DTU ELECTRO
34752 Course Bio-inspired control for robots
Programming exercises

Teacher Assoc. Prof. Silvia Tolu, *stolu@dtu.dk*, b.326, r.106, NRT-Lab
Teaching Assistants (TAs):
Anders Eiersted Molzen, *s194024@student.dtu.dk*, b.326, NRT-Lab
Júlia Rey Vilches, *jrevi@dtu.dk*, b.326, r.110, NRT-Lab

August 2025, **Part II Bio-inspired Control**

Contents

2 Concepts of bio-inspired control

Exercise 2.1 Feedback controller with plant delay

The file `exercise1.py` contains code to run a simple feedback controller with a first-order plant. The output of the system, y, is initialized at 1.0 and adjusted according to the gain, K, for simlen=30 time steps by the difference between its current value and the target value, target=0. The output, y, is then plotted over all time steps.

1. Implement varying delay levels such that the controller sees the output $y(t-\delta)$ where δ is the time delay. Plot the output of the system for delay values 0, 1, 2, and 3 time steps. How does the feedback controller perform in the presence of delay?

   Solution:

2. Vary the gain of the feedback controller. Does this help to deal with feedback delay? What happens to the speed of convergence?

   Solution:

Exercise 2.2 Target reaching with delay

The file `exercise2.py` simulates control of a two-joint arm using a feedback controller model. The arm reaches 4 targets. In each time step, the code does the following:

- The desired wrist position is computed using a min-jerk planner (minjerk.m).
- The desired wrist position is converted into a set of target joint angles (invkinematics.m).
- PD controllers are used to compute the desired torque for these target joint angles (pdcontroller.m).
- The desired torques are passed on to the plant (plant.m) to control the arm and the forward kinematics model (fkinematics.m) to compute the resulting wrist position.

In this exercise, do the following:

1. Add noise to the output of the PD controller by varying the coefficient of variation of the noise. You can use `randn` to generate random numbers. What happens to the reaching at various noise levels?

   Solution:

2. Set the noise coefficient to zero and add a delay to the inputs that the PD controller receives. How does this affect reach performance?

   Solution:

3. Adjust the gain and damping parameters of the PD controllers (kp, and kd). Can the system cope with delays with suitable parameter values?

   Solution:

Exercise 2.3 Feedback and feedforward control

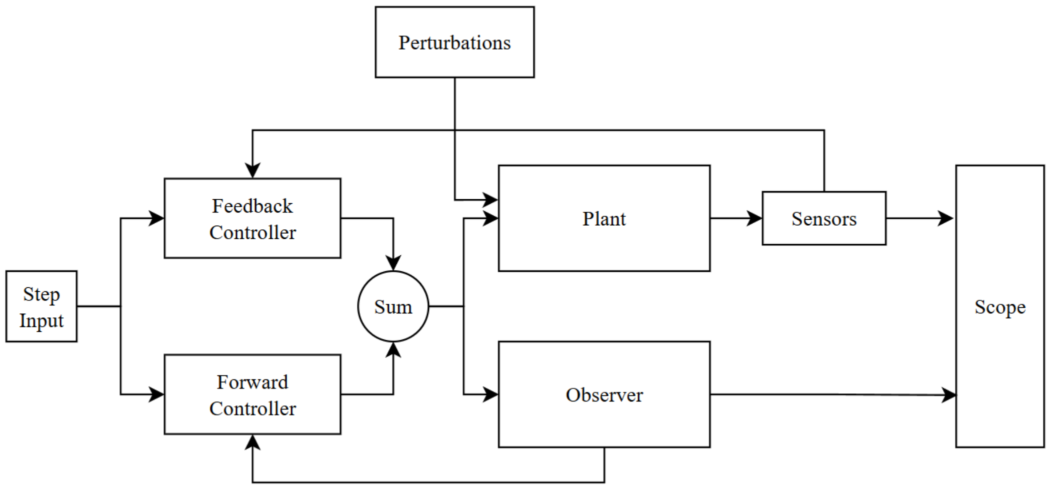

Figure 1: Diagram showing the Feedback-Forward control model.

The file `exercise3.py` simulates control of a single joint arm using a combination of feedback and feed-forward control. In each time step, a forward model of the arm is used to estimate the angle, velocity, and acceleration of the joint. This is used to compute a forward model torque which is combined with a feedback-based torque which is computed using a delayed copy of the actual joint angle. During the movement, a disturbance is applied to the joint. The actual, predicted, delayed, and target trajectory of the joint angle, as well as the perturbation and the total torque applied, is plotted over the time steps of the simulation.

1. Turn off the forward model by setting Kp_forward and Kd_forward to 0.0, turn off the perturbation by setting pert_amp to 0.0, and vary the delay by changing the value of delay in increments from 0.0 to 0.125. What happens when only feedback control is used as the sensory delay increases?

   Solution:

2. With the forward model turned off and the delay set to 0.0, turn the perturbation back on by setting pert_amp to 1. How does the feedback model perform when faced with a disturbance?

   Solution:

3. Turn the forward model back on and turn off the feedback model by setting Kp_feedback and Kd_feedback to 0.0. Turn off the disturbance and set the delay to 0.125. Is the forward model affected by delay?

   Solution:

4. With the feedback model turned off, set the delay to 0.0 and turn the perturbation back on by setting pert_amp to 1. How does the forward model alone deal with perturbation?

   Solution:

Exercise 2.4 Smith predictor with learned model

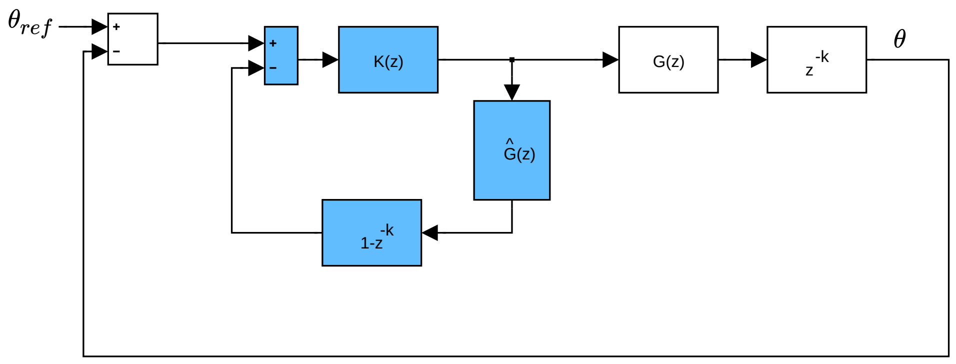

Figure 2: Smith predictor control architecture. $\hat{G}(z)$ is the learned model.

The file `exercise4.py` contains a Smith predictor control architecture as shown in Figure 2, where the plant model is a learned model. The estimated acceleration is computed as a weighted combination of the feedback torque and the estimated velocity. The weights are adjusted using gradient descent with the difference between the actual and estimated acceleration as the learning signal. Training is run for 40 trials, and actual and estimated velocity and acceleration are plotted for the first and last trials.

1. Adjust the learning rate by changing the value of alpha. What effect do different values have on learning? Can the value become too high? Why?

   Answer:

2. Plot the changes in the weights over the learning trials. Do the weights converge?

   Answer:

3. Here, the model is utilized to solve the issue of plant delay. What happens if the delay time is not known exactly? Try changing the `delay_estimate` variable. Does the training still work?

   Answer:

Exercise 2.5 Introduction to CMAC

In this and the following exercise, you will work with a variant of the cerebellar model articulation controller (CMAC) as illustrated in Figure 3.

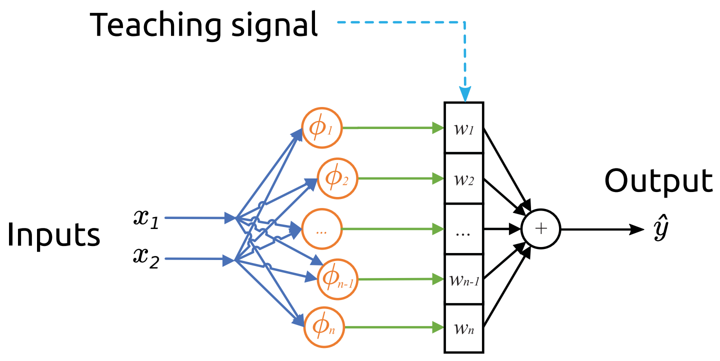

Figure 3: Illustration of the CMAC structure.

The CMAC controller consists of a series of "receptive fields" which can be activated by the input. The idea of the CMAC is to act like a kind of adaptive memory which can be used for feed-forward control. Each receptive field is activated only in a small area of the input space. For a single input dimension, the activation of the receptive fields is computed with Gaussian basis functions like so:

$$\phi_i(x) = \exp\left(-\frac{(x-\mu_i)^2}{\sigma^2}\right), \; i = 1, 2, \ldots, N. \tag{1}$$

The activation of a receptive field ,$\phi_i(x)$, is a number between 0 and 1. A value close to 0 means that the particular receptive field does not match the given input, and a value close to 1 means that the particular receptive field contains information (a memory) about the given input x.

We will begin with the case of a single input and output dimension. Your first task is to use the CMAC to learn a function. Open the `exercise5.ipynb` jupyter notebook for this exercise, where the rest of the instructions will be.

Write your solution here:

Exercise 2.6 Control with CMAC

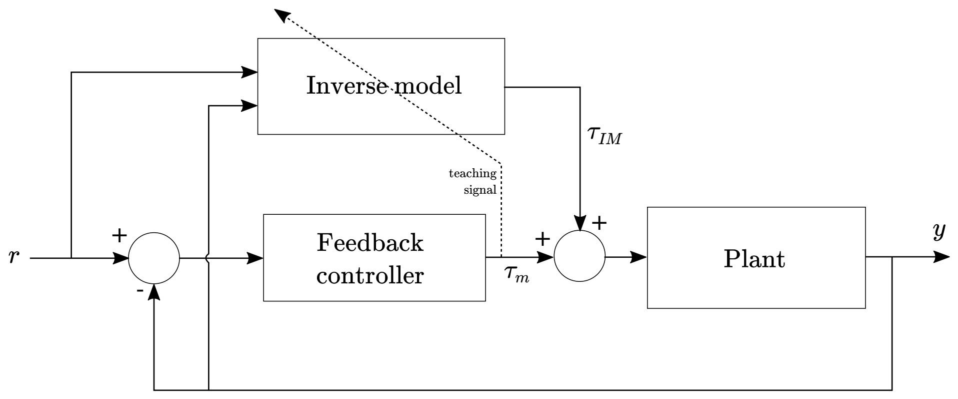

Figure 4: Feedback error learning architecture.

A feedback controller is used to guide the learning process in the beginning and provide a teaching signal $e = \tau_m$. Here, we select the input to the CMAC as the reference and measured angles, such that $x_1 = y = \theta$ and $x_2 = r = \theta_{ref}$
For multidimensional input the bases have to cover all of the input dimensions. If we define the bases in each dimension, in this case 2:

$$\phi_{ki} = \exp\left(-\frac{(x_k - \mu_{ki})^2}{\sigma_k^2}\right),\ i = 1, 2, \ldots, N\ ,\ k = 1, 2\ , \tag{2}$$

we can define an activation matrix like so:

$$B_{ij} = \phi_{1i}\phi_{2j} = \exp\left(-\frac{(x_1 - \mu_{1i})^2}{\sigma_1^2} - \frac{(x_2 - \mu_{2j})^2}{\sigma_2^2}\right). \tag{3}$$

The output and covariance learning rule is then:

$$\hat{y} = \sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}B_{ij}, \tag{4}$$

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} + \beta e B_{ij}. \tag{5}$$

In this exercise, we apply the Feedback Error Learning (FEL) scheme to a single joint arm under the influence of gravity using joint space control. The FEL control architecture is illustrated in Figure 4. In this exercise, we will keep our implementation of the CMAC controller in one file and run experiments in another.

1. In the file `cmac2.py` is a 2D CMAC implementation using the above equations which you can use for this exercise. Open `exercise6.py` and run the code. It simulates a single joint arm with the FEL architecture, using a PD controller and the CMAC (omitted initially). The joint reference signal is chosen as $\theta_{ref} = \frac{\pi}{4}$.

   Outcome:

2. Tune the PD controller to get sufficient performance on the step reference. It should not be too fine-tuned.

   Outcome:

3. Implement a periodic reference trajectory as $\theta_{ref} = A \sin\left(2\pi \frac{t}{T}\right)$, initially with $A = \pi$ and $T = 5$ seconds. Defining an iteration of this trajectory as a trial, the simulation loop should go through `n_trials`.

   Outcome:

4. Implement the CMAC controller into the loop, using the feedback torque τ_m as error signal. An example of how to call the CMAC functions is provided in the class definition file.

   Outcome:

5. Plot the mean square position error in a trial using the given code. After how many trials does the average trial error stop decreasing? Try different learning rates β.

   Outcome:

6. When the CMAC controller learning has converged, try running the controller for one trial where you have increased the reference trajectory frequency by reducing the period T. Does the learned model generalize to the higher frequency, or would it have to learn again? What about lower frequencies?

   Outcome:

7. Is it possible to include the joint velocity as an input to the CMAC? Look at the 2D CMAC implementation code. Would there be any issues involved in expanding the CMAC for input of any dimension?

   Outcome:

Exercise 2.7 Adaptive filter controller

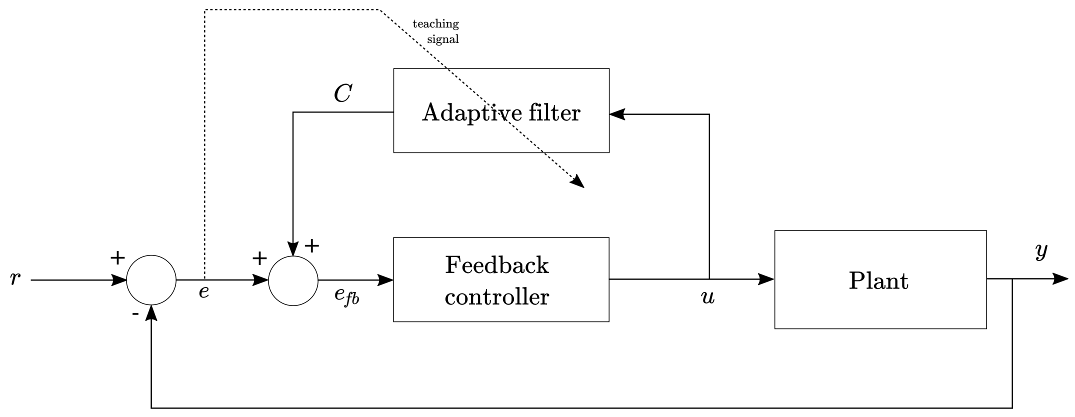

Figure 5: The adaptive filter in a recurrent control architecture.

The structure of the adaptive filter is similar to that of the CMAC. However, there are some distinct differences. The basis functions consists of a fixed set of filters that serves as a short time memory.
Here we will use a second order linear filter, as defined by the following transfer function:

$$G_i(s) = \frac{1}{(\tau_{1i}s+1)(\tau_{2i}s+1)} \tag{6}$$

Given a range of such filters with a range of time constant such that $\tau_{1i} \in [2, 50]$ ms and $\tau_{2i} \in [50, 750]$ ms.
The output C_j of the adaptive filter is a linear combination of the filter outputs $g_i(t)$:

$$C_j(t) = \sum_i w_{ij} g_i(t) \tag{7}$$

We will use the model in the recurrent control architecture that is illustrated in Figure 5.
The error signal $e(t) = r(t) - y(t)$ is summed with a contribution from the adaptive filter controller:

$$e_{fb}(t) = e(t) + C(t). \tag{8}$$

The control signal u is generated from a traditional PD controller given the error e_{fb}. The control signal is the input of each of the filters $G_i(s)$. The weights are updated with the covariance learning rule:

$$\Delta w_{ij} = \beta e_j(t) g_i(t). \tag{9}$$

1. Open `exercise7.py` and paste your experiment code from the previous exercise.
2. Change the simulation loop to use the recurrent architecture with the adaptive filter. Note that for now, the cerebellum should just add a contribution to the position error.
3. Run the experiment with different numbers of filters `n_bases`. How many seems to be the optimal number?

   Outcome:

4. Compare the results with those you obtained with the CMAC. Which algorithm learns faster? Which has the best performance?

   Outcome:

5. (Optional) Expand the code to also give corrections to the velocity error:
   (a) Generate a velocity reference with an analytical derivative of your position trajectory.
   (b) Use this to generate a velocity error.
   (c) Have the adaptive filter cerebellum give corrections to the velocity error as well.
   (d) Plot your results. Did the performance increase?
   Outcome:

A CMAC Rule Derivation

Calculate the gradient here

$$Q(w)\frac{1}{2}e^2 = \frac{1}{2}\tau_m^2 \tag{10}$$

$$\frac{\partial Q}{\partial w_j} = \frac{\partial Q}{\partial \tau_m}\frac{\partial \tau_m}{\partial w_j} = \tau_m \frac{\partial \tau_m}{\partial w_j} \tag{11}$$

$$I\ddot{\theta} = \tau - b\dot{\theta} - g(\theta), \tag{12}$$

where I is the inertia, b is the damping, and $g(\theta)$ is the term of nonlinear gravity.
Given the control law,

$$\tau = \tau_m + \tau_{cmac}, \tag{13}$$

we have

$$\tau_m = I\ddot{\theta} + b\dot{\theta} + g(\theta) - \tau_{cmac} \tag{14}$$

which gives us

$$\frac{\partial \tau_m}{\partial w_j} = -\frac{\partial \tau_{cmac}}{\partial w_j} = -B_j \tag{15}$$

Thus, the update rule should be

$$\Delta w_j = -\beta \frac{\partial Q}{\partial w_j} = -\beta\tau_m(-B_j) = -\beta(-\tau_m)B_j \tag{16}$$

Compared with the earlier CMAC update rule, we have $e = -\tau_m$.