# 6.7950 Final Project Report
# Deep deterministic policy gradients applied to control of web-conveyance systems

Aniruddha Deshpande
ani0203@mit.edu

## Abstract

*Web-conveyance systems form a part of many industrial processes and the control of their lateral motion is a problem of interest. Many commercial processes use simple PID control to keep lateral web-motion in check however a major drawback of this kind of approach is that control is applied only after an error has occurred. Classical LQR has also been applied to the problem, but it assumes an idealised process model where speed and tension are assumed to be constant which is not true in real systems. In real systems another added complication is that all the states of the system are not observed and are accessed only through some noisy measurements. The traditional way of solving this problem is to first do state estimation (eg. Kalman filtering) using some knowledge of the underlying dynamics and then applying LQR to the estimated states. In this project I investigate applying Deep deterministic policy gradients [4] for the first case where the state-information is fully known. I trained the model for sinusoidal and step-input disturbances of a single frequency/duration and tested these models on sinusoidal inputs of varying frequency and step inputs of varying durations. I found that the DDPG model trained on sinusoidal performed significantly better than the one trained on step inputs and even outperformed traditional LQR control in certain test examples. I also studied and implemented the Recurrent deep policy gradient model for the partially observed case, however I did not manage to train the model successfully so far.*

## 1. Introduction and Related Work

Web-conveyance systems appear in several industrial processes such as packaging manufacturing, printing. It involves a long flat sheet (called the web) that propagates along the machine direction, while being supported by rollers at regular intervals. Typical conveyance systems will have multiple such web-segments between roller pairs. In an ideal scenario the web moves only in the machine direction, but there is usually some web-motion in the lateral direction as well. Too much lateral motion leads to inferior product quality downstream and hence there is a need to develop a control strategy to ensure that lateral movement of the web along the entire length of multiple spans is minimised.

Many industrial systems use simple PID control for this problem. However, a downside of this approach is that control action is applied only after an error has occurred. Classical LQR control has also been applied on this problem as well. [3] [2] [6] derive variations of physically motivated models of multiple span web-conveyance systems, where the web is assumed to be a beam and the model is discretized to give a state-space formulation which is then used to derive optimal gain matrices for control. A drawback of these methods however is that they all assume the speed and tension of the web to be constant during operation. Observations (specifically in the process of the client that I'm working on this project for) reveal that this assumption doesn't always hold true and there is significant variations in both speed and tension even during normal operation. Another layer of complication added in real systems is that the full-state of the conveyance system is rarely known, and is accessible only through a few noisy sensor measurements at different locations. The traditional approach of doing this is by performing some kind of state-estimation (eg. Kalman filtering [8]) using the observations and knowledge of the underlying process dynamics and then using the best state estimates to obtain the control action. Drawbacks of this approach are similar to before in that tradition models assume constant speed and tension. These imperfections in known models is what motivated me to apply reinforcement learning to this problem. The approach to this problem would be in two steps; first to learn control for the case where full state information is known and second when we have only noisy measurements. In this project I worked on learning a control policy for the full state information case.

For the full state-information case I experimented with applying deep deterministic policy gradients [4] , a model-free actor critic method that is developed to act over continuous action spaces. It has been applied and tested by the authors on a variety of traditional control tasks such as

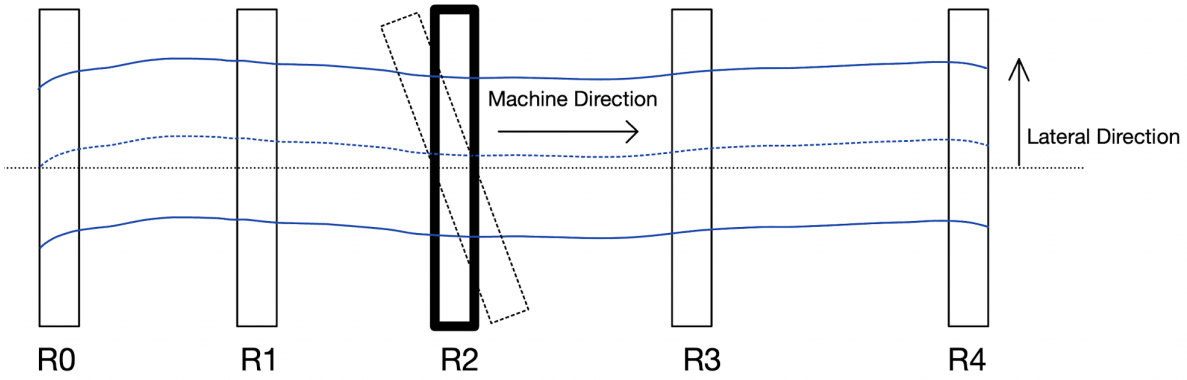



Figure 1. 5-roller pair system in consideration. The lateral position and angle made by the web at roller 0 is the measured input disturbance to the system. Roller 2 is the actuated roller, and can move in and out of plane as seen by the dotted line. The control action is introduced through rhis roller

cartpole, locomotion etc and found to be competitive with traditional approaches with full access to the dynamics.

For the partially observed case, the problem is modeled as a POMDP. The optimal policy for a POMDP would be to use the full history, which we know is too computationally inefficient. I studied and implemented the recurrent deep policy gradient approach from [1] for this problem. The approach is very similar to the DDPG approach , the major difference being the use of LSTM networks to parametrize the actor and critic. The idea is that the memory and model structure of LSTMs will allow for retention of relevant aspects of the history required to derive a good control policy. However, I had trouble getting the model to train well sso far.

In this project I apply the methods on a simulated environment of a web-conveyance system. I compare the performance of these approaches with the traditional LQR control.

## 2. Simulation environment

All the experiments in this project are performed on a simulated environment of a 5 roller-pair system as show in figure 1. It consists of 4 web segments supported by 5 rollers. Roller 2 is the actuated roller, through which the control is applied. It has the ability to move in and out of the plane thereby influencing the lateral position of the web. I use the beam-bending web-model as derived in [2] [6] as my environment. The discrete model of the lateral web dynamics of this system takes the following state-space form.

$$X_{t+1} = AX_t + Bu_t^{control} + Fu_t^{dist} + w_t \qquad (1)$$

where, the state-vector $X_t \in \mathbb{R}^{10}$ the control input $u_t^{control} \in \mathbb{R}^1$ and the measured external disturbance $u_t^{dist} \in \mathbb{R}^2$ are given by:-

$$X = [\, y_1 \quad \frac{dy_1}{dt} \quad y_2 \quad \frac{dy_2}{dt} \quad y_3 \quad \frac{dy_3}{dt} \quad y_4 \quad \frac{dy_4}{dt} \quad z_2 \quad \frac{dz_2}{dt}\,]^T$$

$$\mathrm{u}^{control} = \frac{d^2 z_2}{dt}$$

$$\mathrm{u}^{dist} = [\, y_0 \quad \theta_0]^T$$

$y_i$ refers to the lateral web-position at the $i^{th}$ roller and its derivative is the lateral velocity. $z_2$ refers to the displacement of the actuated roller, roller 2 in the plane perpendicular to the web. The acceleration in this direction to this roller is the control input to the system. $\theta_0$ is the angle made by the edge of the web with the roller at roller 0. This is often too small to be measured and will be set as 0 in the simulation.

The state-space model used here is slightly different from the usual (as well as the form we studied in class) in that it has 2 kinds of inputs, control inputs as well as measured external disturbances. $u^{dist}$ in the setup are measurements of lateral position and web entry-angle at roller 0. These are treated as inputs to the system which are measured by sensors, but cannot be directly controlled.

As mentioned before, this model assumes constant speed and tension. The speed and tension values are used in the derivation of the matrices $A, B$ and $F$. In order to simulate the variations in speed and tension, I derived 40 such matrices $A_i, B_i$ and $F_i$ for different values of speed and tension about the nominal values. Then, in simulating the environment at each step I cycle through different sets of the matrices, thereby having a slightly different update at each step. I also add iid random normal noise to the state-evolution at each step.

When resetting the environment, I have two kinds of resets. The first drives all the state-values to zero. The second,

which I use for training randomly samples values between -0.005 and 0.005 for each state.

For the measured input disturbance to the system I use 2 different kinds of inputs, a step-input and a sinusoidal variation , both in the lateral position measured at roller $y_0$

The cost (negative rewards) for the environment at each step is the quadratic cost

$$c(t) = X_t^T Q X_t + (u_t^{control})^T R u_t^{control} \quad (2)$$

I use $R = I_1$ and $Q$ as a diagonal matrix with all entries being equal to 1, except for the $Q_{3,3} = 10$ and $Q_{5,5} = 1000$ which I found worked well for the LQR controller. By working well, I mean that these values were able to control the lateral motion of the last roller (roller 4) pretty well using LQR control.

## 3. Baseline methods

### 3.1. Extended Linear Quadratic Regulator (ELQR)

The baseline approach I compare performance against is the extended LQR controller. This is slightly different from the LQR controller we studied in class. This difference comes from the fact the state-space model of the system in this case has an additional set of measured but uncontrollable inputs. [7] re-derives the optimal quadratic controller for this variation of a state-space model. The optimal control signal is given by :-

$$u_t^{*control} = -(KX_t + Gu_t^{dist}) \quad (3)$$

For the infinite-horizon case (which is what I will consider), the gain matrix $K$ turns out to be the same as that for regular LQR. The matrix G is given by :-

$$G = K(A - P^{-1}(P - Q))^{-1}F$$
$$P = Q + A^T[P - PB(R + B^T PB)^{-1}B^T P]A$$

## 4. RL methods

### 4.1. Deep deterministic Policy gradients [4]

For the full-state information control problem I use the deep deterministic policy gradients algorithm introduced in [4]. It is an off-policy actor critic method. This works by maintaining 4 different neural networks, an actor network $\mu(s|\theta^\mu)$, a critic network $Q(s, z|\theta^Q)$ a target actor $\mu'(s|\theta^{\mu'})$ and a target critic $Q'(s, z|\theta^{Q'})$. [4] and [5] find that the presence of this target networks helps stabilize the training. These target networks slowly track the weights of the network during training; $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ with $\tau << 1$. As discussed in lecture 10, this serves to alleviate the issue of chasing a "moving target" by varying the targets slowly.

Being an off-policy algorithm, it also uses a replay buffer which in this case stores transitions $(s, a, r, s')$. For each update a mini-batch is sampled randomly from this buffer. It serves the purpose of de-correlating the samples and adding data diversity for training.

The algorithm also adds exploration by adding a random noise to the action policy. The scale of the noise is chosen appropriately based on the scale of the output.

The state-vector used for training here is the concatenated vector $[X_t \quad u_t^{dist}]$, ie. the networks take as input the system state as well as the incoming measured disturbance.

The complete algorithm is Algorithm 1 given in [4]

### 4.2. Recurrent Deep policy gradients [1]

For the partially observed case I planned to experiment with the RDPG network as given in [1] . The algorithm is very similar to the deep deterministic policy gradients algorithm. Some key differences however are the following. First, it uses recurrent neural networks/LSTMs in-place of regular feedforward neural networks for both the actor and the critic networks. This serves the purpose of keeping track of the relevant history required to best internally estimate the true states of the system from the observed trajectory.

Like DDPG this is also an off-policy learning algorithm and uses trajectories stored in the experience replay buffer to the actual network update. A key difference from DDPG is that it samples entire episodes $(o_1, a_1, r_1 ... o_T, a_T, r_T)$ and not just individual transitions because of the recurrent network parametrization. The BPTT backpropagation through time algorithm is used to train the weights of the network.

The complete algorithm is algorithm 1 in [1]. Unfortunately, I was unable to get this algorithm working in time, but intend to experiment with it going forward.

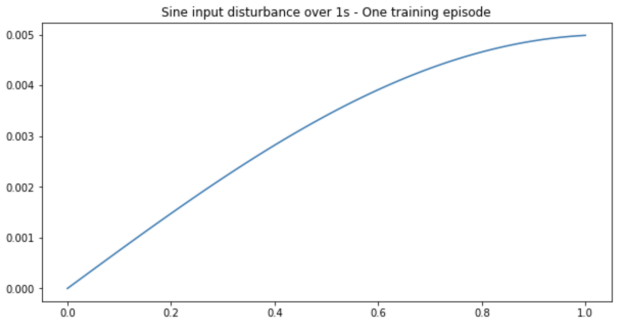
## 5. Evaluation metrics

I will evaluate the performance of the different control strategies for both sine waves of varying frequency and step-response of varying duration as input disturbances to the system. To compare the performance of the different control strategies I will do the following:

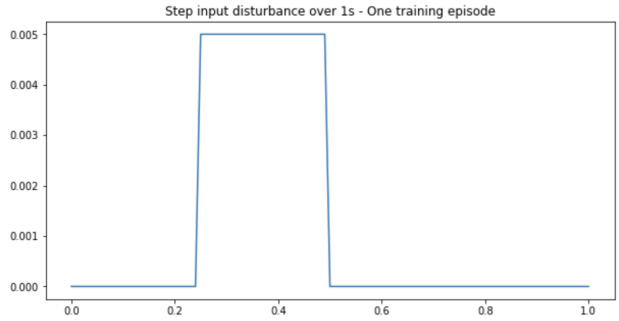
- Plot state evolution for each control strategy
- Compare accumulated rewards averaged over 20 episodes (each initialized randomly) for given Q and R matrices

## 6. Implementation details and Experiments

Standard feedforward networks are used to parametrize the actor and critic networks. For the actor network I used




(a) Sin input disturbance




(b) Step input disturbance

Figure 2. Input disturbances used for training the (1) DDPG-sin (2) DDPG-step model over 1 training episode length

a FCN with 1 hidden layer with 128 hidden units. The Q-network was a FCN with 2 hidden layers of dimension 128. For the Q-network the action is introduced only at the second hidden-layer. The target actor and critic networks are identical to the original networks. I used a learning rate of $10^{-3}$ for the critic and $10^{-4}$ for the policy network. For the update of the target networks used $\tau = 10^{-3}$. During training I used the Ornstein-Uhlenback process as exploration noise as used in [4]. This is a process that is used to model the movement of a particle under friction and is correlated in time. It is given by :-

$$x_{t+1} = x_t + \theta(\mu - x_t) * \Delta t + \sigma\sqrt{\Delta t}\epsilon$$

where $\epsilon$ is zero mean unit variance random noise, $\Delta t$ is the sampling period for discretization. I used $\theta = 0.15$ , $\mu = 0$ , $\sigma = 1$. These are the same parameters for the exploration process as used in [4]. I found that this exploration noise led to better policies being found by the agent as compared to using uncorrelated white noise for exploration. I also started to decay the scale of the noise after 20 iterations in order to gradually keep reducing exploration.

The input states used for the RL formulation are the states $X_t$ concatenated with the measured input disturbance $u_t^{dist}$. I scaled the inputs to the networks to have a maximum/minimum value of +1/-1 before inputting it to the network. This played a significant role in ensuring the network actually trained.

For training I used episodes of length 100 (1 second) and trained for 100 episodes. I found that the agent trained better for these shorter length episodes as compared to longer training episodes. I believe that the reason for this is that, in the beginning when the model is still exploring , if the episode is long the states blow up, which leads to accumulation of bad data in the replay buffer, which is eventually used for training. Keeping the episode shorter meant that even in the case where the control action wasn't well-chosen the states still remained within reasonable bounds. I trained the model for 100 training episodes.

For the evaluation I compare the performance of two DDPG agents that I got to converge to some reasonable policies. The first, I trained for a an input sin(t) disturbance and the second one for a step-input of duration 0.25s. While training each of them, I alternated between giving a positive and negative disturbance in alternate training episodes. This seemed to give better performance on longer test episodes as opposed to training without alternating the disturbance signs. My hypothesized reason for this observation is the following: see figure 2 of the disturbance given for the training episodes in each case. Since the training episode is only 1s long, the sign of the disturbance is only positive in this duration, but in the test set of a longer duration the disturbance becomes negative as well. During training we want to allow the agent to explore that part of the state space where the input disturbance is negative as well, so it learns a good policy for that state too. Alternating the signs of the disturbance during training lets us expose the agent to this situation effectively while still keeping the length of the episodes small (for the reason mentioned before).

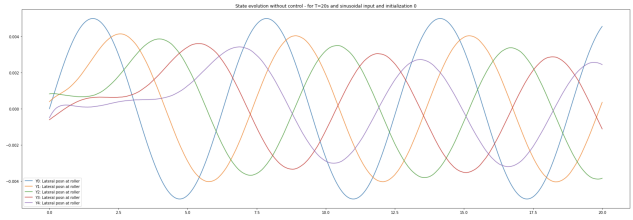
I tested both these models on 6 different input disturbances [1] sin(t) [2] sin(1.5t) [3] sin(2t) [4] Step - 0.25s [5] Step - 0.5s [6] Step - 1s. For each of these I test it for 20 different episodes and average the episodic rewards for comparing performance of the different agents.

Details of the implementation can be found in the colab notebook https://tinyurl.com/ytshy6bw .

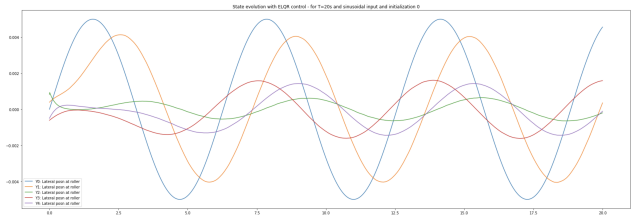
## 7. Results and Discussions

Table 1 shows the performance of (1) ELQR (2) DDPG-sin (3) DDPG-step on the different test input trajectories in terms of mean episodic reward averaged over 20 episodes. Figures 3 4 5 6 shows plots test episodes without control, and with the different control strategies tested.

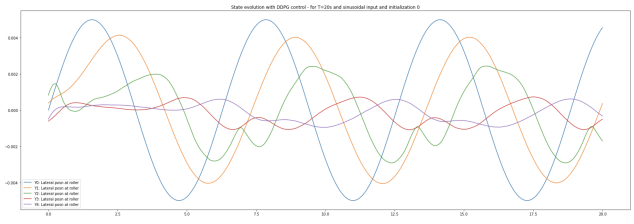
Few things to note here: firstly that the performance of the DDPG agent trained on the step-disturbance input is universally worse than the performance of DDPG agent trained on a sin input. It also performs worse than the ELQR controller with an imperfect system model. I believe the reason for this is that the sinusoidal disturbance adds more 'excita-



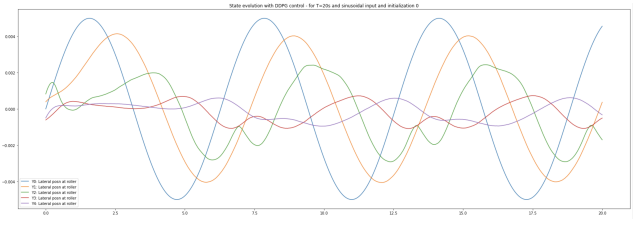
(a) No control



(b) ELQR

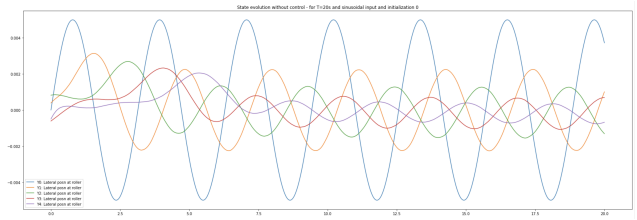


(c) DDPG-sin

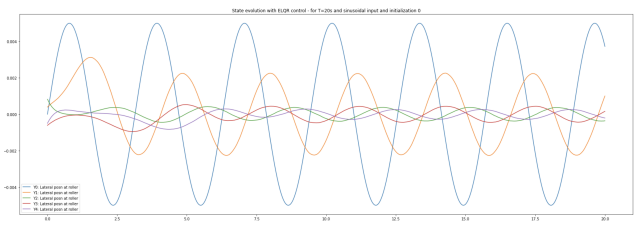


(d) DDPG-step

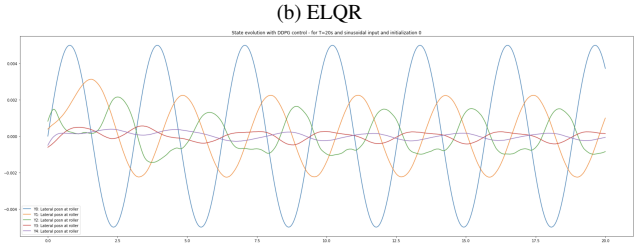
Figure 3. State-evolution for input disturbance = sin(t) and different control strategies



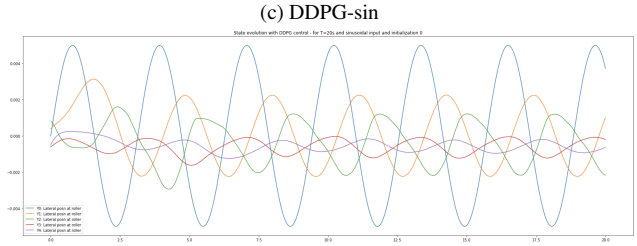
(a) No control



(b) ELQR



(c) DDPG-sin



(d) DDPG-step

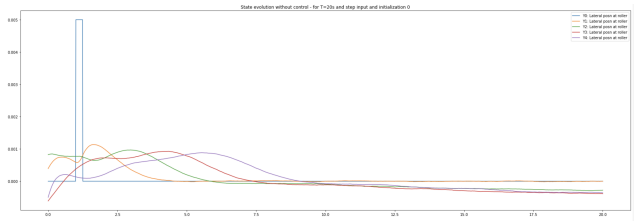
Figure 4. State-evolution for input disturbance = sin(2t) and different control strategies

tion' to the system thereby driving it to more states during the training. This essentially means that giving a sinusoidal disturbance to the system during training is better for exploration. This is just a hypothesis, and I plan to systematically verify this going forward.

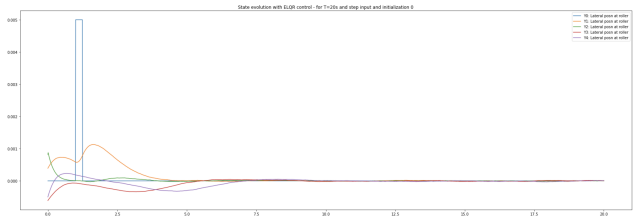
The DDPG-sin agent does not always perform better than the ELQR model as can be seen in the table. However, it does perform better in all the test cases where a sinusoidal input is given and the last step input of a 1 second duration. An interesting observation here is that the DDPG-sin agent actually performs relatively better in the test cases where control is 'harder'. For the step-inputs , a longer step-input is harder to control as it introduces more disturbance to all the states (see figures 5 6). Similarly, for the sin inputs , a higher frequency disturbance actually disturbs the states of the system lesser than one with a smaller frequency (figure 3 4 ). From the table it can be seen that the relative improvement of the DDPG controller over ELQR decreases with increasing frequency.

| | | Control strategy | | |
|---|---|---|---|---|
| | | ELQR | DDPG-sin | DDPG-step |
| Input disturbance | sin(t) | $1.35 \times 10^{-3}$ | $9.08 \times 10^{-4}$ | $1.55 \times 10^{-3}$ |
| | sin(1.5t) | $6.20 \times 10^{-4}$ | $4.32 \times 10^{-4}$ | $1.08 \times 10^{-3}$ |
| | sin(2t) | $3.54 \times 10^{-4}$ | $3.50 \times 10^{-4}$ | $8.59 \times 10^{-4}$ |
| | step-.25s | $2.15 \times 10^{-4}$ | $2.66 \times 10^{-4}$ | $1.27 \times 10^{-3}$ |
| | step-.5s | $2.27 \times 10^{-4}$ | $2.64 \times 10^{-4}$ | $1.26 \times 10^{-3}$ |
| | step-1s | $2.74 \times 10^{-4}$ | $2.71 \times 10^{-4}$ | $1.26 \times 10^{-3}$ |

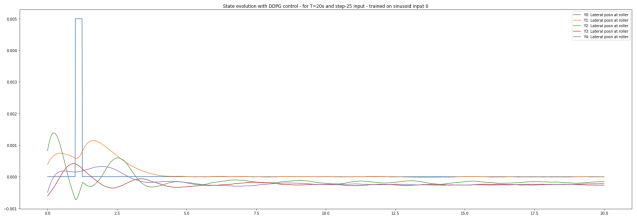
Table 1. Evaluation metrics of ConvLSTM and U-Net model outputs over test set across different thresholds



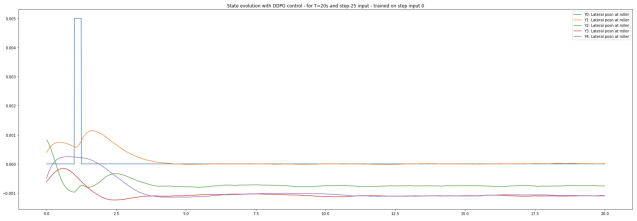
(a) No control



(b) ELQR



(c) DDPG-sin

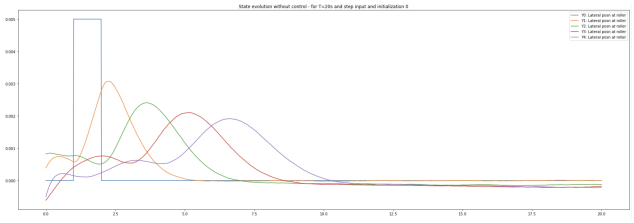


(d) DDPG-step

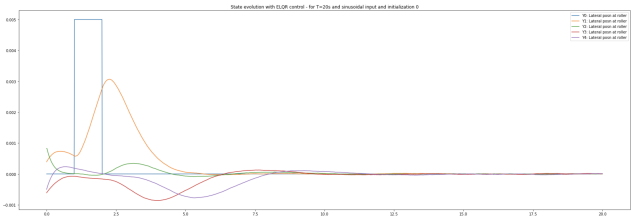
Figure 5. State-evolution for input disturbance = step(0.25s) and different control strategies

## 8. Conclusion and Next steps

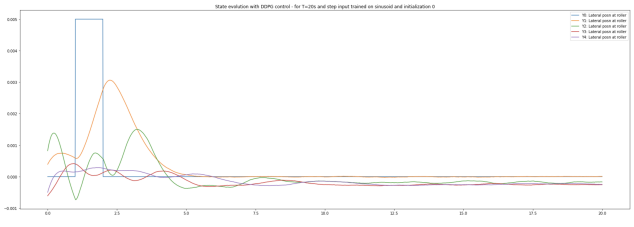
Deep deterministic policy gradients was introduced as an RL approach for continuous control problems specifically and seems to perform well for web-handling in simulation. The model trained on sine input disturbance performs better than the one trained on step input disturbance. I believe that this work is a promising step in apply Deep-RL for control of the packaging manufacturing process I am working on in my research (web-conveyance forms a significant part of this system). The next step for me with this work would be to do more experiments comparing DDPG with other control methods and also adding increased variability to the environment (in terms of noise and increased speed/tension variation). After this I would like to work on the partially observed version of this problem using recurrent DPG. The last step (probably in the distant future) would be to test these approaches on the real system. I would love to hear feedback/suggestions on how I can improve on this work and take it forward!



(a) No control



(b) ELQR



(c) DDPG-sin

Figure 6. State-evolution for input disturbance = step(1s) and different control strategies

## References


[1] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015. 2, 3

[2] Zhao Jin. Application of repetitive control to the lateral motion in a roll-to-roll web system. Master's thesis, University of Waterloo, 2012. 1, 2

[3] CE Kardamilas and GE Young. Stochastic modeling of lateral web dynamics. In *1990 American Control Conference*, pages 474–480. IEEE, 1990. 1

[4] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 1, 3, 4

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 3

[6] Lisa Sievers, Mark J Balas, and Andreas von Flotow. Modeling of web conveyance systems for multivariable control. *IEEE Transactions on Automatic Control*, 33(6):524–531, 1988. 1, 2

[7] Abhinav Kumar Singh and Bikash C Pal. An extended linear quadratic regulator for lti systems with exogenous inputs. *Automatica*, 76:10–16, 2017. 3

[8] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995. 1