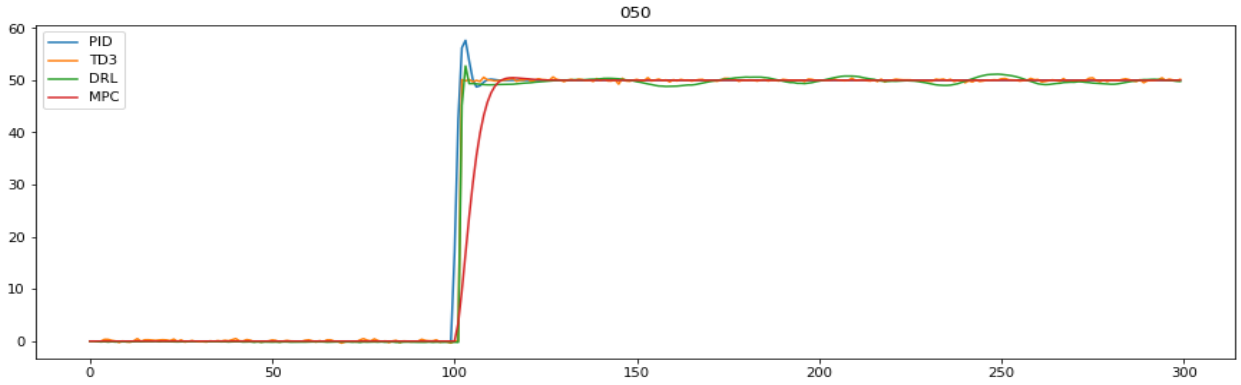

50
DRL
MPC
TD3
PID

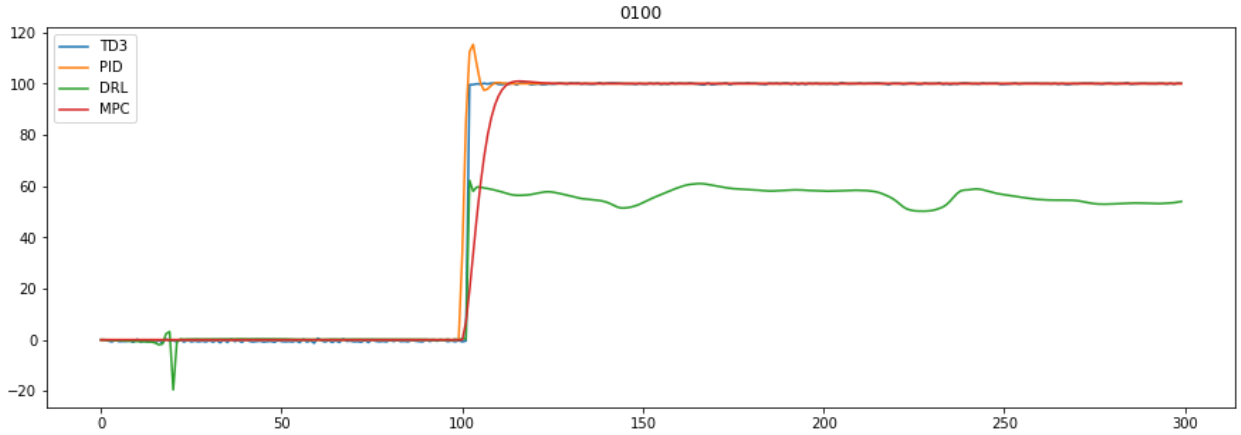

100
DRL
TD3
MPC
PID

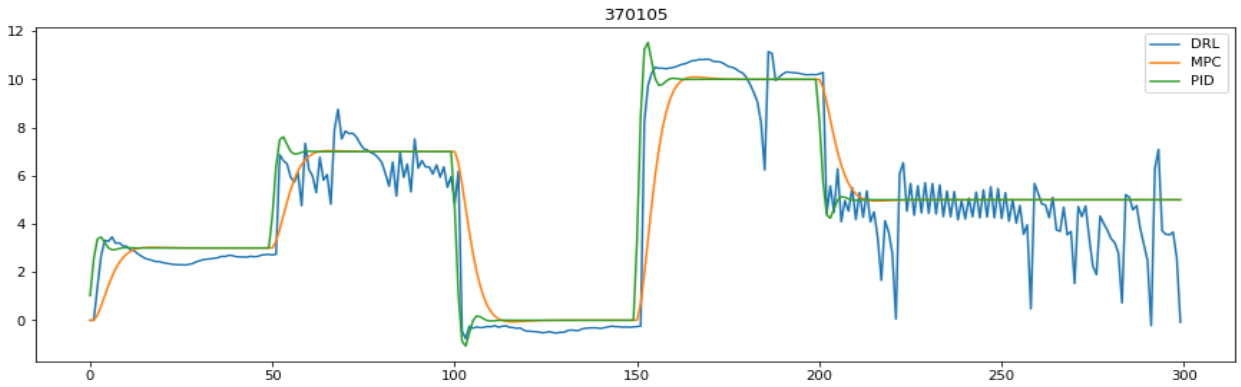

01
DRL
PID
MPC
TD3


05
MPC
PID
TD3
DRL

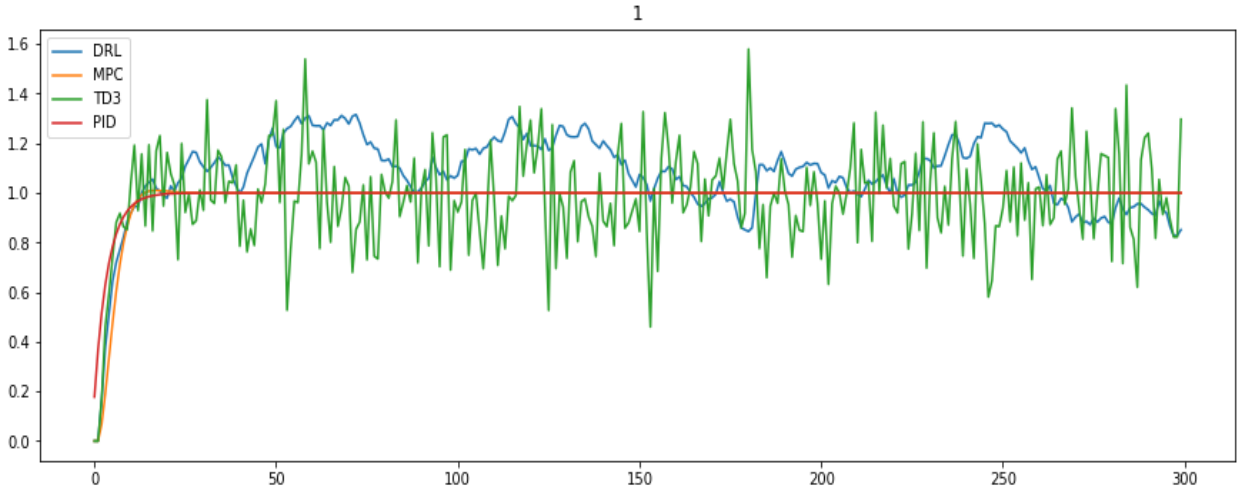

010
DRL
PID
MPC
TD3

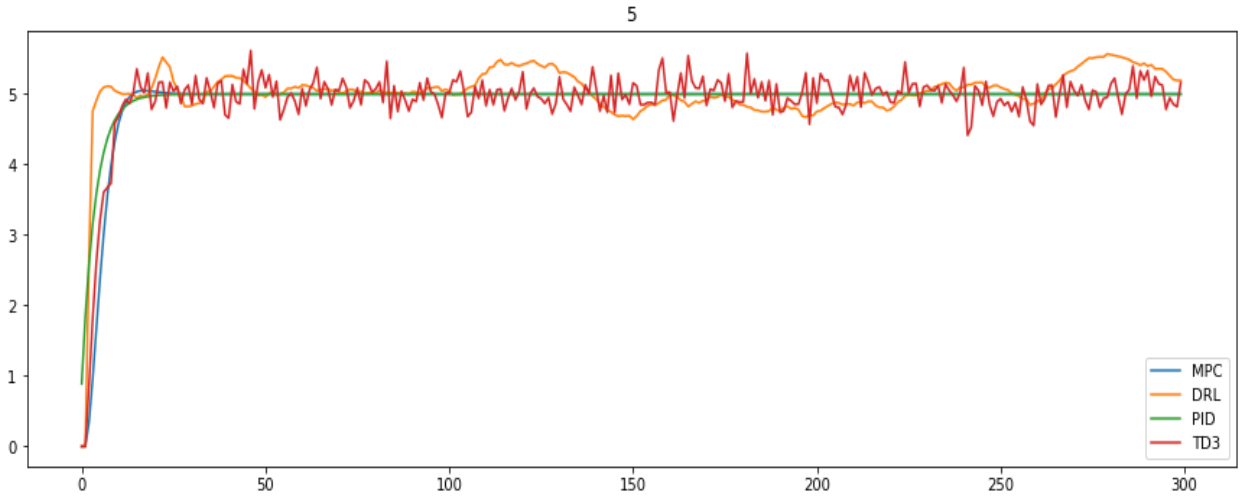

050
PID
TD3
DRL
MPC

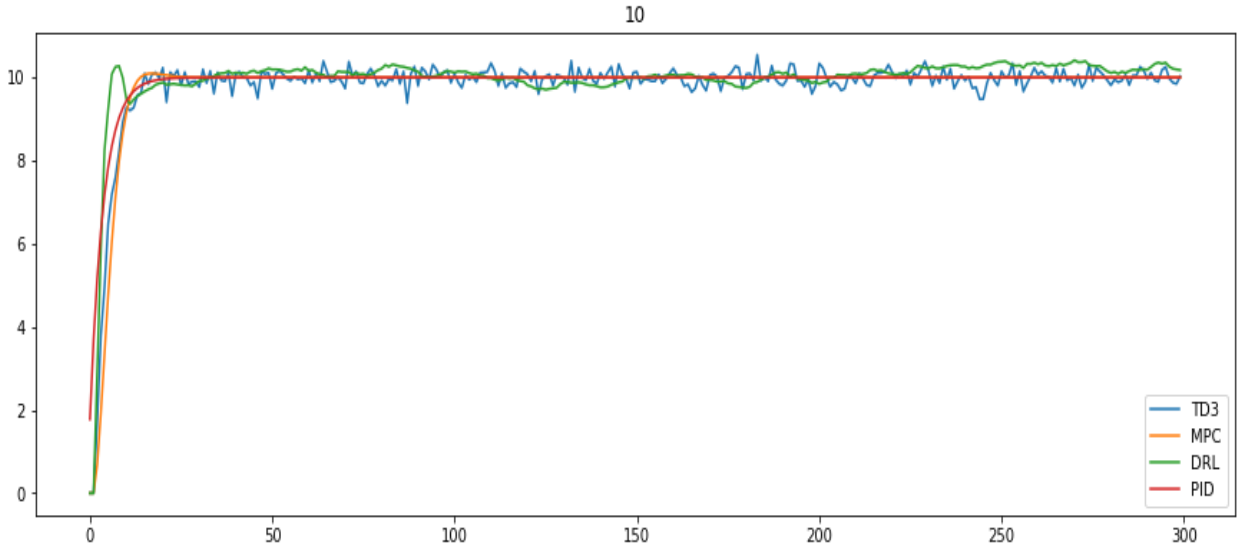

0100
TD3
PID
DRL
MPC

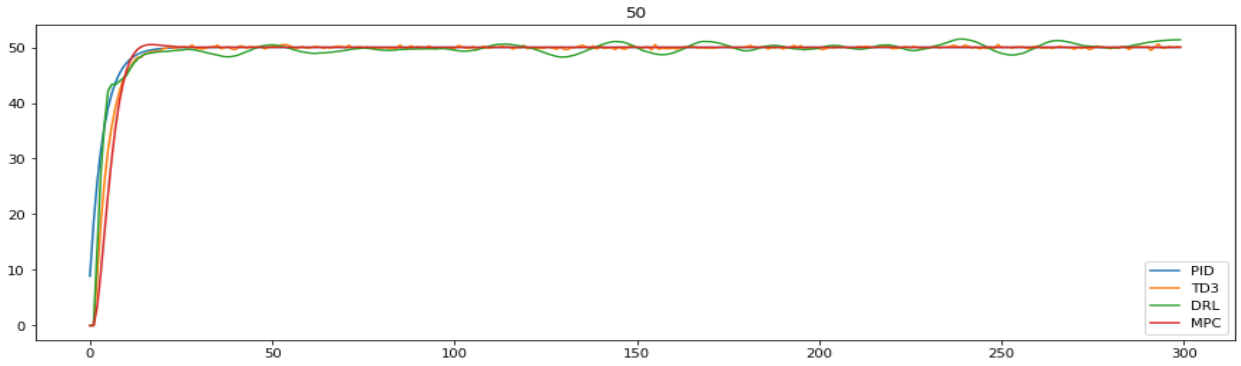

370105
DRL
MPC
PID

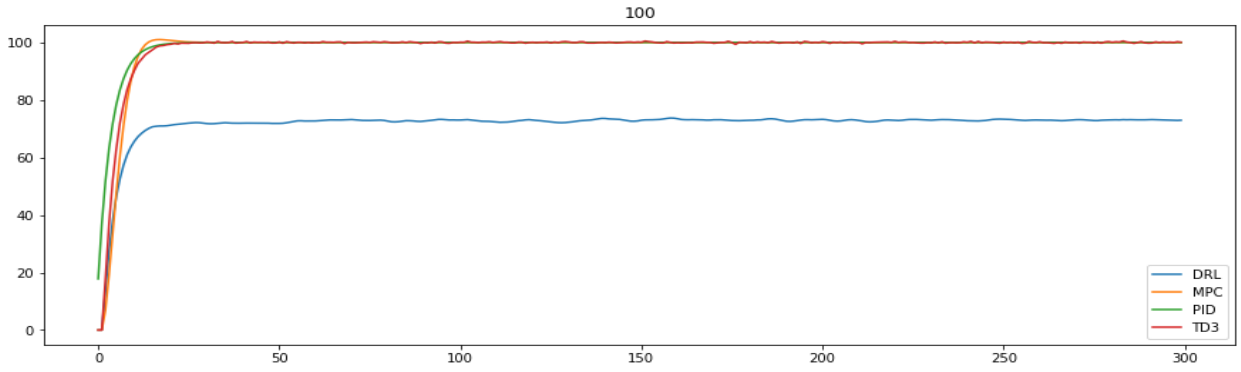
2nd Order

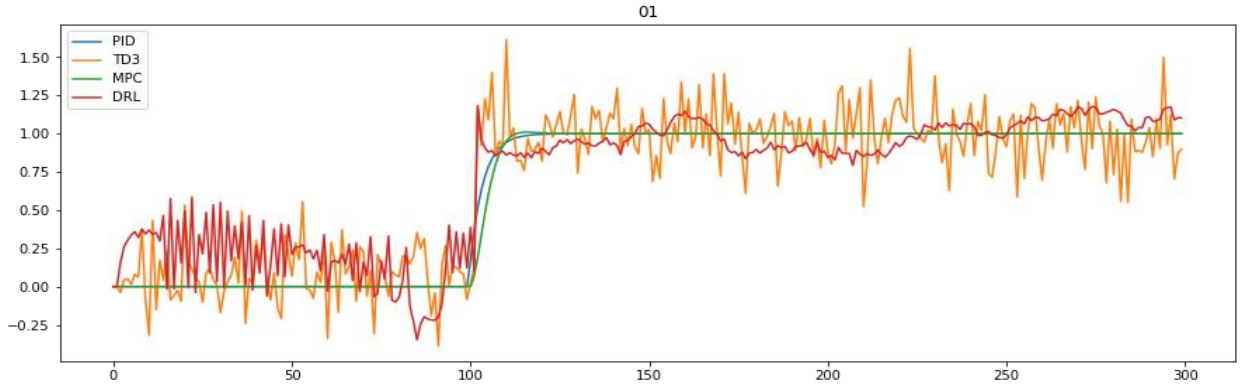


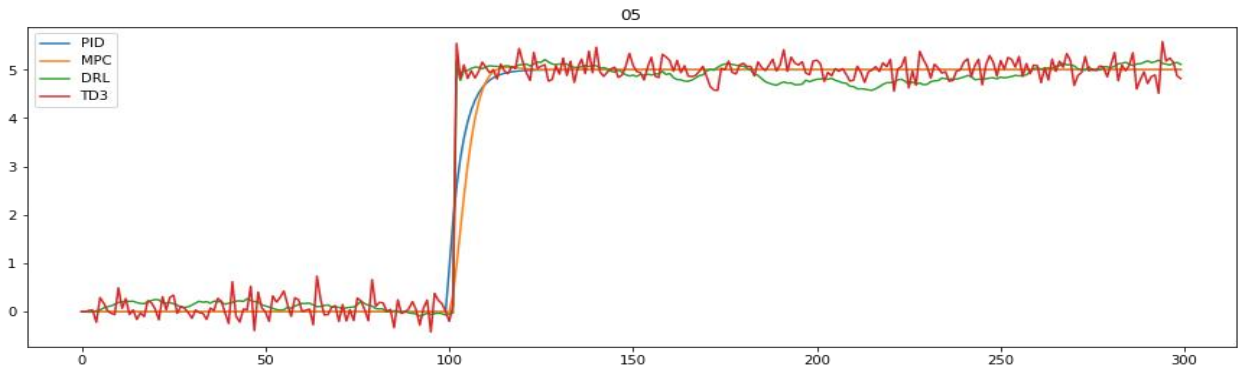


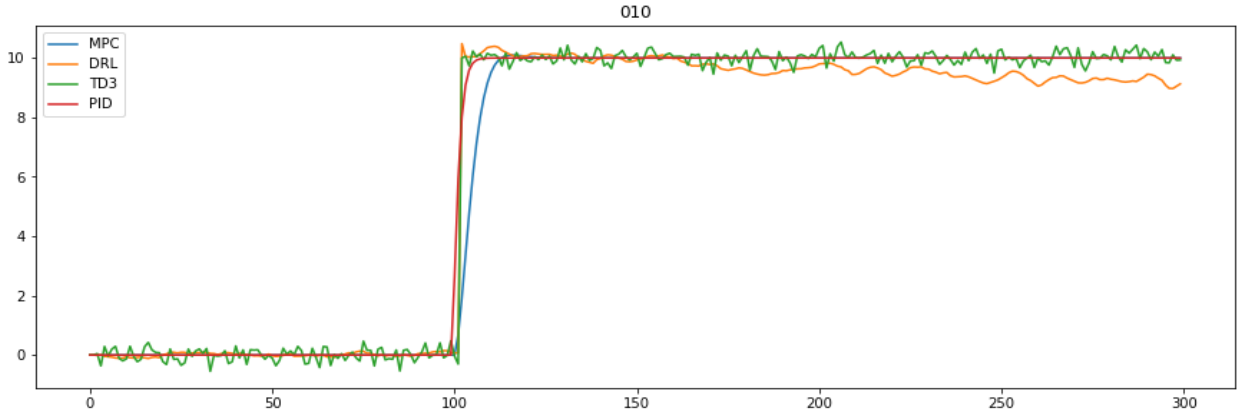


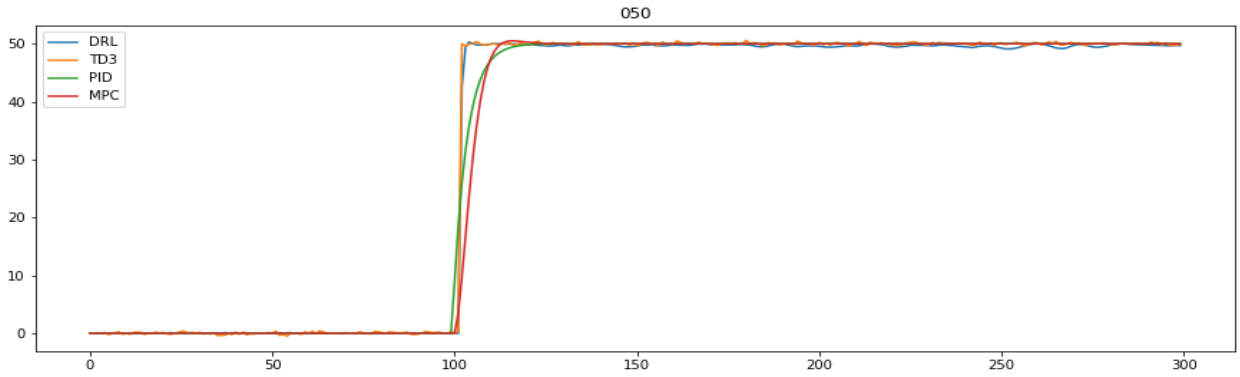

50
PID
TD3
DRL
MPC

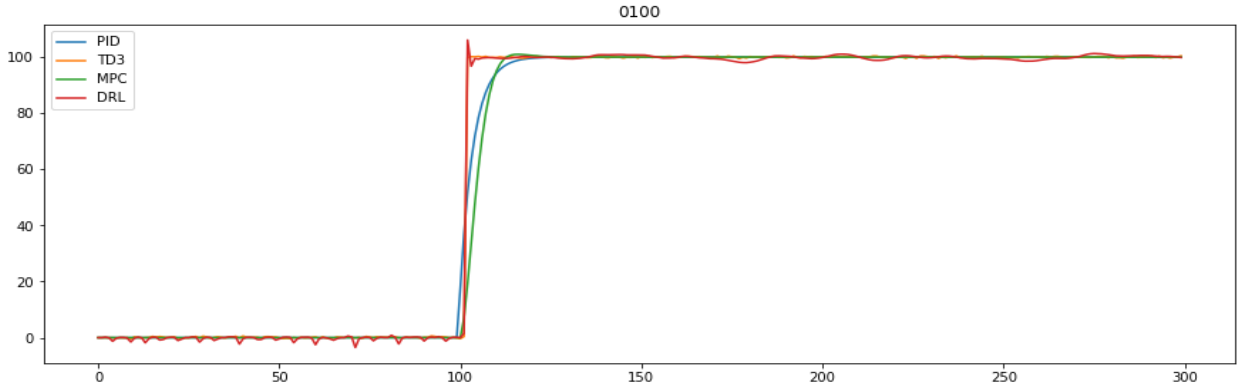

100
DRL
MPC
PID
TD3

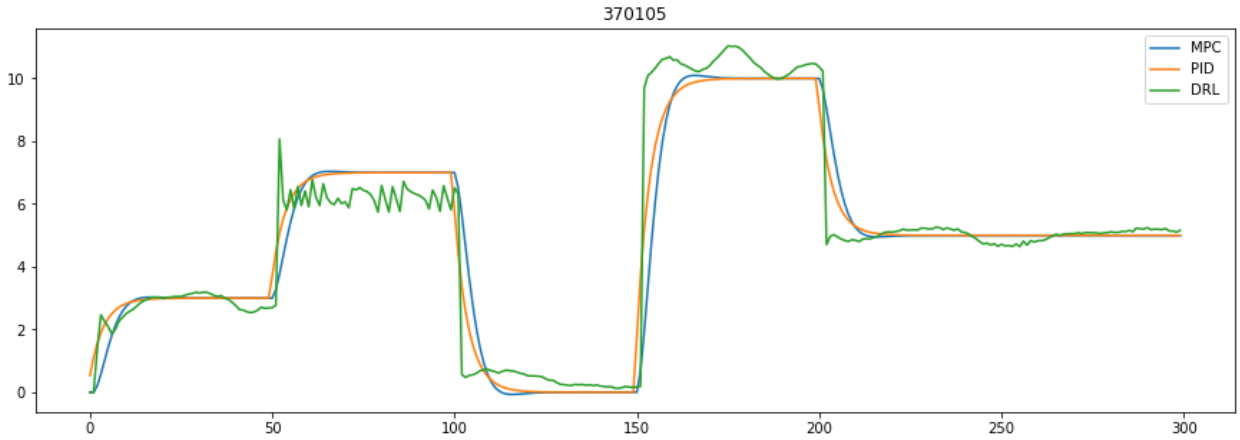

01
PID
TD3
MPC
DRL


05
PID
MPC
DRL
TD3

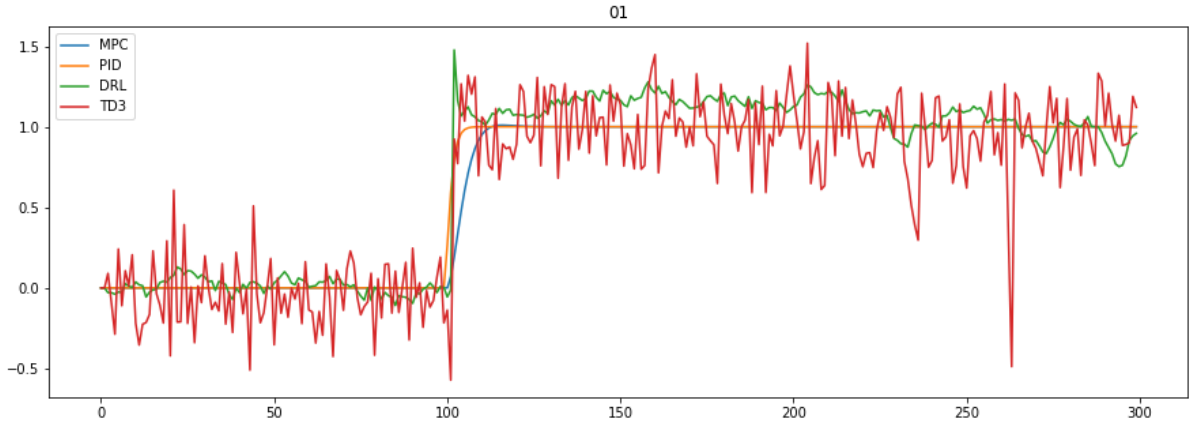

010
MPC
DRL
TD3
PID

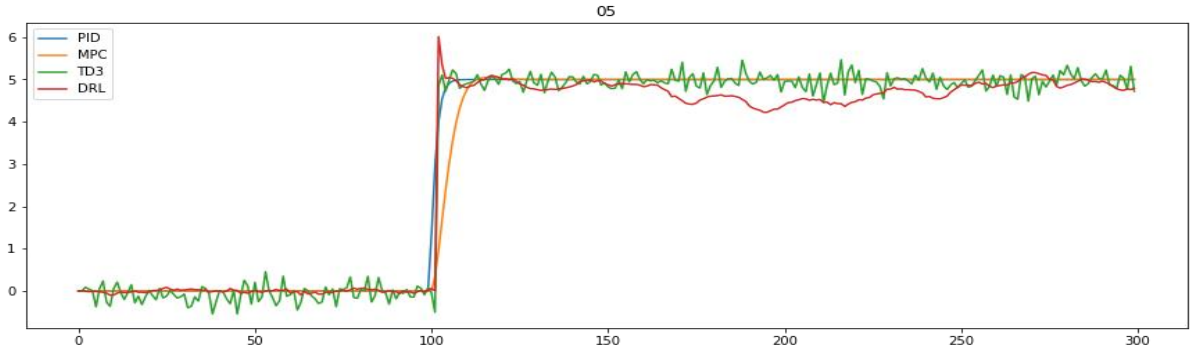

050
DRL
TD3
PID
MPC

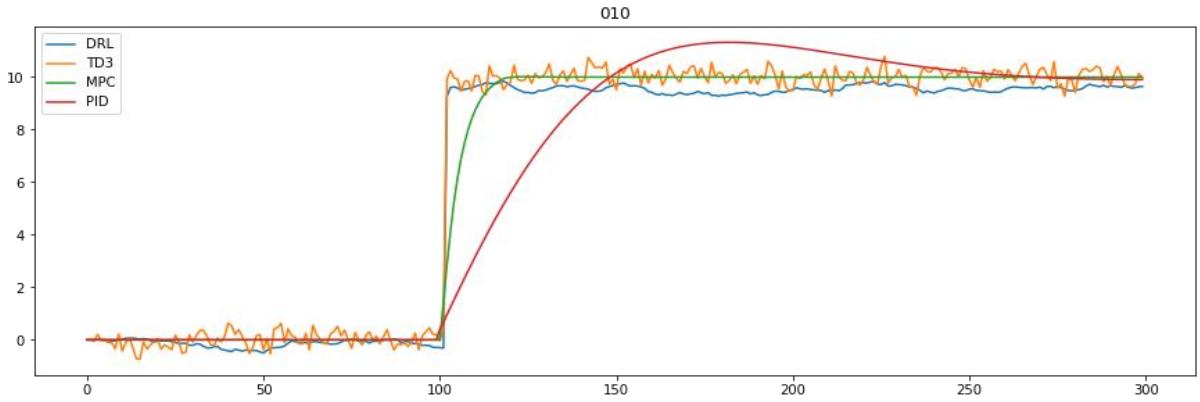

0100
PID
TD3
MPC
DRL

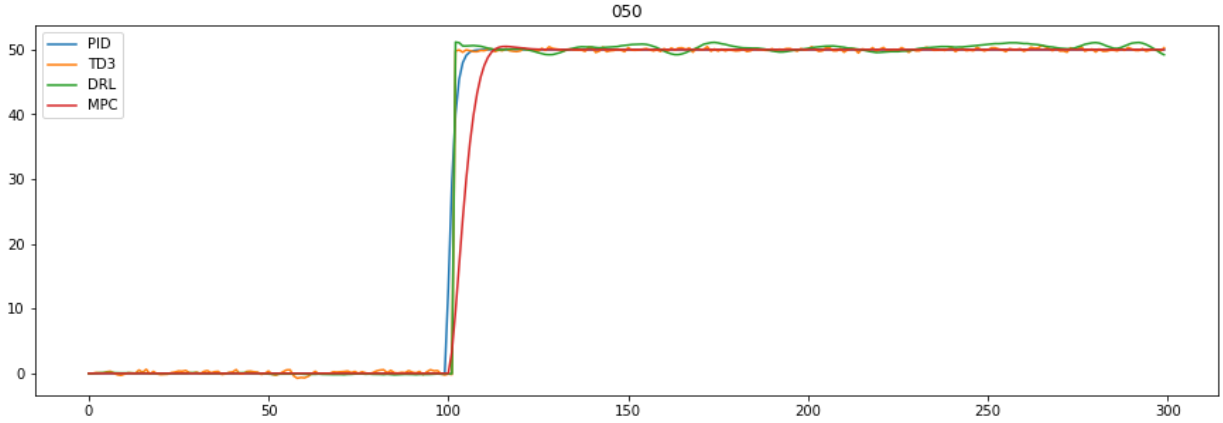

370105
MPC
PID
DRL

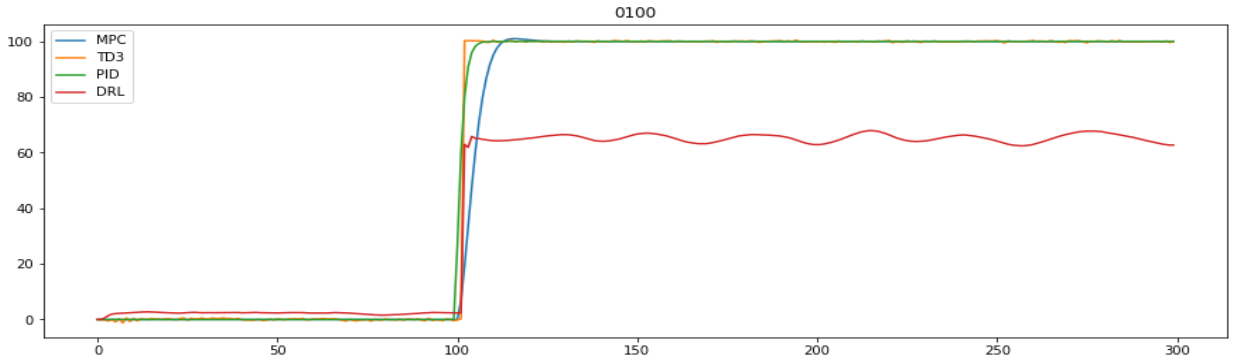
2nd Order: CD

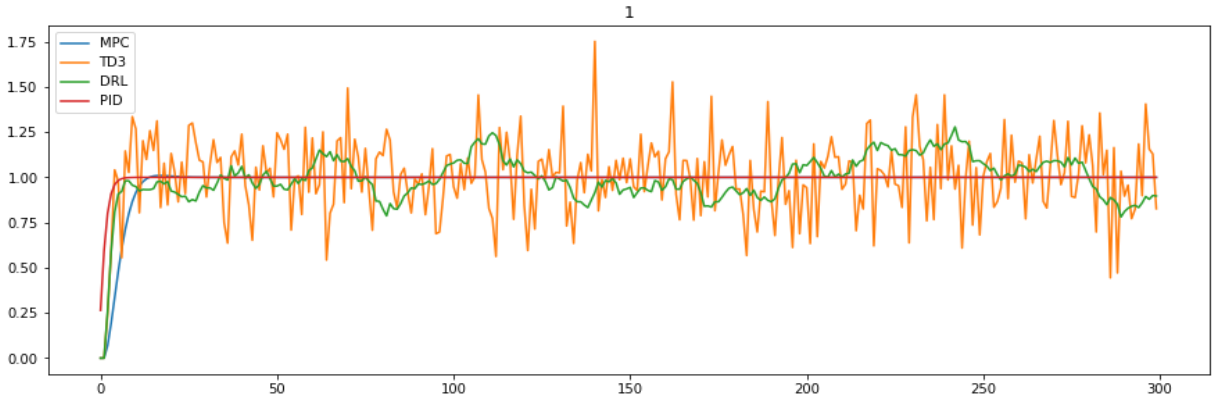

01
MPC
PID
DRL
TD3

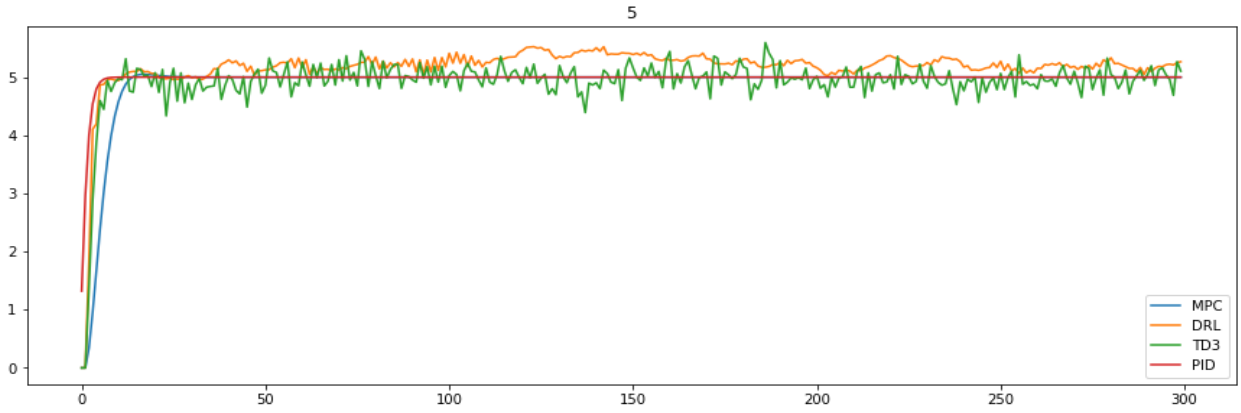

05
PID
MPC
TD3
DRL

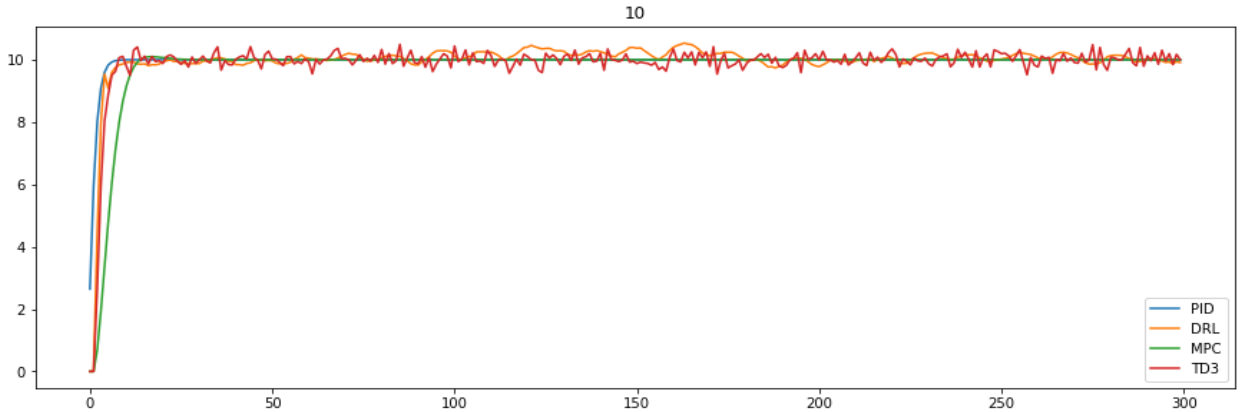

010
DRL
TD3
MPC
PID

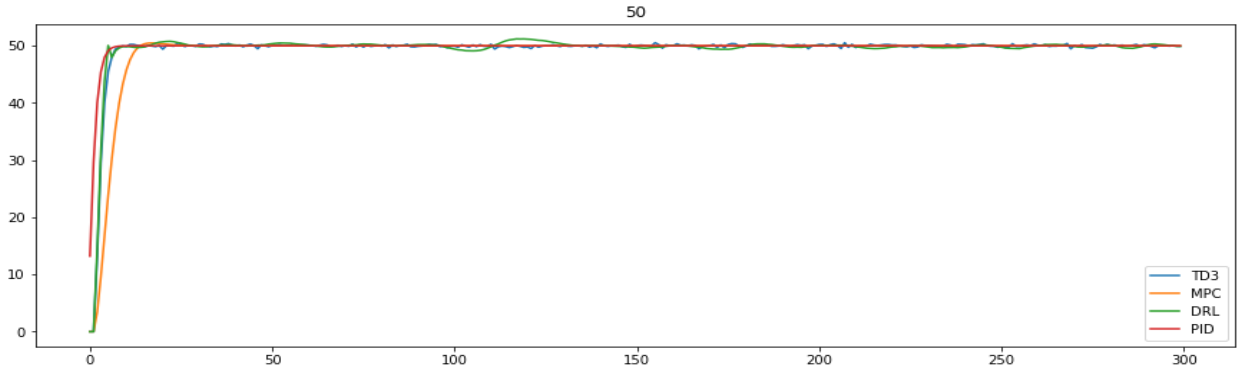

050
PID
TD3
DRL
MPC

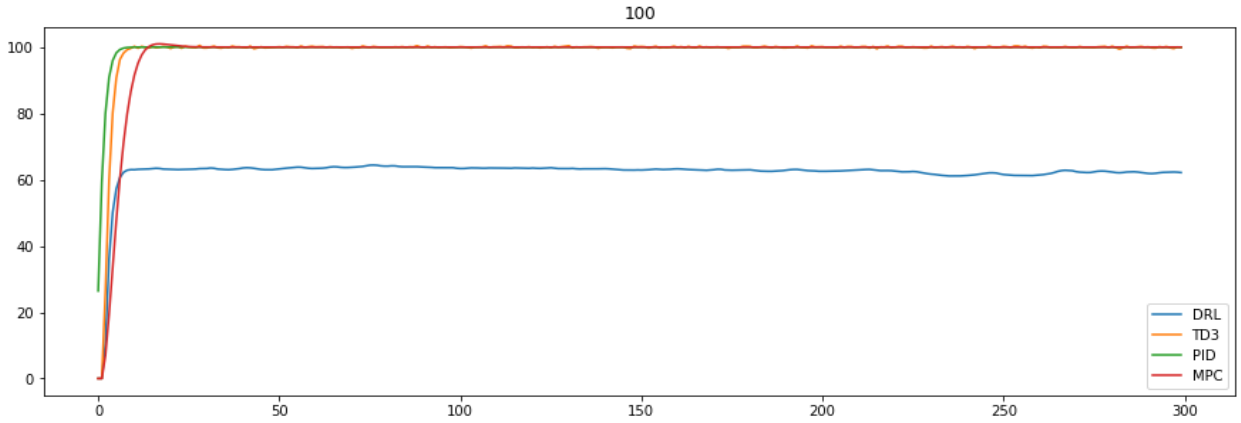

0100
MPC
TD3
PID
DRL
0
20
40
60
80
100
0
50
100
150
200
250
300

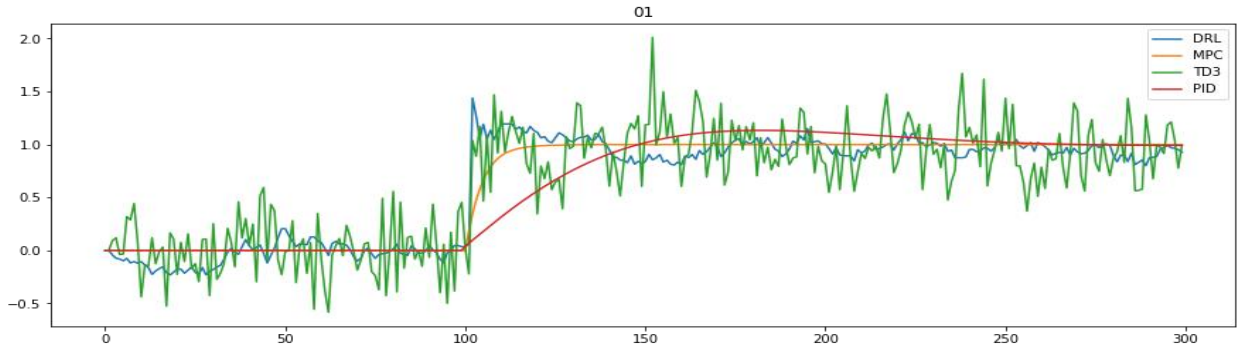

1
MPC
TD3
DRL
PID
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
50
100
150
200
250
300

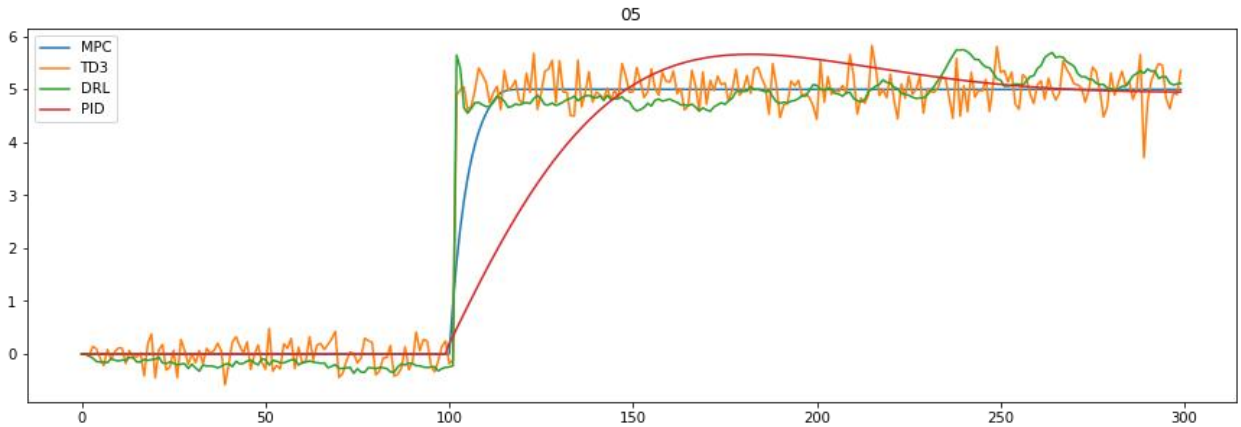

5
MPC
DRL
TD3
PID
0
1
2
3
4
5
0
50
100
150
200
250
300

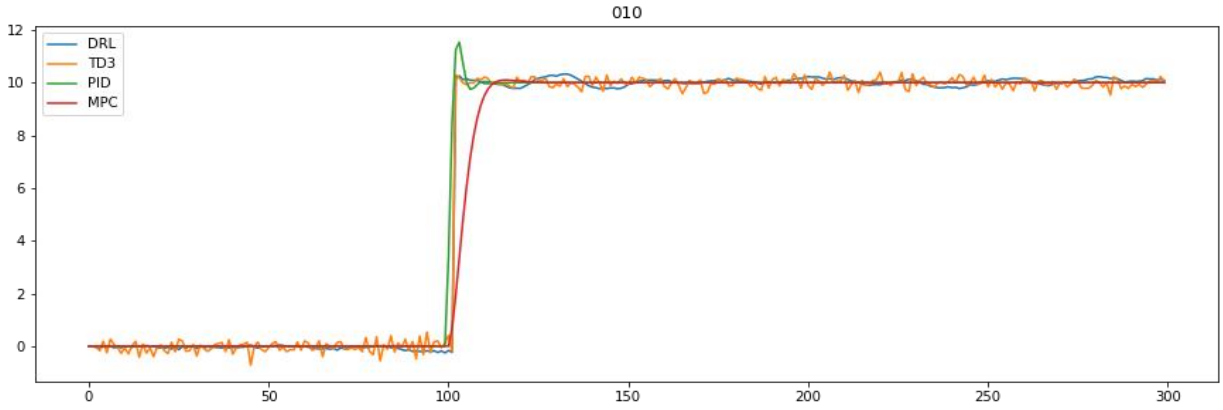

10
PID
DRL
MPC
TD3
0
2
4
6
8
10
0
50
100
150
200
250
300

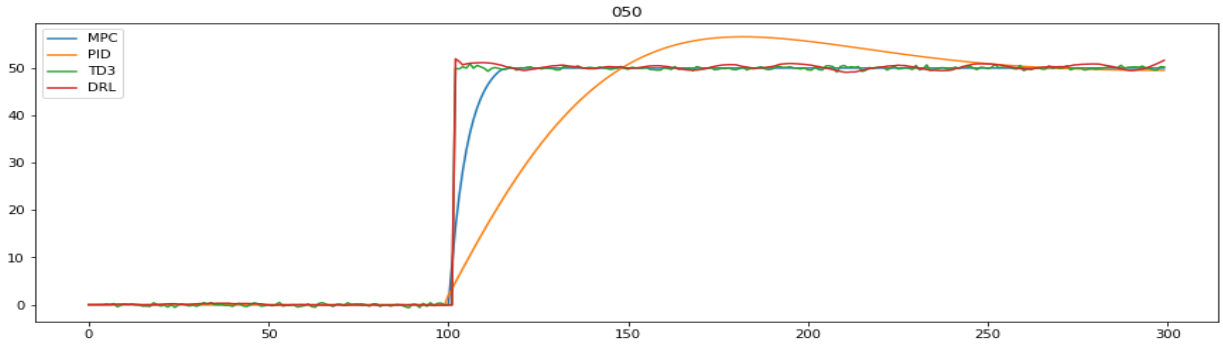

50
TD3
MPC
DRL
PID

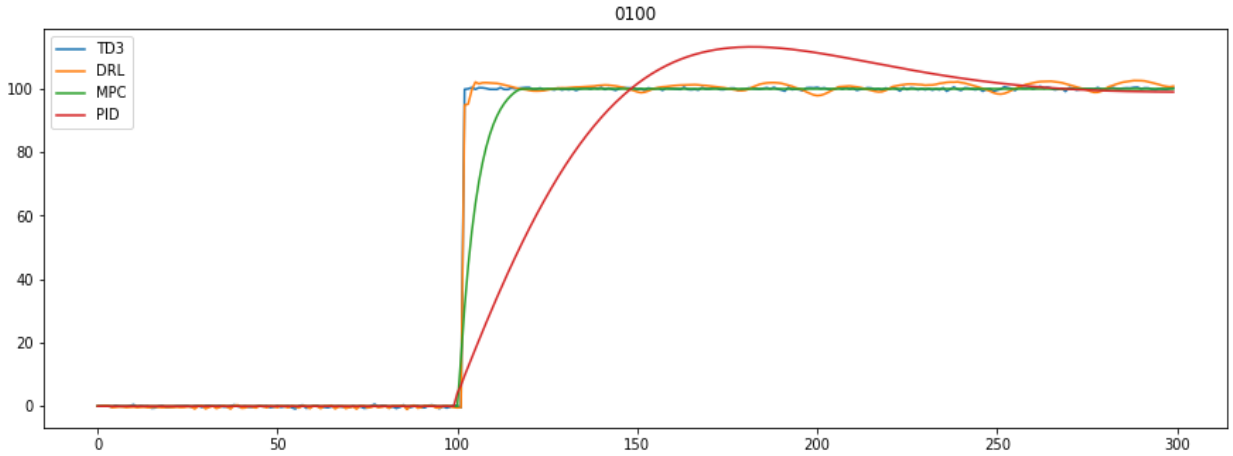

100
DRL
TD3
PID
MPC

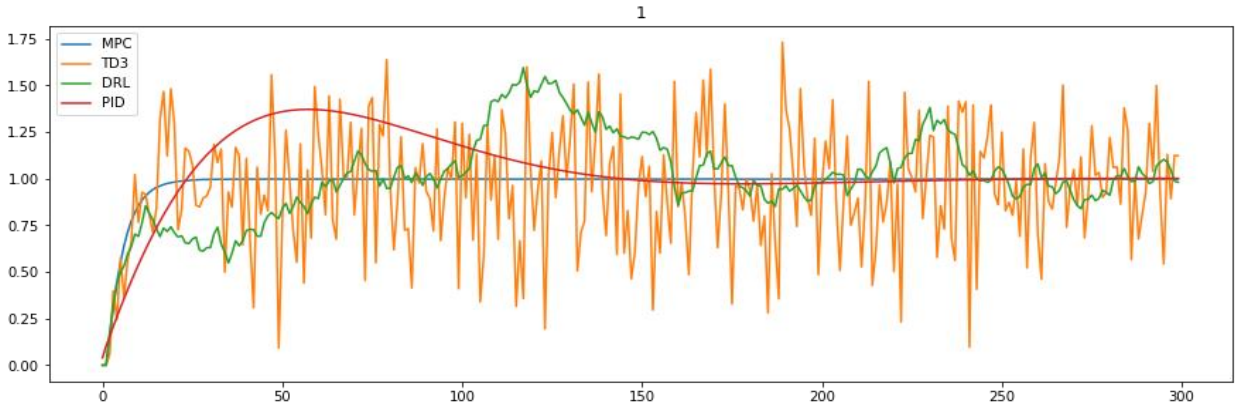

01
DRL
MPC
TD3
PID

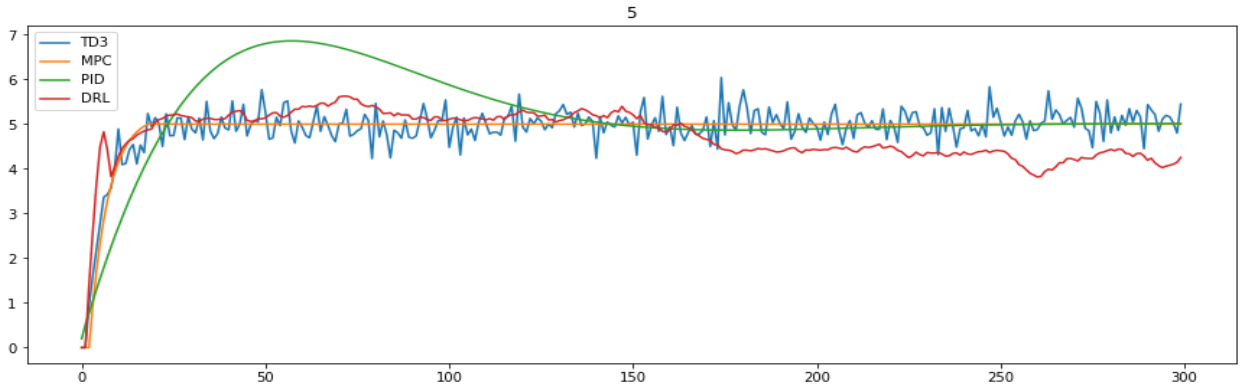

05
MPC
TD3
DRL
PID

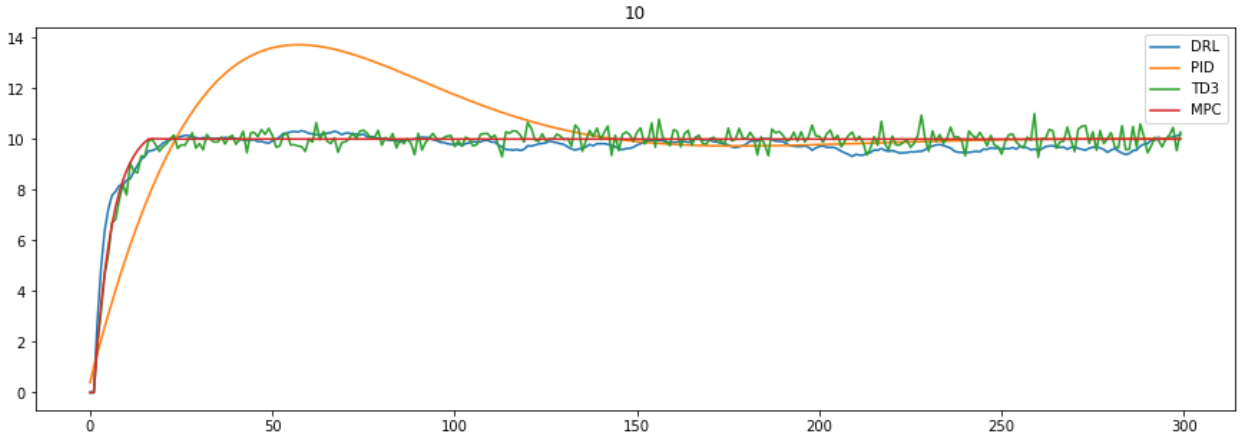

010
DRL
TD3
PID
MPC

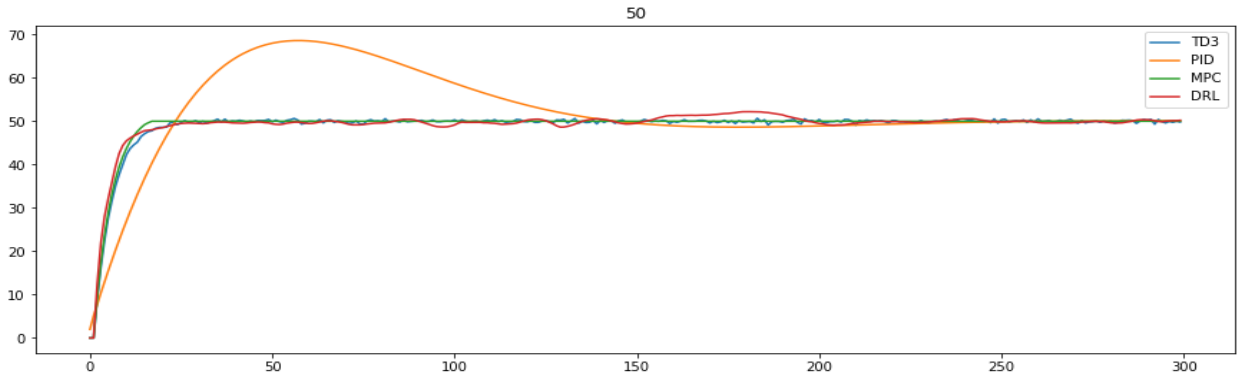

050
MPC
PID
TD3
DRL

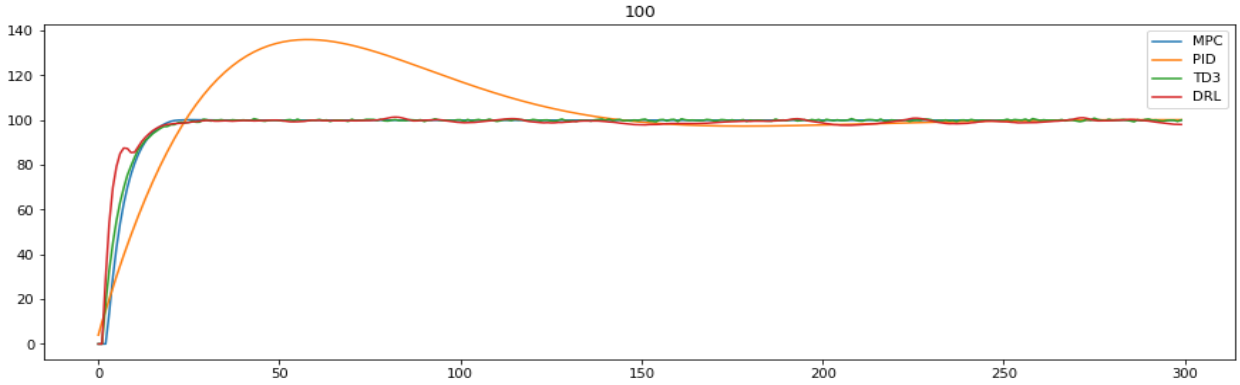

0100
TD3
DRL
MPC
PID

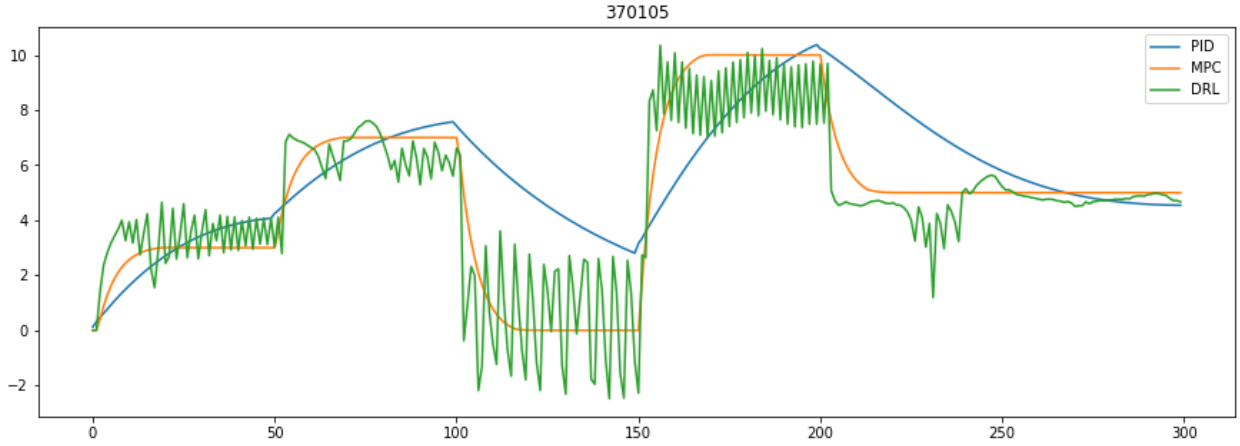

1
MPC
TD3
DRL
PID


5
TD3
MPC
PID
DRL
0
1
2
3
4
5
6
7
0
50
100
150
200
250
300

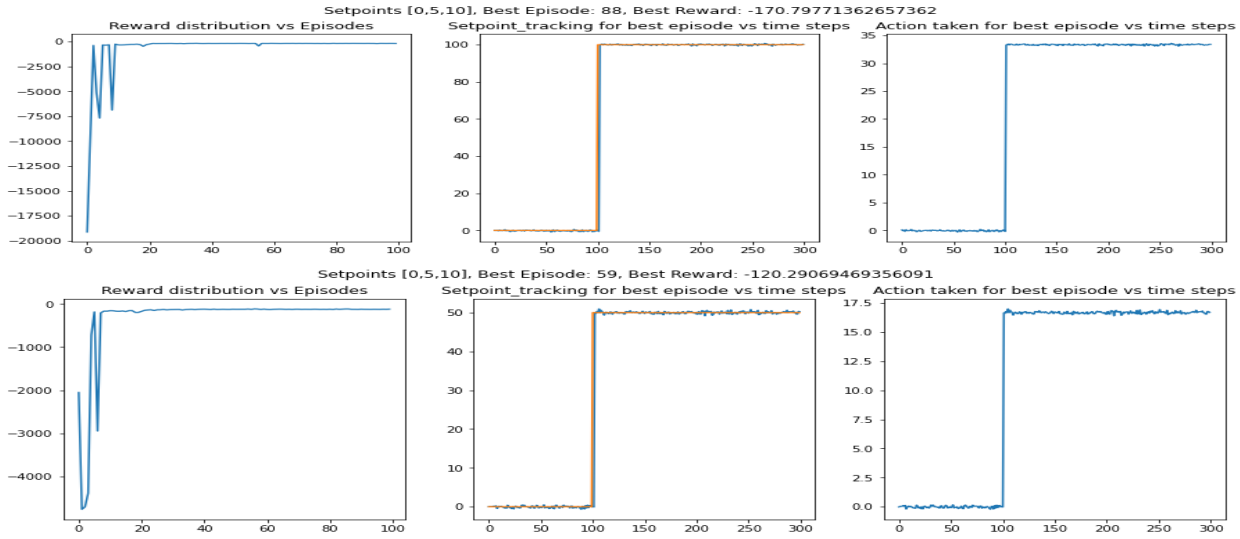

10
DRL
PID
TD3
MPC
0
2
4
6
8
10
12
14
0
50
100
150
200
250
300


50
TD3
PID
MPC
DRL
0
10
20
30
40
50
60
70
0
50
100
150
200
250
300

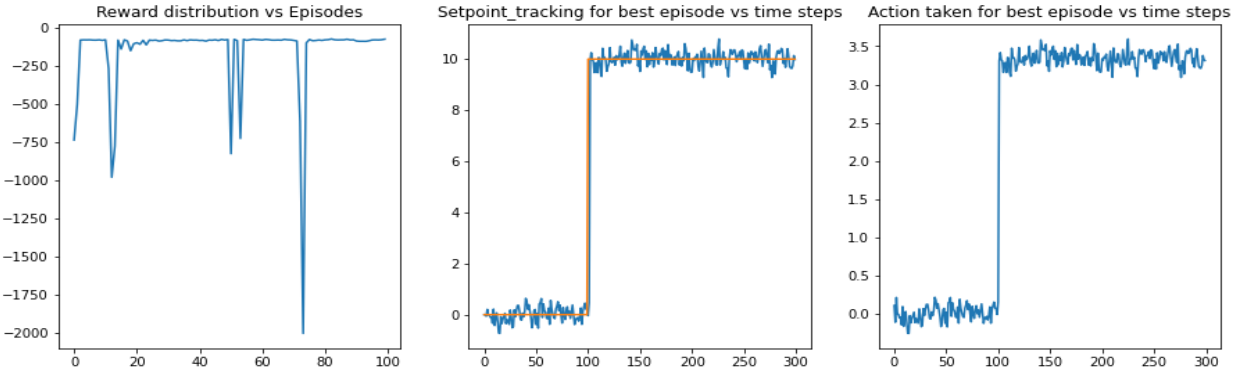




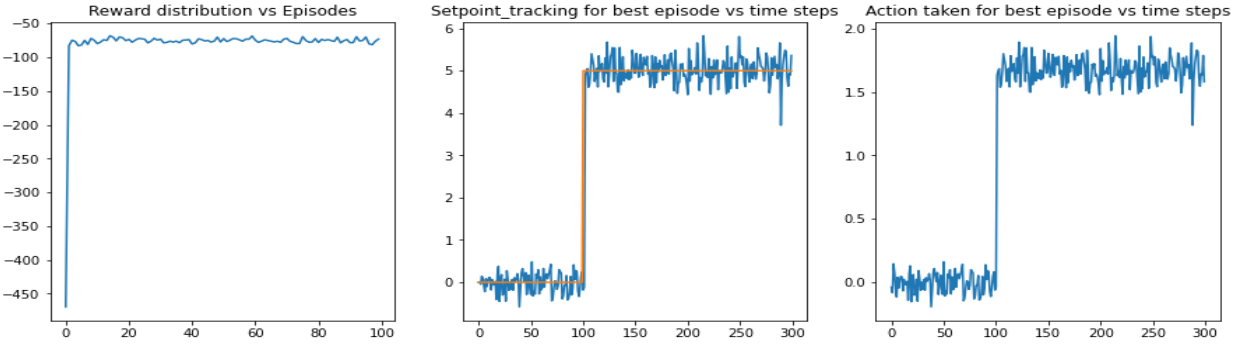
TD3 1st Order Setpoint Tracking:



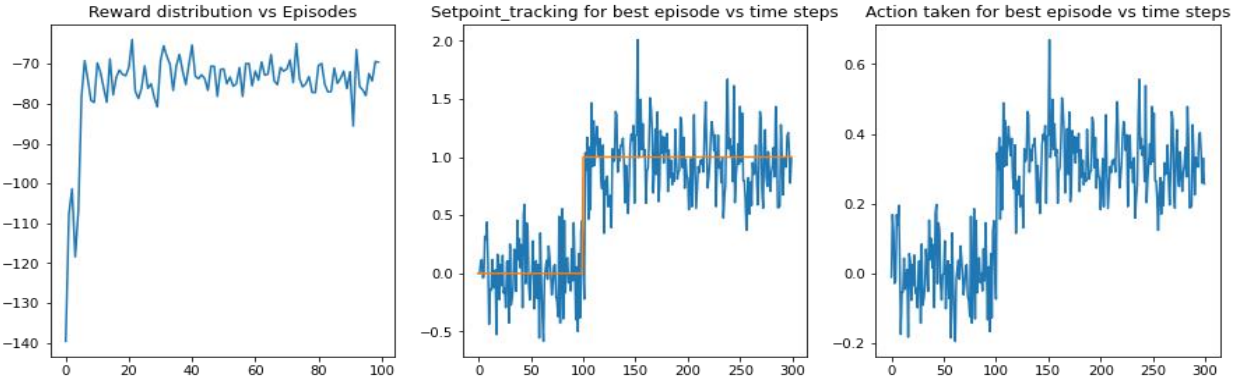
Setpoints [0,5,10], Best Episode: 89, Best Reward: -80.31349628751202



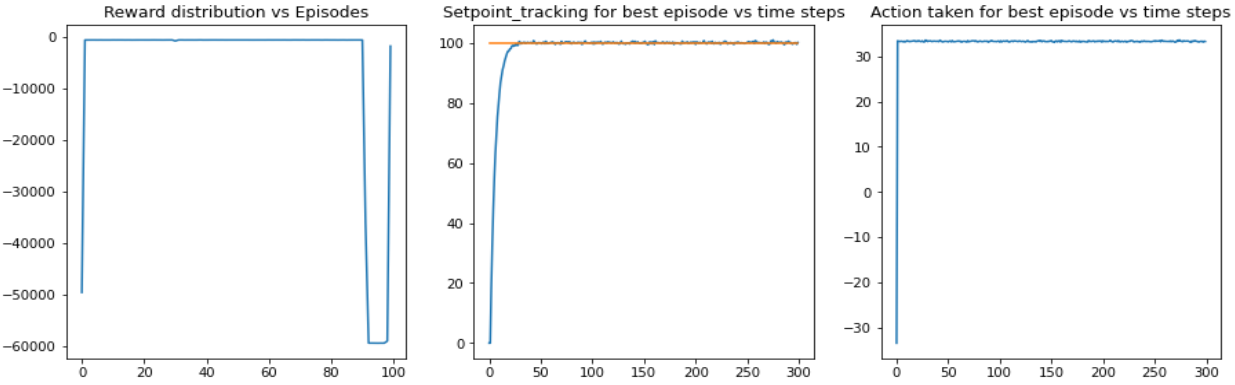
Setpoints [0,5,10], Best Episode: 23, Best Reward: -72.78006070115778



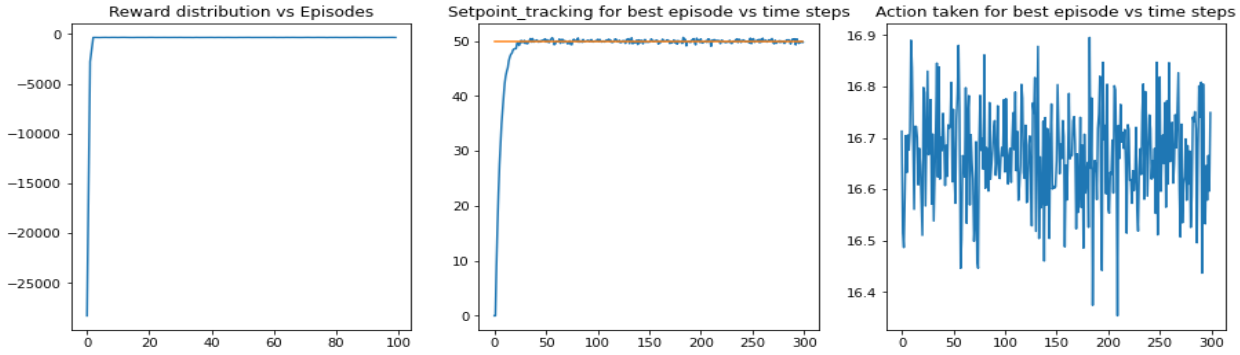
Setpoints [0,5,10], Best Episode: 40, Best Reward: -70.09258896782315



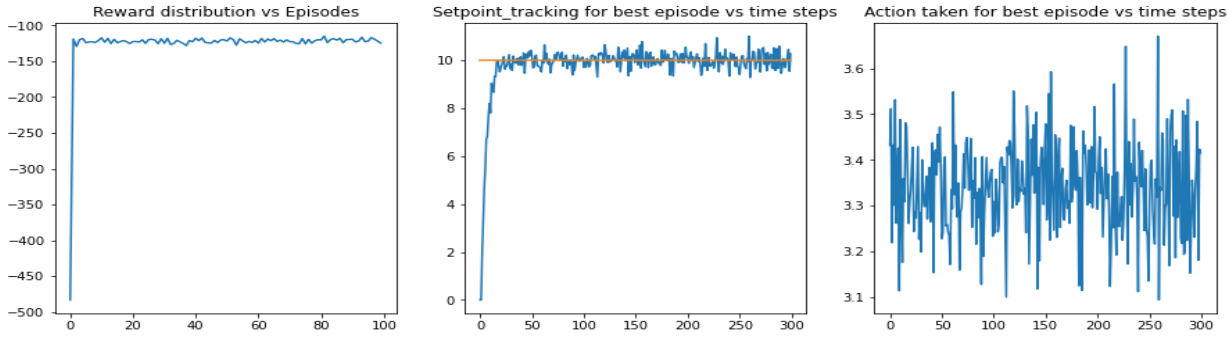
Setpoints [0,5,10], Best Episode: 76, Best Reward: -612.8288309729968



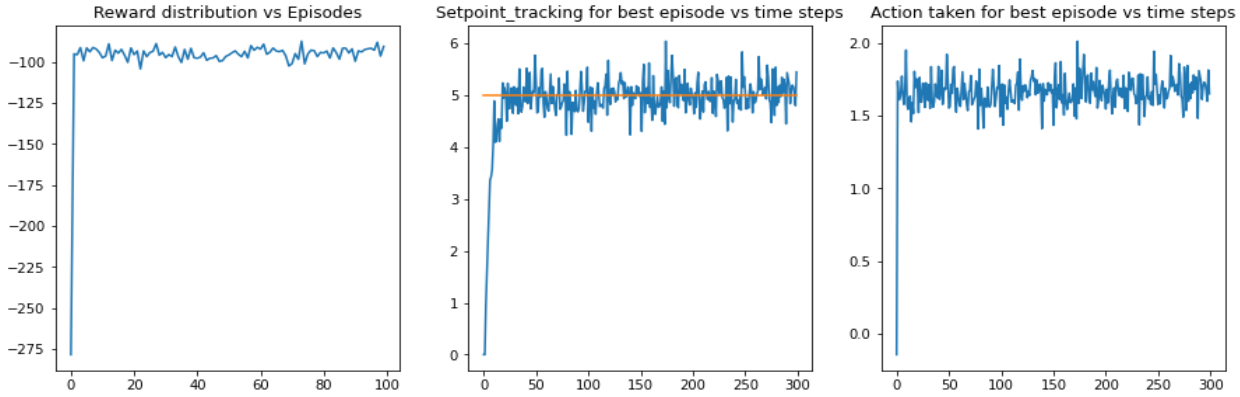
Setpoints [0,5,10], Best Episode: 21, Best Reward: -339.07224192123675



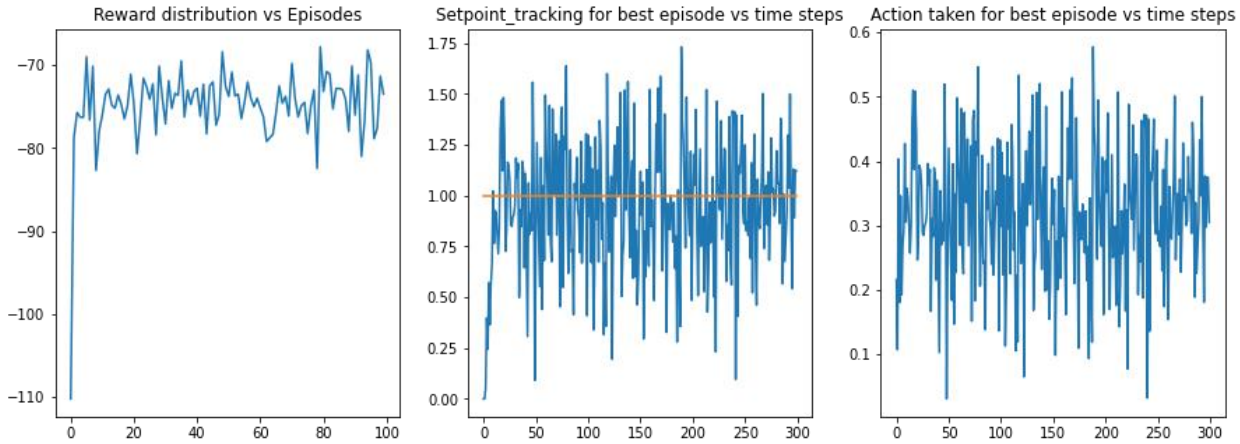
Setpoints [0,5,10], Best Episode: 90, Best Reward: -119.83815235789356



Setpoints [0,5,10], Best Episode: 66, Best Reward: -92.35328802078223

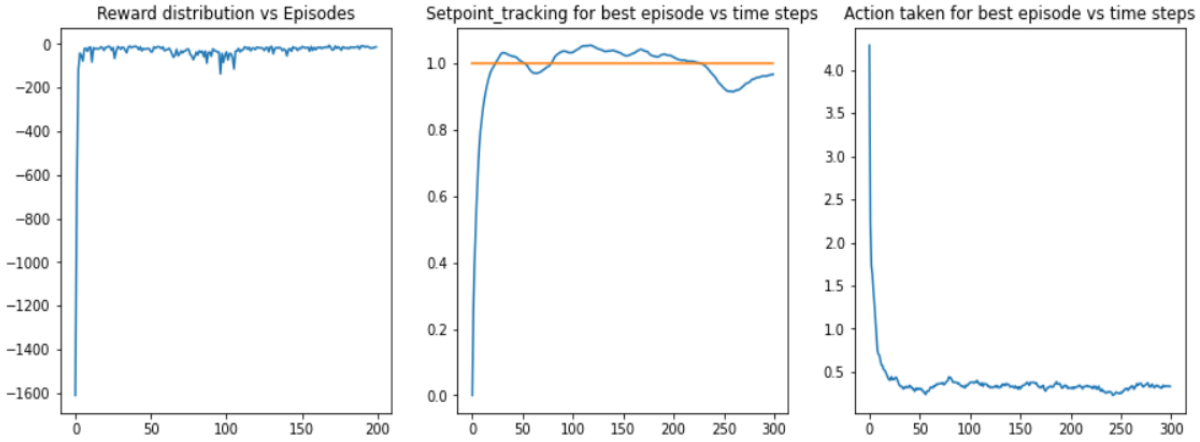


Setpoints [0,5,10], Best Episode: 88, Best Reward: -72.86882151840561

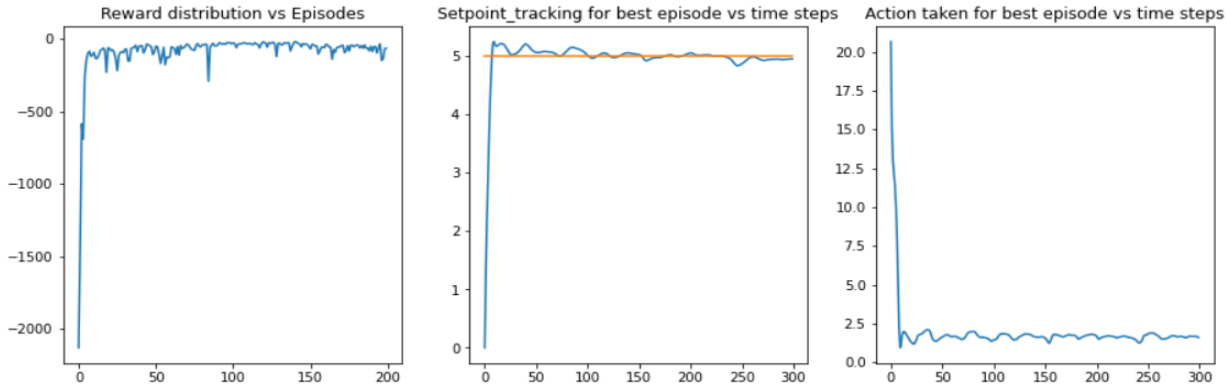


DRL 1st Order Setpoint Tracking:

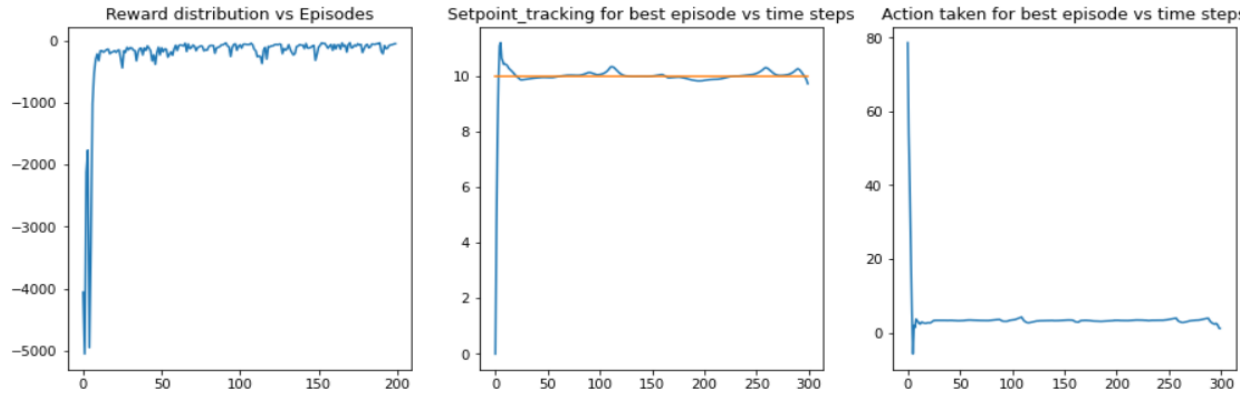
Setpoints [0,5,10], Best Episode: 194, Best Reward: -13.418119691094713



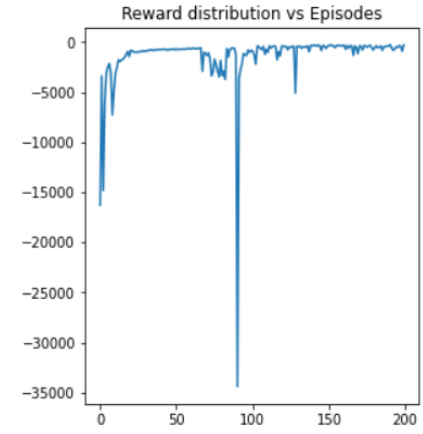
Setpoints [5], Best Episode: 100, Best Reward: -32.18317775561307

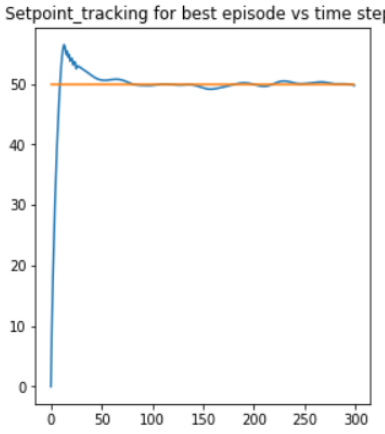


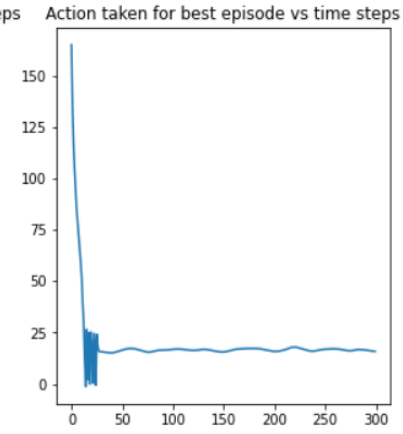
Setpoints [10], Best Episode: 189, Best Reward: -67.46659023194431

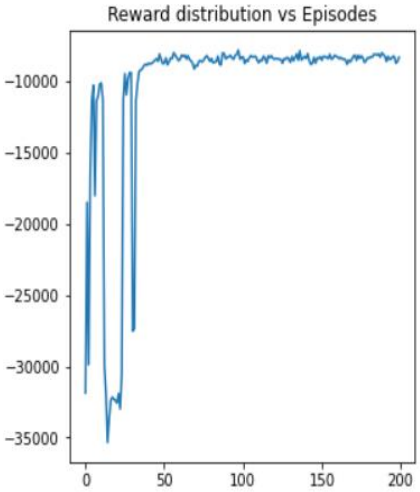


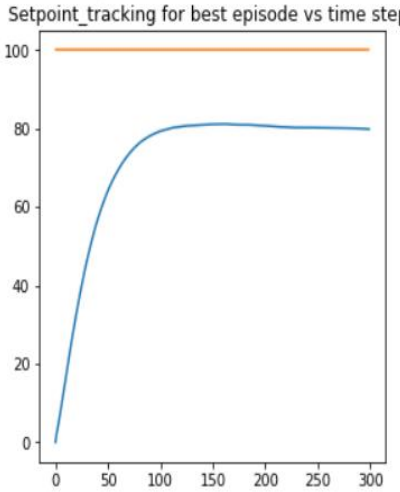
Setpoints [50], Best Episode: 160, Best Reward: -378.4909552151886

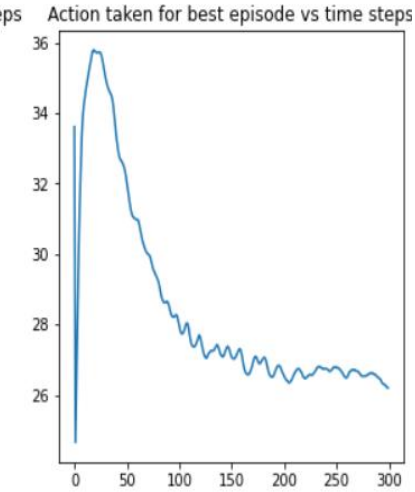




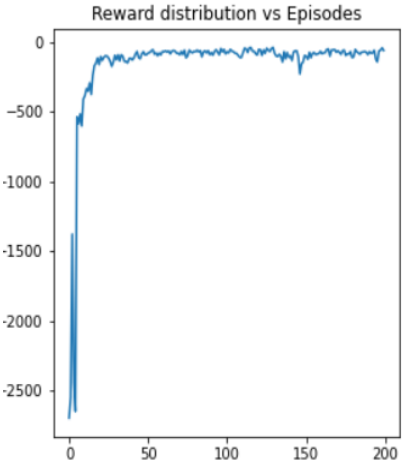


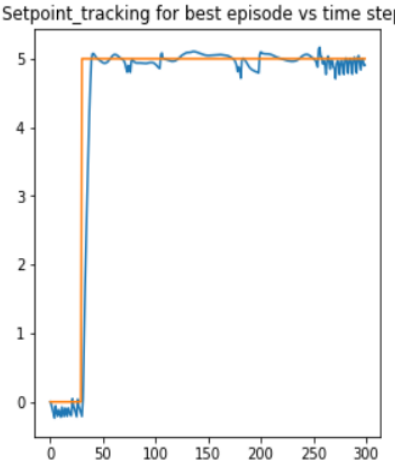


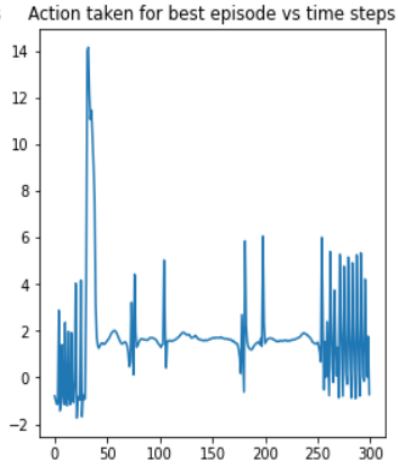




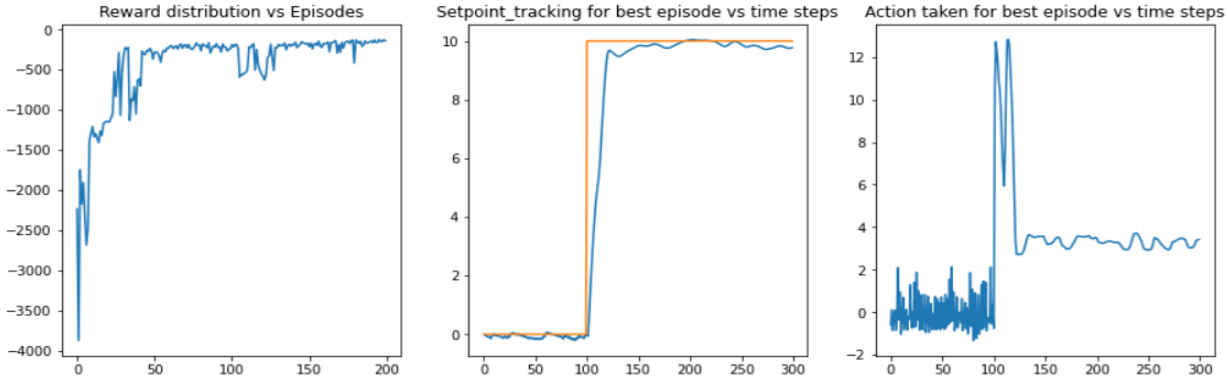
Setpoints [0,5,10], Best Episode: 129, Best Reward: -61.65240897813194



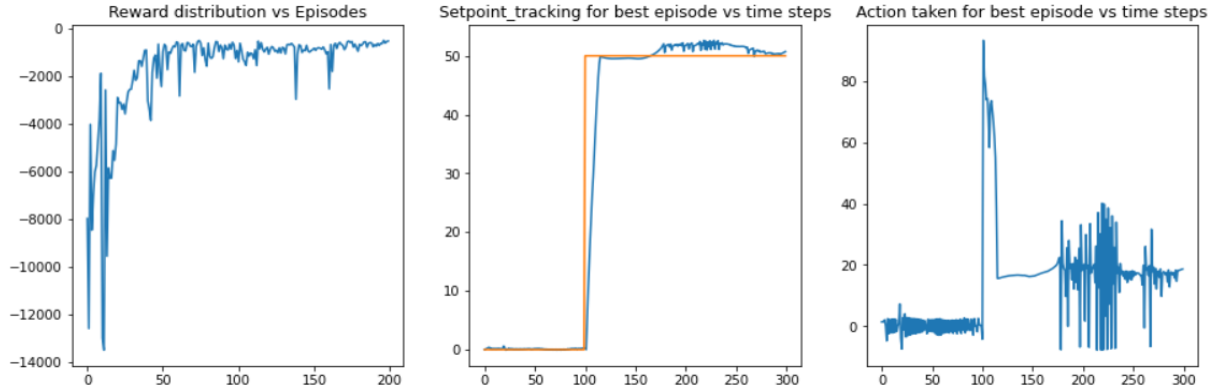


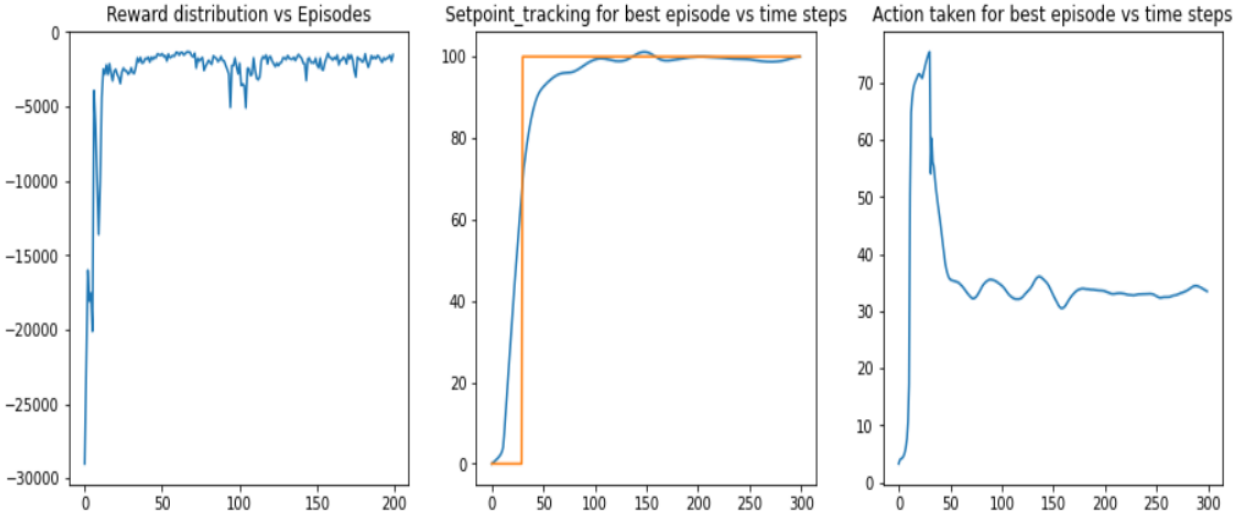


Setpoints [50], Best Episode: 199, Best Reward: -142.28682067187702

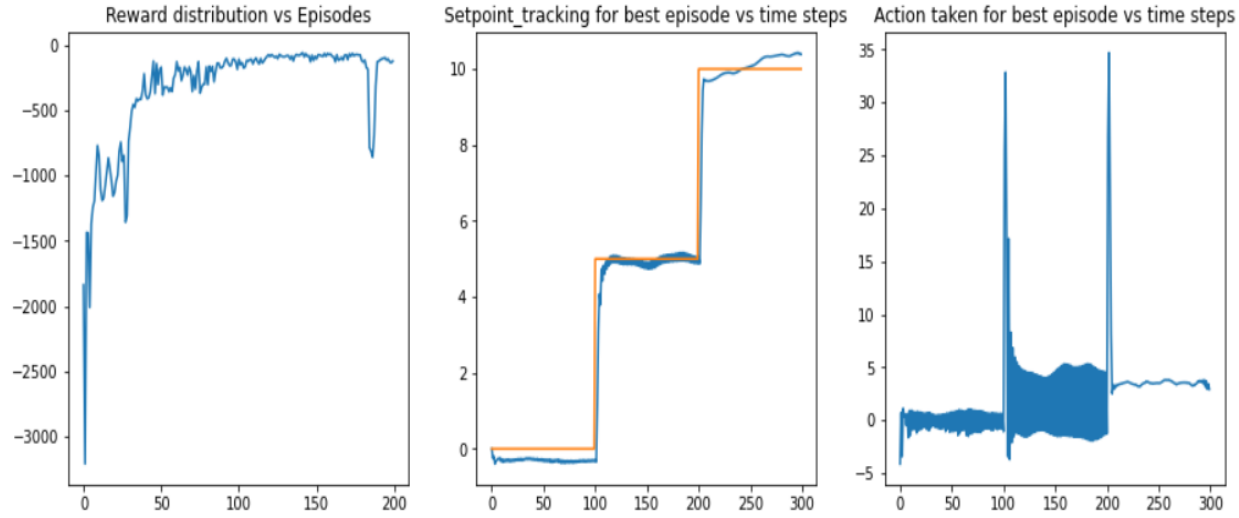


Setpoints [50], Best Episode: 199, Best Reward: -632.2618305348192

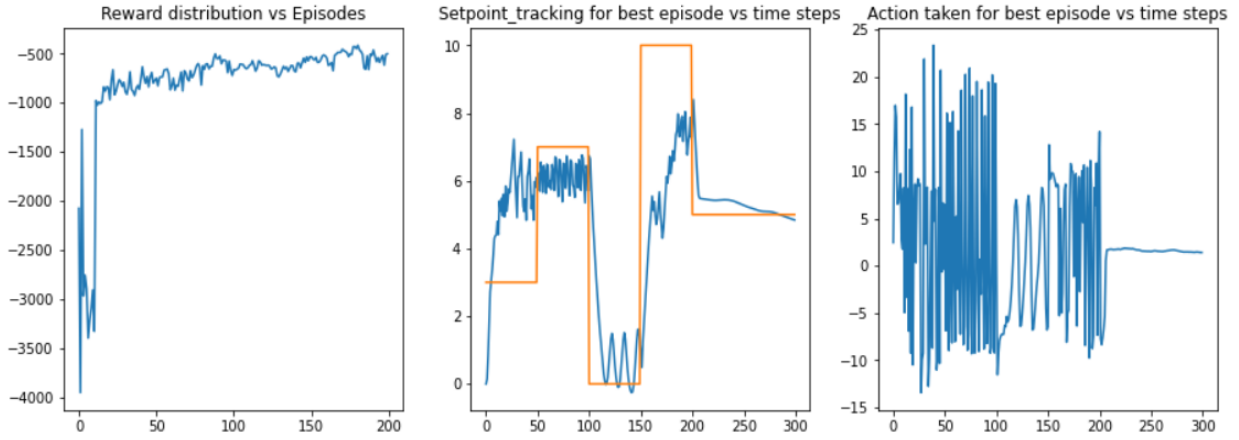




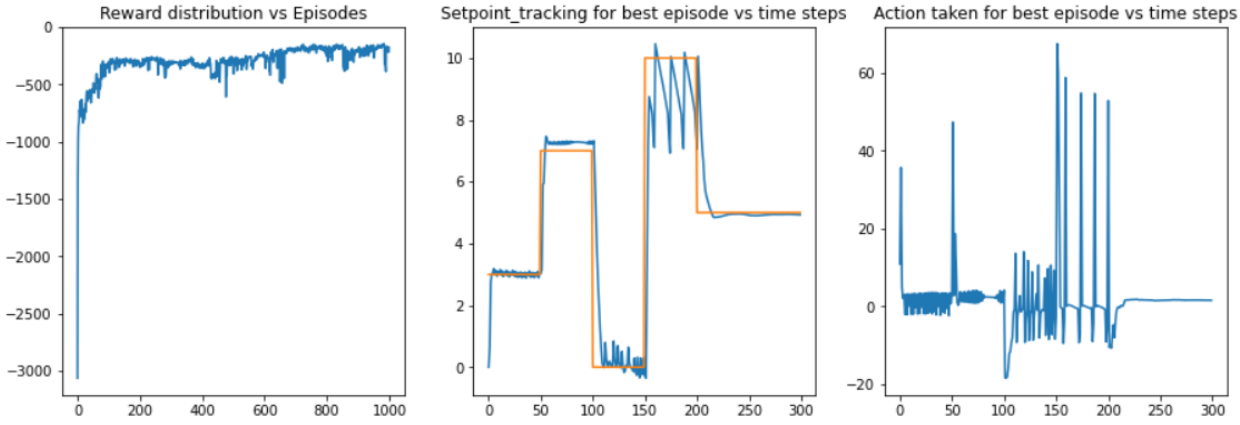
Setpoints [0,5,10], Best Episode: 160, Best Reward: -79.03717415818008



Setpoints [3 7 0 10 5], Best Episode: 182, Best Reward: -468.4319953043679

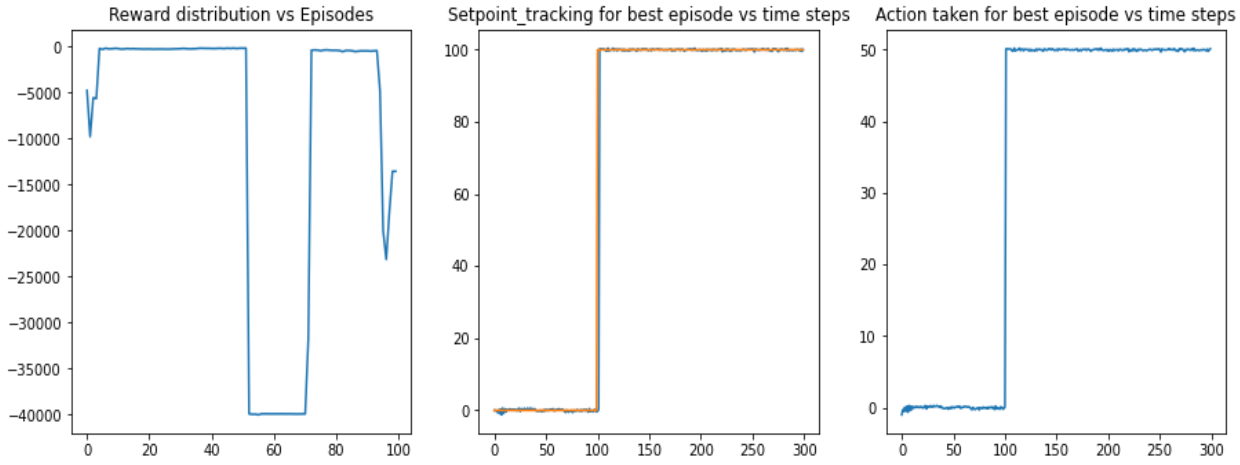


Setpoints [3 7 0 10 5], Best Episode: 984, Best Reward: -157.62784441157544

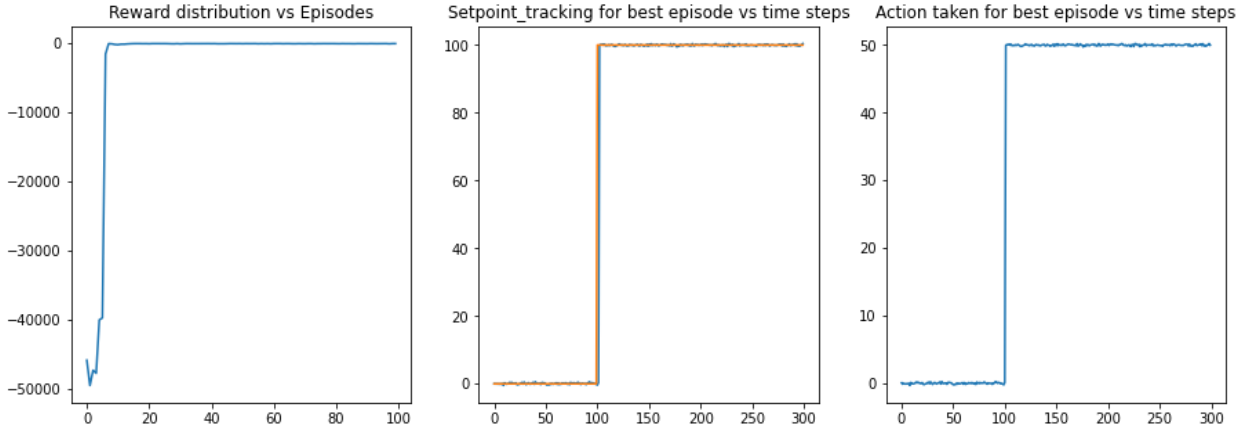


TD3 2nd Order Setpoint Tracking: (1st ,2nd, 3rd represents Overdamped, Critically Damped, Underdamped)

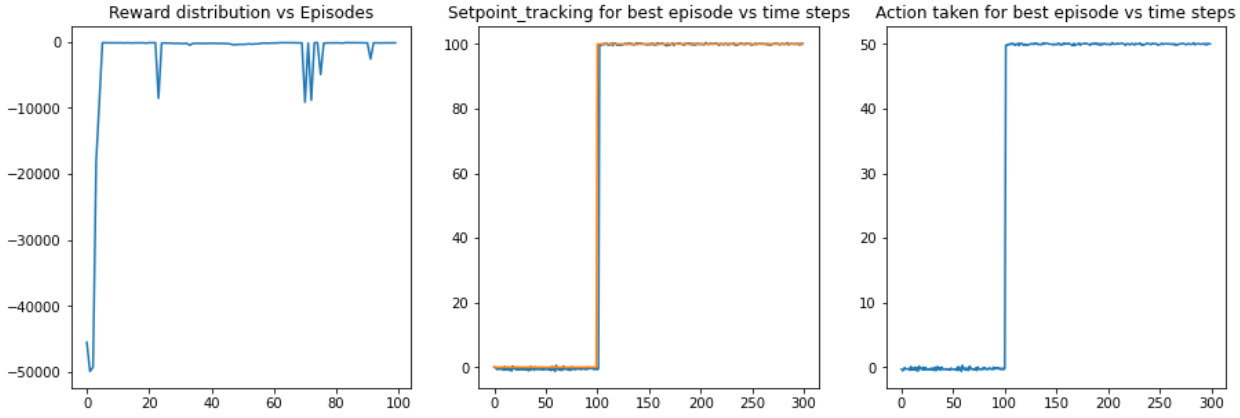
Setpoints [0,5,10], Best Episode: 51, Best Reward: -167.98014394488342



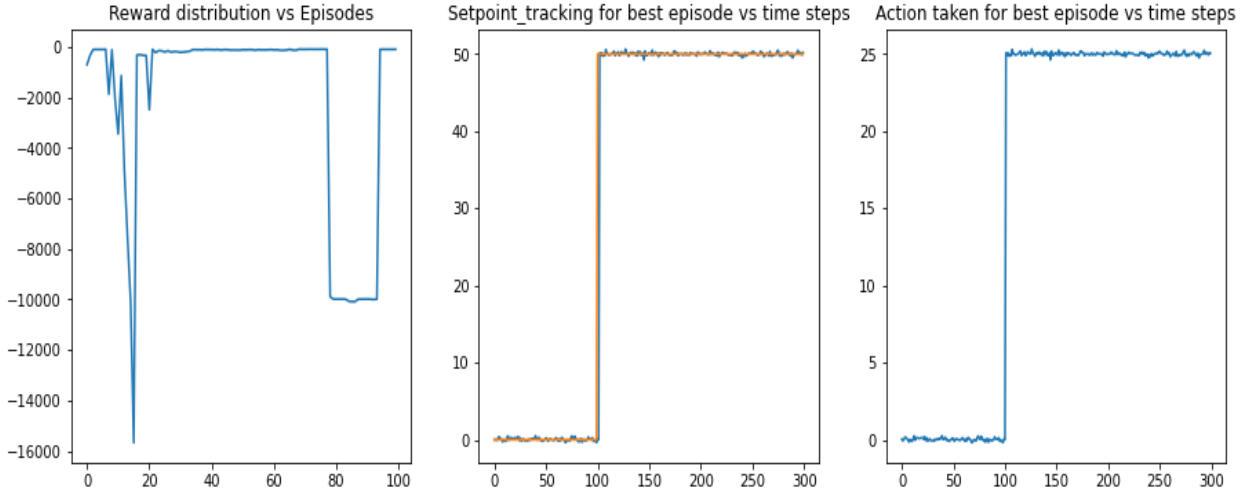
Setpoints [0,5,10], Best Episode: 25, Best Reward: -153.9233411604486



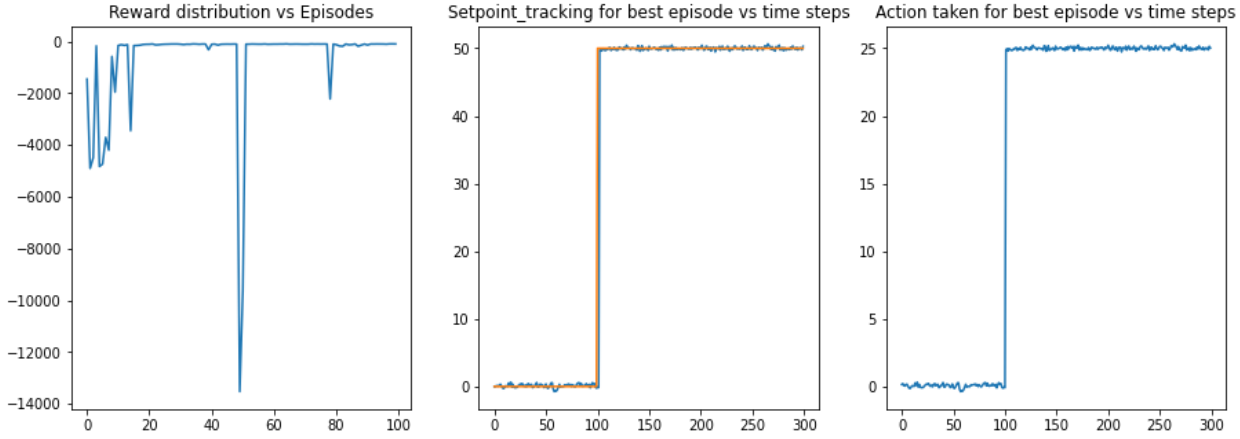
Setpoints [0,5,10], Best Episode: 89, Best Reward: -173.47088667347776



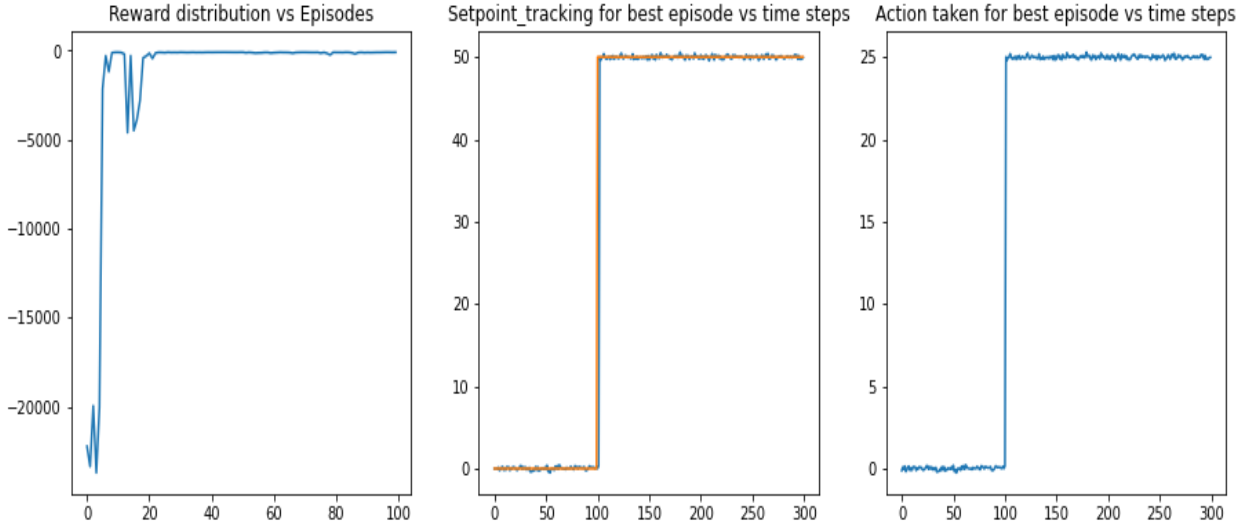
Setpoints [0,5,10], Best Episode: 77, Best Reward: -96.90395651825918



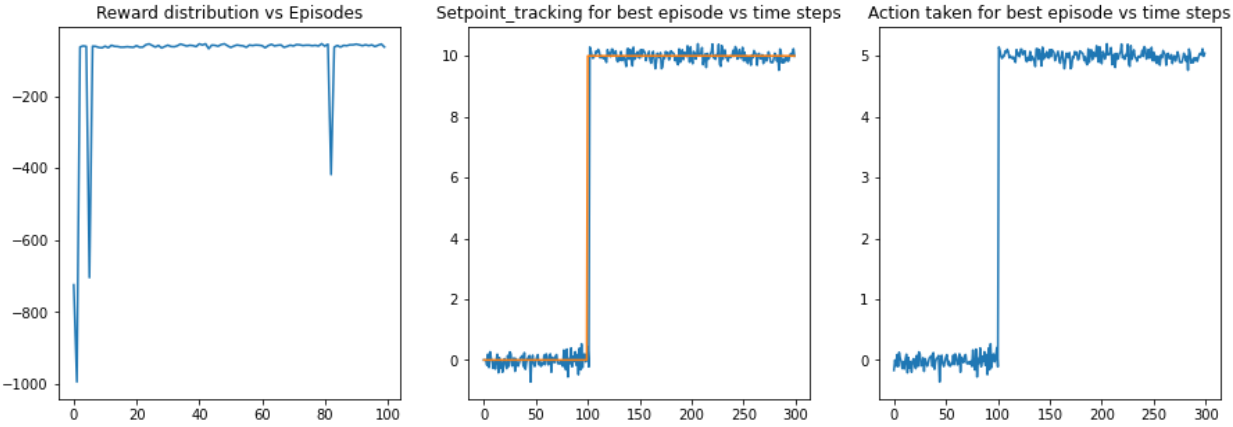
Setpoints [0,5,10], Best Episode: 69, Best Reward: -104.87234284093219



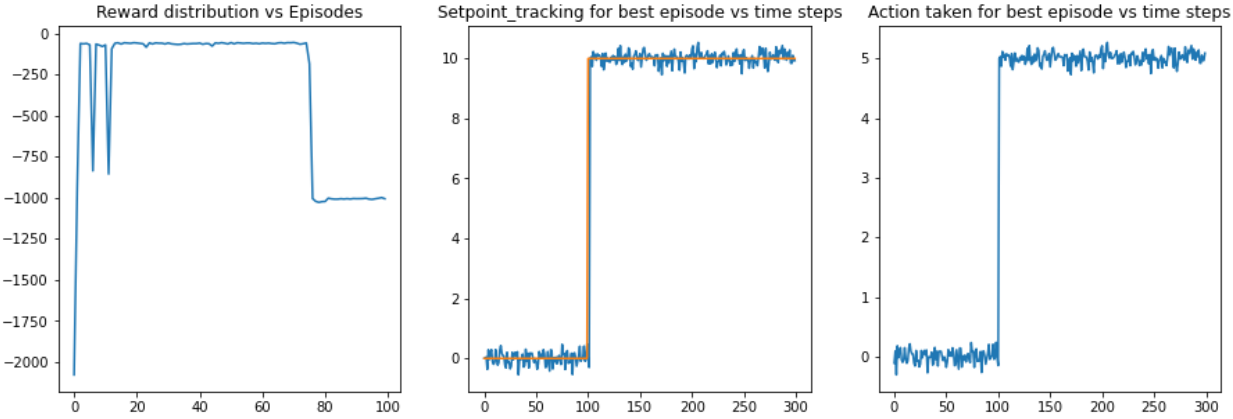
Setpoints [0,5,10], Best Episode: 50, Best Reward: -98.66885422503223



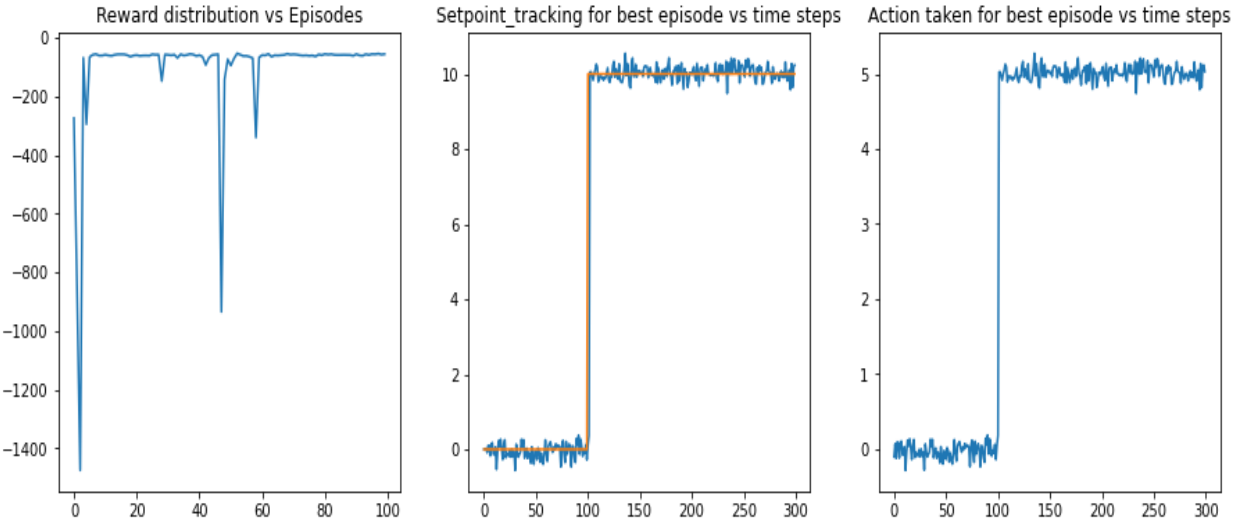
Setpoints [0,5,10], Best Episode: 98, Best Reward: -57.952956807986496



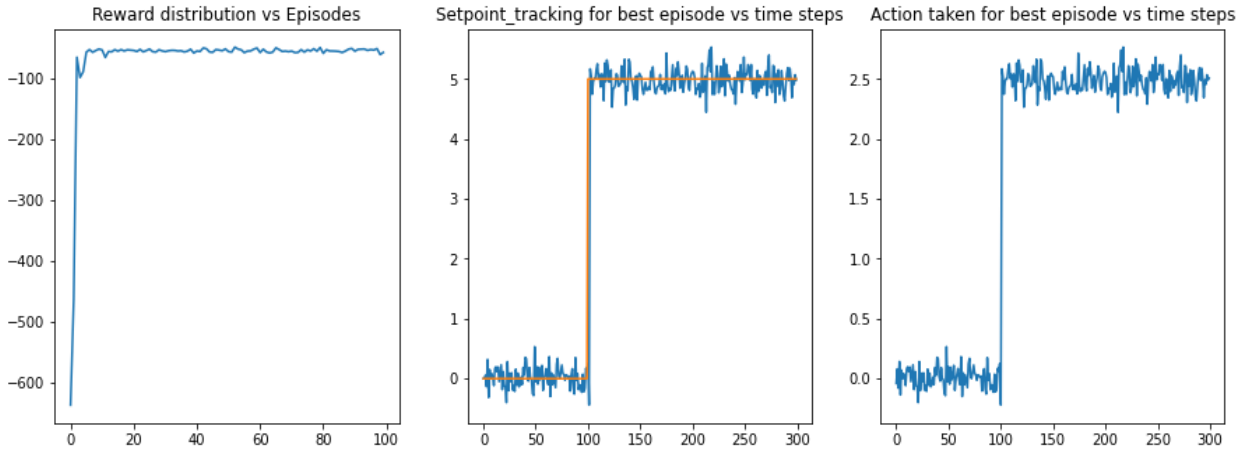
Setpoints [0,5,10], Best Episode: 71, Best Reward: -59.20750313210823



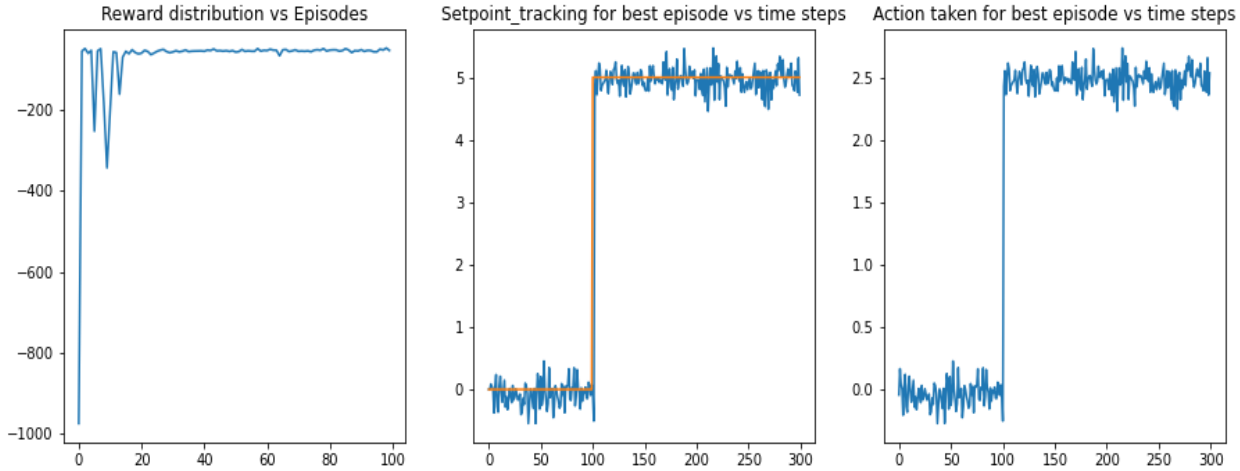
Setpoints [0,5,10], Best Episode: 99, Best Reward: -57.30147964921747



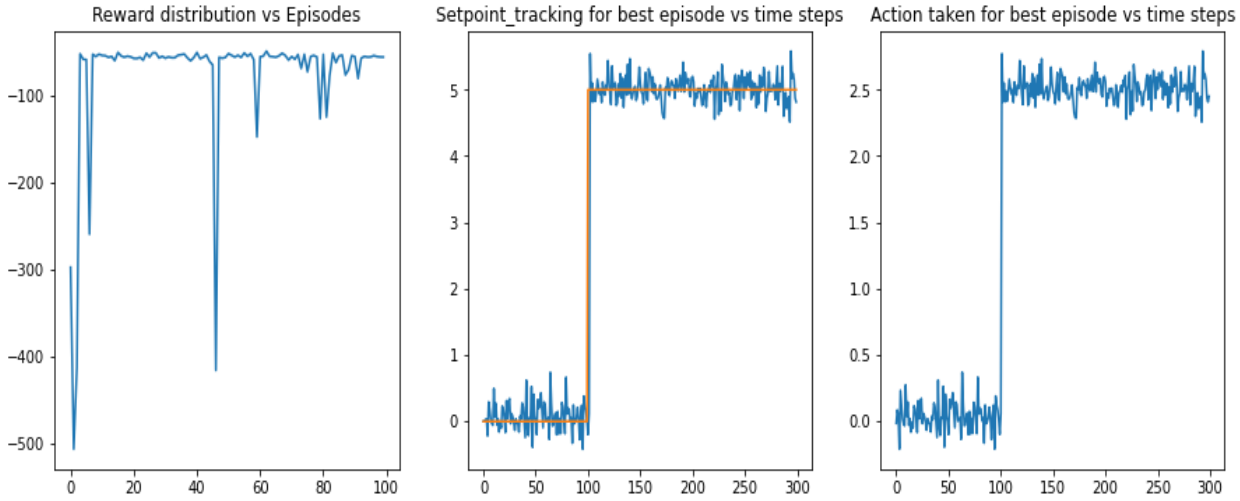
Setpoints [0,5,10], Best Episode: 97, Best Reward: -52.17015933360076



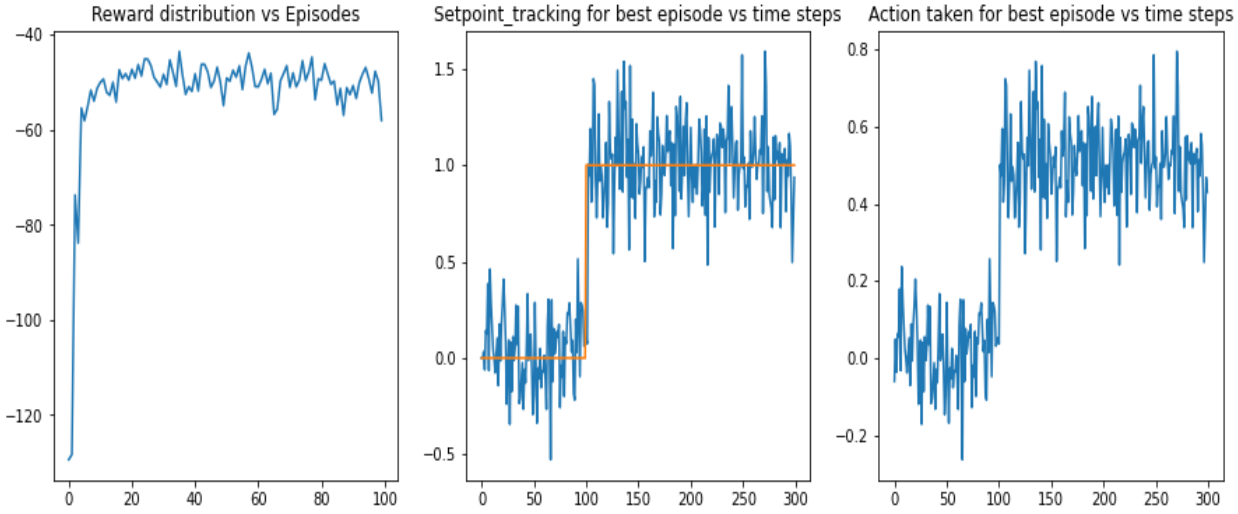
Setpoints [0,5,10], Best Episode: 85, Best Reward: -52.52343735380629



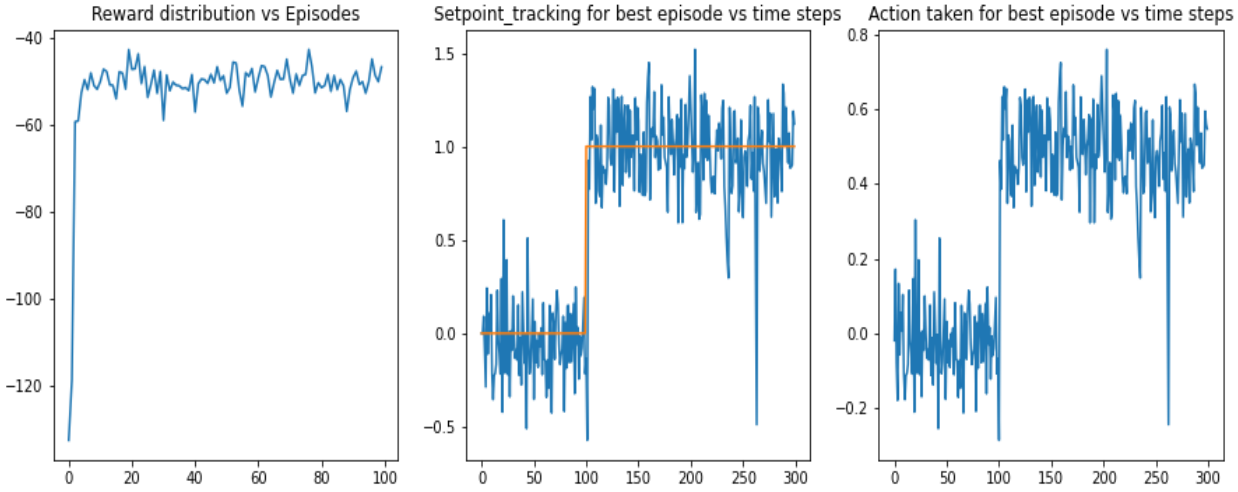
Setpoints [0,5,10], Best Episode: 57, Best Reward: -54.33359185672991



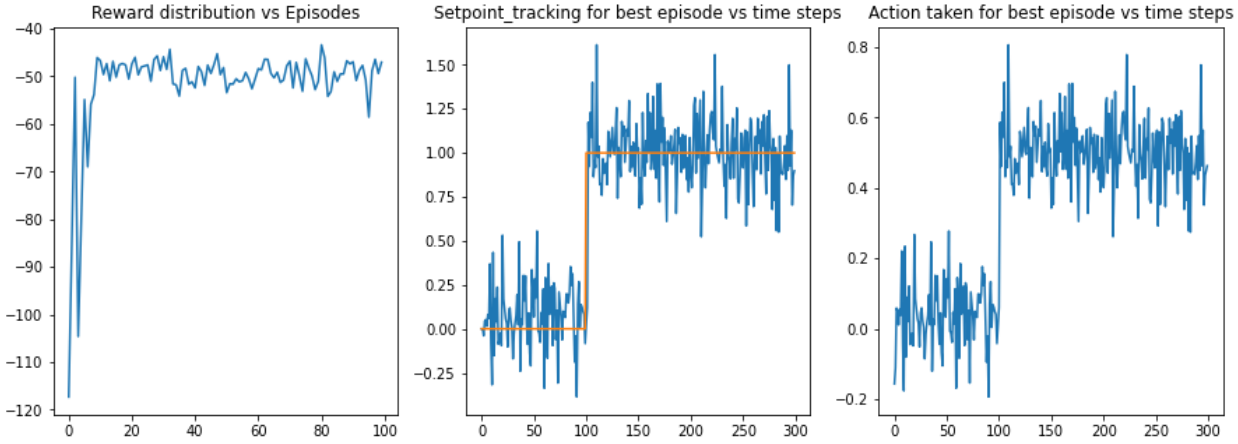
Setpoints [0,5,10], Best Episode: 27, Best Reward: -47.460329235261014



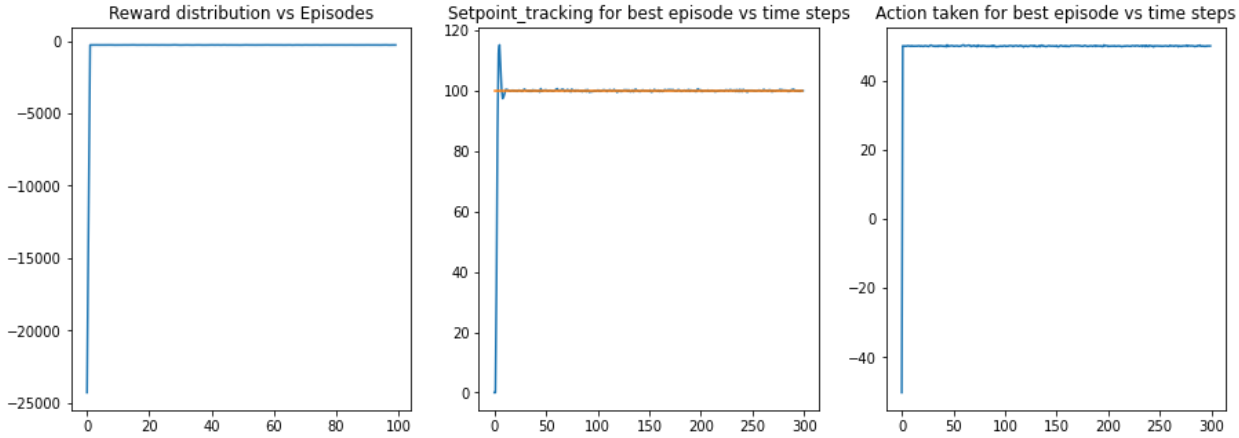
Setpoints [0,5,10], Best Episode: 25, Best Reward: -48.03143173092004



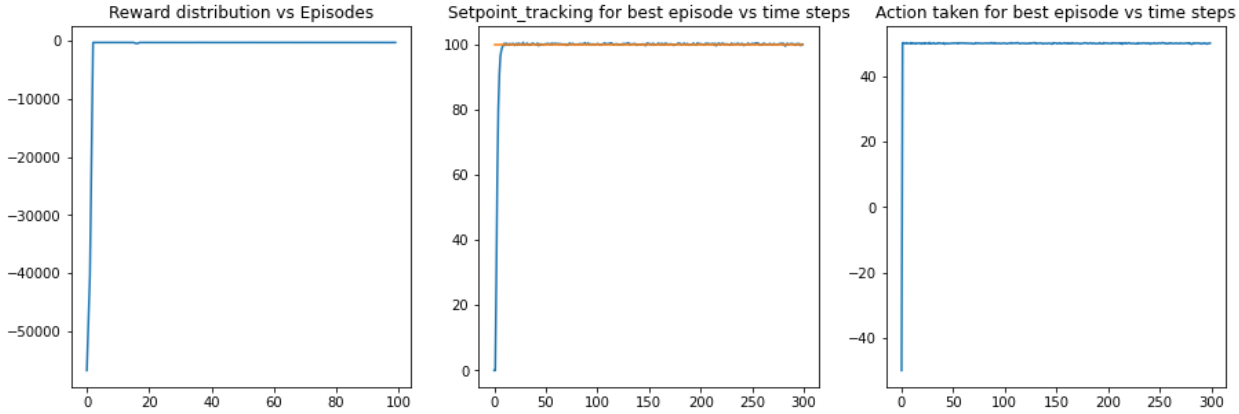
Setpoints [0,5,10], Best Episode: 32, Best Reward: -47.417500580837746



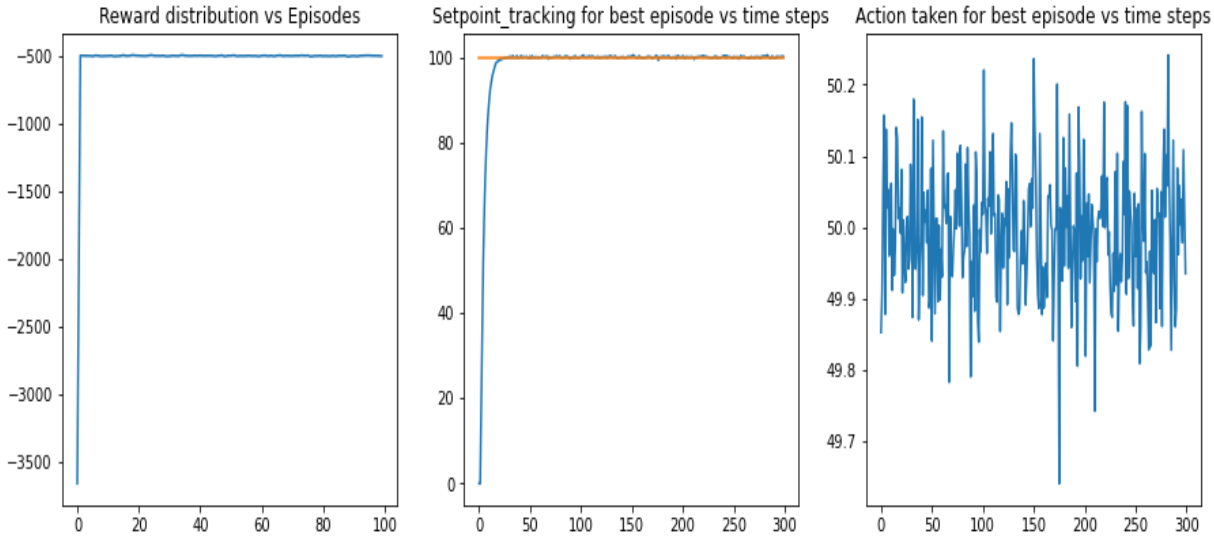
Setpoints [0,5,10], Best Episode: 21, Best Reward: -266.7848373993608



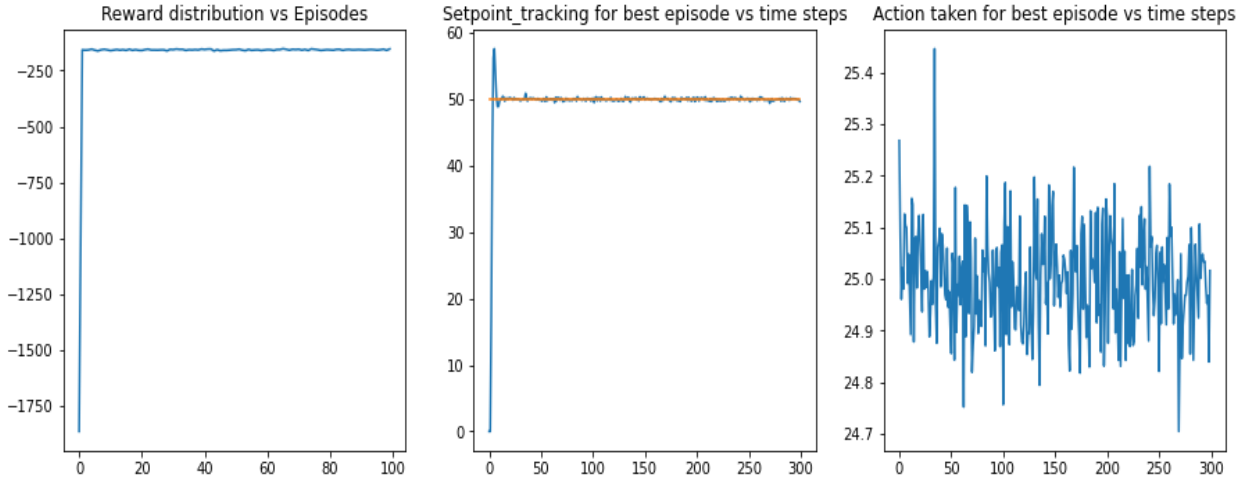
Setpoints [0,5,10], Best Episode: 71, Best Reward: -295.4826570284121



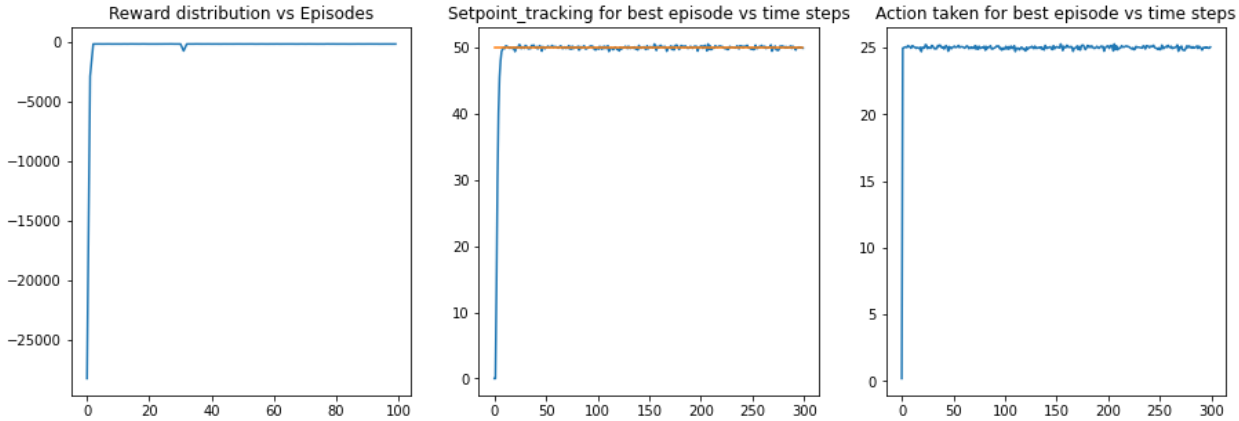
Setpoints [0,5,10], Best Episode: 24, Best Reward: -492.290565020725



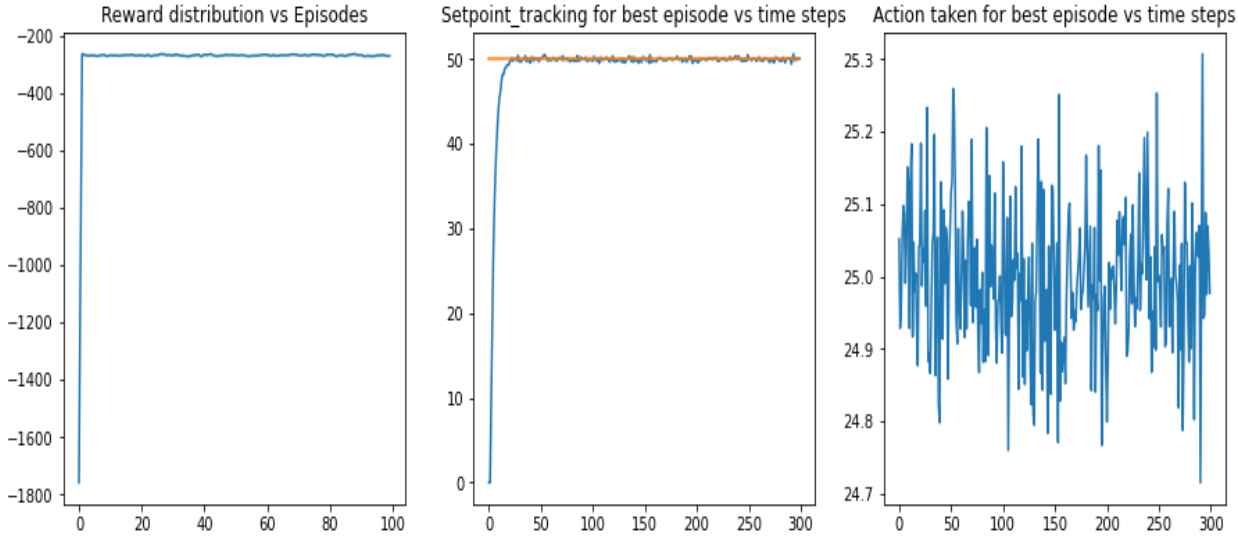
Setpoints [0,5,10], Best Episode: 72, Best Reward: -156.21251941717045



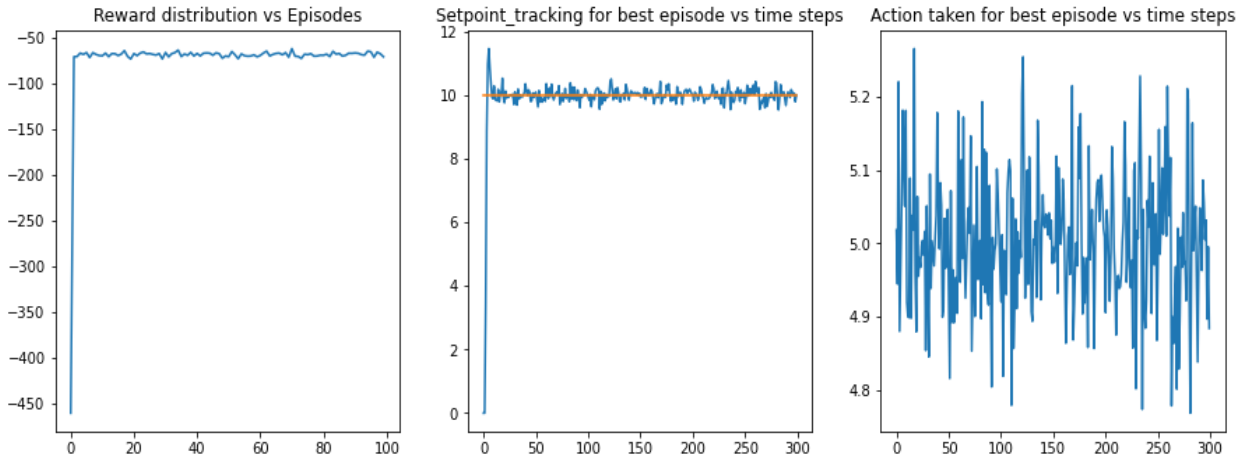
Setpoints [0,5,10], Best Episode: 19, Best Reward: -170.75231499707868



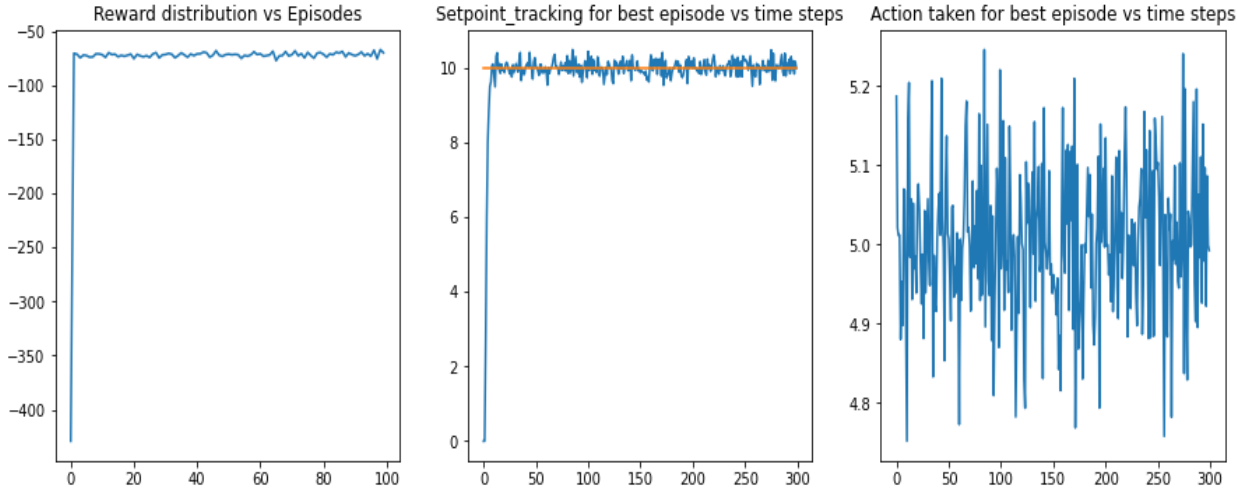
Setpoints [0,5,10], Best Episode: 77, Best Reward: -268.130063952926



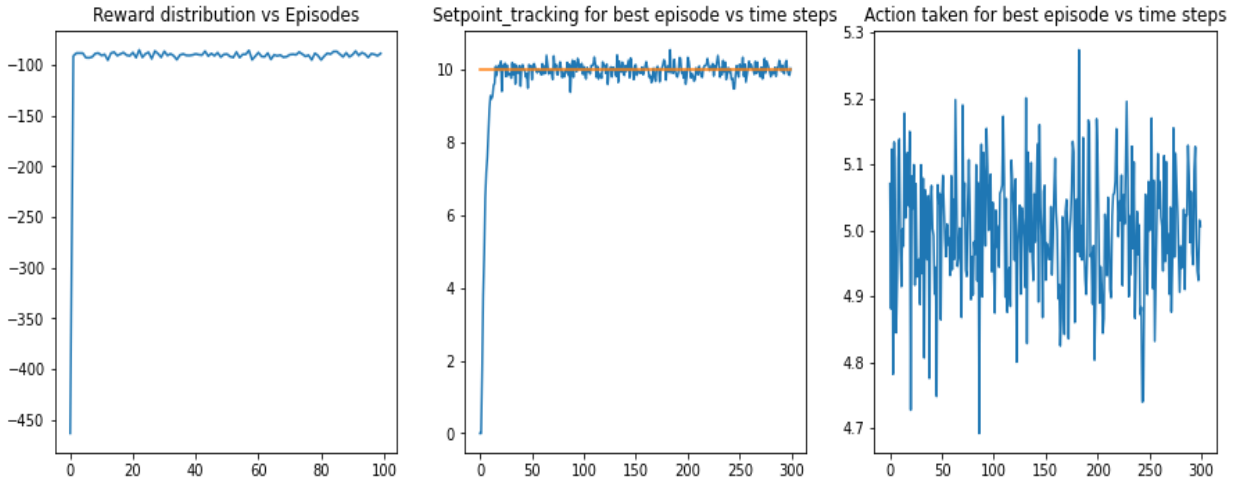
Setpoints [0,5,10], Best Episode: 70, Best Reward: -67.5725268250301



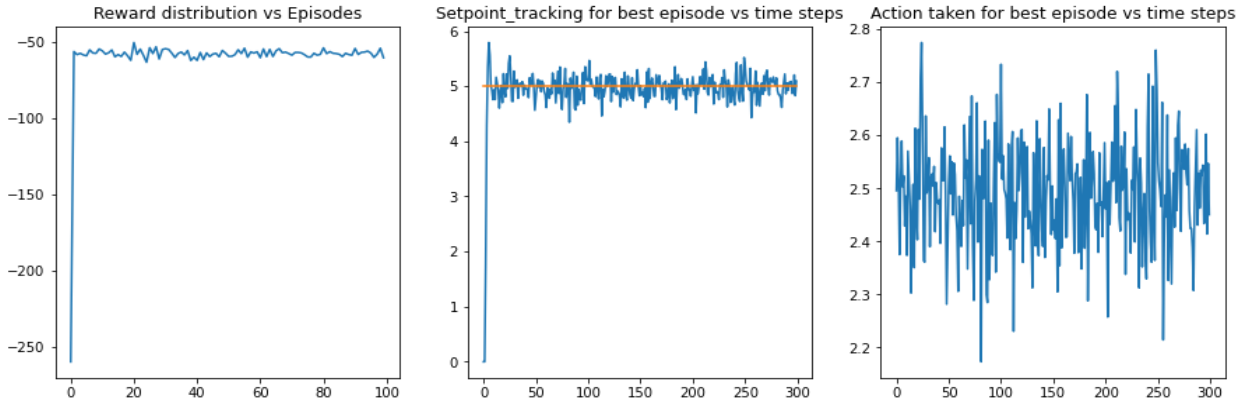
Setpoints [0,5,10], Best Episode: 92, Best Reward: -70.79846522895264



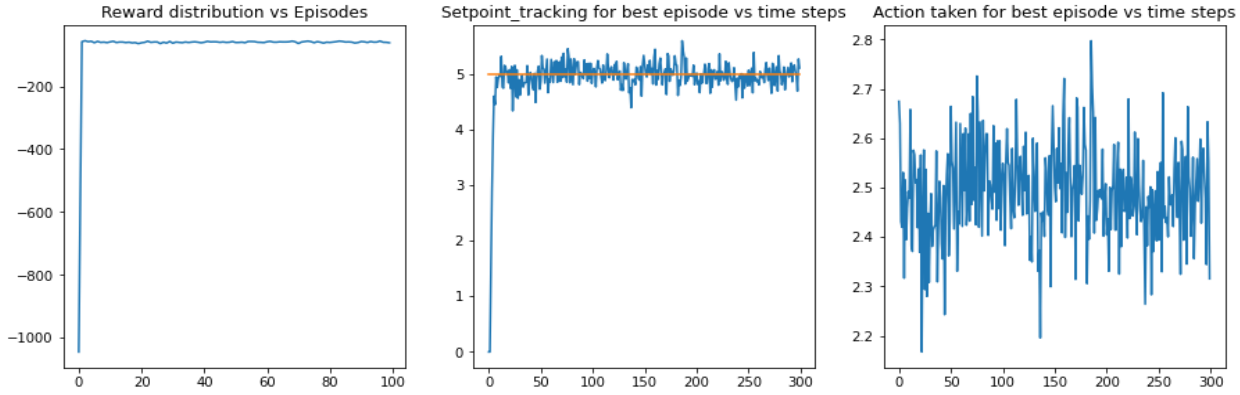
Setpoints [0,5,10], Best Episode: 91, Best Reward: -88.78529926193103



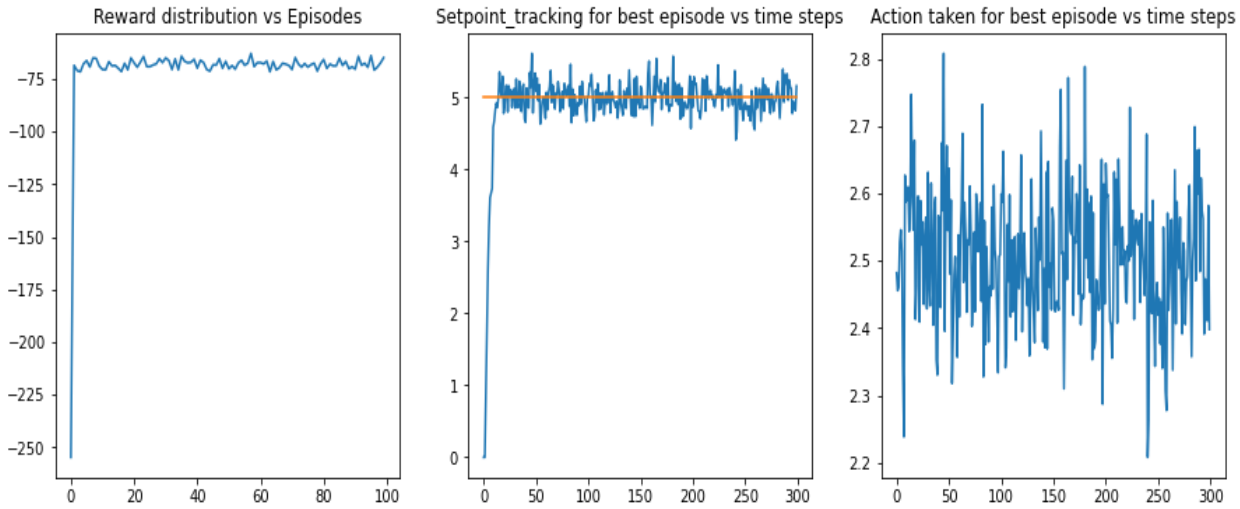
Setpoints [0,5,10], Best Episode: 34, Best Reward: -56.48531258954635



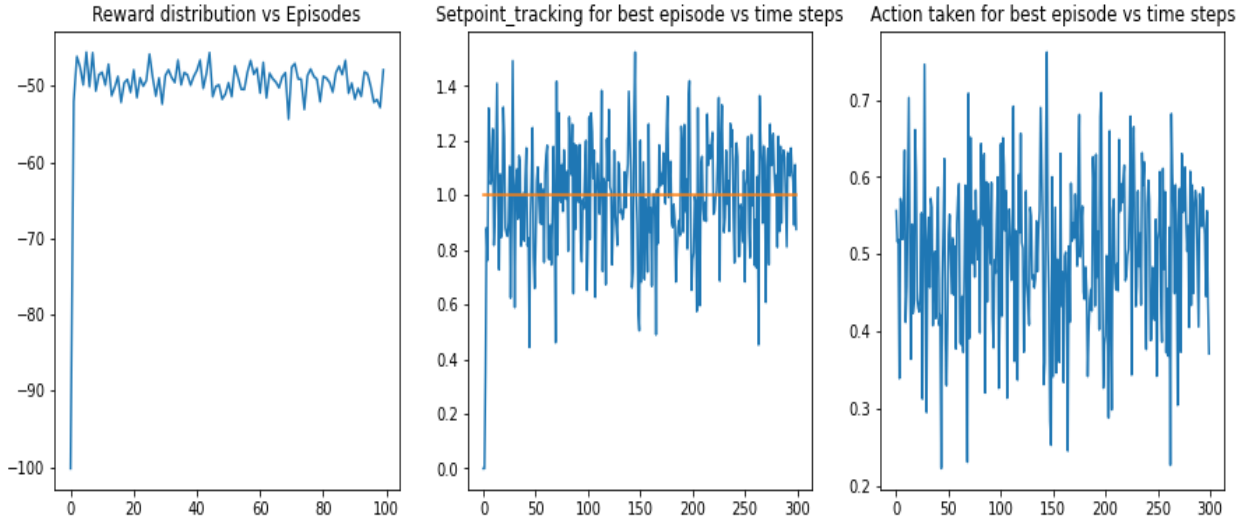
Setpoints [0,5,10], Best Episode: 69, Best Reward: -58.32713574844904



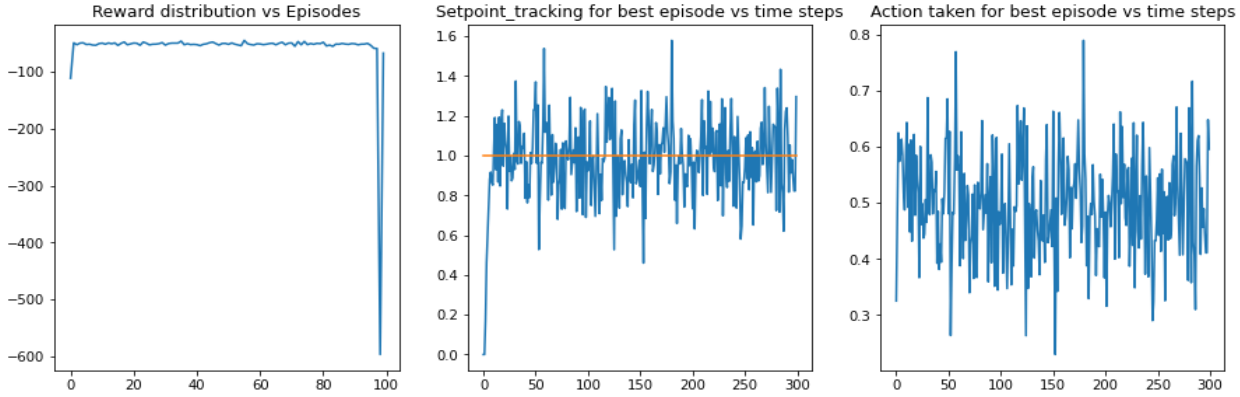
Setpoints [0,5,10], Best Episode: 37, Best Reward: -66.84004126411368



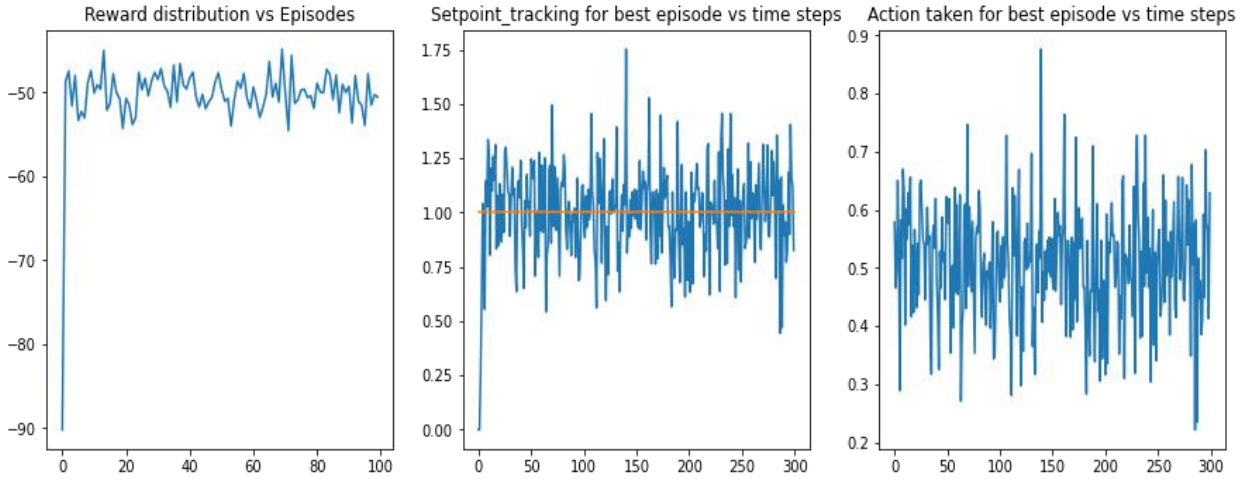
Setpoints [0,5,10], Best Episode: 11, Best Reward: -48.230393793732375



Setpoints [0,5,10], Best Episode: 35, Best Reward: -49.969947068055355



Setpoints [0,5,10], Best Episode: 33, Best Reward: -48.713557782495045



Future Work:

1. Extend this work for MIMO and practical use case systems.
2. Explore Soft-Actor Critic Algorithm in detail, which can serve as an even better substitute to TD3 Algorithms in terms of Reinforcement Learning algorithms, and can come close to beating MPC Algorithm.
3. Explore Process Changes within a given episode, and check for improvements further down the episode.