Robotics: Science and Systems

Advanced Digital Controllers

Zhibin Li
School of Informatics
University of Edinburgh
Content

- State space representation
- Linear–quadratic regulator (LQR controller)
- LQR tracking control
- Control constraints: stability and methods
State space representation
Recap: 1D point-mass dynamics (lecture “Dynamics”).

In general, system dynamics is described by: $\dot{x} = f(x, u)$

Rewrite the point-mass dynamics in this form by matrix, $u$ is the rate change of acceleration, we have

$$\frac{d}{dt} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u$$

$A$: state matrix, $x$: state vector, $B$: input matrix

$u$: control effort
State space equation in continuous time

\[
\frac{d}{dt} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u
\]

Similarly, by specifying the output matrix, e.g., position, we can have y as the output.

\[
y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} u
\]

All this is called the state space equation of the system dynamics.
State space equation in continuous time

\[
\begin{align*}
\frac{d}{dt} & \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} = \\
& \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \\
& \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u
\end{align*}
\]

\[
\begin{align*}
\dot{x} &= Ax + Bu \\
y &= Cx + Du
\end{align*}
\]
Discretization: Tustin’s Method

Exactly the same as in the numerical integration part, the area, incremental change of y, has been approximated by that of the trapezoid formed by the base T and vertices f(kT+ T) and f(kT) as shown by the dashed line.
We have learnt discretization in the numerical simulation previously, state space equation can be done in a similar manner.

Previously, we learned that in discrete-time, we can use trapezoidal discretization to simulate physics numerically:

\[ s_{n+1} = s_n + \frac{(v_{n+1} + v_n)}{2} \cdot dt = s_n + v_n \cdot dt + \frac{1}{2} a_n \cdot dt^2. \]

This is more accurate than the Euler method, which results in a staircase-like approximation (zero-hold).
Discretizing a system

Similarly, we can apply Tustin’s method to each term as follows (work out from high order derivatives first):

\[
x(i + 1) = x(i) + \dot{x}(i)\Delta t + \frac{1}{2}\ddot{x}(i)\Delta t^2 + \frac{1}{4}u(i)\Delta t^3
\]
\[
\dot{x}(i + 1) = \dot{x}(i) + \ddot{x}(i)\Delta t + \frac{1}{2}u(i)\Delta t^2
\]
\[
\ddot{x}(i + 1) = \ddot{x}(i) + u(i)\Delta t
\]
State space equation in discrete time

If we express all these linear equations in the form of matrices and vectors, we obtain *discrete state space equation* that reflects exactly the same underlying physics.

$$\begin{bmatrix}
  x(i+1) \\
  \dot{x}(i+1) \\
  \ddot{x}(i+1)
\end{bmatrix} =
\begin{bmatrix}
  1 & \Delta t & \Delta t^2/2 \\
  0 & 1 & \Delta t \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x(i) \\
  \dot{x}(i) \\
  \ddot{x}(i)
\end{bmatrix} +
\begin{bmatrix}
  \Delta t^3/4 \\
  \Delta t^2/2 \\
  \Delta t
\end{bmatrix} u(i)$$

- **A**: discrete state matrix
- **x**: discrete state vector
- **B**: discrete input matrix
- **u**: control effort

Example of discretization in Matlab, options are zero-hold, Tustin (bi-linear transform).
Analytic discretization of state space equations

It is easier to formulate the continuous SS equations, but how to discretize systematically without manually derivation?

Given the sampling time $T$, the analytic form is:

$$
\begin{align*}
Ad &= (I-A*T/2) \backslash (I+A*T/2); \quad \% \quad \text{inv}(I-A*T/2)*(I+A*T/2) \\
Bd &= (I-A*T/2) \backslash B*T; \quad \% \quad \text{inv}(I-A*T/2)*B*T \\
Cd &= C/(I-A*T/2); \quad \% \quad C*\text{inv}(I-A*T/2) \\
Dd &= C*Bd/2+D
\end{align*}
$$

Note: $A\backslash B \rightarrow \text{inv}(A)*B$. 
Discretization using Matlab

Matlab command `c2d` discretizes the continuous-time SS model using the specified discretization method and given a sampling time of Ts (in seconds), see here.

\[
\text{sysd} = \text{c2d}(\text{sys}, \text{Ts}, \text{method})
\]

Here, we can first use \text{sys} = \text{ss}(A,B,C,D) to create a state space model \text{sys}. 
Example: point mass dynamics

Control effort $u$ is acceleration, output is position $x$.

\[
\frac{d}{dt} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u
\]

\[
y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} u
\]
Example: point mass dynamics

\[
\begin{bmatrix}
  \dot{x} \\
  \ddot{x}
\end{bmatrix} = \begin{bmatrix}
  0 & 1 & 0 \\
  0 & 0 & 1 \\
  0 & 0 & 0
\end{bmatrix} \begin{bmatrix}
  x \\
  \dot{x} \\
  \ddot{x}
\end{bmatrix} + \begin{bmatrix}
  0 \\
  0 \\
  1
\end{bmatrix} u
\]

A=[0 1 0; 0 0 1; 0 0 0];
B = [0; 0; 1];
C=[1, 0, 0];
D=0;
T=1.0; % setting 1 is easy for us to see the result
sys=ss(A,B,C,D)
sysd = c2d(sys,T); % default is zero hold
display('zero hold')
sysd = c2d(sys,T,'zoh')
display('tustin')
sysd = c2d(sys,T,'tustin')
Example: continuous state space

sys =
a =
\[
\begin{bmatrix}
  x1 & x2 & x3 \\
x1 & 0 & 1 & 0 \\
x2 & 0 & 0 & 1 \\
x3 & 0 & 0 & 0
\end{bmatrix}
\]
b =
\[
\begin{bmatrix}
u1 \\
x1 & 0 \\
x2 & 0 \\
x3 & 1
\end{bmatrix}
\]
c =
\[
\begin{bmatrix}
x1 & x2 & x3 \\
y1 & 1 & 0 & 0
\end{bmatrix}
\]

Continuous-time state-space model.

\[
\frac{d}{dt} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u
\]

\[
y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} u
\]
Example: zero-hold method

sysd =
a =
\[
\begin{array}{ccc}
  & x1 & x2 & x3 \\
 x1 & 1 & 1 & 0.5 \\
x2 & 0 & 1 & 1 \\
x3 & 0 & 0 & 1 \\
\end{array}
\]

b =
\[
\begin{array}{c}
  u1 \\
x1 & 0.1667 \\
x2 & 0.5 \\
x3 & 1 \\
\end{array}
\]
c =
\[
\begin{array}{ccc}
  x1 & x2 & x3 \\
y1 & 1 & 0 & 0 \\
\end{array}
\]

Sample time: 1 seconds; Discrete-time state-space model.
Example: Tustin method

\[
\begin{bmatrix}
    x_1(i + 1) \\
    \dot{x}(i + 1) \\
    \ddot{x}(i + 1)
\end{bmatrix}
= \begin{bmatrix}
    1 & \Delta t & \Delta t^2/2 \\
    0 & 1 & \Delta t \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x(i) \\
    \dot{x}(i) \\
    \ddot{x}(i)
\end{bmatrix}
+ \begin{bmatrix}
    \Delta t^3/4 \\
    \Delta t^2/2 \\
    \Delta t
\end{bmatrix} u(i)
\]

\[
\begin{bmatrix}
    x_1 \\
    x_2 \\
    x_3
\end{bmatrix}
= \begin{bmatrix}
    1 & 1 & 0.5 \\
    0 & 1 & 1 \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    x_3
\end{bmatrix}
\]

b =
\[
\begin{bmatrix}
    u_1 \\
    0.25 \\
    0.5 \\
    1
\end{bmatrix}
\]

c =
\[
\begin{bmatrix}
    x_1 & x_2 & x_3
\end{bmatrix}
\]

y_1 = 1 0.5 0.25

d =
\[
\begin{bmatrix}
    u_1 \\
    0.125
\end{bmatrix}
\]

Sample time: 1 seconds; Discrete-time state-space model.
Linear–quadratic regulator (LQR)
LQR stabilization

Continuous-time State-Space (SS) model

\[
\dot{x} = Ax + Bu \\
y = Cx + Du
\]

Discrete-time State-Space (SS) model

\[
x_{k+1} = A_dx_k + B_d u_k \\
y_k = C_d x_k + D_d u_k
\]

Simply the format without subscript “d” and hereafter

\[
x_{k+1} = Ax_k + Bu_k \\
y_k = Cx_k + Du_k
\]
Discrete-time LQR

Linear-quadratic (LQ) state-feedback regulator for discrete-time state-space system

\[ x_{k+1} = Ax_k + Bu_k \]

Cost function \( J \): minimize change of states and the associated efforts

\[
J = \sum_{k=0}^{\infty} (x_k^T Q x_k + u_k^T R u_k + 2x_k^T N u_k)
\]

\[ u_k = -K x_k \]

\[
K = (B^T P B + R)^{-1} (B^T P A + N^T)
\]

\[
P = A^T P A - (A^T P B + N)(B^T P B + R)^{-1}(B^T P A + N^T) + Q
\]

\( P \) is the Riccati Equation. The default value of \( N \) is 0.
Discrete-time LQR

Discrete time Algebraic Riccati Equation (DARE)

\[ P = A^T P A - (A^T P B + N)(B^T P B + R)^{-1}(B^T P A + N^T) + Q \]

This can be solved by iterating the dynamic Riccati equation of the finite-horizon as shown below until it converges.

\[ P_{k-1} = A^T P A - (A^T P_k B + N)(B^T P_k B + R)^{-1}(B^T P_k A + N^T) + Q \]
Discrete-time LQR: C++ example

```cpp
// discrete time algebraic Riccati equation (DARE):
MatrixXd LQRClass::SolveDARE(const MatrixXd &A, const VectorXd &B, const RowVectorXd &C, const double &Q, const double &R)
{
    int sizeP = A.rows();
    MatrixXd P;
    P.setZero(sizeP, sizeP);
    double Pnorm_old = 100.0;
    bool IsFinishRecursion = false;

    int i = 0;
    double eps = 1e-8;

    while (!IsFinishRecursion){
        P = A.transpose()*P*A + C.transpose()*Q*C - A.transpose()*P*B*(I/(R+B.transpose()*P*B))*B.transpose()*P*A;
        if (abs(P.norm()-Pnorm_old) < eps || i>20000){
            IsFinishRecursion = true;
        }
        Pnorm_old = P.norm();
        i++;
        // cout"i: ", i<<" times."
        cout<<"The ARE recursion runs "<><" times."
    }
    return P;
}
Discrete-time LQR: Matlab example

Syntax: \([K,S,e] = \text{dlqr}(A,B,Q,R,N)\)

It returns the calculated optimal gain matrix \(K\).

The closed-loop eigenvalues can be calculated by \(e = \text{eig}(A-B*K)\) which reflects the stability of the closed loop system.

If the norms of all eigenvalues are within the unitary circle, or \(\text{abs(eigenvalue)} < 1\), the discrete closed loop system is stable.
Note that the LQR here uses the **full state feedback** to stabilize its state variables, meaning that any deviation will quickly converge to its equilibrium, ie $x=0$.

For LQR tracking control, further formulation is needed.
LQR tracking control

Note that previous formulation has no reference input, hence, it is merely a self-stabilized system.

Logically, if a reference needs to be tracked, we need to compare the error in some ways. We briefly introduce the use of integral control in state-space formulation.

Integral control feeds back the integral of the error as well as the state of the plant, $x$, by adding an extra state variable $x_I$, defined as the integral of the error.
LQR tracking control

Define tracking error: $e(t) = y(t) - r(t)$.

Define $x_1$ as the integral of the error.

We will then introduce $x_1$ as a new state.
LQR tracking control

Integral of error $x_I$ is

$$x_I(t) = \int^t e(t) \, dt$$

Derivative of $x_I(t)$ is

$$\frac{\partial x_I(t)}{\partial t} = e(t)$$

The *discrete* version is

$$e(k) = y(k) - r(k)$$

$$x_I(k+1) = x_I(k) + e(k)$$
LQR tracking control

The discrete version is

\[ x_I(k+1) = x_I(k) + y(k) - r(k) \]

Recall

\[ y(k) = C \cdot x(k) + D \cdot u(k) \]

So

\[ x_I(k+1) = x_I(k) + C \cdot x(k) + D \cdot u(k) - r(k) \]

Define new state vector as

\[ x_k = [x(k), x_I(k)]^T \]

Hence

\[ x_{k+1} = [x(k+1), x_I(k+1)]^T \]
LQR tracking control

The discrete version of tracking control

\[
x(k+1) = Ax(k) + B \cdot u(k)
\]

\[
\begin{bmatrix}
x(k+1) \\
x_I(k+1)
\end{bmatrix} = \begin{bmatrix}
x(k) \\
x_I(k)
\end{bmatrix} + \begin{bmatrix}
u(k)
\end{bmatrix} + \begin{bmatrix}
r(k)
\end{bmatrix}
\]

\[
x_i(k+1) = x_i(k) + C \cdot x(k) + D \cdot u(k) - r(k)
\]
LQR tracking control

The discrete version of tracking control

\[
x(k+1) = Ax(k) + B \cdot u(k) + C \cdot x(k) + D \cdot u(k) - r(k)
\]
LQR tracking control

The discrete version of tracking control

\[
\begin{bmatrix}
    x(k+1) \\
    x_I(k+1)
\end{bmatrix} =
\begin{bmatrix}
    A & 0 \\
    C & 1
\end{bmatrix}
\begin{bmatrix}
    x(k) \\
    x_I(k)
\end{bmatrix}
+ 
\begin{bmatrix}
    B \\
    D
\end{bmatrix} u(k) + 
\begin{bmatrix}
    0_{3 \times 1} \\
    -1
\end{bmatrix} r(k)
\]

\[
x_{k+1} = A_t x_k + B_t u_k + N r_k
\]

\[
y_k = C_t x_k + D_t u_k
\]
LQR tracking control of an inverted pendulum

\[ \ddot{\theta} = \frac{\tau + m g r_c \cdot \sin(\theta) - c \cdot \dot{\theta}}{I} \]

\[ I = m \cdot r_c^2 \]

\[ \theta \approx \sin(\theta) \]

m=32;
g=9.81;
rc=0.5; % pivot to COM vector length
lc=1/3*m*rc^2; % approximation of inertia tensor
l=lc+0.5*m*rc^2;
c=0.1;
LQR tracking control of an inverted pendulum

\[
A = \begin{bmatrix} 0 & 1 \\ \frac{mgrc}{I} & -\frac{c}{I} \end{bmatrix}; \quad \text{\% B is viscous coefficient at ankle}
\]

\[
B = \begin{bmatrix} 0 \\ \frac{1}{I} \end{bmatrix};
\]

\[
C = \begin{bmatrix} 1 & 0 \end{bmatrix};
\]

\[
D = [0];
\]

\[
T = 0.005;
\]

\[
sysd = \text{c2d(}\text{ss}(A,B,C,D),T);\]

\[
Ad = sysd.a;
\]

\[
Bd = sysd.b;
\]

\[
Cd = sysd.c;
\]

\[
Dd = sysd.d;
\]
LQR tracking control of an inverted pendulum

\[
\begin{align*}
    A_t &= [A_d \ zeros(2,1); C_d 1]; \\
    B_t &= [B_d; D_d]; \\
    C_t &= [C_d 0]; \\
    D_t &= [D_d];
\end{align*}
\]

\[
\begin{align*}
pos &= 1; \\
\text{vel} &= 1; \\
\text{err} &= 1e6; \\
Q &= \text{diag}([\text{pos} \ \text{vel} \ \text{err}]); \% \text{same number as state variables} \\
R &= 5e-1; \quad \% \text{same number as control input} \\
[K, P, E] &= \text{dlqr}(A_t, B_t, Q, R);
\end{align*}
\]
Influence of weights

Weight: how much a variable is penalized.
Example 1: pos = 1; vel = 1; err = $1e6$; K = [12372, 2550, 293]
Influence of weights

Weight: how much a variable is penalized.
Example 1: \( \text{pos} = 1; \ \text{vel} = 1; \ \text{err} = 1e3; \ K = [1985.8, 409.3, 40.7] \)
Limitations of unconstrained LQR

The LQR control design is unconstrained, ie the optimal feedback is independent of physical constraints.

The disadvantage appears in real world problems where all inputs, physical quantities, are *inevitably limited*, such as the inverted pendulum control, rocket landing control etc..

This problem can be reformulated as constrained LQR problem, or some other advanced predictive methods, such as MPC.
Constraint of control output
Control constraint/saturation

**Clipping**: approach is formulated assuming that the control effort has an unlimited range. First the control action is calculated, then it is hard-limited to keep the resulting values within the specified range, eg \( \min \leq u^*(t) \leq \max \).
Constraints of output by clipping

Clipping is to simply apply:

```plaintext
if u(i) > u_max
   u(i) = u_max;
elseif u(i) < -u_min
   u(i) = -u_min;
end
```
LQR control with output saturation by clipping

LQR position tracking

Reference: square wave (magnitude 1 m);

Control constraint: max acceleration 10 m/s.

Result: unstable divergence.

*PID control with output saturation by simple clipping has exactly the same issue, thus omitted here for brevity.
LQR control with output saturation by clipping

Saturation of control effort:

1. Bang-bang oscillation
2. Instantaneous switch of control effort
3. Damage of real systems
Anti-windup: proportionally deduct the amount of overflow action via feedback, combined with saturation.
Anti-windup: adding feedback of saturation term

Feedback gain $Ka$ is the proportional gain to deduct the quantity that overflows. For example, $Ka=1.0$ means deduction of the exact amount that saturates.
LQR control with anti-windup

Case study in comparison to LQR with clipping.

![Block diagram of LQR control system with anti-windup](image-url)
LQR control with anti-windup

LQR position tracking

1. Position tracking with zero steady state error given enough time.
2. Longer rise up time for step-response due to limited control effort
3. No divergence
LQR control with anti-windup

Saturation of control effort:

1. Deduction of excessive control effort during saturation;
2. Slight discontinuous of using proportional feedback $K_a$ for anti-wind up regulation.
3. Reduced time of saturation
LQR control with anti-windup

Comparison of control efforts
PID control for rocket landing

Manually tuned PD controllers
PID control for rocket landing

Saturation of control output and loss of stability
Summary

What are the features of LQR controller?

- A state-space controller
- Stabilize full states
- Model based approach
- LQR is also an instantaneous controller without looking at the future reference.
- Centralized controller that considers coupling effects between state variables.
Advantages of LQR:

- Generally more stable, less tuning or trial and error
- Naturally a state stabilizer
- Introducing integral of error terms makes it a tracking controller

Disadvantages of LQR:

- Loss of intuition of output behaviors, eg under-, critical-damped response.
- Required a model of the system, design is more complex compared to PID.
- Sensitive to model accuracy (model parameters)
Summary

Constraints of control output/effort:

- **Clipping** technique: simple constraints represented by saturation: easy to implement, it comes with cost if
  - control output is too limited
  - Disturbance is too large

Then, the whole control system becomes unstable if saturation is significant.

- **Anti-windup**: introducing feedback of the saturated quantity into LQR integral control.

- *Constrained optimization* (more info in the MPC lecture)