Neural optimal feedback control with local learning rules

Friedrich

Optimal Feedback Control (OFC)

- Optimal Feedback Control (OFC) is a framework from control theory that describes how to compute actions (or controls) to achieve a goal in the most efficient way, especially in systems with uncertainty, like noise or delayed feedback.
    - OFC continuously adjusts actions based on real-time sensory feedback, using an internal model to predict future states and minimize a cost (like error or effort).

Kalman Filtering and Linear-Quadratic Regulator (LQR)

- Classical approaches like Kalman filtering and linear-quadratic regulators (LQR) work well in theory, but they're not biologically plausible.
- The **Kalman Filter (for Estimation)** is an algorithm used to **estimate** the hidden state of a system (like position or velocity) from noisy observations.
    - It combines predictions from a model with actual noisy observations to get the best estimate of the current state.
    - Example: You're tracking a moving object (like your hand). You don’t see it clearly due to noisy visual input, but you can predict where it’s going based on how it was moving. The Kalman filter fuses that prediction and the noisy observation to guess where your hand really is.
- The **Linear-Quadratic Regulator (LQR) (for Control)** is a method to compute the optimal action (**control**) to apply to a system to minimize a cost, assuming the system's dynamics and noise are known and linear.
    - Given a current estimate of the state, it outputs a control action that minimizes a long-term cost function (usually quadratic—e.g., penalties on error and effort).
    - Example: It helps you smoothly move your hand to a target while using the least amount of energy and keeping the motion accurate.
- Why these aren’t biologically plausible:
    - They require exact knowledge of system dynamics and noise statistics.
    - They often involve matrix operations, like inversion and solving Riccati equations—hard to do with real neurons.
    - They assume instant access to clean data (**no delays**) and global updates across the system.
    - Learning in these frameworks isn’t local (i.e., neurons can’t just use their own input/output to update weights).

- So, while Kalman filters and LQRs are mathematically elegant, the brain probably doesn't implement them directly, which is why papers like this one try to build **neural approximations that use biologically plausible rules**.

Core Idea of Bio-OFC

- The authors aim to model how the brain might solve control problems like moving your hand to a target despite sensory noise and delay, without needing to know the exact equations of motion or noise levels. Classical approaches like Kalman filtering and linear-quadratic regulators (LQR) work well in theory, but they're not biologically plausible.
- So they build a network that:
    - Uses adaptive Kalman filtering for estimating the system state from delayed/noisy observations.
    - Uses policy gradient (a reinforcement learning method) for learning a control policy, avoiding backpropagation and non-local updates.
    - Learns both the system dynamics (A, B, C) and Kalman gain (L) using online local learning rules, meaning updates are only based on information locally available at the synapse.

1. **Bio-OFC Architecture**:
    - It contains neurons representing:
        - **State estimates** ($\hat{x}_t$),
        - **Prediction errors** (e_t),
        - **Control actions** (u_t).
    - **Kalman filtering is modified** to work with delayed observations ($y_{t-\tau}$).
2. **Learning Rules**:
    - **Local plasticity rules** update system parameters using Hebbian-like updates.
    - Control weights are updated using **policy gradients with eligibility traces**, simulating dopamine-like global signals.
3. **No Prior Knowledge Assumed**:
    - Doesn't assume knowledge of noise covariances or dynamics.
    - Works **online** with **single-phase learning** (no alternating "learning" and "execution" phases).
4. **Biological Plausibility**:
    - **No weight transport** or centralized computation.
    - Supports **delayed feedback**, something prior models ignored.
5. **Experiments**:
    - **Double integrator system** (a basic control problem),
    - **Human hand-reaching task** (comparison with motor adaptation experiments),
    - **2D winged flight** (tests performance in nonlinear and delayed scenarios).

Table 1: **Limitations of previously proposed neural implementations of OFC.** Presence or absence of different properties in previously proposed neural models, and their comparison to Bio-OFC. Guide to symbols: ✓: true, ✗: false, ✓✗: partially true, N/A: not applicable.

	[4]	[6]	[5]	[7]	[8]	Bio-OFC
delayed sensory feedback	✗	✗	✗	✗	✗	✓
control included	✗	✓	✗	✗	✗	✓
noise covariance agnostic	✗	✓	✗	✗	✗	✓
online system identification	✗	✓	✗	✓	✓✗	✓
local learning rules	N/A	✓	N/A	✗	✓	✓
tractable latent size	✗	✓	✗	✓	✓	✓
absence of inner loop	✓	✗	✓	✓	✗	✓
single phase learning/execution	N/A	✗	N/A	✓	✓	✓

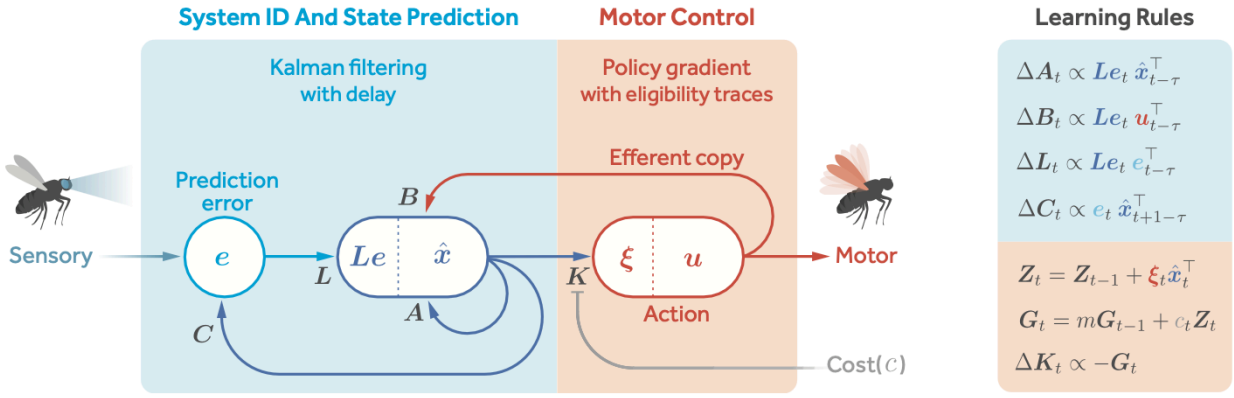

Figure 1: **The circuit and learning rules of the Bio-OFC algorithm.** Our circuit is comprised of two main parts. First (in blue), the circuit performs Kalman filtering. Then (in red), the circuit performs control using policy gradients with eligibility traces. Triangular arrowheads denote synaptic connections and the flat arrowhead denotes the modulatory effect of the cost signal.

Problem Formulation

2.1 Problem formulation

We model the environment as a linear dynamical system driven by control input and perturbed by Gaussian noise. The true state of the system $\boldsymbol{x}$ is hidden and all the animal has access to are the observations $\boldsymbol{y}$ that are assumed to be linear functions of the state corrupted by Gaussian noise.

$$\text{dynamics:} \qquad \boldsymbol{x}_{t+1} = \boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{v}_t \tag{1}$$

$$\text{observation:} \qquad \boldsymbol{y}_t = \boldsymbol{C}\boldsymbol{x}_t + \boldsymbol{w}_t \tag{2}$$

Here $\boldsymbol{v}_t \sim \mathcal{N}(0; \boldsymbol{V})$ and $\boldsymbol{w}_t \sim \mathcal{N}(0, \boldsymbol{W})$ are independent Gaussian random variables and the initial state has a Gaussian prior distribution $\boldsymbol{x}_0 \sim \mathcal{N}(\hat{\boldsymbol{x}}_0, \boldsymbol{\Sigma}_0)$.

The goal is to estimate the latent state $\hat{\boldsymbol{x}}$ in order to design a control $\boldsymbol{u}$ that minimizes expected cost

$$\text{expected cost:} \qquad J = \mathbb{E}\left[\sum_{t=0}^{T} c(\boldsymbol{x}_t, \boldsymbol{u}_t)\right] \tag{3}$$

$$\text{control:} \qquad \boldsymbol{u}_t = k(\hat{\boldsymbol{x}}_t) = \arg\min J \tag{4}$$

where $c(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is the instantaneous cost associated with state $\boldsymbol{x}_t$ and action $\boldsymbol{u}_t$. As the environment dynamics is not known to the animal a priori, the parameters $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ must be learned online.

1. Dynamics of the system (Eq. 1):

$$x_{t+1} = Ax_t + Bu_t + v_t$$

- x_t: the **hidden state** at time t (e.g., position and velocity of a limb).
- u_t: the **control input** at time t (e.g., muscle activation).
- A: state transition matrix (how state evolves on its own).
- B: control matrix (how control input affects the state).
- v_t: **process noise**, drawn from a Gaussian distribution with covariance V.
  - This equation describes how the system evolves over time under control and noise.

2. Observation model (Eq. 2):

$$y_t = Cx_t + w_t$$

- y_t: **observed signal**, like sensory feedback.
- C: observation matrix (how the hidden state is projected into observations).
- w_t: **observation noise**, also Gaussian with covariance W.
  - You don't see the true state x_t; instead, you observe a noisy version of it, y_t.

3. Objective:

Estimate the **hidden state** $\hat{x}_t$ (using something like Kalman filtering), and use it to generate a control input u_t that minimizes the **expected cumulative cost**:

$$J = \mathbb{E}\left[\sum_{t=0}^{T} c(x_t, u_t)\right]$$

- $c(x_t, u_t)$: cost function — penalizes being far from the goal, or using too much control (like energy).
- Goal: choose $u_t = k(\hat{x}_t)$ to minimize this cost over time.

- Challenge: You don't know the system parameters A, B, and C. The system must learn them online from data – just like a brain learning how to move in a new environment.
- Recap

- A: State transition matrix — tells you how the current state x_t evolves into the next state x_{t+1} **without any control input**.
  - > Example: if you're coasting on a bike, A predicts where you'll go just based on your momentum.
- B: Control matrix — tells you how your **actions** u_t (like pushing pedals) influence the next state.
  - > Larger B means more effect from your inputs.
- C: Observation matrix — maps the true state x_t to the **observable signals** y_t.
  - > Think of it like a camera or a sensor that only shows a noisy version of what's really going on.

- **Online Learning**
  - "Online" learning means the system updates its knowledge step-by-step as data comes in — it doesn't wait for all the data to be collected.
  - The system does not know how the world works ahead of time (i.e., it doesn't know A, B, C), so it has to infer them while interacting with the environment.
- In classical control, you assume A,B,C are known — you can compute everything ahead of time.
- But in the brain (or in real adaptive systems), you have to figure them out on the fly based on:
  - What you did (u_t)
  - What you saw (y_t)
  - And your estimate of what the system's state was ($\hat{x}_t$)
- This is also called system identification — learning the internal model of how actions and the environment evolve.
- In this paper, the authors develop local learning rules (i.e. biologically plausible updates) that allow a neural network to gradually estimate these matrices just using its **local activity** — which is pretty cool.

- **Local Activity**
  - By local activity, we mean information that's available at a single neuron and its connections — specifically:
    - The neuron's own activity (e.g. firing rate),
    - Activity of its presynaptic (input) and postsynaptic (output) partners,
    - Possibly a global modulatory signal (like dopamine).
  - In short: no need to access other parts of the network or perform matrix-wide computations. That's what makes a learning rule biologically plausible.

Kalman Estimation and Control

2.2 Kalman estimation and control

The Kalman filter [16, 2] is an estimator of the latent state $\hat{\boldsymbol{x}}_t$ (and its variance) via a weighted summation of the current observation and the prediction of the internal model based on prior measurements. However, in biologically realistic situations the sensory feedback is always delayed. This means that the control signal $\boldsymbol{u}_t$ has to be issued, and thus the state $\boldsymbol{x}_t$ estimated, before $\boldsymbol{y}_t$ has been observed. The appropriately modified Kalman filter computes the posterior probability distribution of $\boldsymbol{x}_t$ given observations $\boldsymbol{y}_{t-\tau}, ..., \boldsymbol{y}_0$ where $\tau \geq 1$ is the delay. We start here with the case of $\tau = 1$ for which the recursive updates of the mean $\hat{\boldsymbol{x}}_t$ and the variance $\boldsymbol{\Sigma}_t$ are

$$\hat{\boldsymbol{x}}_{t+1} = \boldsymbol{A}\hat{\boldsymbol{x}}_t + \boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{L}_t(\boldsymbol{y}_t - \boldsymbol{C}\hat{\boldsymbol{x}}_t) \tag{5}$$

$$\boldsymbol{\Sigma}_{t+1} = (\boldsymbol{A} - \boldsymbol{L}_t\boldsymbol{C})\boldsymbol{\Sigma}_t\boldsymbol{A}^\top + \boldsymbol{V} \tag{6}$$

where $\boldsymbol{L}_t$ is known as the Kalman gain matrix which optimally combines the noisy observations $\boldsymbol{y}_t$ with the internal model and is given by

$$\boldsymbol{L}_t = \boldsymbol{A}\boldsymbol{\Sigma}_t\boldsymbol{C}^\top(\boldsymbol{C}\boldsymbol{\Sigma}_t\boldsymbol{C}^\top + \boldsymbol{W})^{-1}. \tag{7}$$

The Kalman filter is optimal in the sense that it minimizes the mean-squared error $\mathbb{E}[\boldsymbol{e}_t\top\boldsymbol{e}_t]$ with prediction error (innovation) $\boldsymbol{e}_t = \boldsymbol{y}_t - \boldsymbol{C}\hat{\boldsymbol{x}}_t$.

The output feedback law, also known as policy, (4) simplifies if the cost J is quadratic in $\boldsymbol{u}_t$ and $\boldsymbol{x}_t$:

$$\boldsymbol{u}_t = -\boldsymbol{K}\hat{\boldsymbol{x}}_t \tag{8}$$

and is known as linear-quadratic regulator (LQR). The control gain, $\boldsymbol{K}$, is found by solving a matrix Riccati equation (cf. Supplementary Material). Linear policies have been successfully applied to a variety of control tasks [17].

- **Estimate Part**

  (5) Update of the state estimate (mean):

  $$\hat{x}_{t+1} = A\hat{x}_t + Bu_t + L_t(y_t - C\hat{x}_t)$$

  Let's unpack this:

  - $A\hat{x}_t + Bu_t$: your **prediction** for what the next state should be.
  - $y_t - C\hat{x}_t$: the **prediction error** (aka "innovation") — how much your prediction differs from what was observed.
  - L_t: the **Kalman gain** — it decides how much to trust the observation vs. your internal model.

  (6) Update of the estimate variance (uncertainty):

  $$\Sigma_{t+1} = (A - L_tC)\Sigma_tA^\top + V$$

  This tracks your **uncertainty** about the state. As you get new observations, this should ideally go down.

(7) The Kalman gain formula:

$$L_t = A\Sigma_t C^\top (C\Sigma_t C^\top + W)^{-1}$$

This formula decides how to **weight your prediction error** when updating your belief. It adapts based on:

- Your confidence in your current state estimate (Σ_t),
- And how noisy the observations are (W).

- Kalman filtering is optimal in the sense that it minimizes the mean squared prediction error, $e_t = y_t - C\hat{x}_t$

- **Control Part**

  Linear-Quadratic Regulator (LQR) is a classic method in control theory for computing the **optimal way to act** (i.e., choose control inputs u_t) in a linear system so that you minimize a given **cost function** over time.

  It's called:

  - **Linear**: because the system dynamics are linear (as in $x_{t+1} = Ax_t + Bu_t + \text{noise}$),
  - **Quadratic**: because the cost function is quadratic in both the state and control.

  Quadratic cost functions are used because they:

  1. **Penalize deviation** from the desired state (like being far from a goal),
  2. **Penalize excessive control effort** (e.g., you don't want to use too much energy).

  A typical form of the cost is:

$$J = \sum_{t=0}^{T} \left(x_t^\top Q x_t + u_t^\top R u_t \right)$$

  - $x_t^\top Q x_t$: penalizes how far the state is from zero (or a goal), scaled by matrix Q,
  - $u_t^\top R u_t$: penalizes large control inputs, scaled by R.

- Quadratic terms are nice because they're:
  - Smooth and convex → easy to optimize,
  - Closed-form solutions exist (e.g., for computing K).

  (8) Control policy:

$$u_t = -K\hat{x}_t$$

  - K is the **control gain** — tells you how strongly to respond to the estimated state.
  - This is known as a **Linear Quadratic Regulator (LQR)**.

- More details:

- K is the **control gain matrix**.
- It tells you **how strongly to react to your current estimated state**.
- It is computed based on the system dynamics and the cost matrices Q, R by solving the **Riccati equation** (a specific matrix equation).

In practice:

- If K is large, the controller reacts **aggressively** to deviations (big corrections),
- If K is small, it's more **cautious** (gentle corrections).

- What does LQR give you?
    - A **feedback controller**: it maps your state estimate $\hat{x}_t$ directly to the control input u_t,
    - It's **optimal**: minimizes the total cost over time,
    - And it's **efficient**: once K is computed, control is just a simple matrix multiplication at every step.