Autonomous Robotics
Winter 2024
Abhishek Gupta
TAs: Karthikeya Vemuri, Arnav Thareja
Marius Memmel, Yunchu Zhang

Slides borrowed from many sources – Sidd Srinivasa, Sanjiban Choudhury
Class Outline

State Estimation
- Robotic System Design
- Filtering
- Localization
- SLAM

Control
- Feedback Control
- PID Control
- MPC
- LQR

Planning
- Search
- Heuristic Search
- Motion Planning
- Lazy Search

Learning
- Imitation Learning
- Policy Gradient
- Actor-Critic
- Model-Based RL
HW 2 due on Feb 2

HW3 out on Feb 3 (Saturday) barring no hiccups in testing 😊
Recap of Pure Pursuit

From tracking to optimal control

Linear Quadratic Regulator
Aerial combat in which aircraft *pursues* another aircraft by pointing its nose directly towards it

Similar to carrot on a stick!
Rationale: Controller should leverage model!

\[ \dot{x} = v \cos \theta \]
\[ \dot{y} = v \sin \theta \]
\[ \dot{\theta} = \omega = \frac{v}{R} = \frac{v \tan \delta}{L} \]

PID control doesn’t directly utilize the fact that we know the kinematic car model.
Key Idea:

The car is always moving in a circular arc
Pure Pursuit Controller

- Assume the car is moving with fixed steering angle
Consider a reference at a lookahead distance

Problem: Can we solve for a steering angle that guarantees that the car will pass through the reference?
Solution: Compute a circular arc

We can always solve for a arc that passes through a lookahead point

Note: As the car moves forward, the point keeps moving
Pure pursuit: Keep chasing lookahead

1. Find a lookahead and compute arc
2. Move along the arc
3. Go to step 1
Equations of Motion

\[ \dot{x} = v \cos \theta \]
\[ \dot{y} = v \sin \theta \]
\[ \dot{\theta} = \omega = \frac{v}{R} = \frac{v \tan \delta}{L} \]

\[ \tan \delta = \frac{L}{R} \rightarrow R = \frac{L}{\tan \delta} \]
\[
\dot{x} = f(x, u)
\]

\[
\begin{bmatrix}
    x \\ y \\ \theta \\
\end{bmatrix}
\]

- X-COORDINATE
- Y-COORDINATE
- HEADING

\[
\begin{bmatrix}
    v \\
    \delta
\end{bmatrix}
\]

- SPEED
- STEERING ANGLE

\[
\begin{bmatrix}
    1 \\
    u
\end{bmatrix}
\]
Pure pursuit: Control law derivation

Whiteboard
Computing the Arc Radius

\[(R_{pp} - b)^2 + a^2 = R_{pp}^2\]

\[R_{pp} = \frac{a^2 + b^2}{2b}\]
Computing the Arc Radius

$$R_{pp} = \frac{a^2 + b^2}{2b}$$

$$\begin{bmatrix} a \\ b \end{bmatrix} = R(-\theta) \left( \begin{bmatrix} x_{\text{ref}} \\ y_{\text{ref}} \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix} \right)$$

Different than cross-track error (this is ref. position in robot frame; vice versa for cross-track error)
Computing the Steering Angle

\[ R_{pp} = \frac{a^2 + b^2}{2b} \]

\[ \tan \delta = \frac{L}{R_{pp}} \]
Question: How do I choose L?
Controller Design Decisions

1. Get a reference path/trajectory to track
2. Pick a reference state from the reference path/trajectory
3. Compute error to reference state
4. Compute control law to minimize error

- Option 1: Bang-bang control
- Option 2: PID control
- Option 3: Pure-pursuit control

Are we done?
Recap of Pure Pursuit

From tracking to optimal control

Linear Quadratic Regulator
Controller Design Decisions

Option 1: Bang-bang control
Option 2: PID control
Option 3: Pure-pursuit control

Very particular cost function

Model Agnostic

Very simplistic model
Control as an Optimization Problem

For a sequence of H control actions
1. Use model to predict consequence of actions (i.e., H future states)
2. Evaluate the cost function

Compute optimal sequence of H control actions (minimizes cost)
Generalized Problem: Optimal Control

- Minimize sum of costs, subject to dynamics and other constraints

\[
\min_{u_1:T} \sum_{t=1}^{T} g(x_t, u_t) + G(x_T, u_T)
\]

s.t. \( x_{t+1} = Ax_t + Bu_t \)

Can be costs like smoothness, preferences, speed
Can be constraints like velocity/acceleration bounds
Linear Quadratic Regulator

- **Linear** system (model)
- **Quadratic** cost function to minimize

\[ x_{t+1} = Ax_t + Bu_t \]

\[ \sum_t x_t^\top Q x_t + u_t^\top Ru_t \]
Linear System

- **Linear** system (model)
- **Quadratic** cost function to minimize

\[ x_{t+1} = Ax_t + Bu_t \]
\[ \sum_t x_t^\top Q x_t + u_t^\top Ru_t \]

\[
x_{t+1} = A x_t + B u_t \\
(N \times 1) \quad (N \times N)(N \times 1) \quad (N \times M)(M \times 1)
\]

STATE → NEXT STATE  \hspace{1cm} CONTROL → NEXT STATE
Example: Inverted Pendulum (Linear System)

\[ mgl \sin \theta + \tau = ml^2 \ddot{\theta} \]
\[ \ddot{\theta} = \frac{g}{l} \sin \theta + \frac{\tau}{ml^2} \approx \frac{g}{l} \theta + \frac{\tau}{ml^2} \]

\[
\begin{bmatrix}
\theta_{t+1} \\
\dot{\theta}_{t+1}
\end{bmatrix} = \begin{bmatrix}
1 & \Delta t \\
\frac{g}{l} \Delta t & 1
\end{bmatrix} \begin{bmatrix}
\theta_t \\
\dot{\theta}_t
\end{bmatrix} + \begin{bmatrix}
0 \\
\Delta t
\end{bmatrix} \frac{\tau}{ml^2}
\]
Quadratic Cost Function

- **Linear** system (model)
- **Quadratic** cost function to minimize

\[
x_{t+1} = Ax_t + Bu_t \\
\sum_t x_t^\top Q x_t + u_t^\top Ru_t
\]

\[
x_t^\top Q x_t \quad (1 \times N)(N \times N)(N \times 1)
\]

**STATE COST**

\[
u_t^\top Ru_t \quad (1 \times M)(M \times M)(M \times 1)
\]

**CONTROL COST**
Example: Inverted Pendulum (State Cost)

\[ x_t^\top Q x_t \]  

\[ = \begin{bmatrix} \theta_t \\ \dot{\theta}_t \end{bmatrix}^\top \begin{bmatrix} Q_{\theta\theta} & Q_{\theta\dot{\theta}} \\ Q_{\dot{\theta}\theta} & Q_{\dot{\theta}\dot{\theta}} \end{bmatrix} \begin{bmatrix} \theta_t \\ \dot{\theta}_t \end{bmatrix} \]

\[ = Q_{\theta\theta} \theta_t^2 + 2Q_{\theta\dot{\theta}} \theta_t \dot{\theta}_t + Q_{\dot{\theta}\dot{\theta}} \dot{\theta}_t^2 \]

\[ Q > 0 \iff z^\top Q z > 0, \ \forall z \neq 0 \]
Example: Inverted Pendulum (Control Cost)

\[ u_t^\top Ru_t \]  

\[ = \frac{\tau_t}{ml^2} \left[ R_{\tau\tau} \right] \frac{\tau_t}{ml^2} \]

\[ = R_{\tau\tau} \left( \frac{\tau_t}{ml^2} \right)^2 \]

\[ R > 0 \iff z^\top Rz > 0, \ \forall z \neq 0 \]
Example: Inverted Pendulum
How do we solve for controls?

Dynamic programming to the rescue!

Start from timestep $T-1$ and solve backwards.
Bellman Equation for Dynamic Programming

- **Linear** system (model)
- **Quadratic** cost function to minimize

\[ x_{t+1} = Ax_t + Bu_t \]
\[ \sum_t x_t^\top Q x_t + u_t^\top R u_t \]

\[ J^*(x_t) = \min_{u_t} x_t^\top Q x_t + u_t^\top R u_t + J^*(x_{t+1}) \]

**MINIMUM COST, STARTING FROM** \( x_t \) **IMMEDIATE COST** **MINIMUM FUTURE COST, STARTING FROM** \( x_{t+1} \)
\[ J_0(x) = \min_u x^\top Q x + u^\top R u = x^\top Q x = x^\top P_0 x \]

Minimized with \( u = 0 \)

\( P_0 = Q \)

Note that the cost is quadratic in \( x \)

\( x_{\text{ref}} = 0 \)
Take one step towards the start: Time-to-go = 1

\[ J_0(x) = \min_u x^\top Q x + u^\top Ru = x^\top Q x = x^\top P_0 x \]

\[ J_1(x) = \min_u x^\top Q x + u^\top Ru + J_0(Ax + Bu) \]

Solve for control at timestep T-1, accounting for impact on the future, through dynamics.
Take one step towards the start: Time-to-go = 1

\[ J_1(x) = \min_{u} x^\top Q x + u^\top R u + J_0(Ax + Bu) \]
Value Iteration (Horizon = 1)

\[ J_1(x) = \min_u \left[ x^\top Q x + u^\top R u + (Ax + Bu)^\top P_0 (Ax + Bu) \right] \]

\[ \nabla_u [\cdot] = 2Ru + 2B^\top P_0 (Ax + Bu) = 0 \]

\[ u = - (R + B^\top P_0 B)^{-1} B^\top P_0 A x \]

\[ J_1(x) = x^\top P_1 x \]

\[ P_1 = Q + K_1^\top R K_1 + (A + BK_1)^\top P_0 (A + BK_1) \]

\[ K_1 = - (R + B^\top P_0 B)^{-1} B^\top P_0 A \]
Turns into a recursion at time-to-go = \( i \)

\[
K_i = -(R + B^\top P_{i-1} B)^{-1} B^\top P_{i-1} A
\]

\[
P_i = Q + K_i^\top R K_i + (A + B K_i)^\top P_{i-1} (A + B K_i)
\]

\[
u = K_i x, \quad J_i(x) = x^\top P_i x
\]

Optimal controller is linear in \( x \)

Optimal cost is quadratic in \( x \)

**RUNTIME:** \( O(H(n^3 + m^3)) \)
The LQR algorithm

Algorithm OptimalValueControl(A, B, Q, R, time-to-go):

    if time-to-go == 0:
        return Q, 0
    else:
        P_{i-1} = OptimalValueControl(A, B, Q, R, time-to-go - 1)
        K_i = -(R + B^T P_{i-1} B)^{-1} B^T P_{i-1} A
        P_i = Q + K_i^T R K_i + (A + B K_i)^T P_{i-1} (A + B K_i)
        return K_i, P_i

Optimal controller is linear in x

Optimal cost is quadratic in x
Unpacking LQR intuitively

\[ K_i = -(R + B^\top P_{i-1} B)^{-1} B^\top P_{i-1} A \]

\[ P_i = Q + K_i^\top R K_i + (A + BK_i)^\top P_{i-1} (A + BK_i) \]

\[ u = K_i x, \quad J_i(x) = x^\top P_i x \]
Unpacking LQR intuitively

\[ K_i = -\left( R + B^T P_{i-1} B \right)^{-1} B^T P_{i-1} A \]

Recall Kalman Filtering

Set \( A, B = I \)

\[ \frac{B^T P_{i-1} A}{R + B^T P_{i-1} B} \]

Tradeoff between future cost \( P_{i-1} \) and current cost \( R \)

\[ \frac{P_{i-1}}{R + P_{i-1}} \]
Unpacking LQR intuitively

\[ P_i = Q + K_i^\top R K_i + (A + B K_i)^\top P_{i-1} (A + B K_i) \]

- Current state cost
- Current action cost
- Optimal cost in the future based on dynamics
For **linear** systems with **quadratic** costs, we can write down very efficient algorithms that return the optimal sequence of actions!

- Special case where dynamic programming can be applied to continuous states and actions (typically only discrete states and actions)

- Many LQR extensions: non-linear systems, linear time-varying systems, trajectory following for non-linear systems, arbitrary costs, etc.
Recap: Course Overview

- MDPs and RL
- Imitation Learning
- Solving POMDPs

- Filtering/Smoothing
- Localization
- Mapping
- SLAM

- Search
- Motion Planning
- TrajOpt
- Stability/Certification

Mapping

SLAM
Class Outline

- **State Estimation**
  - Robotic System Design
  - Filtering
  - Localization
  - SLAM

- **Planning**
  - Search
  - Heuristic Search
  - Motion Planning
  - Lazy Search

- **Control**
  - Feedback Control
  - PID Control
  - MPC
  - LQR

- **Learning**
  - Imitation Learning
  - Policy Gradient
  - Actor-Critic
  - Model-Based RL