# Robot Learning
# Weekly Exercise 2

Marc Toussaint & Wolfgang Hönig
Learning & Intelligent Systems Lab, Intelligent Multi-Robot Coordination Lab, TU Berlin
Marchstr. 23, 10587 Berlin, Germany

Summer 2024

## 1 Work with the Literature

[The links to literature sometimes point to journal sites, but they should be accessible from within TU Berlin.]

a) Have a look at Eq. (1) of

H. T. Siegelmann, B. G. Horne, and C. L. Giles. Computational capabilities of recurrent NARX neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2):208–215, 1997. URL: https://ieeexplore.ieee.org/abstract/document/558801/

This paper describes a classical "NARX" model. Consider the discrete time dynamics

$$v_{t+1} = v_t + u_{t-3} \tag{1}$$
$$p_{t+1} = p_t + \tau v_{t-2} \tag{2}$$
$$y_t = p_t \ , \tag{3}$$

with variables $(p_t, v_t)$, controls $u_t$, and sensor observation $y_t$. $\tau \in \mathbb{R}$ is a fixed constant. (In words: the control directly adds to the velocities – but with a delay of 3 steps! And the velocities add to the position – but with a delay of 2 steps! And we only observe position $p_t$, not velocities.)

Could the "NARX" model described in the paper above learn this dynamics? How would you have to choose $n_u$ and $n_y$?

We rewrite the dynamics in NARX form:

$$y_t = p_t = p_{t-1} + \tau v_{t-3} \tag{4}$$
$$= p_{t-1} + \tau[v_{t-4} + u_{t-7}] \tag{5}$$
$$= p_{t-1} + \tau[\tfrac{1}{\tau}(p_{t-1} - p_{t-2}) + u_{t-7}] \tag{6}$$
$$= 2p_{t-1} - p_{t-2} + \tau u_{t-7} \ . \tag{7}$$

Which is in NARX form for $n_u = 7$ and $n_y = 2$.

b) Also have a look at Eq. (1) of

M. P. Deisenroth, D. Fox, and C. E. Rasmussen. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, 2015-02. URL: http://ieeexplore.ieee.org/document/6654139/, doi:10.1109/TPAMI.2013.218

This is also called a state-space model. How can you define $x_t$ for our dynamics above so that it can be represented in that form (1)?

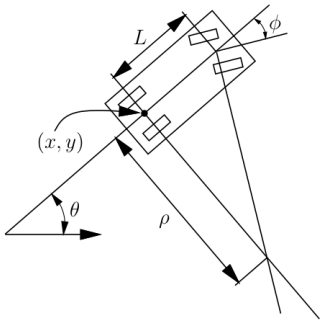
Go back to the original equations (1-3). We need to define $x_t$ to be "all relevant things on the RHS" of eqns (1-3). $x_t$ definitely needs to contain $p_t$ to predict $y_t$, but also $v_{t-2}$ to predict $p_{t+1}$, and therefore also $v_{t-2}$ and $u_{t-5}$ to predict $v_{t-1}$. So let's try

$$x_t = \begin{pmatrix} p_t \\ v_{t-2} \\ u_{t-1} \\ u_{t-2} \\ u_{t-3} \\ u_{t-4} \\ u_{t-5} \end{pmatrix}, \quad \text{and} \quad x_{t+1} = \begin{pmatrix} p_{t+1} \\ v_{t-1} \\ u_t \\ u_{t-1} \\ u_{t-2} \\ u_{t-3} \\ u_{t-4} \end{pmatrix} = \begin{pmatrix} p_t + \tau v_{t-2} \\ v_{t-2} + u_{t-5} \\ u_t \\ u_{t-1} \\ u_{t-2} \\ u_{t-3} \\ u_{t-4} \end{pmatrix} = f(x_t, u_t) \ . \tag{8}$$

## 2 System Identification of a Simple Car

Consider the dynamics model of a first order car with states $q = (x, y, \theta)^\top$ (position and orientation), actions/controls $u = (s, \phi)^\top$ (speed and steering wheel angle), and known dynamics

$$\dot{q} = f(q, u) = \begin{pmatrix} s\cos\theta \\ s\sin\theta \\ \frac{s}{L}\tan\phi \end{pmatrix}. \tag{9}$$




Here, $L$ is the distance between the wheels and not known.

a) Assume you have an example trajectory $D = \{(x_t, y_t, \theta_t, s_t, \phi_t)\}_{t=1}^n$, where individual datapoints were sampled at 10Hz. Formulate an optimization problem that computes the "best" $L$ for the given data.

$$L^* = \underset{L}{\text{argmin}} \sum_{t=1}^{n-1} \left(L - \frac{s_t}{\dot{\theta}_t}\tan\phi_t\right)^2, \tag{10}$$

where $\dot{\theta}_t$ is estimated numerically from the data (which is why we have $n-1$ rather than $n$ data points). A basic estimate is $\dot{\theta}_t \approx \frac{d(\theta_{t+1}, \theta_t)}{\Delta t}$, where $d(\theta_{t+1}, \theta_t)$ is a distance metric in SO(2).

An alternative error function is

$$L^* = \underset{L}{\text{argmin}} \sum_{t=1}^{n-1} \left(\dot{\theta}_t - \frac{s_t}{L}\tan\phi_t\right)^2. \tag{11}$$

b) Find a closed-form solution for your optimization problem in a).

Since it's a quadratic problem, we can find the global extrema when the gradient is zero:

$$2\sum_{t=1}^{n-1} \left(L^* - \frac{s_t}{\dot{\theta}_t}\tan\phi_t\right) = 0 \tag{12}$$

$$(n-1)L^* = \sum_{t=1}^{n-1} \frac{s_t}{\dot{\theta}_t}\tan\phi_t \tag{13}$$

(In other words, taking the mean over the estimated individual $L$'s.)

## 3 Mountain Car Dynamics Learning

This is a coding exercise. Please bring your laptop and connect to the HDMI in the tutorial to show your results. (If you upload a pdf, just include a screenshot of results in the pdf.)
Install the mountain car simulation of *gymnasium* (https://gymnasium.farama.org/) using

```
pip install gymnasium[classic-control]
```

The following code simulates a few steps and collects data for a dynamics learning problem:

```
import gymnasium as gym
import numpy as np
env = gym.make('MountainCarContinuous-v0', render_mode='human')

# for this problem observation=state

u_dim = env.action_space.shape[0]
x_dim = env.observation_space.shape[0]
```

```
data_input = np.zeros((0,x_dim+u_dim))
data_target = np.zeros((0,x_dim))
n_data = 200

x_state, info = env.reset()

for t in range(n_data):
    # u_controls = env.action_space.sample()  # agent policy that uses the observation and info
    u_controls = np.sin([.01*t])
    x_prev = x_state
    x_state, reward, terminated, truncated, info = env.step(u_controls)
    # terminated = a terminal state (often goal state, sometimes kill state, typically with pos/neg reward) is reached;
    #    formally: the infinite MPD transitions to a deadlock nirvana state with eternal zero rewards
    # truncated = the simulation is 'artificially' truncated by some time limited - that's actually formally inconsistent to the definition of an infinite MDP

    data_input = np.vstack([data_input, np.concatenate([x_prev, u_controls])])
    data_target = np.vstack([data_target, x_state])

    if terminated or truncated:
        if truncated:
            print('-- truncated -- should not happen!')
        else:
            print('-- terminated -- goal state reached')
        x_state, info = env.reset()

env.close()

print('input data:', data_input.shape)
print('output data:', data_target.shape)
```

a) Increase the amount of data you collect (e.g. to $n = 1000$) and learn a regression from the input to output. Use whatever ML techniques you learned about in previous courses. Also linear regression is an option, which should work particularly well if you happen to include $\cos(3x_0)$ as a feature (where $x_0$ is the first entry of $x$: the position; see the domain documentation).

b) The above might not work well (in the sense of generalizing to the full state space), because the controller generating the data (`u_controls = np.sin([.01*t])`) is not very explorative. Play around with alternatives that generate much better data for learning.

Following the typical 'grid sampling' of frequency inputs when analyzing linear systems in frequency space, I modified exploration to

```
freq = 1+t//100
u_controls = np.sin([.01*freq*t])
```

c) Assume that you could only observe the position $x_0$ of the car, not the velocity $x_1$. As the state is not fully observable, you'll need to learn an autoregression model with longer input window. Modify the code above so that the data only contains positions and controls as input, and predicts the next position.

Simple for this simulation: store data $x_{t\text{-}2}, x_{t\text{-}1}, u \mapsto x_t$ with $x$=position only. In this particular simulation, the difference $x_{t\text{-}1} - x_{t\text{-}2} = v_{t\text{-}1}$ is the velocity.

d) [Added for the tutorial session, to show you an easy way of how to make use of a learned model.] First, since we know this is a physical system with observed position $q$ and velocity $\dot{q}$, let's also treat is like that: The *forward* dynamics is a mapping $q, \dot{q}, u \mapsto \ddot{q}$, while the *inverse* dynamics is the mapping $q, \dot{q}, \ddot{q} \mapsto u$. Learn the inverse dynamics function (define $\ddot{q}$ as the change in velocity by a simulation step). Then use the inverse dynamics to impose a PD behavior

$$\ddot{q}^* = k_p(q^* - q) - k_d\dot{q}$$

with $q^* = 2$ and $k_p = m/\tau^2$ , $k_d = 2m\xi/\tau$ (exactly as in last exercise solution), and $\tau = 50$, $\xi = 0.9$.

# References

[1] M. P. Deisenroth, D. Fox, and C. E. Rasmussen. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, 2015-02. URL: http://ieeexplore.ieee.org/document/6654139/, doi:10.1109/TPAMI.2013.218.

[2] H. T. Siegelmann, B. G. Horne, and C. L. Giles. Computational capabilities of recurrent NARX neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2):208–215, 1997. URL: https://ieeexplore.ieee.org/abstract/document/558801/.