Deep Reinforcement Learning Applied to Dynamic Systems

Student(s)

Billy Toshi
Emre Yavaş
Mahmut Yasir Esmek
Muhtasham Oblokulov

Faculty Member(s)

Prof.Dr.Ahmet Onat

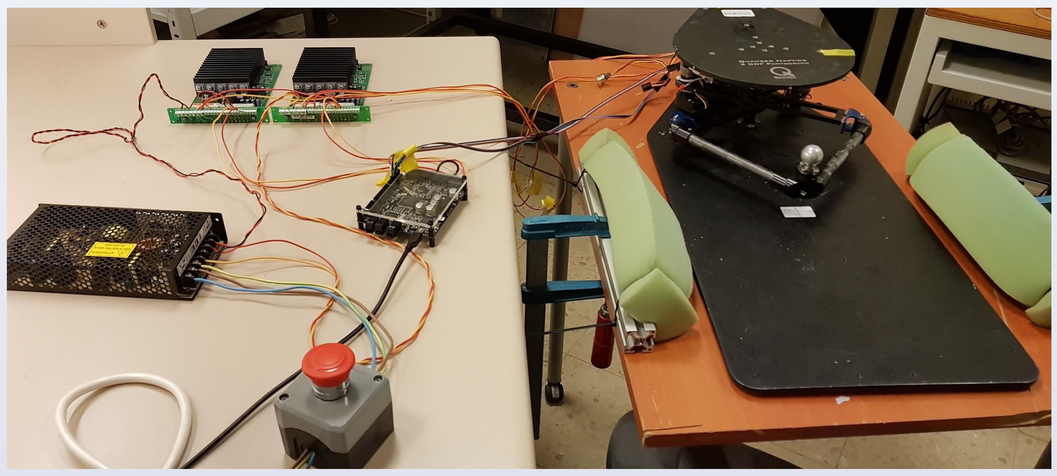
Assistant(s)

Vahid Tavakol
Arda Ağababaoğlu

Sabancı Üniversitesi

PURE
PROGRAM FOR UNDERGRADUATE RESEARCH

ABSTRACT



Deep learning is typically used for pattern classification on static datasets, such as face recognition from streaming video, based on prior experience. We are working on implementing deep learning methods in the framework of reinforcement learning, applied to dynamic systems, such as robot manipulators.

In this project we applied algorithms to a physical two d.o.f. pantograph in the lab. We port the algorithms to the robot controller (based on Matlab), running the algorithm and analyzing the data.

INTRODUCTION

Reinforcement Learning is a type of Machine Learning algorithms which allows software agents and machines to automatically determine the ideal behaviour within a specific states to maximize its performance. Deep Learning is a class of methods and techniques that employs artificial neural networks with multiple layers of increasingly richer functionality.

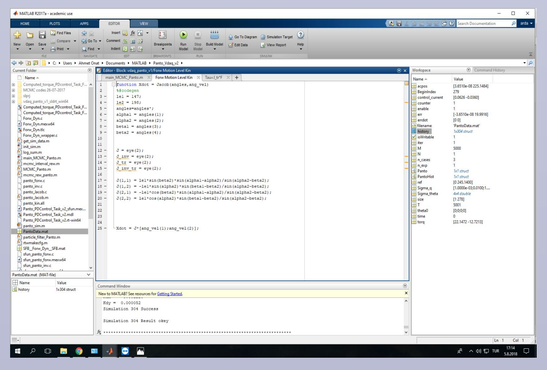
In industry and robotics, kinematics of the plant and the controller mechanisms are the main issues. In this project, we have the kinematics of the pantograph and we want to apply PID controls to it. However we do not know the PID values of the system and it is difficult to determine. Therefore we use deep reinforcement learning algorithms to find the appropriate PID values. At the end, we expect a method which learns to the control values of the robot, so that the robot will follow the motion paths that are supplied to it.

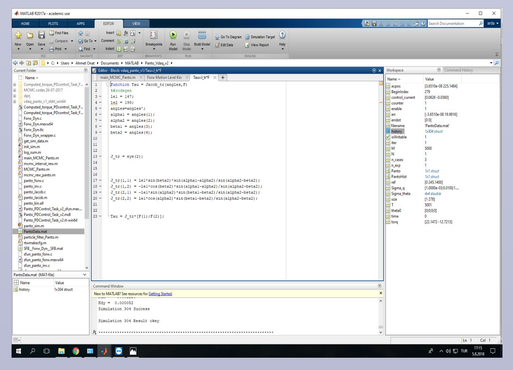
METHODS

Our methods mainly consists of 3 parts. Kinematics of the system, Matlab/simulink diagrams and ddpg learnig algorithm.

a) Kinematics of the System

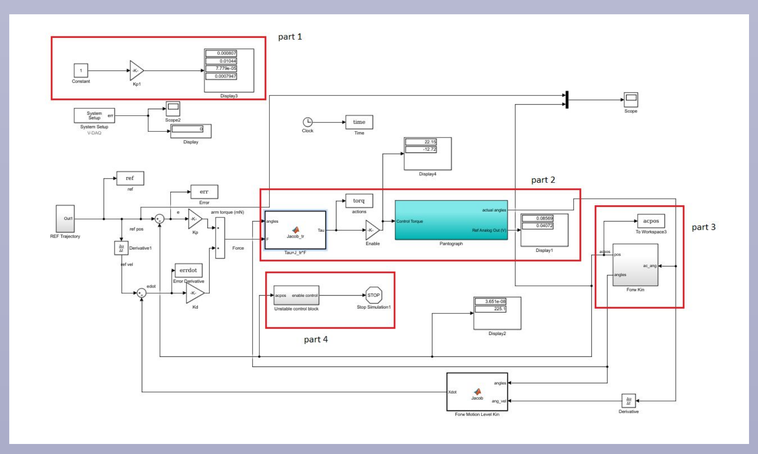
Kinematics of the system and the equations behind it was ready for us. The only thing to do for this part is just converting them to Jacobian Matrix and applying them into the Matlab.





b) Matlab/Simulink

Simulink diagram consists of many parts but, the most important ones will be explained below.

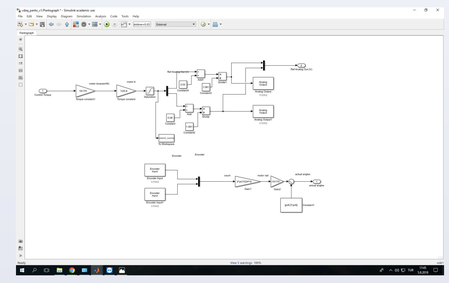


Part 1)

For each experiment, the values of the controller parameters are changed iteratively by the ddpg learning algorithm. This changed values are shown in this part.

Part 2)

From the reference part we can give a trajectory to the system. This reference is converted to Force and from feedback we have the angle of encoders. These force and angle values are converted to the Tork with jacobian matrix. Then this needed tork value is converted to the voltage value that we need to supply the system to track the trajectory in the blue box.

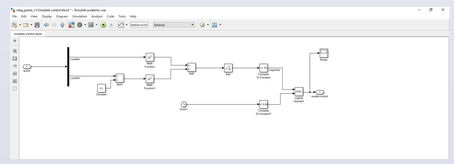


Part 3)

From blue box, we can also obtain the actual angle values of the encoders easily. With these angle values we calculate the actual position of the pantograph in this part.

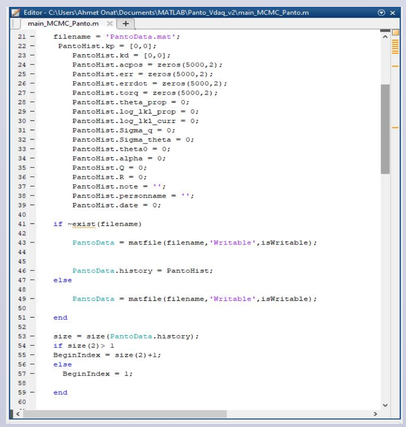
Part 4)

When system is working, system can behave unexpectedly. For sake of learning we need to prevent this behaviours. Simulink stops running when the pantograph exceeds specific borders in this part.

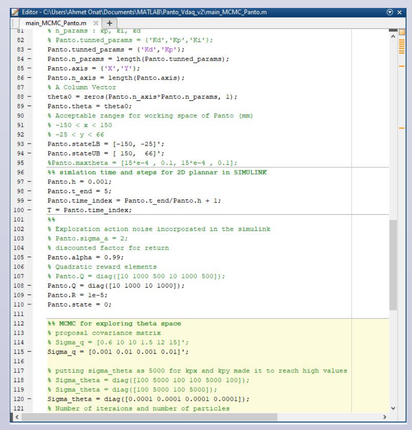


c) DDPG Algorithm

As we mentioned before, in the simulink part, the controller variables are determined by the ddpg algorithm iteratively and these variables applied on the simulink. Each time these variables are changed by code and rewards are recorded. Determining variables is the action and the error is our reward and algorithm learns the true values of the controller variables by recording the each action-reward pairs.



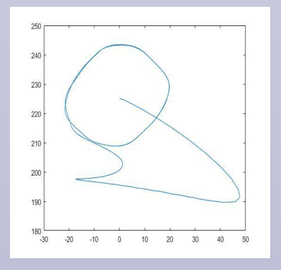
```
Editor - C:\Users\Ahmet Onat\Documents\MATLAB\Panto_Vdaq_v2\main_MCMC_Panto.m
main_MCMC_Panto.m
filename = 'PantoData.mat';
 PantoHist.kp = [0,0];
    PantoHist.kd = [0,0];
    PantoHist.acpos = zeros(5000,2);
    PantoHist.err = zeros(5000,2);
    PantoHist.errdot = zeros(5000,2);
    PantoHist.torq = zeros(5000,2);
    PantoHist.theta_prop = 0;
    PantoHist.log_lkl_prop = 0;
    PantoHist.log_lkl_curr = 0;
    PantoHist.Sigma_q = 0;
    PantoHist.Sigma_theta = 0;
    PantoHist.theta0 = 0;
    PantoHist.alpha = 0;
    PantoHist.Q = 0;
    PantoHist.R = 0;
    PantoHist.note = '';
    PantoHist.personname = '';
    PantoHist.date = 0;

if ~exist(filename)

    PantoData = matfile(filename,'Writable',isWritable);


    PantoData.history = PantoHist;
else

    PantoData = matfile(filename,'Writable',isWritable);

end

size = size(PantoData.history);
if size(2)> 1
BeginIndex = size(2)+1;
else
  BeginIndex = 1;

end
```

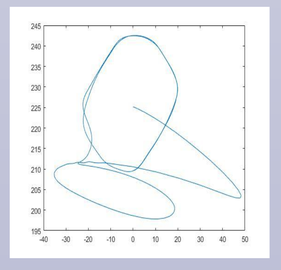


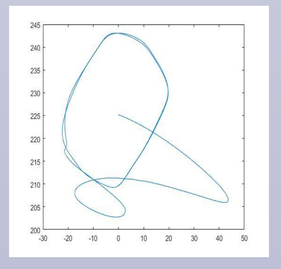
```
Editor - C:\Users\Ahmet Onat\Documents\MATLAB\Panto_Vdaq_v2\main_MCMC_Panto.m
main_MCMC_Panto.m
% n_params : Kp, Ki, Kd
% Panto.tunned_params = {'Kd','Kp','Ki'};
Panto.tunned_params = {'Kd','Kp'};
Panto.n_params = length(Panto.tunned_params);
Panto.axis = {'X','Y'};
Panto.n_axis = length(Panto.axis);
% A Column Vector
theta0 = zeros(Panto.n_axis*Panto.n_params, 1);
Panto.theta = theta0;
% Acceptable ranges for working space of Panto (mm)
% -150 < x < 150
% -25 < y < 66
Panto.stateLB = [-150, -25]';
Panto.stateUB = [ 150,  66]';
%Panto.maxtheta = [15*e-4 , 0.1, 15*e-4 , 0.1];
%% simlation time and steps for 2D plannar in SIMULINK
Panto.h = 0.001;
Panto.t_end = 5;
Panto.time_index = Panto.t_end/Panto.h + 1;
T = Panto.time_index;
%%
% Exploration action noise incorporated in the simulink
% Panto.sigma_a = 2;
% discounted factor for return
Panto.alpha = 0.99;
% Quadratic reward elements
% Panto.Q = diag([10 1000 500 10 1000 500]);
Panto.Q = diag([10 1000 10 1000]);
Panto.R = 1e-5;
Panto.state = 0;

%% MCMC for exploring theta space
% proposal covariance matrix
% Sigma_q = [0.6 10 10 1.5 12 15]';
Sigma_q = [0.001 0.01 0.001 0.01]';

% putting sigma_theta as 5000 for Kpx and Kpy made it to reach high values
% Sigma_theta = diag([100 5000 100 100 5000 100]);
% Sigma_theta = diag([100 5000 100 5000]);
Sigma_theta = diag([0.0001 0.0001 0.0001 0.0001]);
% Number of iterations and number of particles
```

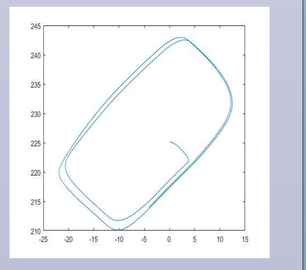
EXPERIMENTS

For deep reinforcement learning we should have a huge amount of data. Since we started to collect data 2 weeks later, we still need to collect too much data. Therefore we couldn't find the appropriate controller values, but we have taken a step.









CONCLUSIONS & FUTURE WORK

After doing more experiments, algorithm will learn the exact values of the controller parameters and these techniques can be used in industry.

For future work, these algorithm can be developed to find the kinematics of the system and the jacobian matrix's values, because the kinematics of the machines, plants, vehicles are too complicated to calculate.

REFERENCES

Deep Reinforcement Learning for Continuous Control. (n.d.). Retrieved from https://www.ias.informatik.tu-darmstadt.de/uploads/Site/EditPublication/Ramstedt_BscThesis_2016

P., T., H., J., J., A., N., T., . . . Erez. (2016, February 29). Continuous control with deep reinforcement learning. Retrieved from https://arxiv.org/abs/1509.02971Reinforcement

Learning for a Dexterous Manipulation Task. (n.d.). Retrieved from https://www.ias.informatik.tu-darmstadt.de/uploads/Main/Abschlussarbeiten/ValerianMarg

Watkins, C. J., & Dayan, P. (n.d.). Q-learning. Retrieved from https://link.springer.com/article/10.1007/BF00992698