Dynamical control of single leg using Physics Informed AI

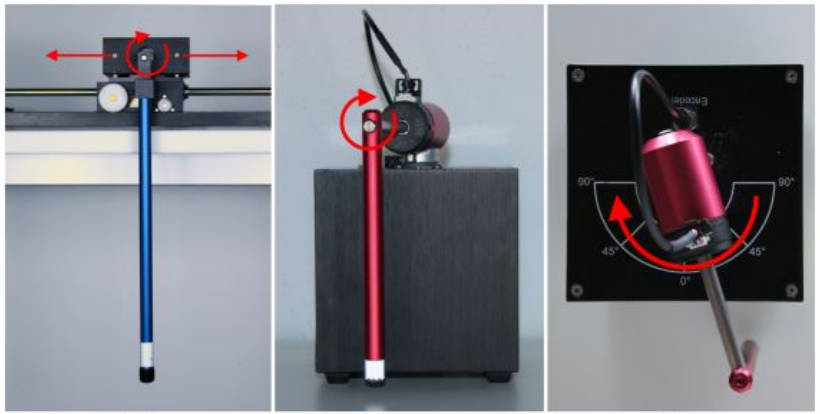
By:
Shivansh Pandey

Supervisor:
Professor Shakti S. Gupta

Literature review

Cartpole and Furuta control using DeLan 4EC

Deep lagrangian networks with energy control



Lutter, M., Listmann, K., & Peters, J. (n.d.). control for under-actuated systems.

DeLan 4EC vs system identification

- DeLaN is capable of learning the underlying ODE and energies from data using the joint configurations and motor torques.

- DeLaN learns the mass-matrix, the centrifugal, Coriolis, gravitational and frictional forces as well as the potential and kinetic energy using unsupervised learning.

- Knowledge of the kinematic structure is not needed

Forward model - $$\underbrace{\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \mathbf{H}(\mathbf{q})\dot{\mathbf{q}}\right)\right)^T}_{:=\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}} + \frac{\partial V}{\partial \mathbf{q}} = \sum_i \boldsymbol{\tau}_i \quad (1)$$

Inverse model- $$\mathbf{H}^{-1}(\mathbf{q})\left(\sum_i \boldsymbol{\tau}_i - \dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} + \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \mathbf{H}(\mathbf{q})\dot{\mathbf{q}}\right)\right)^T - \frac{\partial V}{\partial \mathbf{q}}\right) = \ddot{\mathbf{q}}.$$

H is mass matrix simplified as- $$\hat{\mathbf{H}} = \hat{\mathbf{L}}(\mathbf{q};\theta)\,\hat{\mathbf{L}}^T(\mathbf{q};\theta) + \epsilon\,\mathbf{I} \qquad \hat{V} = \hat{V}(\mathbf{q};\psi) \qquad (2)$$

(L is lower triangular matrix)

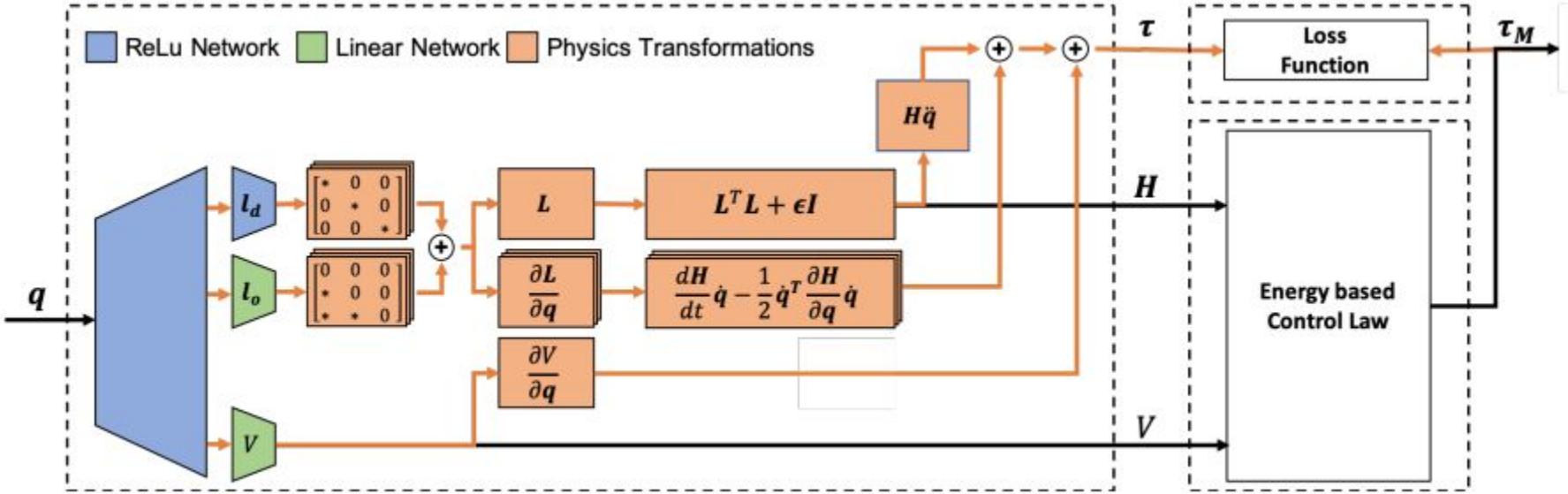
Online Learning:

The network parameters can be learned online and end-to-end, by minimizing the error of the ODE using the samples {q, qÛ, qÜ, τ M } recorded on the physical system, i.e. minimizing the li norm between the prediction of Equation 1 and the observed motor torque τ M . Therefore, the superposition of the different forces is learned supervised, while the decomposition into inertial, Coriolis, centripetal and gravitational forces is learned unsupervised.

Things DeLan 4EC does

Adheres to the Lagrangian Mechanics

Ensures energy conservation



Incorporating friction

$$\boldsymbol{\tau}_{fi} = -\left(\tau_{C_v} + \tau_{C_s}\, \exp\left(-\dot{\mathbf{q}}_i^2/v\right)\right) \operatorname{sign}(\dot{\mathbf{q}}_i) - d\,\dot{\mathbf{q}}_i \qquad (4)$$

Friction coefficients- $\phi = \{\tau_{C_v}, \tau_{C_s}, v, d\}$

Since the frictional force τ f is a function of the generalized coordinates, the frictional force is a non-conservative and generalized force and can simply be added to the Lagrange Euler ODE (Equation 1) i.e. $\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \underbrace{\dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \mathbf{H}(\mathbf{q})\dot{\mathbf{q}}\right)\right)^T}_{:=\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}} + \frac{\partial V}{\partial \mathbf{q}} = \sum_i \boldsymbol{\tau}_i \qquad (1)$

Loss function-

$$(\theta^*, \psi^*) = \arg\min_{\theta,\psi} \ell_i \left(\hat{f}(\theta, \psi), \ \ddot{\mathbf{q}} \right) + \ell_i \left(\hat{f}^{-1}(\theta, \psi), \ \boldsymbol{\tau}_M \right) + \lambda \Omega(\theta, \psi) \tag{3}$$

where Ω is the l_2 weight regularization.

Adding energy conservation (Equation 7) and energy coherence (Equation 8 & Equation 9) to the optimization problem of Equation 3 yields the loss for DeLaN 4EC.

$$\tilde{T}(\mathbf{q}_{t+\delta t}; \theta) = \hat{T}(\mathbf{q}_t; \psi) + \dot{\mathbf{q}}_t^T \mathbf{H} \ddot{\mathbf{q}}_t \delta_t + \frac{1}{2} \dot{\mathbf{q}}_t^T \dot{\mathbf{H}} \dot{\mathbf{q}}_t^T \delta_t \tag{8}$$

$$\tilde{V}(\mathbf{q}_{t+\delta t}; \psi) = \hat{V}(\mathbf{q}_t; \psi) + \dot{\mathbf{q}}_t^T \frac{\partial \hat{V}}{\partial \mathbf{q}} \delta_t . \tag{9}$$

$$\begin{aligned} \dot{E} = \dot{\mathbf{q}}^T (\boldsymbol{\tau}_M + \boldsymbol{\tau}_F) &= \dot{T} + \dot{V} \\ &= \dot{\mathbf{q}}^T \mathbf{H} \ddot{\mathbf{q}} + \frac{1}{2} \dot{\mathbf{q}}^T \dot{\mathbf{H}} \dot{\mathbf{q}} + \dot{\mathbf{q}}^T \frac{\partial V}{\partial \mathbf{q}} . \end{aligned} \tag{7}$$

Control Law

Energy of the pendulum: Ep

Desired energy: E ∗

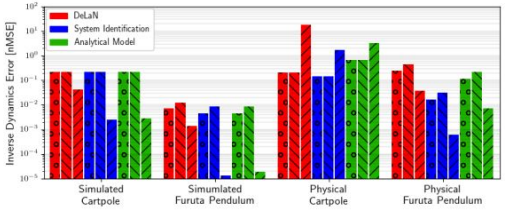
For systems with high friction an additional term to compensate the friction of the actuator can be added.

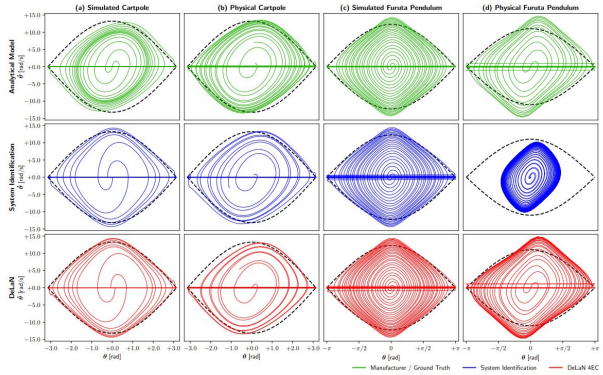
$$\mathbf{u}_E = k_E \left(E_p - E^*\right) \operatorname{sign}\left(\dot{\mathbf{q}}_p \cos(\mathbf{q}_p)\right) + \mathbf{K}_p \left(\mathbf{q}_a^* - \mathbf{q}_a\right) \qquad (10)$$

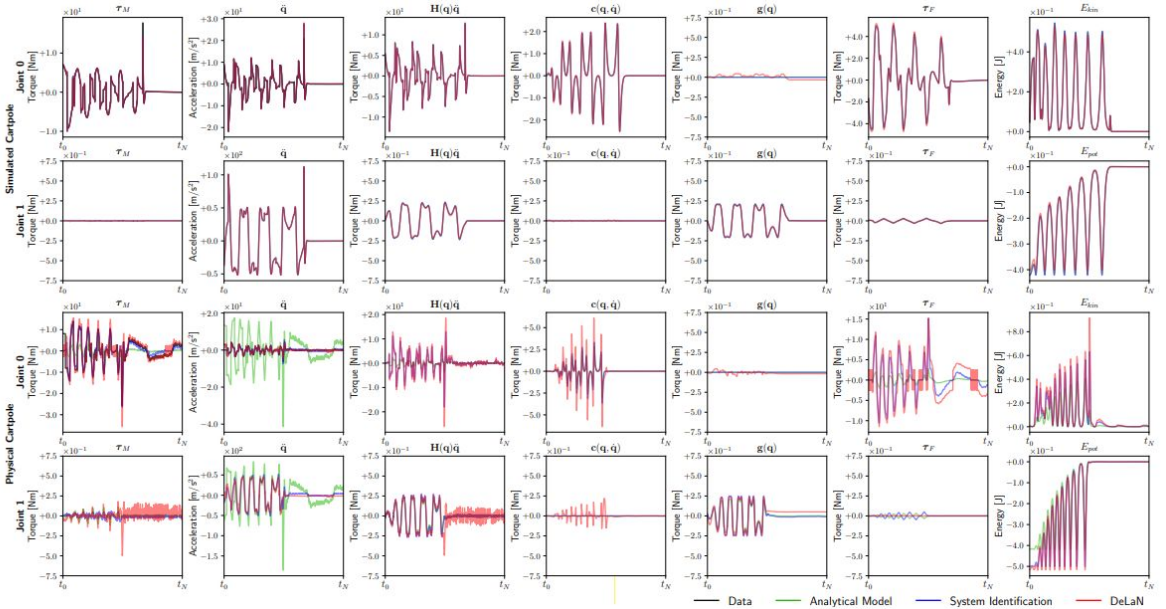
uE = controlled torque

For swing-up task the systems are first stabilized to the desired energy E ∗ of the balancing point using energy control and then balanced at the unstable equilibrium using a PD-controller.

Result







$$\text{nMSE} = \frac{\sum_{i=0}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2}{\sum_{i=0}^{N} \|\mathbf{x}_i + \delta\|_2^2} \qquad (11)$$

Result (offline control evaluation)

Only for the passive joint DeLaN performs slightly worse due to the noise.

For every other case it performed better or equivalent to system identification and analytical model.

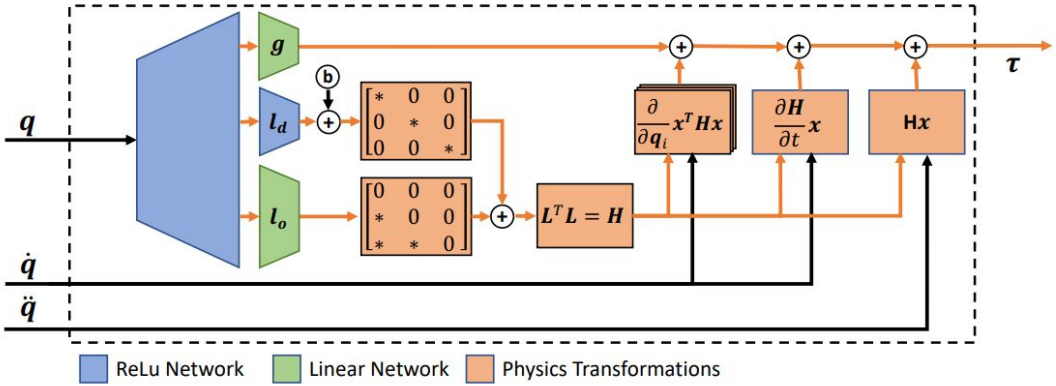
TABLE I

PERCENTAGE OF SUCCESSFUL SWING-UPS OF SIMULATED AND PHYSICAL CARTPOLE AND FURUTA PENDULUM FOR THE DIFFERENT MODELS.

	Cartpole		**Furuta**	
Model	Sim	Real	Sim	Real
Analytic Model	1.00	1.00	1.00	1.00
System Identification	1.00	1.00	0.93	0.00
DeLaN	1.00	1.00	1.00	0.90

Deep Lagrangian Networks:

Using Physics as Model Prior for Deep Learning



Lutter, M., Ritter, C., Peters, J., Science, C., Universit, T., Hochschulstr, D., Networks, D. L., Mechanics, L., & Mechanics, L. (2019). Published as a conference paper at ICLR 2019. 1–17.

Forward model - $$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \underbrace{\dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \mathbf{H}(\mathbf{q})\dot{\mathbf{q}}\right)\right)^T}_{:=\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}} + \frac{\partial V}{\partial \mathbf{q}} = \sum_i \boldsymbol{\tau}_i \quad (1)$$

Inverse model- $$\mathbf{H}^{-1}(\mathbf{q})\left(\sum_i \boldsymbol{\tau}_i - \dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} + \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \mathbf{H}(\mathbf{q})\dot{\mathbf{q}}\right)\right)^T - \frac{\partial V}{\partial \mathbf{q}}\right) = \ddot{\mathbf{q}}.$$

$$\hat{\mathbf{H}} = \hat{\mathbf{L}}(\mathbf{q};\theta)\,\hat{\mathbf{L}}^T(\mathbf{q};\theta) + \epsilon\,\mathbf{I} \qquad \hat{V} = \hat{V}(\mathbf{q};\psi) \quad (2)$$

H is mass matrix simplified as-

(L is lower triangular matrix)

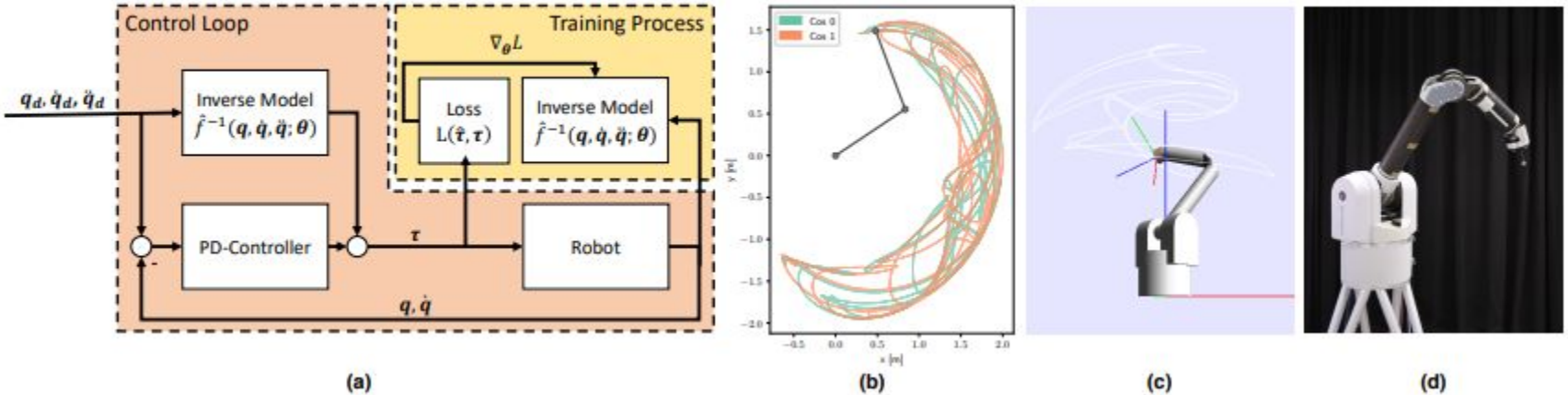
.

2R robot training

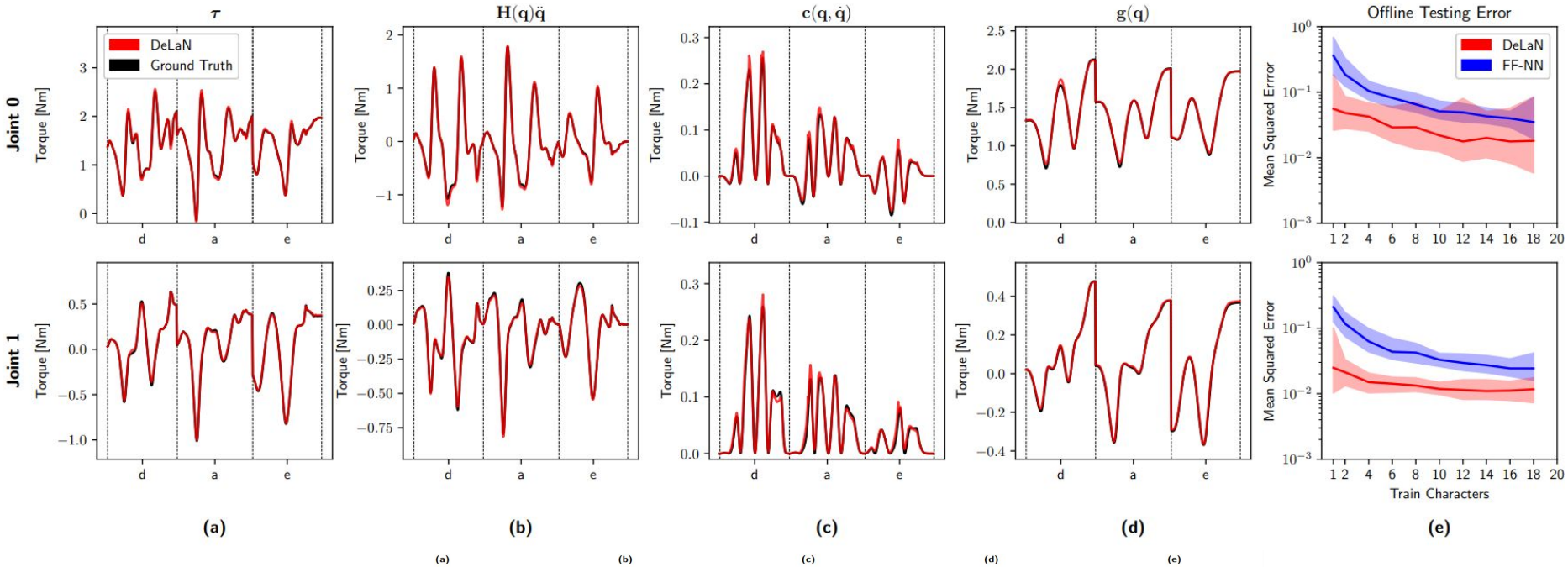
Tff is returned from the network

Control Law implemented (Torque values are updated at 200 Hz frequency)

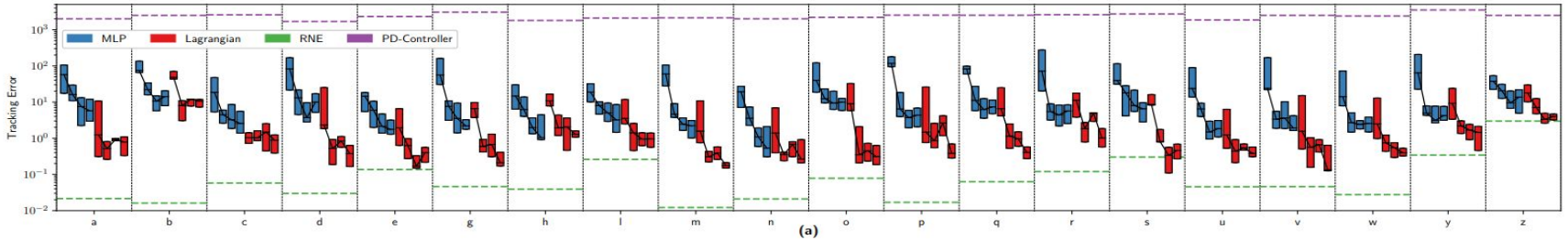
$$\boldsymbol{\tau} = \mathbf{K}_p\,(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_d\,(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) + \boldsymbol{\tau}_{ff} \quad \text{with} \quad \boldsymbol{\tau}_{ff} = \hat{f}^{-1}(\mathbf{q}_d, \dot{\mathbf{q}}_d, \ddot{\mathbf{q}}_d)$$

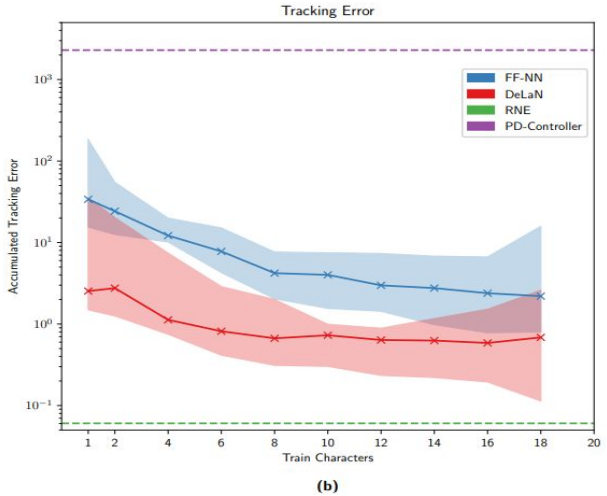


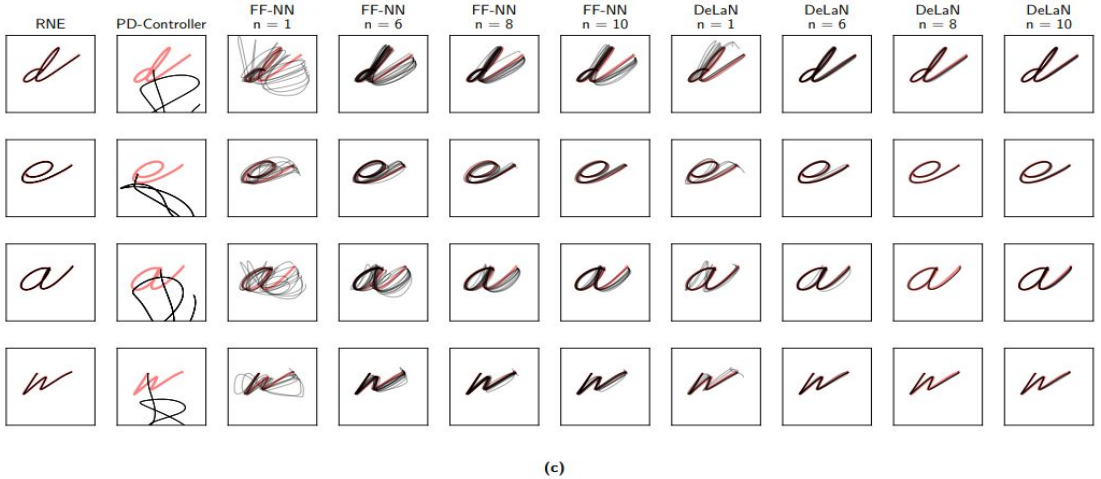
Testing result



Testing for various n_dof





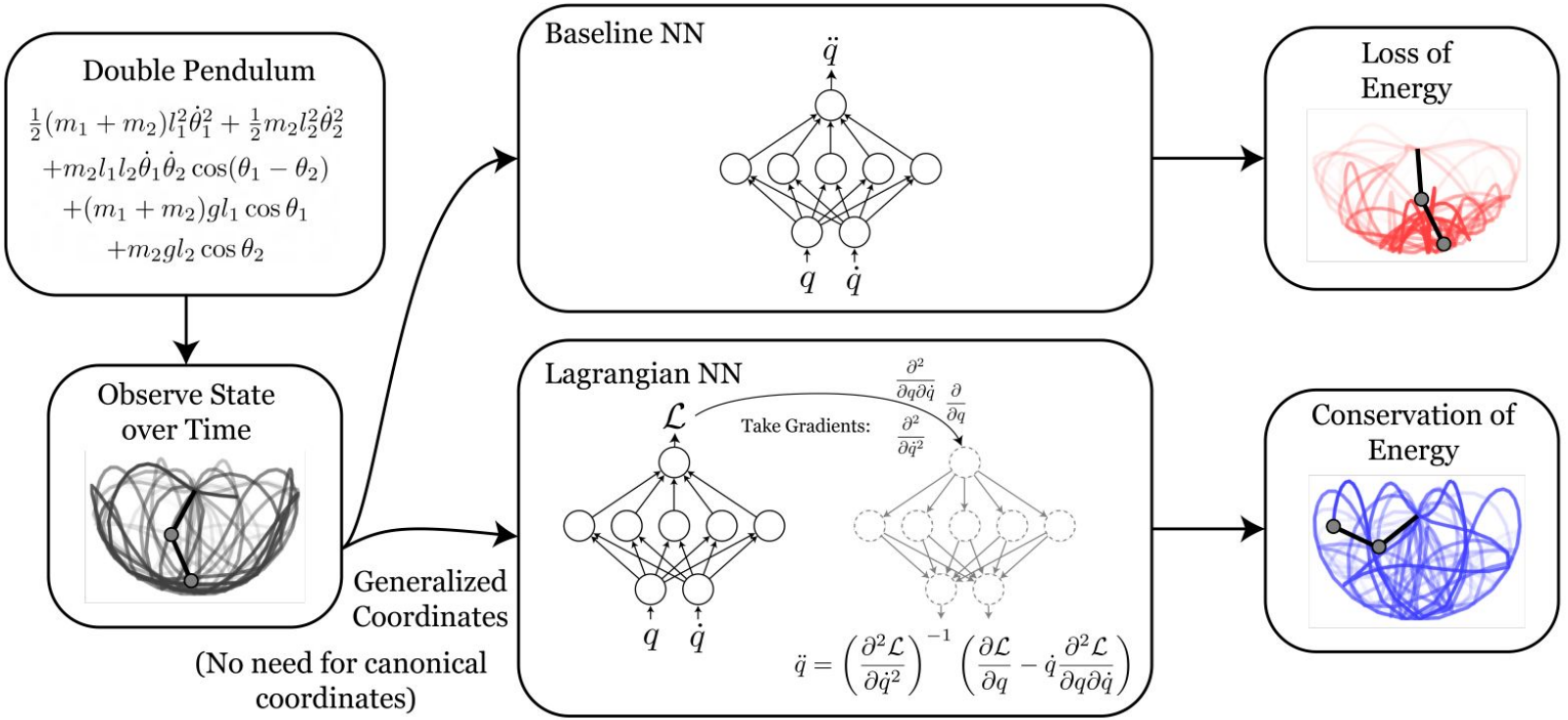


Lagrangian Neural Networks

Cranmer, M., Greydanus, S., Hoyer, S., Battaglia, P., & Spergel, D. (n.d.). Agrangian eural etworks. 1–9.

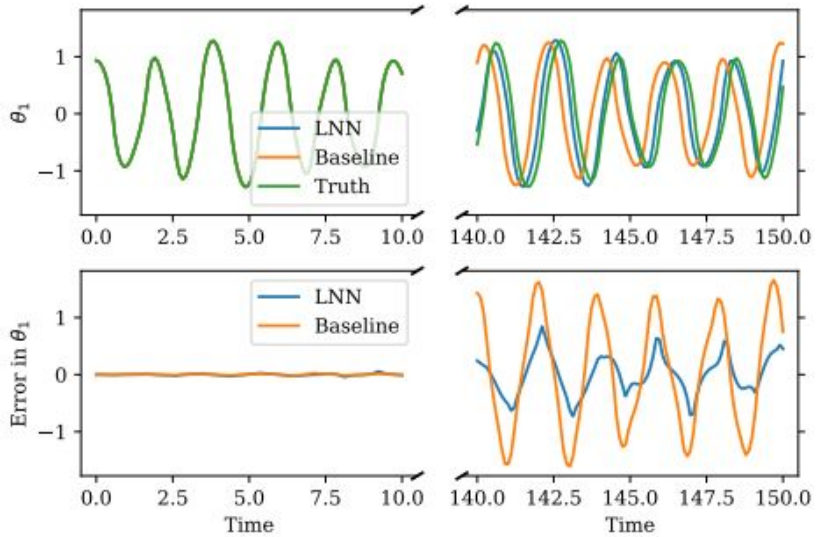
Network Overview

Predicting Lagrangian through neural network and then feeding this to the Euler Lagrangian equation to get q double dot or required torques

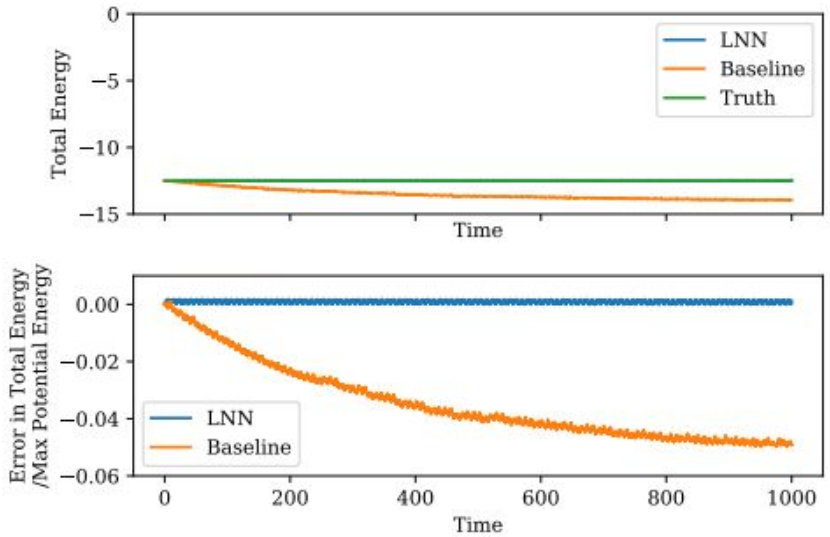


Double Pendulum

LNN almost exactly conserved the true energy over time, whereas the baseline did not. Averaging over 40 random initial conditions with 100 time steps each, the mean energy discrepancy between the true total energy and predicted was 8% and 0.4% of the max potential energy for the baseline and LNN models.



(a) Error in angle



(b) Error in energy

Comparison with DeLaN

DeLaN. A closely related previous work is “Deep Lagrangian Networks”, or DeLaN (Lutter et al., 2019), in which the authors show how to learn specific types of Lagrangian systems. They assume that the kinetic energy is an inner product of the velocity: T = ˙q TMq˙, where M is a q-dependent positive definite matrix. This approach works well for rigid body dynamics, which includes many systems encountered in robotics. However, many systems do not have this kinetic energy, including, for example, a charged particle in a magnetic field, and a fast-moving object in special relativity. Other Lagrangian-based approaches include Gupta et al. (2019); Qin (2019).

	Neural net	Neural ODE	HNN	DeLaN	LNN (ours)
Can model dynamical systems	✓	✓	✓	✓	✓
Learns differential equations		✓	✓	✓	✓
Learns exact conservation laws			✓	✓	✓
Learns from arbitrary coords.	✓	✓		✓	✓
Learns arbitrary Lagrangians					✓

Michael Lutter repository

https://github.com/milutter/deep_lagrangian_networks

Libraries required

- Numpy
- Matplotlib
- Torch
- Jax
- Dill
- Haiku
- Warnings
- Time

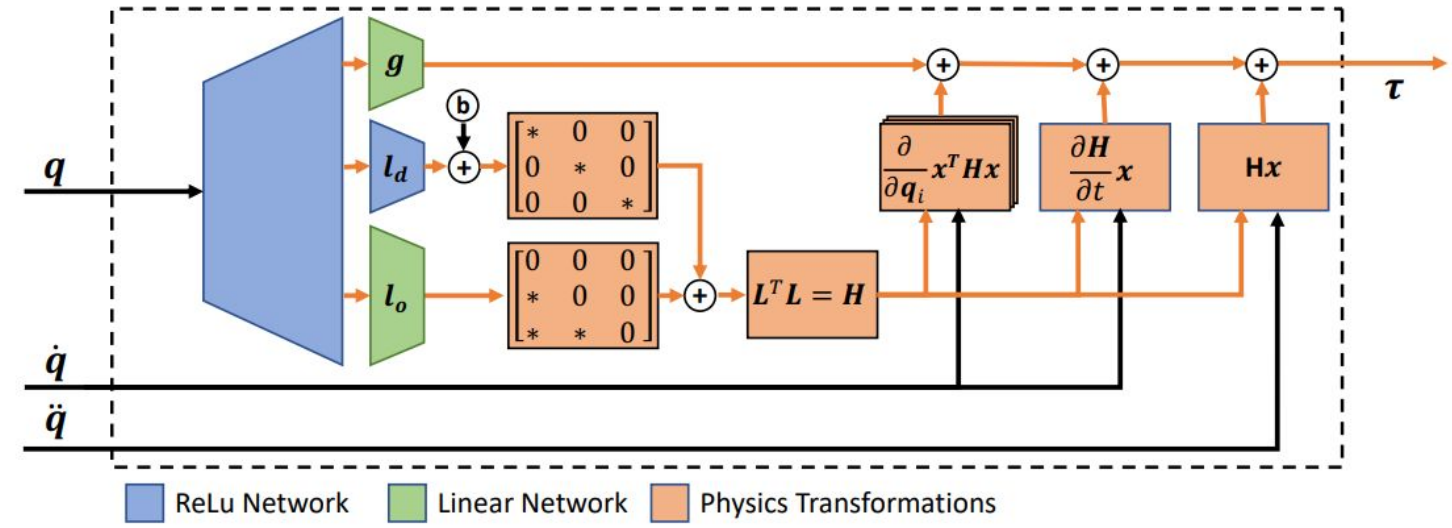
Classes and functions

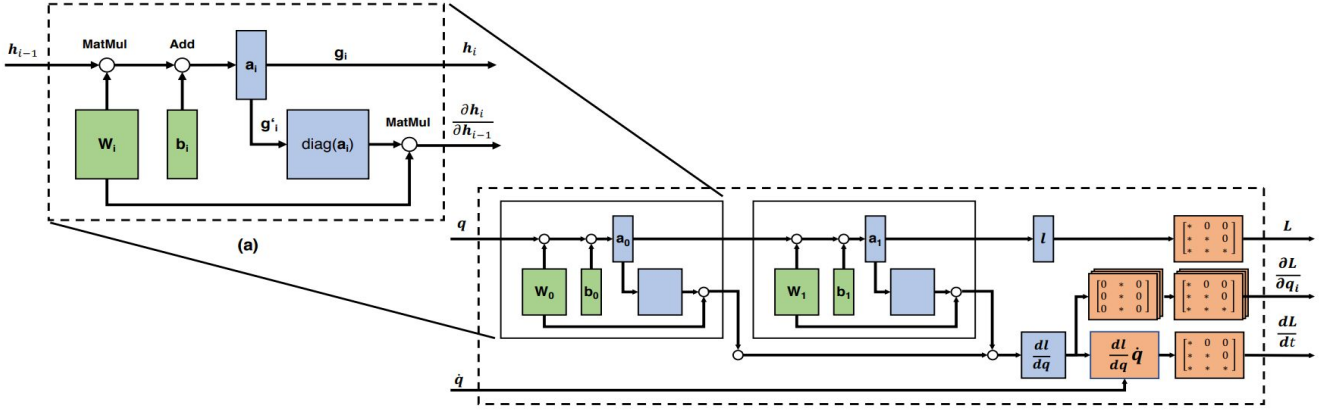
- Replay memory-
- load_dataset- load data file for training and testing
- LowTri - input- n (size of L = nxn) l (l = elements in L) ; Returns L
  n= n_dof

$$\hat{\mathbf{H}} = \hat{\mathbf{L}}(\mathbf{q};\theta)\,\hat{\mathbf{L}}^{T}(\mathbf{q};\theta) + \epsilon\,\mathbf{I}$$

  (mass matrix)

- Langrangian Layer - input- n_dof, q, qd (q dot), qdd (q double dot), activation, input size
- Deep Langrangian - input- n_dof, **kwargs (hyperparameters), q, qd (q dot), qdd (q double dot)


q
q̇
q̈
g
l_d
l_o
b
L^T L = H
∂/∂q_i x^T Hx
∂H/∂t x
Hx
τ
ReLu Network
Linear Network
Physics Transformations


h_{i-1}
MatMul
Add
a_i
g_i
h_i
W_i
b_i
g'_i
diag(a_i)
MatMul
∂h_i/∂h_{i-1}
(a)
q
q̇
a_0
a_1
W_0
b_0
W_1
b_1
l
dl/dq
dl/dq q̇
L
∂L/∂q_i
dL/dt

Functions in Deep Lagrangian (Class)

- _init_ - Parameters which are n_width, n_depth, b_init, b_diag_init, w_init, g_hidden, g_hidden, p_sparse, diagonal_epsilon
- _dyn_model (q, q_dot, q_double_dot) - Through 3 different layers computes Mass matrix (2 layers used) and g (= derivation of potential energy) and uses these in Euler-Lagrangian equations. $\underbrace{\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{H}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\left(\frac{\partial}{\partial \mathbf{q}}\left(\dot{\mathbf{q}}^T \mathbf{H}(\mathbf{q})\dot{\mathbf{q}}\right)\right)^T}_{:=\mathbf{c}(\mathbf{q},\dot{\mathbf{q}})} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}$ returns- Tau predicted, Mass matrix, Coriolis and centrifugal forces, Mg, Kinetic energy, potential energy, Kinetic energy derivative, potential energy derivative.
- Forward (q, q_dot, q_double_dot) - returns- Tau predicted, derivative of (K.E. + P.E.)
- For_dyn (q, q_dot, q_double_dot)- returns- Tau predicted
- Inv_dyn (q, q_dot, q_double_dot)- returns- q double dot predicted
- Energy (q, q_dot, q_double_dot)- returns- K.E. + P.E.
- Energy_dot (q, q_dot, q_double_dot)- returns- derivative of (K.E. + P.E.)

Training

N_dof = 2

Training data- train_labels, train_qp, train_qv, train_qa, train_p, train_pd, train_tau

Hyper = {.....} (define hyperparameters for the neural network

Delan_model = DeepLagrangianNetwork(n_dof, **hyper)

optimizer= Adam optimizer

Generate replay memory

Training Loop-

Training Loop

Get Tau predicted and dE/dt predicted from delan_model(q, q_dot, q_double_dot)

E-L differential loss = mean.mse(Tau predicted, Train_tau)

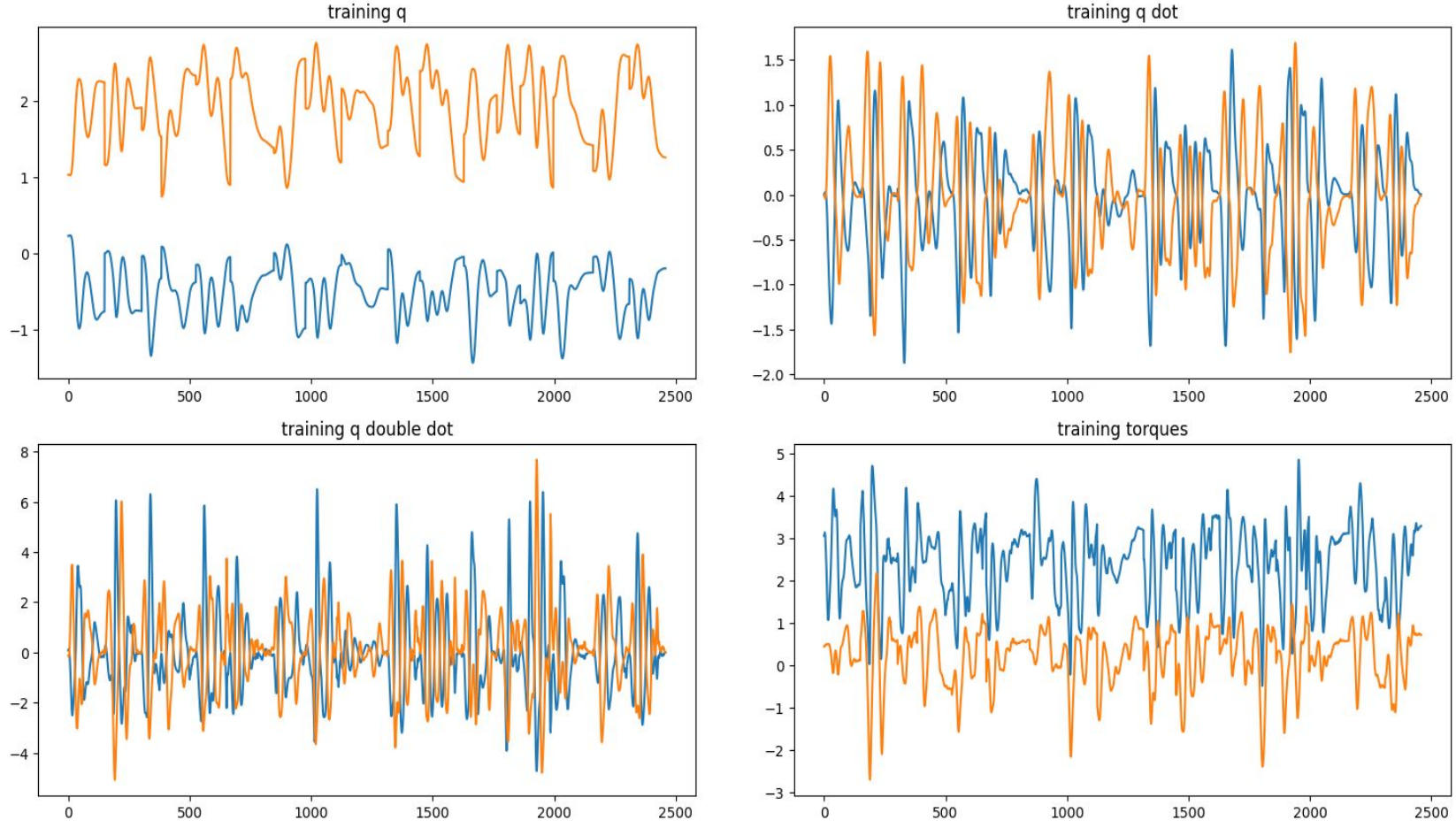
Power conservation loss = mean.mse(dE/dt predicted, dE/dt)

[dE/dt = transpose(q_dot) * train_tau]

Total loss= E-L differential loss + Power conservation loss

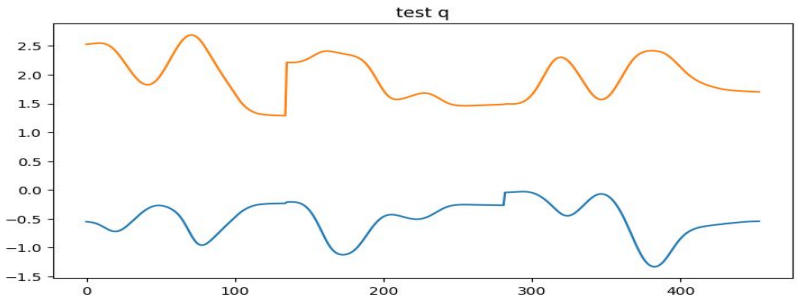
Backpropagate and optimize the network

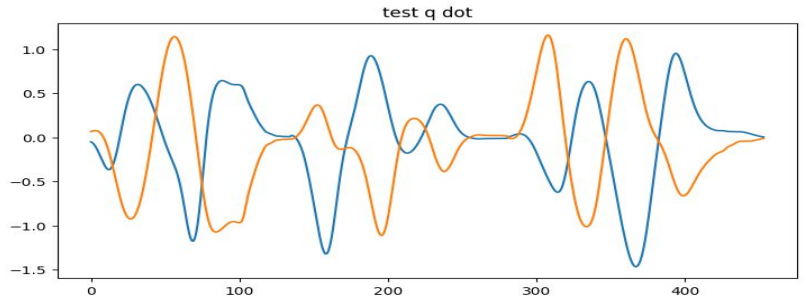
Update the predicted data

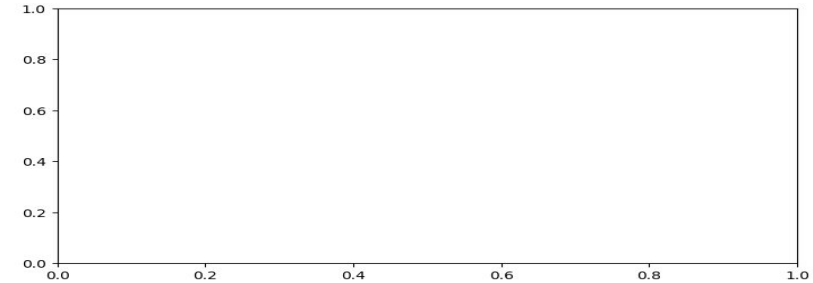

training q
training q dot
training q double dot
training torques

Testing

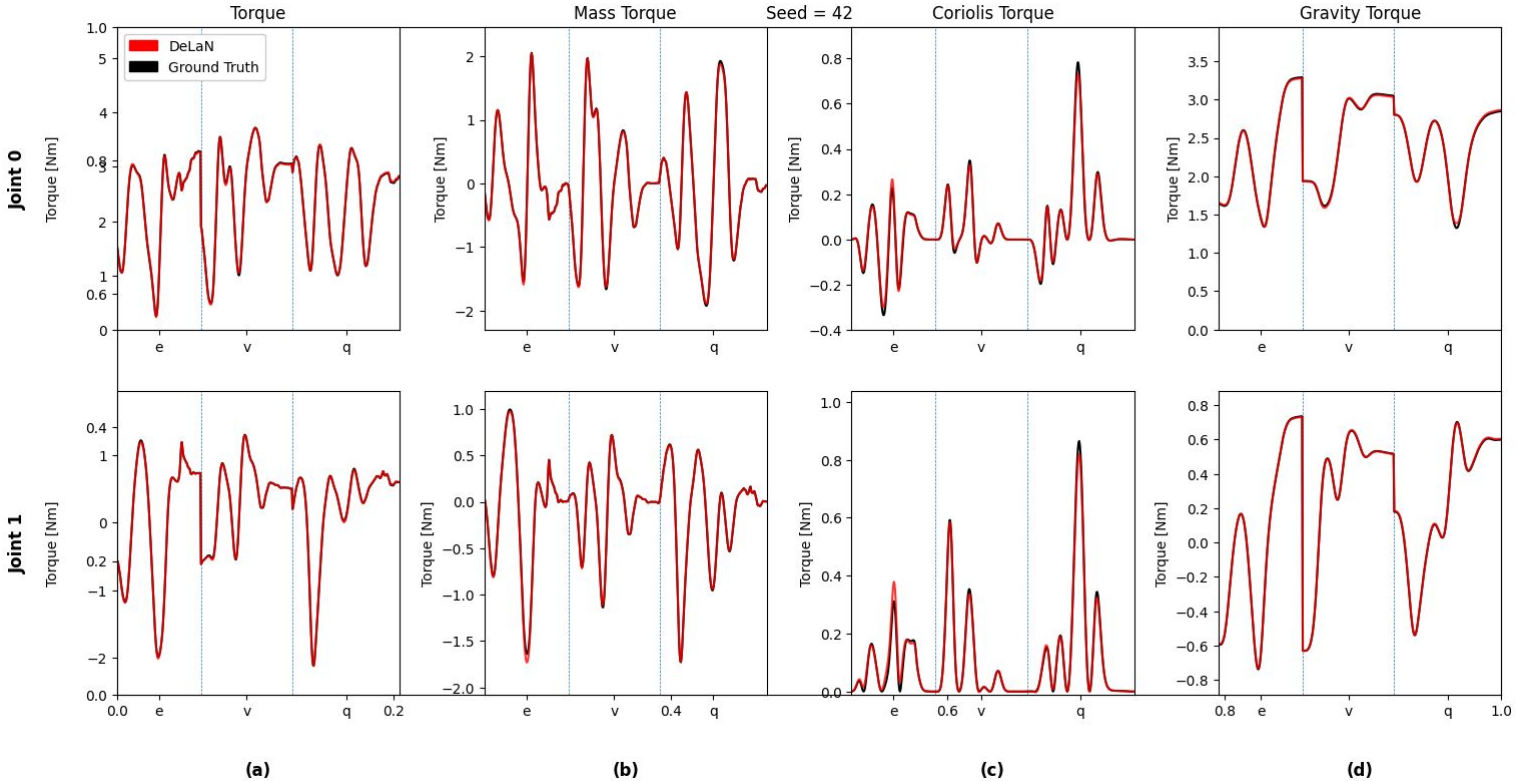
Testing data - qp, qv, qa, p, pd, tau, m (mass torque), c (coriolis torque), g (gravity torque)





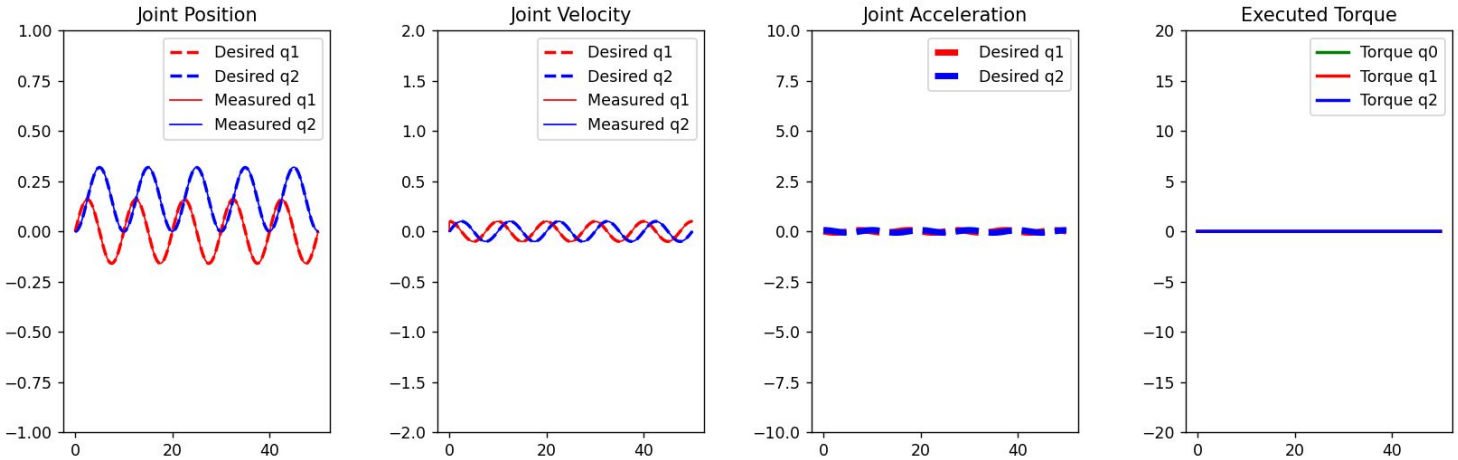


Results from test data



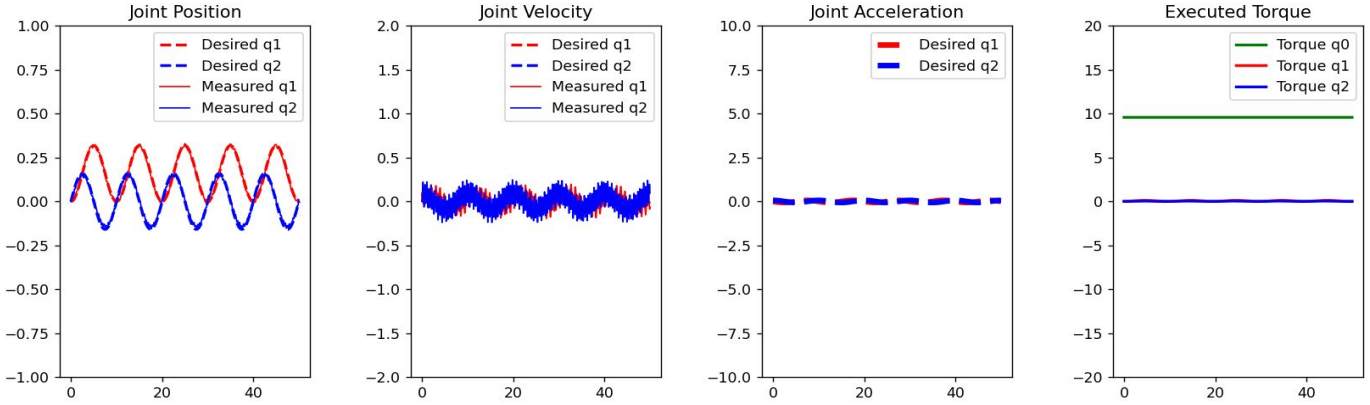
Training DeLaN for single leg

Built-in python function (calculateInverseDynamics) was used to get training and testing data for torque values.

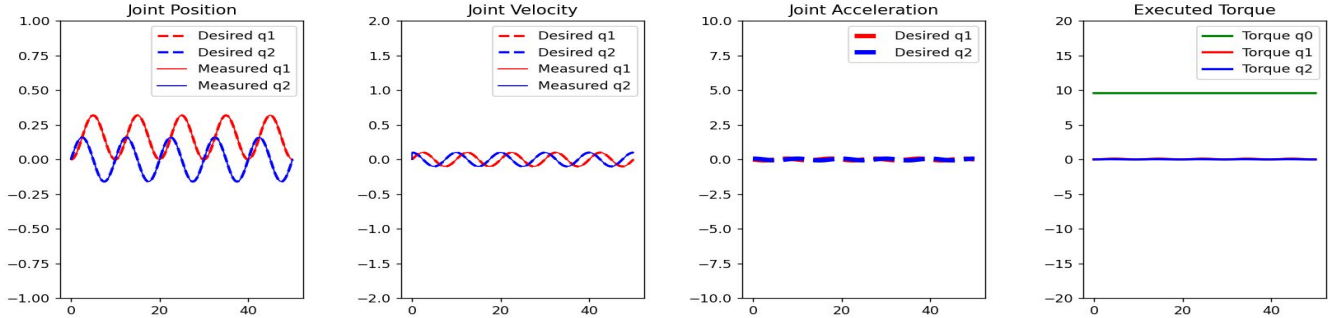


Testing results

DeLan torque Simulation



Analytical torque simulation



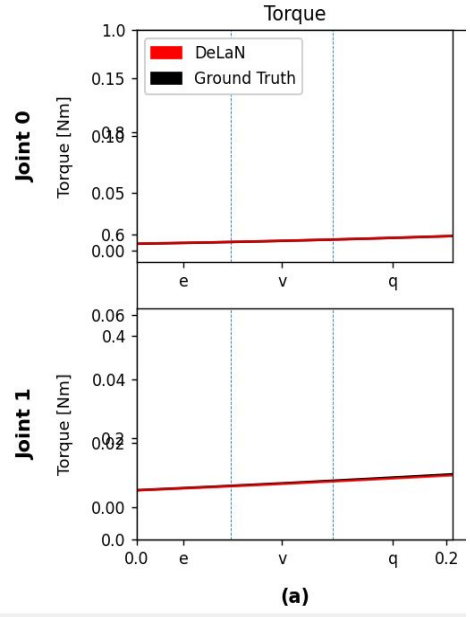
DeLaN performance

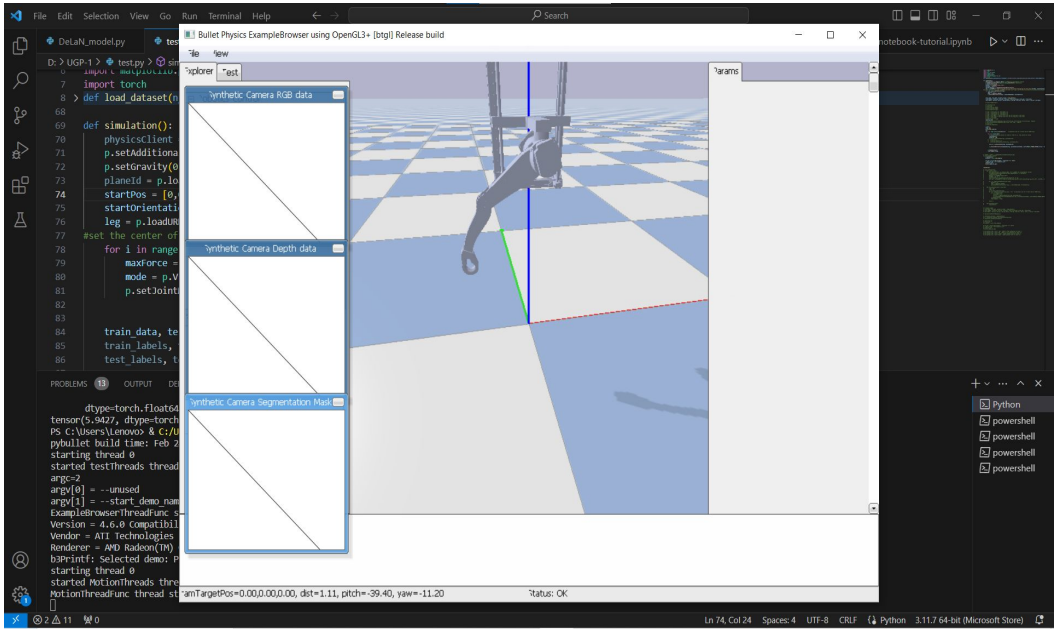
```
Training Deep Lagrangian Networks (DeLaN):
Epoch 00001:  Time = 003.0s, Loss = 3.171e-03, Inv Dyn = 2.862e-03 ± 5.265e-03, Power Con = 5.146e-06 ± 1.269e-05
Epoch 00002:  Time = 006.1s, Loss = 2.282e-03, Inv Dyn = 2.110e-03 ± 2.755e-03, Power Con = 3.532e-06 ± 4.909e-06
Epoch 00003:  Time = 009.0s, Loss = 2.040e-03, Inv Dyn = 1.885e-03 ± 2.464e-03, Power Con = 3.134e-06 ± 4.374e-06
Epoch 00004:  Time = 012.4s, Loss = 1.496e-03, Inv Dyn = 1.378e-03 ± 1.815e-03, Power Con = 2.237e-06 ± 3.185e-06
Epoch 00005:  Time = 015.5s, Loss = 5.321e-04, Inv Dyn = 4.864e-04 ± 7.026e-04, Power Con = 7.020e-07 ± 1.189e-06
Epoch 00006:  Time = 019.4s, Loss = 3.118e-05, Inv Dyn = 2.987e-05 ± 3.954e-05, Power Con = 1.809e-08 ± 4.357e-08
Epoch 00007:  Time = 022.7s, Loss = 4.247e-06, Inv Dyn = 3.844e-06 ± 5.561e-06, Power Con = 7.543e-09 ± 1.254e-08
Epoch 00008:  Time = 026.8s, Loss = 1.959e-06, Inv Dyn = 1.776e-06 ± 2.522e-06, Power Con = 3.359e-09 ± 6.602e-09
Epoch 00009:  Time = 030.1s, Loss = 9.968e-07, Inv Dyn = 9.152e-07 ± 1.232e-06, Power Con = 1.514e-09 ± 3.673e-09
Epoch 00010:  Time = 033.2s, Loss = 5.867e-07, Inv Dyn = 5.443e-07 ± 6.667e-07, Power Con = 8.045e-10 ± 2.277e-09
```

```
Evaluating DeLaN:

Performance:
                Torque MSE = 7.353e-07
    Power Conservation MSE = 2.840e-09
      Comp Time per Sample = 1.173e-03s / 852.4Hz
```

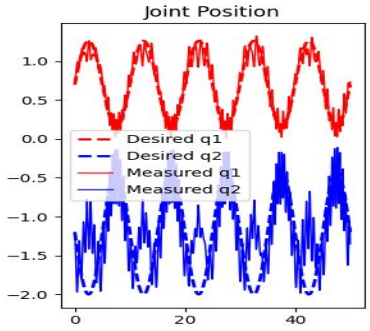
Torque validation and Pybullet simulation

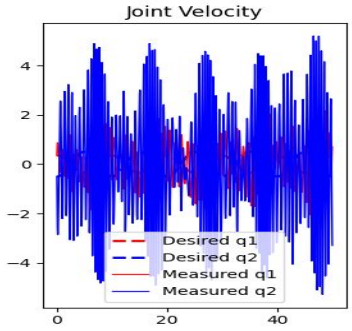


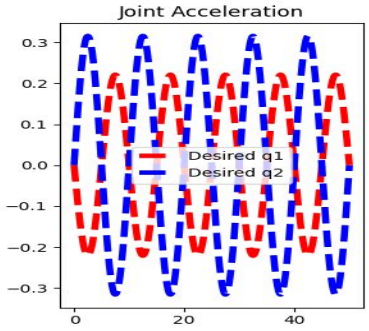


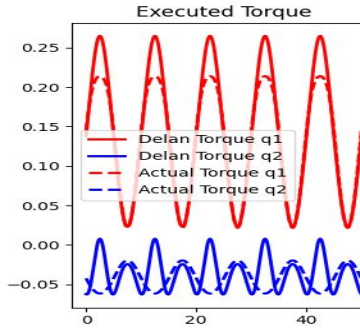
Increasing Epochs

10 epochs

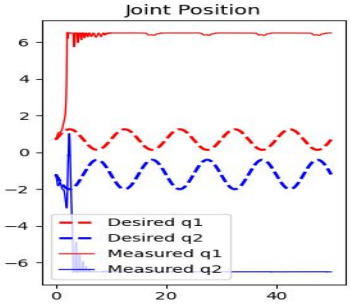


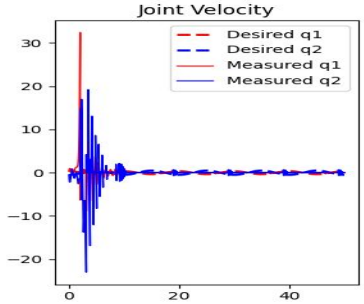


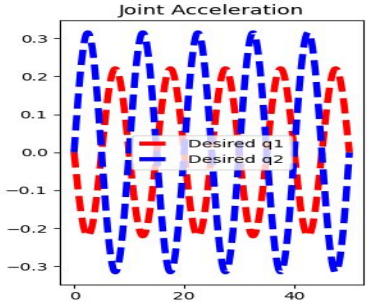


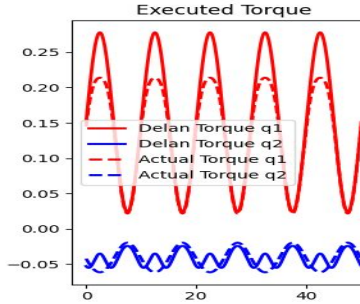


100 epochs

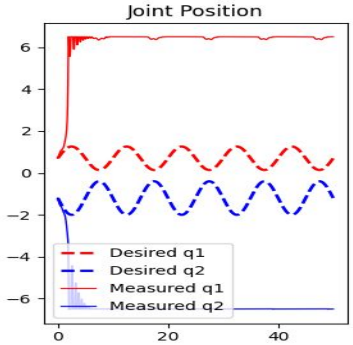


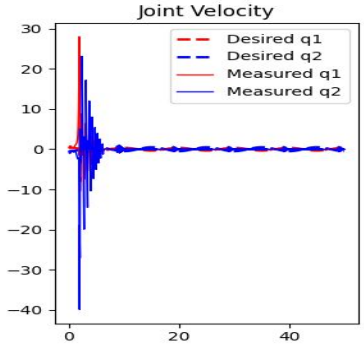


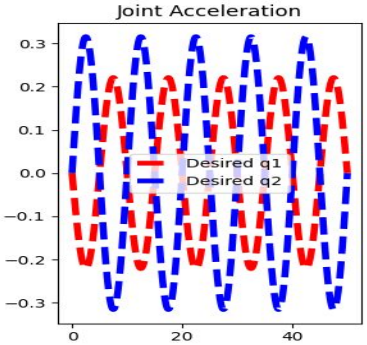


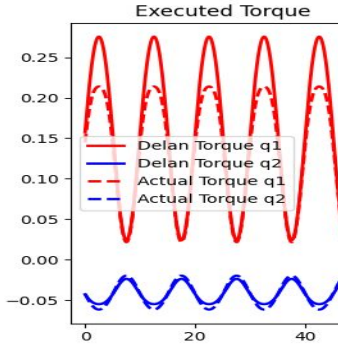


10000 epochs



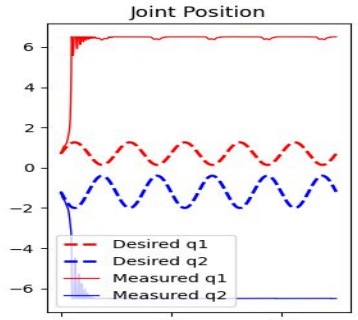


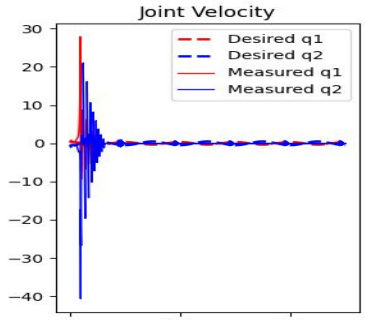


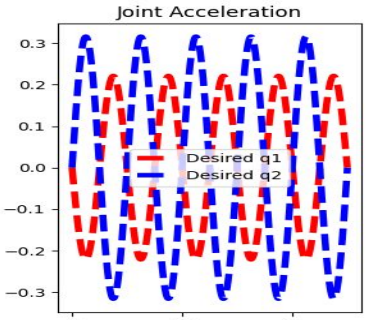


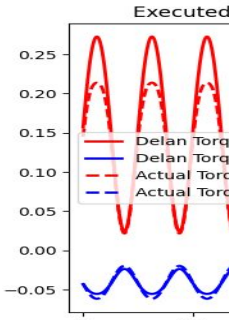
Changing learning rates

5 x 10^(-03)



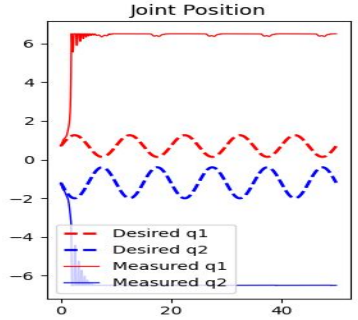


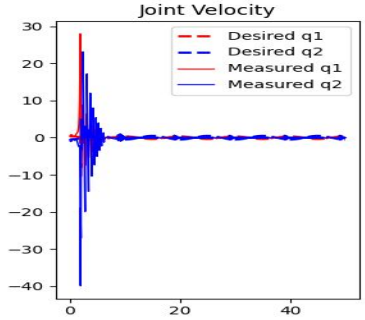


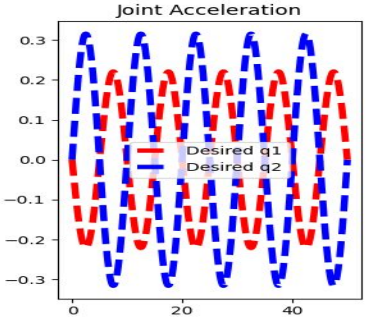


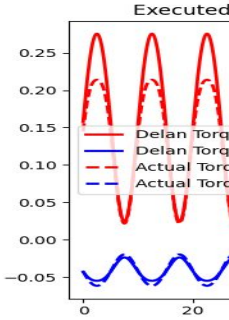
5 x 10^(-04)

(default in code)



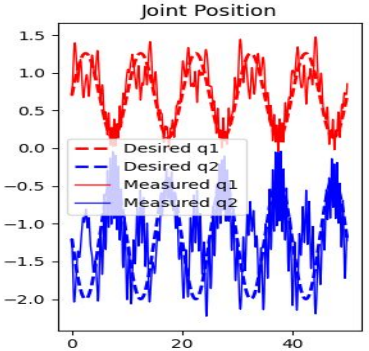


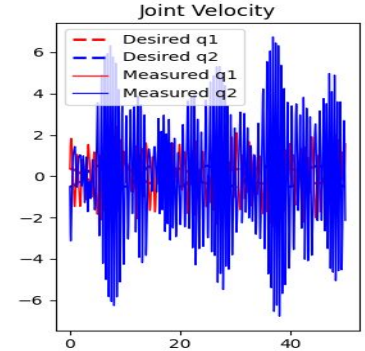


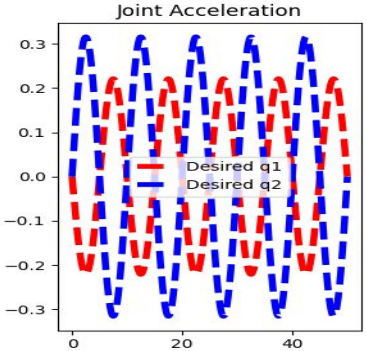


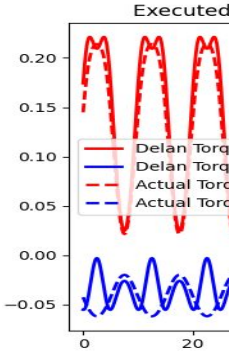
5 x 10^(-05)

(1000 epochs)



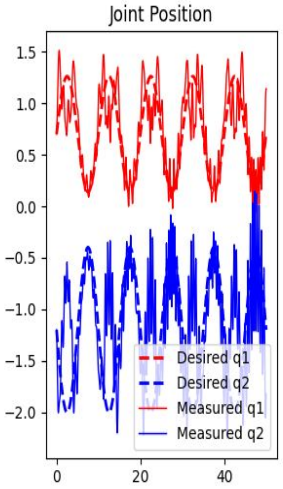


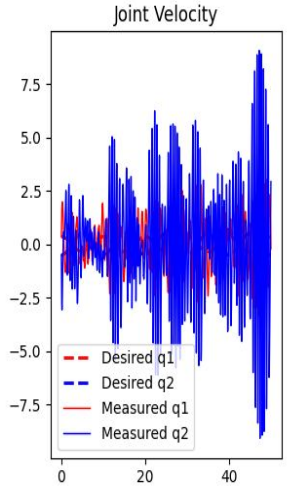


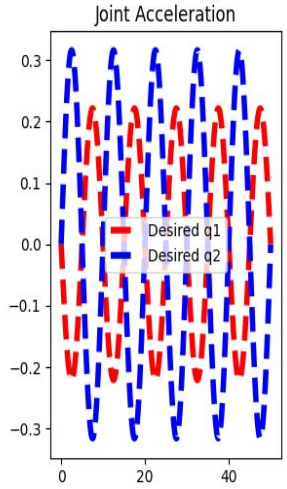


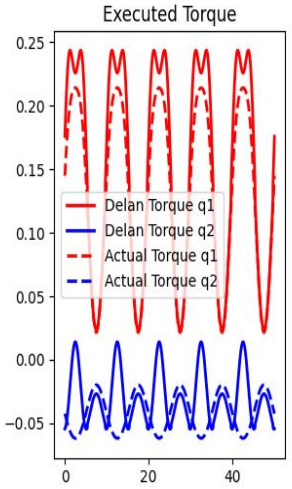
Learning rate (10,000 epochs)

5 x 10^(-05)

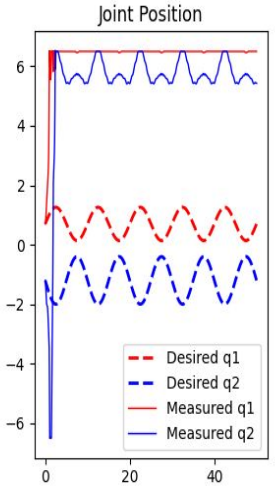


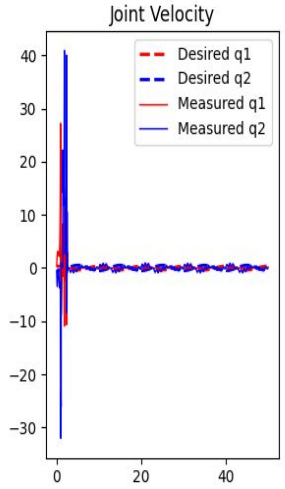


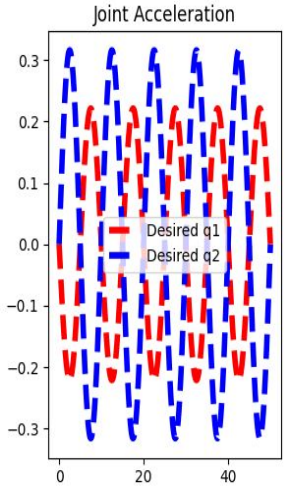


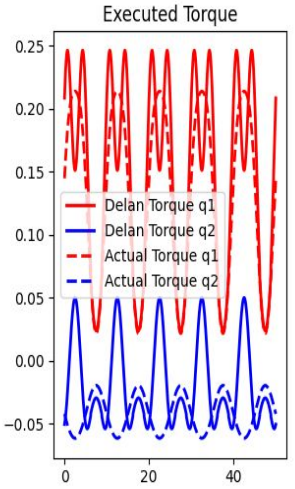


5 x 10^(-06)

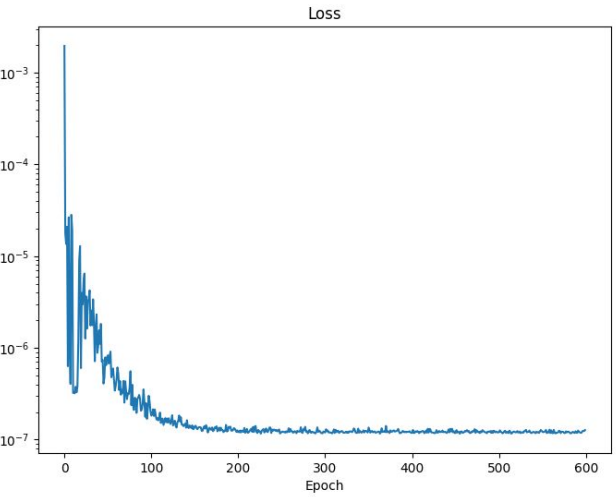




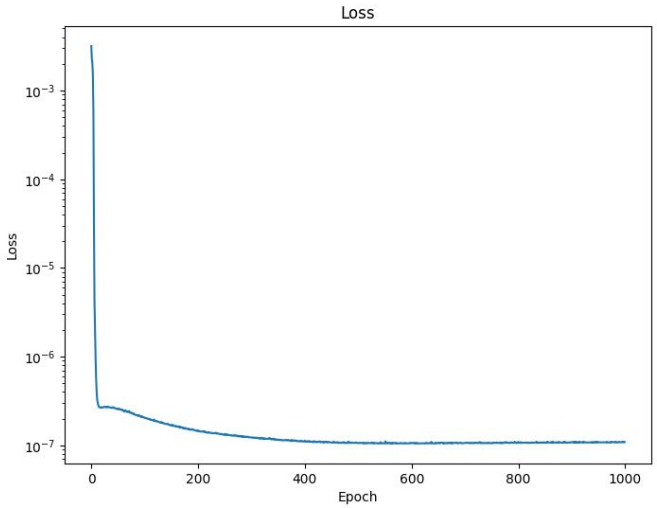




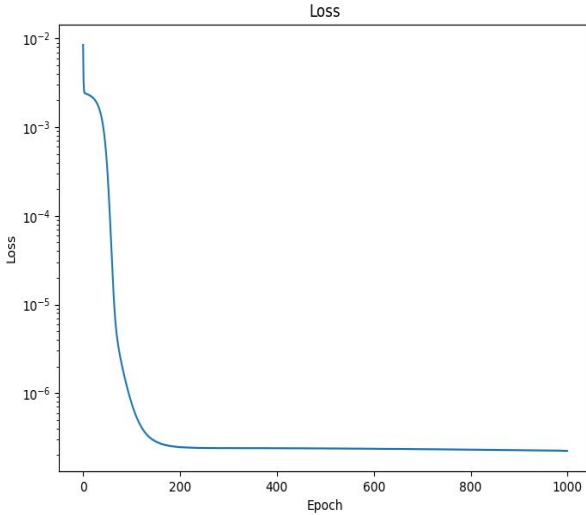
Training Loss



Learning rate = 5 x 10^(-03)



Learning rate = 5 x 10^(-04)

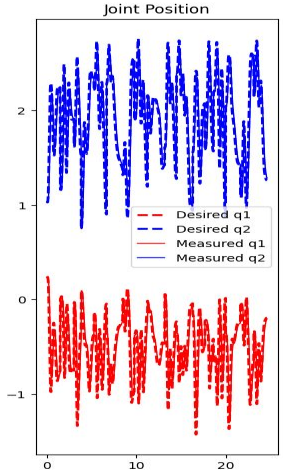


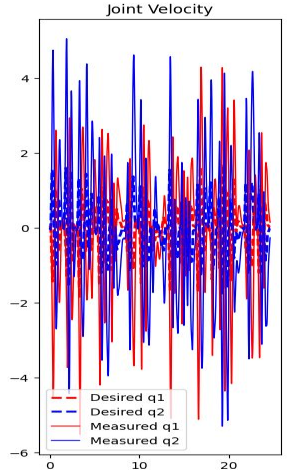
Learning rate = 5 x 10^(-05)

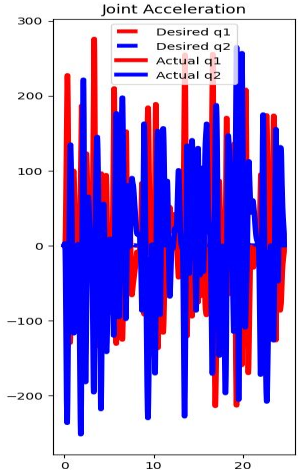
Arbitrary data

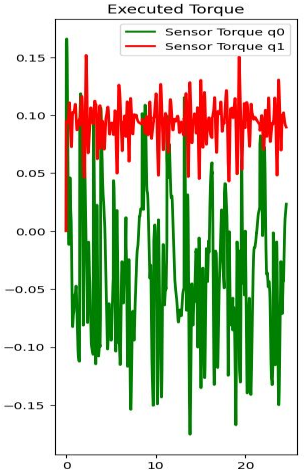
Training-

(1000 epochs)

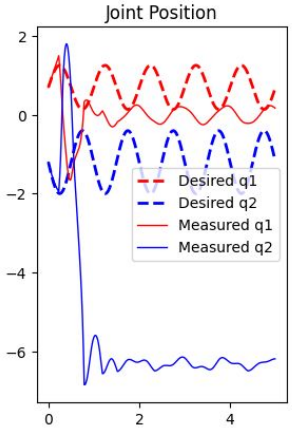


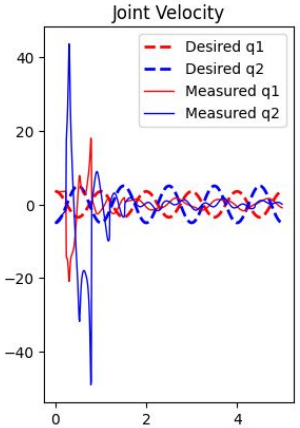


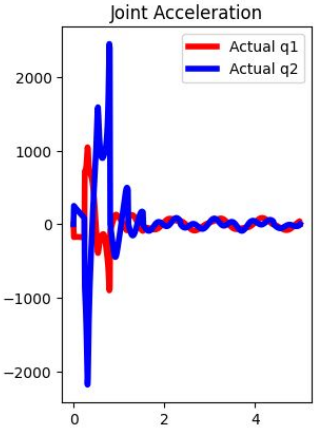


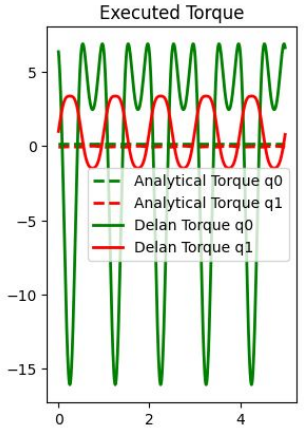


Testing-



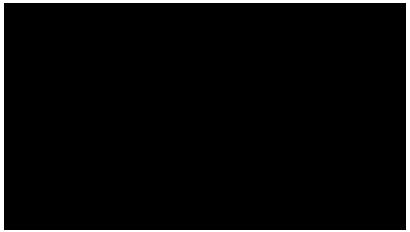






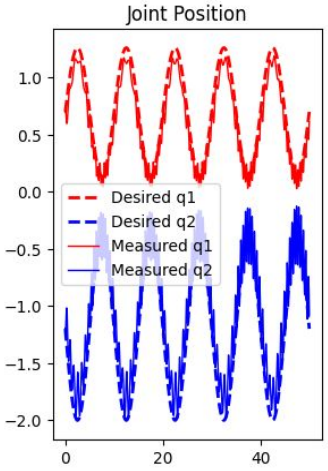
Performance after multiplying by a factor

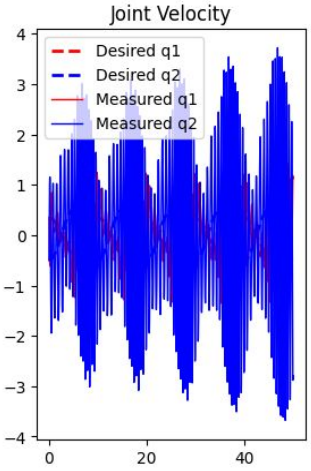
Through Inverse Dynamics

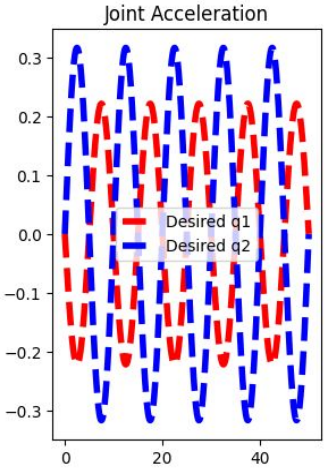


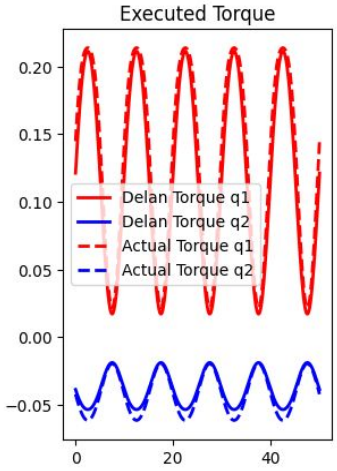
Tracking error p = 0.1276
Tracking error v = 0.6964

Through DeLaN









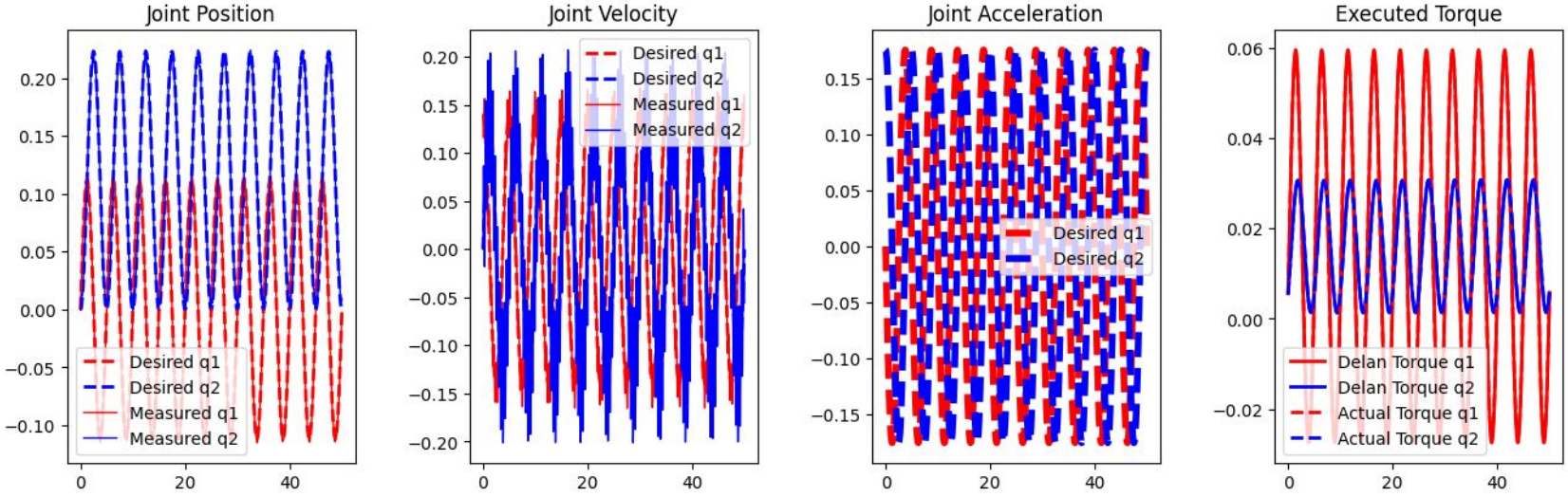
Change in frequency between train and test data

Amplitude = 0.7 x A [A = Amplitude used in training data = 1/(2pi) = 0.16 rad]

Frequency = 0.2 Hz [Frequency used in training data = 0.1 Hz]

Tracking error p = 0.00277 [Mean Absolute Error of joint position]

Tracking error v = 0.02915 [Mean Absolute Error of joint velocity]

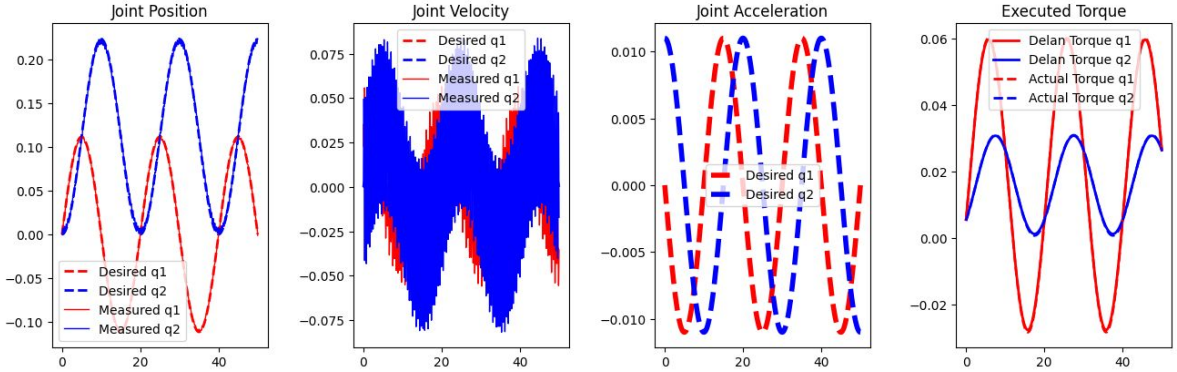


Amplitude = 0.7 x A

Frequency = 0.05

Tracking error p = 0.00168

Tracking error v = 0.0198

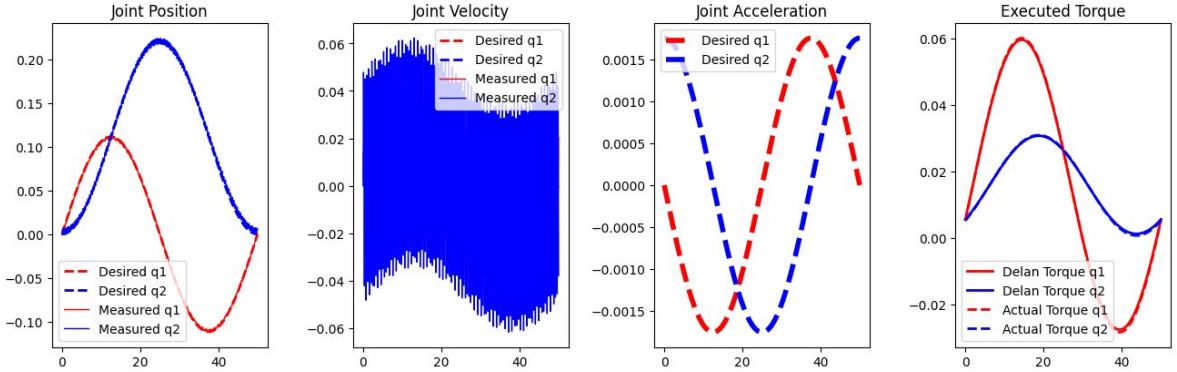


Amplitude = 0.7 x A

Frequency = 0.02

Tracking error = 0.00176

Tracking error v = 0.01965



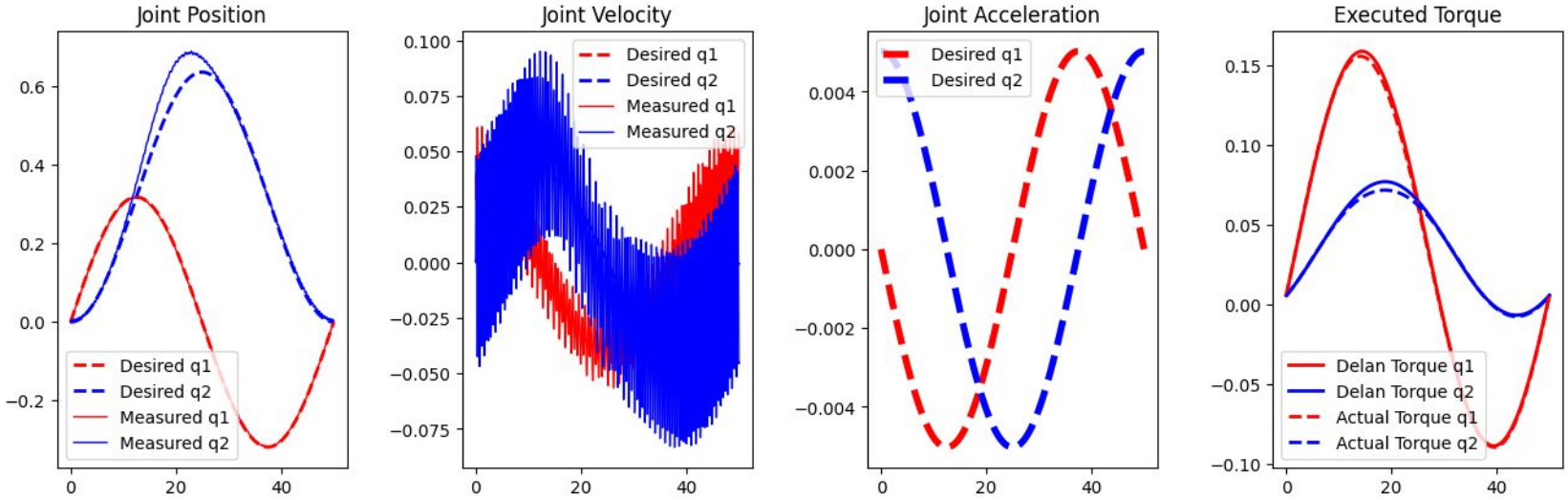
Increasing Amplitude

Amplitude = 2 x A

Frequency = 0.02

Tracking error p = 0.0166

Tracking error v = 0.0194



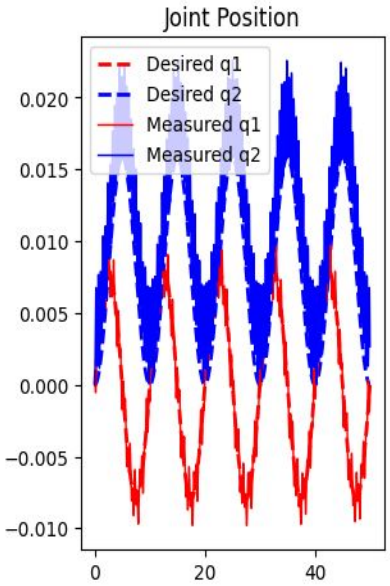
Decreasing Amplitude

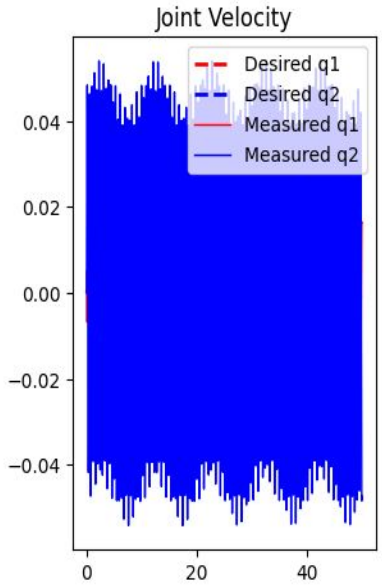
Amplitude = 0.05 x A

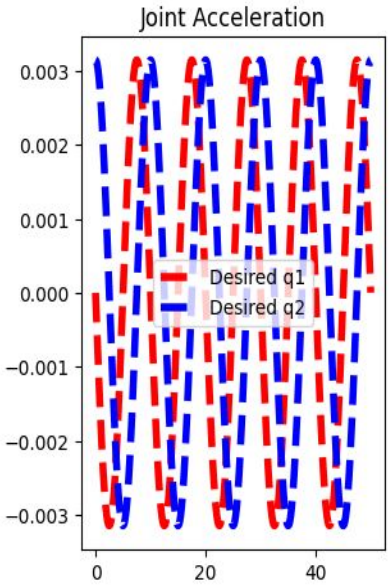
Frequency = 0.1

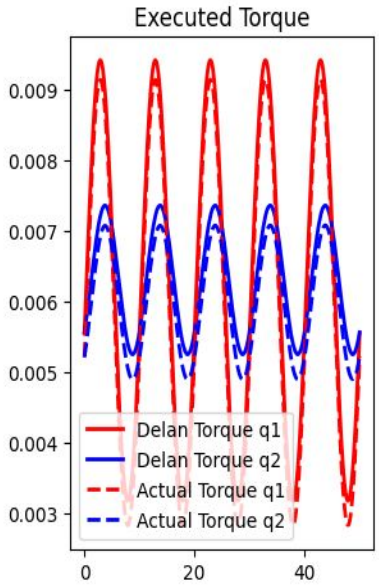
Tracking error p = 0.0209

Tracking error v = 0.0198

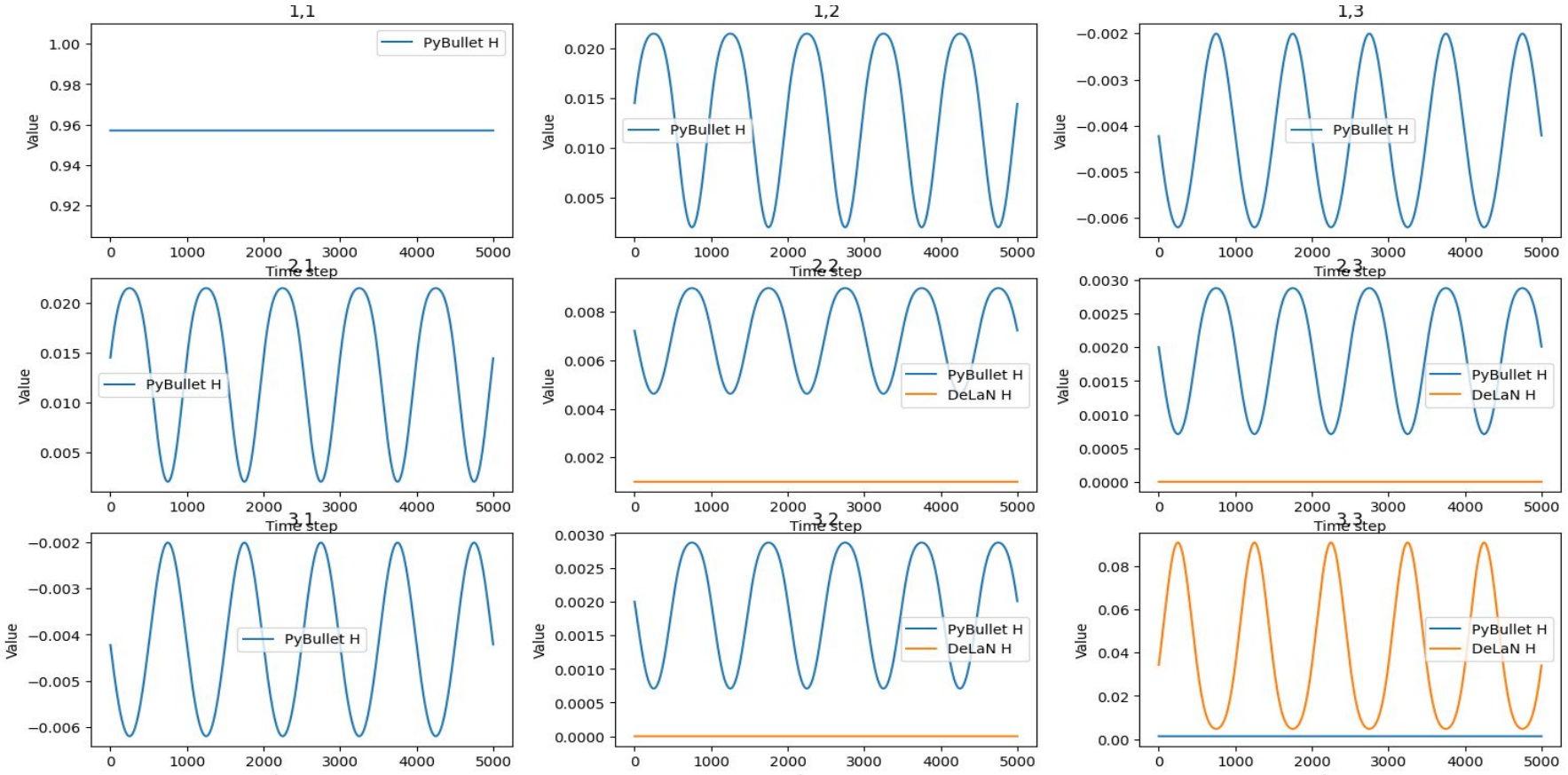








Comparison between Mass Matrix



Conclusion

- Torque can only be predicted for joint angle for which it is trained
- Hence Amplitude and offset should be inside training data
- Prediction is independent of frequency (for a range of frequency)
- It cannot learn PD control torque