iLQR of wheeled robot

Jamie Wong

March 2023

1 Introduction

iLQR(iterative linear quadratic regulator) is a extended form of LQR for non-linear dynamic model. In this report, the kinematic model of wheeled robot will be introduced, and the iLQR algorithm will be explained.

2 LQR

2.1 Problem description

LQR assumes the model is locally linear and time-varied, and we represent it in a matrix form as:

$$f(x_t, u_t) = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t$$

We also define a time-varied cost function to find the optimal trajectory that obeys the model. Mathematically, our objective is:

$$\min_{u_1,...,u_T} \sum_{t=1}^{T} c(x_t, u_t) \qquad \text{s.t.} x_t = f(x_{t-1}, u_{t-1}) \tag{1}$$

Therefore, if the system dynamics f (model) and the cost function c is known, we can plan our optimal controls (actions) using a trajectory optimization method.

$$\tau = \{x_1, u_1, x_2, u_2, ..., x_T, u_T\}$$

LQR takes quadratic form cost and first order state-feedback regulator:

$$c(x_t, u_t) = \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T C_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T c_t \tag{2}$$

$$u_t = K_t x_{t-1} + k_t \tag{3}$$

2.2 Backward recursion

Backward recursion means solving the problem in backward sequence, which is the typical procedure of dynamic programming. In this section, first, solution of a single time stamp is described, second, the reason why dynamic programming approach is suitable for this problem will be shown.
Cost weight matrix C_t in (1) could be transformed to blocked form:

$$C_t = \begin{bmatrix} C_{x_t,x_t}, C_{x_t,u_t} \\ C_{u_t,x_t}, C_{u_t,u_t} \end{bmatrix} \in \mathbb{R}^{N_x+N_u \times N_x+N_u} \tag{4}$$

$$c_t = \begin{bmatrix} c_{x_t} \\ c_{u_t} \end{bmatrix} \in \mathbb{R}^{N_x \times N_u} \tag{5}$$

$$c(x,u) = \frac{1}{2}[x^T C_{x,x} x + x^T C_{x,u} u + u^T C_{u,x} x + u^T C_{u,u} u] + x^T c_x + u^T c_u \tag{6}$$

Since x_T is fixed, $\min_{u_{T-1}} c_(x_{T-1}, u_{T-1})$ is an individual optimization problem. So u_{T-1} could be solved:

$$\min_{u_{T-1}} Q(x_{T-1}, u_{T-1}) \rightarrow u_{T-1} = K_{T-1} x_{T-1} + k_{T-1} \tag{7}$$

$$\nabla_{u_{T-1}} Q = x^T C_{x,u} + u^T C_{u,u} + c_u = 0 \tag{8}$$

$$u = -C_{u,u}^{-} 1(C_{u,x} x + c_u) \tag{9}$$

According to the form of regulator $x_t = K_t x_{t-1} + k_t$, coeffitiants are derived:

$$K = -C_{u,u}^{-1} C_{u,x}, \quad k = -C_{u,u}^{-1} c_u \tag{10}$$

At present, optimal policy of last step is solved. The rest will be solved with dynamic programming approach.

2.2.1 Dynamic Programming approach in Reinforcement Learning

When a MDP(Markov Decision Process) is fully known, which means the dynamics is modeled and the states are fully measured, it can be solved with dynamic programming. There are two main properties of Reinforcement Learning solving approach, one is backup depth, the other is backup width. When backup depth is high, history of experiment is recorded and used in Monte-Carlo way. When backup width is high, each backup can be reused to reduce repetitive searching. Dynamic programming is a low depth and high width approach. Dynamic programming algorithm:

If we konw the solution to subproblems $v_*(s')$, Then solution $v_*(s')$ can be found by one-step lookahead:

$$v_*(s') = \underbrace{\max_{a \in \mathcal{A}} \mathcal{R}_x^a}_{\text{maximize reward}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')}_{\text{accumulated value}} \tag{11}$$

Another explicit form of solution is represented by Q-value, which is another term to represent value at state s when action a is taken:

$$Q(s,a) = \mathcal{R}_x^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \tag{12}$$

The solution can be found by maximize $Q(s,a)$ w.r.t a:

$$Q_*(s,a) = \max_{a \in \mathcal{A}} (\mathcal{R}_x^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')) \tag{13}$$

In dynamic programming, $v_*(s')$ is in dependent of a, therefore it is eqivalent to (11).

2.2.2 Value function of LQR

When u is figured out, the value function is:

$$V(x) = \begin{bmatrix} x \\ u \end{bmatrix}^T C \begin{bmatrix} x \\ u \end{bmatrix} + \begin{bmatrix} x \\ u \end{bmatrix}^T c \tag{14}$$

$$= \text{const} + \frac{1}{2}[x^T C_{x,x} x + 2x^T C_{x,u}(Kx+k) + (Kx+k)^T C_{u,u}(Kx+k)] + x^T c_x + c_u^T (Kx+k) \tag{15}$$

$$= \text{const} + \frac{1}{2} x^T \underbrace{(C_{x,x} + 2C_{x,u}K + KC_{u,u}K)}_{V} x + x^T \underbrace{(c_x + C_{x,u}k + K^T C_u + K^T C_{u,u}k)}_{v} \tag{16}$$

In (13), $\mathcal{P}_{ss'}$ corresponds to the system dynamics:

$$x_T = F_{T-1} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + f_{T-1} \tag{17}$$

Then

$$V(x_T) = \text{const} + \frac{1}{2} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} F_{T-1}^T V_T F_{T-1} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T (F_{T-1}^T V_T f_{T-1} + F_{T-1}^T v_T) \tag{18}$$

Q-value function $Q(x_{T-1}, u_{T-1})$ is adding V_T to reward value $c_(x,u)$:

$$Q(x_{T-1}, u_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T C_{T-1} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T c_{T-1} + V_T \tag{19}$$

$$= \text{const} + \frac{1}{2} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T \underbrace{(C_{T-1} + F_{T-1}^T V_T F_{T-1})}_{Q_{T-1}} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T \underbrace{(c_{T-1} + F_{T-1}^T V_T f_{T-1} + F_{T-1}^T}_{q_{T-1}} \tag{20}$$

In summary, the Q-value function is represented as:

$$Q(x,u) = \begin{bmatrix} x \\ u \end{bmatrix}^T Q \begin{bmatrix} x \\ u \end{bmatrix} + \begin{bmatrix} x \\ u \end{bmatrix}^T q \tag{21}$$

2.3 Forward recursion

By solving (10) and back propagate according to (16) (17) (20), $[u_{T-1}, ..., u_1]$ is calculated. Then states are to be updated for next round of problem solving. The system dynamics for forward recursion is:

$$x_{t+1} = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t \tag{22}$$

$$= Ax_t + Bu_t \tag{23}$$

$$= [A + BK_t]x_t^T + Bk_t \tag{24}$$

2.4 LQR algorithm

Backward recursion:
for t = T to 1:

$$Q_t = C_t + F_t^T V_{t+1} F_t$$
$$q_t = c_t + F_t^T V_{t+1} f_t + F_t^T v_{t+1}$$
$$Q(x_t, u_t) = \text{const} + \frac{1}{2} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T Q_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T q_t$$
$$u_t = \arg\min_{U_t} Q(x_t, u_t) = K_t x_t + k_t$$
$$K_t = -Q_{u_t,u_t}^{-1} Q_{u_t,x_t}$$
$$k_t = -Q_{u_t,u_t}^{-1} q_{u_t}$$
$$V_t = C_{x_t,x_t} + 2C_{x_t,u_t} K_t + K_t C_{u_t,u_t} K_t$$
$$v_t = c_{x_t} + C_{x_t,u_t} k + K_t^T C_{u_t} + K_t^T C_{u_t,u_t} k_t$$
$$V(x_t) = \text{const} + \frac{1}{2} x_t^T V_t x_t + x_t^T v_t$$

Forward recursion:
for t = 1 to T:

$$u_k = K_t x_t + k_t$$
$$x_{t+1} = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t$$

3 iLQR

For nonlinear model, instead of least-square optimization problem (7), Newton-Method is a feasible tool.

3.1 Newton-Method

For quadratic cost function, the minimization problem can be solved with Newton-Method in one step.

$$\begin{aligned} x_* &= \arg\min_x ax^2 + bx + c \\ &= -\frac{f'(x)}{f''(x)} \\ &= -\frac{b}{2a} \end{aligned}$$

Any nonlinear function can be approximated in 2-order Taylor-Expansion form around $\hat{X}$. Therefore, the minimization can be achieved with Newton-Method. One step is not enough, because there is residual due to approximation, thus iterative procedure is taken.

$$f(X) - f(\hat{X}) = H(X - \hat{X})^2 + g(X - \hat{X}) + \text{const} \tag{25}$$

Where H is Hessian matrix of $f(X)$ and g is the gradient matrix. The optimal $X - \hat{X}$ is $-\frac{g}{2H}$. Then $\hat{X}$ is updated and continue being used in next iteration. In fact, $f(X)$ is simply differentiated so as to transform $f(X)$ into quadratic form.

3.2 Non-linear system dynamics linear quadratic planning

Non-linear system dynamics can be transformed into quadratic form by differentiating around current state. Then the local quadratic problem is solved exactly the same as LQR.

$$\underbrace{f(x_t, u_t) - f(\hat{x_t}, \hat{u_t})}_{\bar{f}(\delta_{x_t}, \delta_{u_t})} \approx \underbrace{\nabla_{x_t, u_t} f(\hat{u_t}, \hat{u_t})}_{F_t} \begin{bmatrix} x_t - \hat{x_t} \\ u_t - \hat{u_t} \end{bmatrix} \tag{26}$$

$$\underbrace{c(u_t, u_t) - c(\hat{u_t}, \hat{u_t})}_{\bar{c}(\delta_{x_t}, \delta_{u_t})} \approx \frac{1}{2} \begin{bmatrix} x_t - \hat{x_t} \\ u_t - \hat{u_t} \end{bmatrix}^T \underbrace{\nabla^2_{x_t, u_t} c(\hat{u_t}, \hat{u_t})}_{C_t} \begin{bmatrix} x_t - \hat{x_t} \\ u_t - \hat{u_t} \end{bmatrix} + \begin{bmatrix} x_t - \hat{x_t} \\ u_t - \hat{u_t} \end{bmatrix}^T \underbrace{\nabla_{x_t, u_t} c(\hat{x_t}, \hat{u_t}))}_{c_t} \tag{27}$$

3.3 iLQR algorithm

$C_t = \nabla^2_{x_t, u_t} c(\hat{u_t}, \hat{u_t})$
$c_t = \nabla_{x_t, u_t} c(\hat{x_t}, \hat{u_t})$
$F_t = \nabla_{x_t, u_t} f(\hat{x_t}, \hat{u_t})$
solve the LQR problem, and apply $u_t = \hat{u_t} + K_t(u_t - \hat{u_t}) + k_t$
Forward recursion to update states $\hat{x_t}$

4 Kinematic model of wheeled robot

Under the assumption that wheels doesn't slip on the ground, the system dynamics is in non-holonomic type. The input of the system is $u = [v \quad \delta]^T$.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v\tan\delta}{L} \end{bmatrix} \quad \text{where L is the length of wheel base} \tag{28}$$

Descretized form is:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t + \Delta_t v\cos\theta \\ y_t + \Delta_t v\sin\theta \\ \theta_t + \frac{\Delta_t v\tan\delta}{L} \end{bmatrix} \tag{29}$$

Referring to (26), the kinematic dynamics is transformed:

$$\begin{aligned} \nabla_{x_t,u_t} f(\hat{x}_t, \hat{u}_t) &= \begin{bmatrix} \frac{\partial f}{\partial x_t} & \frac{\partial f}{\partial u_t} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \Delta_t v\sin\theta & \Delta_t\cos\theta & 0 \\ 0 & 1 & \Delta_t v\cos\theta & \Delta_t\sin\theta & 0 \\ 0 & 0 & 1 & \frac{\Delta_t\tan\delta}{L} & \frac{\Delta_t v}{L\cos\delta^2} \end{bmatrix} \end{aligned} \tag{30}$$