Machine Learning for Robotics RBE 577

Project 2: Push & Learn: Learning Dynamics through learning

DUE: Tuesday February 24 at 11:59 pm.

All written components should be typeset using LaTeX. A template is provided on Overleaf. However, you are free to use your own format as long as it is in LaTeX.

Submit two files **a)** your code as a zip file **b)** the written report as a pdf file. If you work in pairs, only **one** of you should submit the assignment, and write as a comment the name of your partner in Canvas. Please follow this formatting structure and naming:

1 Overview

In this programming assignment, you will develop different models to predict the motion of an object pushed by a robot. The goal is to compare the accuracy and effectiveness of:

- A physics-based model.
- A neural network-based model.
- A hybrid model that incorporates both physics and neural networks.

Your task is to implement these models, train them on provided data, and analyze their performance using appropriate evaluation metrics.

2 Core Requirements

1. Implement three different models:
    - **Physics Model:** Uses known physical equations to predict object motion.
    - **Neural Network Model:** A data-driven model that learns the push dynamics from data.
    - **NN Model + Physics:** A model that combines the physics-based equations with neural network predictions.
2. Compute and compare the loss of each model.
3. Visualize and analyze the results with plots comparing predictions and ground truth data.
4. Summarize your findings and discuss the strengths and limitations of each approach.

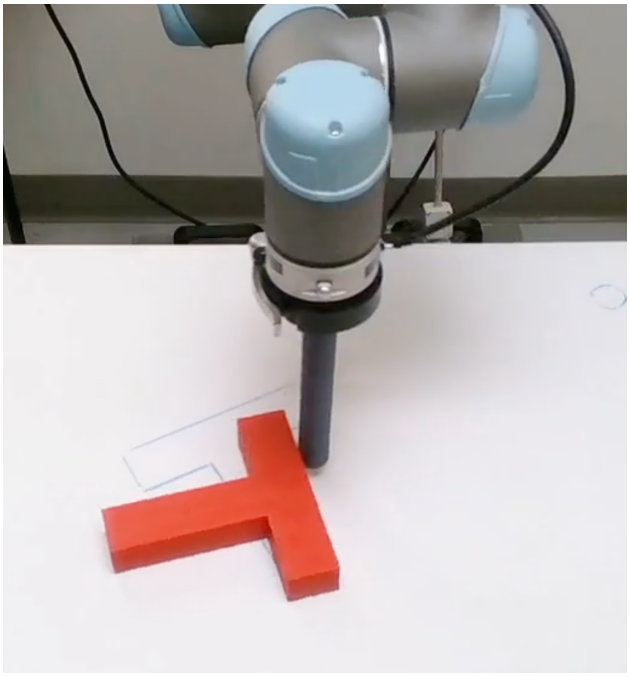

Figure 1: Example of a UR10 Robot pushing a T Block

3 Project Structure

Your submission should follow this structure:

```
project_YOUR_NAME.pdf
project_YOUR_NAME.zip
    src
        config
            default.yaml
        lib
            models.py
            physics.py
        helpers
            utils.py
            config.py
        main.py
        train.py
        results
            training_curves.png
            predictions.png
        requirements.txt
        README.md
```

4 Implementation Details

5 Physics-Based Model (30 points)

In this section, you are required to implement a **physics-based model** to predict the motion of an object subject to a push from a robotic manipulator. The objective is to compute the final position and orientation of the object given the initial conditions and the applied push parameters.

5.1 Provided Data

You are provided with a dataset containing:

- The **initial state** of the object: $\mathbf{s}_{start} = [x_0, y_0, \theta_0]$, representing its initial position and orientation.
- The **final state** of the object: $\mathbf{s}_{final} = [x_f, y_f, \theta_f]$, representing its observed position and orientation after the push. *This final state serves as the ground truth and is stored in the file* `data_y_cracker_box.npy`.
- The **push parameters** θ_{push}, d, D, which are provided in the file `data_x_cracker_box.npy` and represent the applied force characteristics.
- The **push duration** T, which denotes the time interval over which the push is applied.
- The **mass** m and **size** s of the object are specified in the `config/default.yaml` file.
- The **physical equations** governing the motion of the object under the influence of the applied force (see Appendix).

5.2 Implementation Requirements

Your task is to implement a function that predicts the **final state** $\mathbf{s}_{predicted}$ of the object using the provided physics equations. Specifically, you must:

- Use the provided physics-based equations to model the motion of the object.
- Implement numerical integration to update the object's position and orientation over the duration T.
- Compare the predicted final state $\mathbf{s}_{predicted}$ with the ground truth final state $\mathbf{s}_{final}$ from the dataset.
- Compute the prediction error using the Mean Squared Error (MSE):

$$\text{Loss} = \frac{1}{N} \sum ||\mathbf{s}_{predicted} - \mathbf{s}_{final}||^2 \tag{1}$$

- Provide an analysis of the physics model's performance.

5.3 Expected Inputs and Outputs

- **Input:** $\mathbf{s}_{start}, T, \theta_{push}, d, D$ (initial state, push duration, push parameters)
- **Output:** $\mathbf{s}_{predicted}$ (predicted final state)

5.4 Analysis and Discussion

Upon implementing the physics model, you must analyze its performance by addressing the following questions:

- How well does the predicted final state $\mathbf{s}_{predicted}$ match the observed final state $\mathbf{s}_{final}$?

Your findings should be documented with relevant explanations and visualizations, such as plots comparing predicted and actual trajectories.

5.5 Neural Network Model (30 points)

Train a neural network (MLP) to learn the push dynamics from data.

- Define an MLP with at least one hidden layer.
- Train the network using the given dataset.
- Compute and visualize loss curves.
- Compare the network's performance with the physics model.

5.6 Hybrid Physics + Neural Network Model (40 points)

Implement a model that uses both physics and neural networks.

- Use the physics model's predictions as input features to the neural network.
- Train the network and evaluate if it improves prediction accuracy.
- Analyze how much the physics model contributes to performance.

6 Evaluation and Discussion

- Compare the loss values for all three models.
- Analyze the advantages and disadvantages of each approach.
- Discuss under what conditions each model would be preferred.
- Provide plots showcasing the accuracy of the predictions.

Appendix

Push Physics Model

1. Object Properties

$$\text{Mass: } m$$
$$\text{Size: } s$$
$$\text{Moment of Inertia: } I = \frac{1}{12}ms^2 \text{ (for square object)}$$

2. Push Parameters

$$\text{Rotation: } \theta_0 \text{ (initial orientation)}$$
$$\text{Side: } d \text{ (contact point distance)}$$
$$\text{Distance: } D \text{ (total push distance)}$$

3. Velocity Profile

$$v(t) = v_{max}\left(\frac{1}{2}\sin\left(\frac{2\pi t}{T} - \frac{\pi}{2}\right) + \frac{1}{2}\right)$$
$$v_{max} = \frac{2D}{T}$$

4. Motion Equations

a) Angular Motion:

$$\text{Torque: } \tau = F_t d = mv(t)d$$
$$\text{Angular acceleration: } \alpha = \frac{\tau}{I}$$
$$\text{Angular velocity: } \dot{\theta} = \int \alpha\, dt$$
$$\text{Angular position: } \theta = \int \dot{\theta}\, dt$$

b) Linear Motion (Local Frame):

$$\dot{x} = -v(t)\cos(\theta)$$
$$\dot{y} = -v(t)\sin(\theta)$$
$$x = \int \dot{x}\, dt$$
$$y = \int \dot{y}\, dt$$

5. Frame Transformation

$$\begin{bmatrix} x_{global} \\ y_{global} \end{bmatrix} = \begin{bmatrix} \cos(\theta_0) & -\sin(\theta_0) \\ \sin(\theta_0) & \cos(\theta_0) \end{bmatrix} \begin{bmatrix} x_{local} \\ y_{local} \end{bmatrix}$$

6. Numerical Integration

For each timestep Δt:

1. Velocity: $v_i = v(t_i)$
2. Torque: $\tau_i = m v_i d$
3. Angular Update:
$$\begin{cases} \alpha_i = \tau_i / I \\ \Delta\theta_i = \frac{1}{2}\alpha_i \Delta t^2 \\ \theta_i = \theta_{i-1} + \Delta\theta_i \end{cases}$$
4. Position Update:
$$\begin{cases} \Delta x_i = -v_i \cos(\theta_i)\Delta t \\ \Delta y_i = -v_i \sin(\theta_i)\Delta t \\ x_i = x_{i-1} + \Delta x_i \\ y_i = y_{i-1} + \Delta y_i \end{cases}$$

Final State

$$s_{final} = [x_{global}, y_{global}, \theta_{final}]$$