Dynamic System Identification of a Haptic Wrist using ANN

F. Haghverd[1]
[1]Department of Electrical and Computer Engineering, Edmonton – AB, Canada
e-mail: haghverd@ualberta.ca

Abstract – **Model-based control systems are among most efficient and reliable robotic control systems. However, extracting manipulator's dynamic modelling is challenging. In this course project, a one-layer MLP neural network is used to model the lumped function H, which allows for estimation of joint torques based on current joint states. The model is investigated using RMSE and NRMSE metrices. It, also, is further investigated by comparing the performance with MATLAB MLP and RBF modellings that we previously discussed in the course. Compare to them, the model shows good results. However, generally speaking the model performance does not seam trustable for control implementation, and there is a need for upgrading and improving model performance before implementation.**

I. INTRODUCTION

Approaching robotic control systems, there is a class of high potential algorithms that utilize manipulators' inverse dynamics to generate compensating torques for nonlinear terms. By compensating system nonlinearities completely, these kinds of algorithms allow for designing more efficient controllers, as well as more accurate and reliable performance. Generally speaking, modelling inverse dynamics of a robotic system involves computing the required command torques in each joint given a particular desired joint configuration, i.e. joints' angle (q), angular velocity ($\dot{q}$) and angular acceleration ($\ddot{q}$). Worth mentioning that desired torque in each joint depends on the given desired trajectory to all joints.

Early days robotic research, approached dynamic modelling problem using rigid body analytical modelling tools, e.g. Lagrange formulation and Newton-Euler method (Hitzler et al. 2019). However, deriving the manipulator's inverse dynamics analytically is not considered efficient as it does not accurately consider all nonlinear system behaviours, especially frictional and backlash responses. Moreover, this common rigid body modelling tools lead to significantly tedious and computationally heavy computations.

To overcome the mentioned limitations of classical modelling approaches, data-driven learning-based algorithms have been explored recently. This approach involves learning the unknown nonlinear inverse dynamics for the robot based on input data derived from the robot while tracking optimized trajectories which excite the dynamic system (Wu et al., 2010). In this regard, huge portion of literature mainly relies on different regression algorithms. However, Yilmaz et al. deployed a neural network to infer the inverse dynamics model of the da Vinci Research Kit (Yilmaz et al., 2020).

In this course project, we tend to perform the explicit system dynamic identification of a haptic wrist in the Dr. Jagersand's lab in CS building. By explicit, I mean explicitly finding the maping function between the torque and joint states, modelling the inverse dynamics. The dataset containing joint states, angle, velocity acceleration and joint torques is collected while the robot is tracking the randomly designed Gaussian trajectory. A one-layer neural network will be used to find the lumped mapping function.

II. Related Works

Urrea et al. designed and compared different parameter identification algorithms for their 3 DOF SCARA arm, including least squares, extended Kalman filter, Adaptive Linear Neuron (Adaline) neural networks, Hopfield recurrent neural networks, and genetic algorithms (Urrea Pascal, 2018). Their assessment showed promising results for the parameters identified by the least square and Adaline neural network method. However, they mentioned how noisy data could be problematic when using least-square optimization. In this case, the least-square method can be combined with Kalman filters (Poignet Gautier, 2000). Also, Horvath et al. recently integrated extended Kalman filter methods into a neural network algorithm to also consider uncertainty during the training (Horvath Neuner, 2019). In another recent effort, Ting et al. developed a parameter identification algorithm based on the Bayesian method which not only overcome the challenge of working with noisy data by automatically detecting noise in input and output, also contain a post-processing step to preclude any physical inconsistency. Based on the premise that manipulator's dynamic always depends on current joint configuration, Hitzler et al. preseted a real-time deep learning algorithm tested on five different datasets, which showed promising results in terms of generalisation, adaptability and convergence of the network. Almost all the above mentioned works, have identified system dynamic parameters implicitly to use them for finding the system dynamic modelling with Lagrange-Euler formulation. However, Yilmaz et al. utilized 3-layer NNs for each joint to perform inverse dynamic identification of the da Vinci Research Kit explicitly (Yilmaz et al., 2020). This papre is the main refrence and inspiration fo this course project.

III. Background and Problem Formulation

Robotic dynamic equation is commonly written based on Lagrange–Euler formulation as

$$\tau = M(q)\ddot{q} + C(q,\dot{q}) + G(q) \tag{1}$$

.

where:

q - 3×1 vector of joint angles
$\dot{q}$ - 3×1 vector of joint velocities
$\ddot{q}$ - 3×1 vector of joint accelerations
τ - 3×1 vector of joint torques
M - 3×3 mass matrix
C - 3×1 vector containing velocity product terms
G - 3×1 vector of gravity terms

However, as mentioned, modelling manipulator's dynamic using that formulation extracting dynamic parameters from CAD modellings can be inaccurate and tedious. Hence, the problem is formulated here, instead, to find the mapping function H from joint states to torque, explicitly, as in Eqn.2.

$$\hat{\tau} = H(q, \dot{q}) \tag{2}$$

where $\hat{\tau}$ is the inferred joint torques based on joint states using lumped function H. It is also further investigated how does the model performance changes using both q and $\dot{q}$ or each of them as input feature.

IV. Methodology

A. Dataset

Formulated problem requires a dataset containing set of joint angles and velocities as inputs and corresponding joint torques as outputs. Data recording starts with generating desired trajectory. Here, based on joint limits, the desired trajectory is generated as a normally distributed series of joint angles. The size of the Gaussian distributions for generating desired joint positions is set to 4000, with zero-mean and standard deviations based on each joint angle limit. Next, data collection experiment is performed by commanding the robot to track the assigned trajectory and collecting joint states by subscribing to the the Joint State topic. Gaussian generated joint angles are commanded to the robot controller using built in ROS service 'JointMove' which is in fact a built-in position controller in robot computer . Respectively reaching each joint position form the trajectory set, a joint state listener function collects the published joint state includibg time, joint position, velocity, and torque (effort). All listened states are collected as a JSON data set. This data collection procedure led to 4000-data-sample recorded. The python code for data generating and collection is provided in the attached files.

B. Modelling

A one-layer MLP neural network (NN) is used here as a black-box model to identify the lumped function H. The NN is modelled as depicted in Fig.1:

- One input layer of 6 neurons for the angular position(q) and velocity ($\dot{q}$) of all joints.
- One hidden layer of 64 neurons.
- Rectified Linear Unit (ReLU) activation functions were used for each layer.
- One output layer of 3 neurons for inferred joint torques.

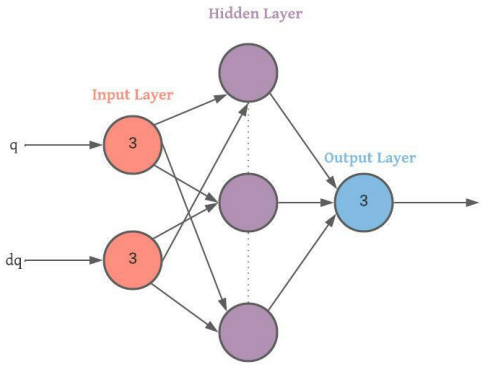
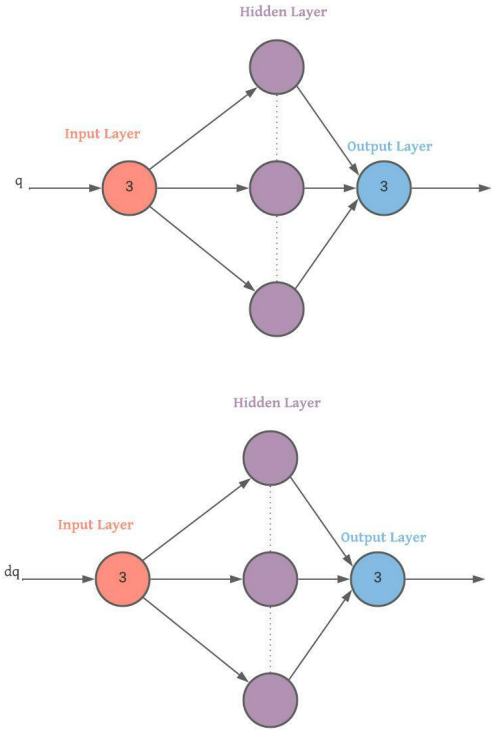


Fig. 1. Network structure for both q and dq in features

It is implemented using PyTorch and you can find the link to the Google Colab notebook at the top of attached python cod. First, the recorded JSON dataset is loaded and modified to the required shape of 4000×6 input (X) from position (q) and velocity($\dot{q}$) and 4000×3 output (Y) from effort. Also, in two separate blocks, X is shaped only using q or $\dot{q}$, to compare the model performance to previous two-feature model, Fig. 2.



y

Fig. 2. Network structures for only q or dq in features; a) q. b) dq.

Next, the whole input data is normalised using z score method, also known as standardization. In z-score normalization, normalised value (z-score) of every value in each dataset is calculated such that mean and standard deviation of all normalised values is 0 and 1 respectively. Therefore, based on mean and standard deviation of whole data, z-score is used for standardizing scores on the same scale. Following formula is used here to perform a z-score normalization on every value in the input dataset:

$$z = \frac{x - \mu}{\sigma} \tag{3}$$

where:

z - z-score value corresponding to each input data value

x - Original input data value

μ - Mean of whole data

σ - standard deviation of whole data

I performed the normalization on whole input data before splitting data, so that I do not need to denormalize at any point. However, for new unseen data, there is a need for denormalization based on previously calculated mean and standard deviation. Also, for checking correct normalization, you can observe that mean and standard deviation of normalized X is 0 and 1 respectively.

Afterward, the data is splitted into training, validation and testing datasets where training is used for the learning process, validation for evaluating the best model among epochs, and testing for evaluating the final model on unseen data.

The NN is modeled with explained structure. We use the mean squared loss function for both the training criteria and also the validation loop. Adam optimizer is used in PyTorch. Batch size is set to 32 and max epoch number to 1000. After training each epoch, in the validation loop, the loss is computed for validation dataset and compared with the best loss value to recognize and save the best model. In this part, also, two separate blocks are provided for when having two features or one feature, as it affects the NN structure.

Finally, for model evaluation, to be consistent with metrices used in the literature, root mean square error (RMSE), Eqn.4, and normalised RMSE (NRMSE), Eqn. 5, is used. Last block in the code, loads saved best model and uses hat model to estimate τ for both training dataset and unseen data which is the testing dataset.

$$RMSE = \sqrt{\frac{\Sigma_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}} \tag{4}$$

$$NRMSE = \frac{\sqrt{\frac{1}{N}\Sigma_{i=1}^{N}(y_i - \hat{y}_i)^2}}{y_{max} - y_{min}} \tag{5}$$

where:

N - number of data points

y - actual output

$\hat{y}$ - estimated output

V. Results and Discussion

Table I and Fig. 1 depict how RMSE and NRMSE change for choosing different features and for testing and training dataset. As expected, testing errors are higher than training errors, but the difference is acceptable, and so there is no concern over overfitting. Generally speaking, the error ranges are acceptable but they are not great, and for real world implementation, we should improve and upgrade model accuracy. I have added normalisation and changed the model structure iteratively to increase the model accuracy, and the resulting model is the best version achieved. There may be, also, problems with the dataset and how it is collected. Because dq feature is too sparse that may affect the training result.

Investigating how choosing different features affect the model performance, as expected, only choosing dq is not performing good. Also, based on the nature of the problem, I expected to get much better results when using both q and dq compared to only q. However, as you can see in the table and the bar chart, they are almost the same. I believe, the mentioned problem with sparse dq is also the reason for the observation here too.

TABLE I

Performance evaluation of our model

Features	**q , dq**	**q**	**dq**
Training RMSE	0.1266	0.1276	0.1641
Training NRMSE	0.1492	0.1504	0.1934
Testing RMSE	0.1292	0.1293	0.1669
Testing NRMSE	0.1466	0.1469	0.1895

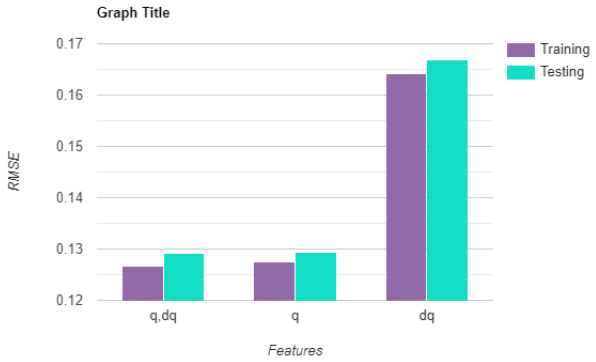


Fig. 3. RMSE for training and testing dataset using different features

To perceive a better perspective over the results, I also loaded the dataset into MATLAB (it may take a while if you try running the MATLAB codes), and used previously performed MLP and RBF NN for estimating our model here too. The results for comparing our method with them is depicted in the II and Fig.4. Our MLP clearly depicts better performance compared to MATLAB MLP and RBF.

TABLE II

Performance evaluation for our MLP and MATLAB MLP and RBF

	My MLP	**MATLAB MLP**	**MATLAB RBF**
Training RMSE	0.1266	0.1670	0.1210
Testing RMSE	0.1292	0.1675	0.1505

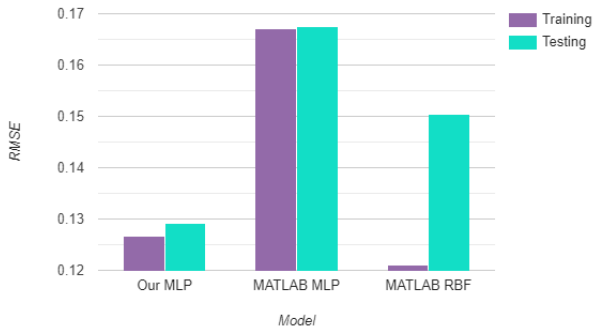


Fig. 4. RMSE for training and testing datase for our MLP and MATLAB MLP and RBF

VI. References

[1] Hitzler, K., Meier, F., Schaal, S., Asfour, T. (2019). Learning and Adaptation of Inverse Dynamics Models: A Comparison. In 2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids). https://doi.org/10.1109/humanoids43949.2019.9035048.

[2] Poignet, P., Gautier, M. (n.d.). Comparison of weighted least squares and extended Kalman filtering methods for dynamic identification of robots. In Pro- ceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automa- tion. Symposia Proceedings (Cat. No.00CH37065). https://doi.org/10.1109/robot.2000.845296.

[3] Urrea, C., Pascal, J. (2018). Design, simulation, comparison and evaluation of parameter identification methods for an industrial robot. In Computers Electrical Engineering (Vol. 67, pp. 791–806). https://doi.org/10.1016/j.compeleceng.2016.09.004 Wu, J., Wang, J., You, Z. (2010).

[4] Wu, J., Wang, J., You, Z. (2010). An overview of dynamic parameter identification of robots. Robotics and computer-integrated manufacturing, 26(5), 414-419.

[5] Yilmaz, N., Wu, J. Y., Kazanzides, P., Tumerdem, U. (2020). Neural Network based Inverse Dynamics Identification and External Force Estimation on the da Vinci Research Kit. In 2020 IEEE International Conference on Robotics and Automation (ICRA). https://doi.org/10.1109/icra40945.2020.9197445.

[6] Stürz, Y. R., Affolter, L. M., Smith, R. S. (2017). Parameter Identification of the KUKA LBR iiwa Robot Including Constraints on Physical Feasibility. In IFAC-PapersOnLine (Vol. 50, Issue 1, pp. 6863–6868). https://doi.org/10.1016/j.ifacol.2017.08.1208.

[7] Montazeri, A., West, C., Monk, S. D., Taylor, C. J. (2017). Dynamic modelling and parameter estimation of a hydraulic robot manipulator using a multi-objective genetic algorithm. In International Journal of Control (Vol. 90, Issue 4, pp. 661–683). https://doi.org/10.1080/00207179.2016.1230231.

VII. Running Code Instructions

For each time you want to run sequence of block, please start with the restart block at the beginning and the run the blocks respectively, based on what features you want to use. Also, dataset 'ID-data.json' need to be uploaded first to the Google Colab.