HW2: Recursive iLQR for Planar Manipulator

EECS 599/491

Overview

In this assignment, you will implement the recursive iLQR algorithm for a 3-link planar robot to track a sequence of viapoints forming a figure-8 shape. The provided Python code includes dynamics, kinematics, and a visualization loop. Your task is to fill in the core iLQR logic for optimization.

Mathematical Model

System Dynamics

We assume a first-order integrator model:

$$x_{t+1} = Ax_t + Bu_t, \quad A = I, \quad B = I \cdot \Delta t$$

Cost Function

We minimize a cost over T steps:

$$J = \sum_{t \in \mathcal{T}_\ell} \|f(x_t) - f_t^{ref}\|_Q^2 + \sum_t \|u_t\|_R^2$$

where $f(x_t)$ maps joint angles to end-effector position/orientation and $\mathcal{T}_\ell$ is the set of via-point indices.

Helper Function Explanations

- `logmap(f, f0)`: computes position and orientation error on the $\mathbb{R}^2 \times S^1$ manifold.
- `fkin(x, param)`: forward kinematics mapping joint angles to end-effector (x, y, θ).
- `fkin0(x, param)`: computes all joint positions for visualization of robot links.
- `Jkin(x, param)`: analytical Jacobian of the end-effector w.r.t. joint angles.
- `f_reach(x, param)`: returns tracking error $(f(x_t) - f_t^{ref})$ and Jacobian at each via-point, transformed into object frame.

iLQR Algorithm Tasks

1. Compute Gradients

At each via-point index t_ℓ:

$$
\begin{aligned}
L_u &= R u_t \\
L_x &= J_t^\top Q(f(x_t) - f_t^{ref}) \\
L_{xx} &= J_t^\top Q J_t
\end{aligned}
$$

2. Backward Pass

For $t = T - 2$ to 0:

$$
\begin{aligned}
Q_x &= L_x + A^\top V_x \\
Q_u &= L_u + B^\top V_x \\
Q_{xx} &= L_{xx} + A^\top V_{xx} A \\
Q_{ux} &= B^\top V_{xx} A \\
Q_{uu} &= R + B^\top V_{xx} B
\end{aligned}
$$

Then compute:

$$k_t = -Q_{uu}^{-1} Q_u, \quad K_t = -Q_{uu}^{-1} Q_{ux}$$

Update:

$$V_x = Q_x - Q_{ux}^\top Q_{uu}^{-1} Q_u, \quad V_{xx} = Q_{xx} - Q_{ux}^\top Q_{uu}^{-1} Q_{ux}$$

3. Forward Rollout with Line Search

Apply updated control:

$$u_t^{\text{new}} = u_t + \alpha k_t + K_t(x_t - \hat{x}_t)$$

Propagate dynamics:

$$x_{t+1}^{\text{new}} = A x_t + B u_t^{\text{new}}$$

Student Coding Tasks

You are provided with a full Python scaffold. Complete the following TODO sections (Line 155-200):

- Compute L_x and L_{xx} at via-points
- Implement the backward pass equations
- Perform the forward pass and cost reduction check

TODO Codes

```
# === Step 1: Compute gradients ===
Lu = uref * param.r
Lx = np.zeros([param.nbVarX, param.nbData])
Lxx = np.zeros([param.nbVarX, param.nbVarX, param.nbData])
for t in range(len(tl)):
    # TODO: Compute Lx
    # Lx[:, tl[t]] = ...
    # TODO: Compute Lxx
    # Lxx[:, :, tl[t]] = ...

# === Step 2: Backward Pass ===
Vx = Lx[:, -1]
Vxx = Lxx[:, :, -1]
for t in range(param.nbData - 2, -1, -1):
    # TODO: Compute Qx, Qu, Qxx, Qux, Quu
    # TODO: Compute k[:, t], K[:, :, t]
    # TODO: Update Vx and Vxx

# === Step 3: Forward Pass ===
alpha = 1
while True:
    xtmp[:, 0] = x0
    for t in range(param.nbData - 1):
        # TODO: Compute du and utmp
        # TODO: Forward simulate xtmp
    # TODO: Evaluate new cost and accept if better
```

Submission

- Submit your completed Python script with all "TODO" sections filled in to implement the iLQR logic.
- Your implementation should reproduce a visualization similar to the provided reference video example.

Hints

- You are encouraged to review the ECE599 Lecture Notes on LQR and iLQR, particularly "Algorithm 1" for a step-by-step outline.
- Visualization code is included after line 200 of the script. You may modify it as needed to enhance clarity or match your implementation style.