Principles of Robot Autonomy I

Planning and Control: Trajectory optimization, trajectory planning

Principles of Robot Autonomy I

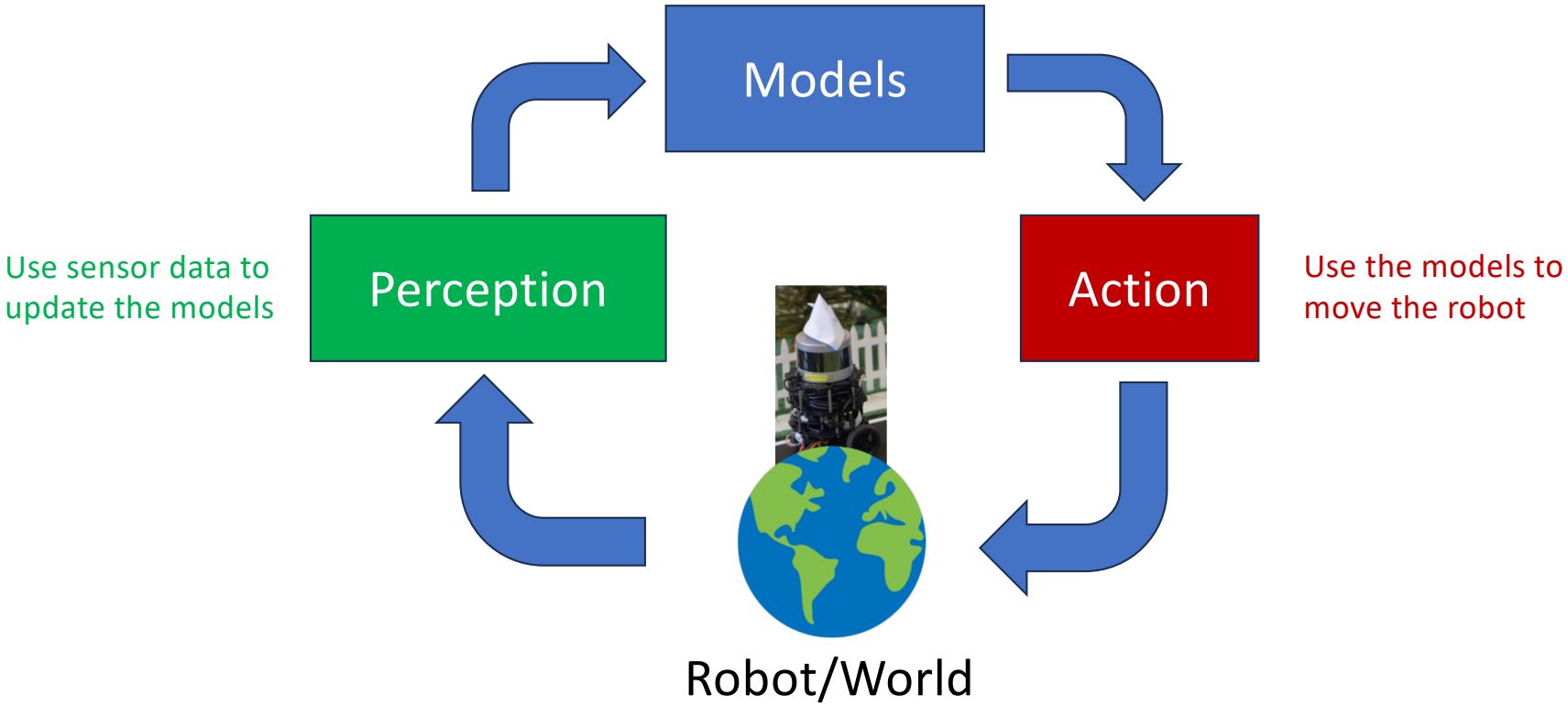
Announcements:

- Lab section 2 this week: will work with TurtleBots
- HW 1 due Thurs, Oct 9

Lecture 5:

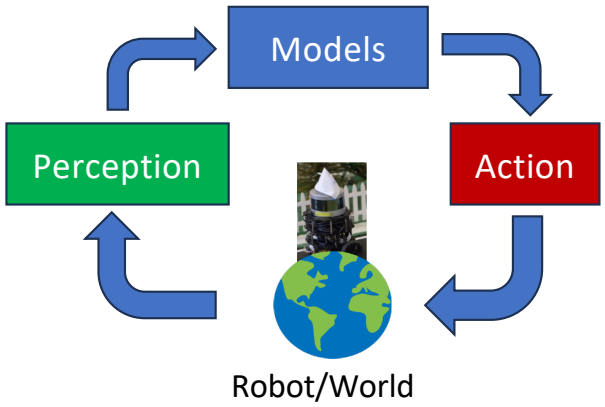
- Trajectory Optimization
- Kinodynamic sampling based traj opt
- Spline based traj opt

Full Stack Autonomy: the Perception-Action Loop



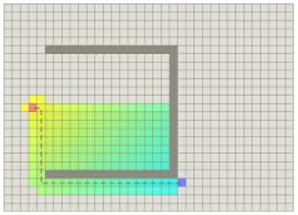
Action: Planning and Control Stack

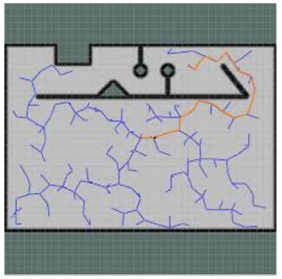
Use the models to move the robot



Path planning:

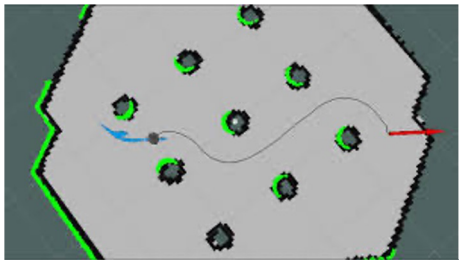
- A* planning on a graph
- RRT, RRT*





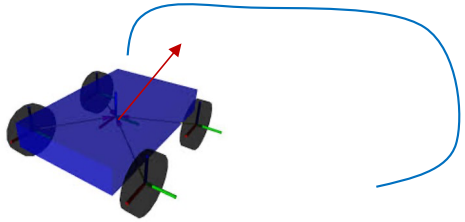
Trajectory optimization:

- Optimal control
- Numerical solvers



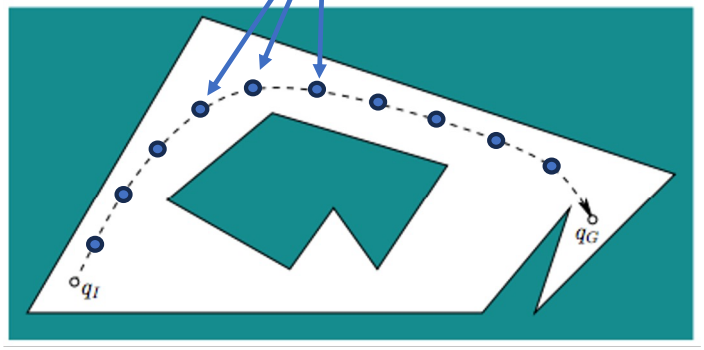
Trajectory following:

- Kinematic/dynamic models
- PID control
- LQR, gain scheduled LQR



Trajectory Optimization

Waypoint poses from path planner
(e.g. A*, RRT*)



Trajectory optimization problem:

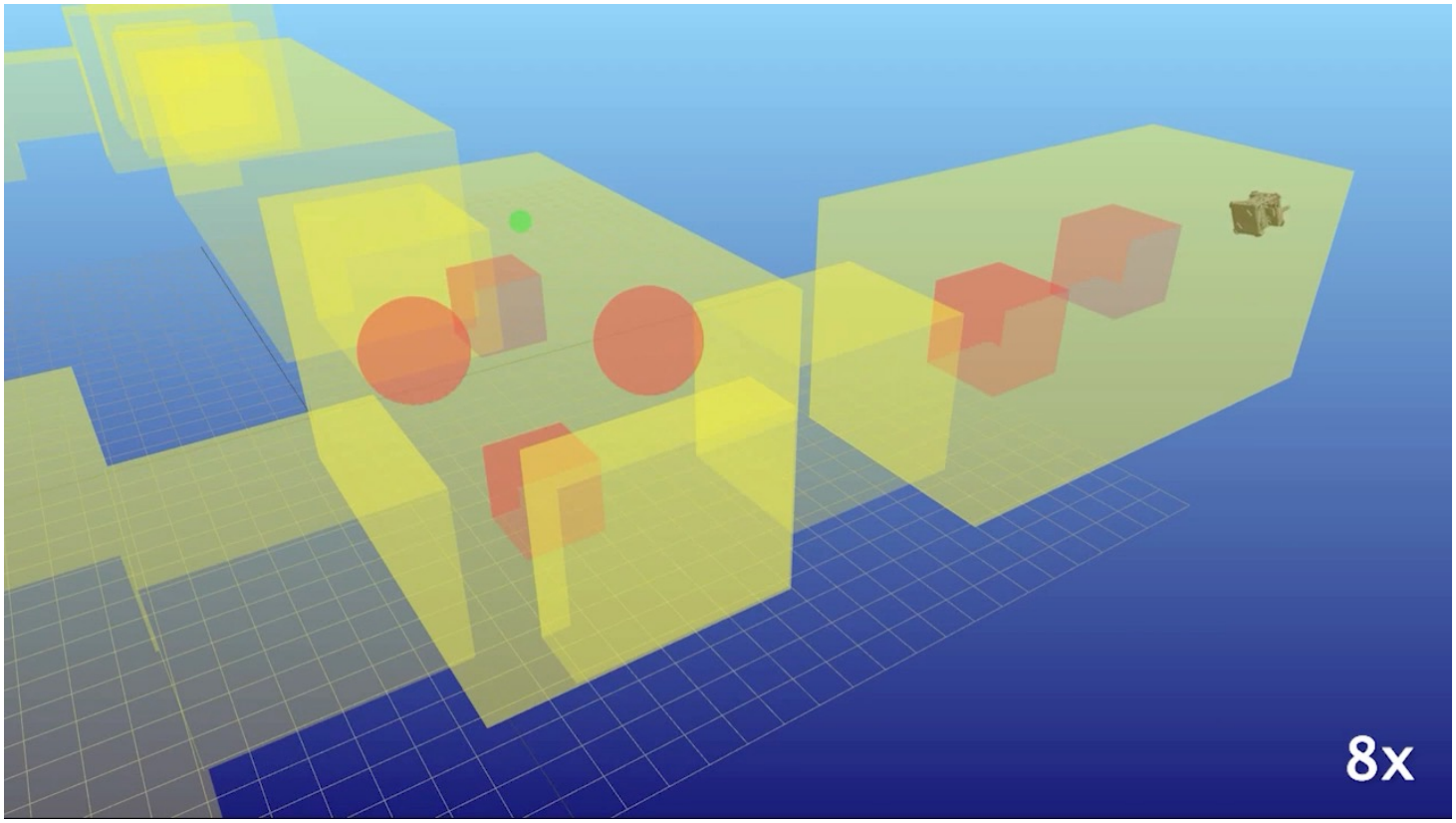
$$\min_{u_{0:T}} \sum_{t=0}^{T} c_t(x_t, u_t)$$ Traj. cost

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t)$$ Dynamic constraints

$$x_t \in \mathbb{X}_{\text{Free}}$$ Collision constraints

Goal: Find a dynamically feasible trajectory (close to the path) so the robot doesn't hit things!

Trajectory Optimization for an AstroBee Flying inside the ISS



Trajectory Optimization for a drone flying through a NeRF environment map

Adamkiewicz, M., Chen, T., Caccavale, A., Gardner, R., Culbertson, P., Bohg, J. and Schwager, M., Vision-only robot navigation in a neural radiance world. *IEEE Robotics and Automation Letters*, 2022.

Trajectory Optimization for High-Speed Car

Li, Yanbo, Zakary Littlefield, and Kostas E. Bekris, IJRR, 2016.

Trajectory Optimization for Dynamic Manipulator

Quang-Cuong Pham, Stéphane Caron, Puttichai Lertkultanon, Yoshihiko Nakamura. IJRR 2017.

Optimal control problem: path planning, traj opt, and feedback control combined

The problem:

$$\min_u \int_0^T c(x(t), u(t))\, dt + h(x(T))$$

Stage cost

Terminal cost

$$\text{subject to} \quad \dot{x} = f(x(t), u(t))$$

Dynamic/kinematic constraints

$$x(0) = x_0$$

Initial conditions

$$x(t) \in \mathcal{X}_{free}$$

Safety, collision avoidance constraints

- Typically cannot solve directly. Must break down into simpler probs
- See AA203 for more details
- Good reference: D. K. Kirk. Optimal Control Theory: An introduction. 2004.

Form of optimal control solution

- **Closed loop:** If a functional relationship can be found in form

$$\mathbf{u}^*(t) = \pi(\mathbf{x}(t), t)$$ **Control law or control policy**

- **Open loop:** If the optimal control law is determined as a function of time for a specified initial state value

$$\mathbf{u}^*(t) = \mathbf{f}(\mathbf{x}(t_0), t)$$ **Feedforward or open loop control**

- A good compromise: two-step design

$$\mathbf{u}^*(t) = \mathbf{u}_d(t) + \pi(\mathbf{x}(t), \mathbf{x}(t) - \mathbf{x}_d(t))$$

Reference trajectory

feedforward control from traj opt

Trajectory-tracking law (closed-loop), from feedback control

Tracking error

Traj opt: Discrete time open-loop control

- We want to find

$$u^*_{0:T} = (u^*_0, u^*_1, \ldots, u^*_T)$$
$$x^*_{0:T} = (x^*_0, x^*_1, \ldots, x^*_T)$$

$$\text{such that}$$
$$x^*_{t+1} = f(x^*_t, u^*_t)$$

"Dynamically feasible"

- In general, two broad classes of methods:

1. Direct methods: transcribe infinite problem into finite dimensional, nonlinear programming (NLP) problem, and solve NLP ⇒ "First discretize, then optimize"
2. Indirect methods: attempt to find a minimum point "indirectly," by solving the necessary conditions of optimality with **Pontryagin's Minimum Principle** ⇒ "First optimize, then discretize"

Direct method: Transcribe to Discrete Time

$$\min_{u} \int_0^T c(x(t), u(t))\, dt + h(x(T))$$

$$\text{subject to} \quad \dot{x} = f(x(t), u(t))$$

$$x(0) = x_0$$

$$x(t) \in \mathcal{X}_{free}$$

1st Order Euler DIscretization

$$\min_{u_{0:T}} \sum_{0}^{T} c(x_t, u_t)$$

$$\text{subject to} \quad x_{t+1} = f(x_t, u_t)$$

$$x_0 = x(0)$$

$$x_t = \mathcal{X}_{free}$$

Nonlinear program (NLP) –
generic numerical optimization problem

E.g. Linear Quadratic Regulator (LQR)

The problem:

Quadratic state cost

Quadratic control cost

$$\min_{u_{0:T-1}} \sum_{t=0}^{T-1} (x_t^T Q x_t + u_t^T R u_t) + x_T^T Q_T x_t$$

$$\text{subject to} \quad x_{t+1} = A_t x_t + B_t u_t$$

Linear dynamics

$$x_0 = x(0)$$

Initial conditions

- Regulates state to $x(t) \to 0$ and keeps it there
- Exact **closed loop** solution by Kalman in 1960s for continuous and discrete time.
- Still widely used. See AA212 for more details.
- Often used as a module in more complex control problems.

LQR Solution

Linear feedback: $u_t = -K_t x_t$

Kalman gain: $K_t = (R + B^T P_{t+1})^{-1} B^T P_{t+1} A$

Riccati Recursion: $P_{t-1} = A^T P_t A - A^T P_t B (B^T P_t B + R)^{-1} B^T P_t A + Q$

Final condition: $P_T = Q_T$

- Solve Riccati backward in time starting from T -> 0 to get K_t before applying control

E.g. Quadratic Tracking Objective

The problem:

Quadratic state tracking cost **Quadratic control cost**

$$\min_{\delta u_{0:T-1}} \sum_{t=0}^{T-1} \Big((x_t - x_t^{ref})^T Q (x_t - x_t^{ref}) + (u_t - u_t^{ref})^T R (u_t - u_t^{ref}) \Big) + (x_T - x_t^{ref})^T Q_T (x_T - x_T^{ref})$$

$$\text{subject to} \quad x_{t+1} = f(x_t, u_t)$$

Nonlinear dynamics

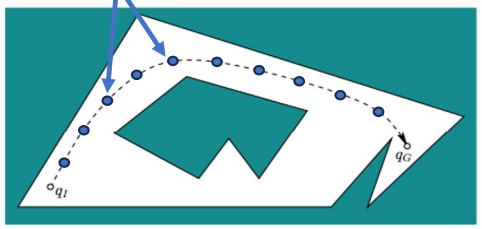
$$x_0 = x(0)$$

Initial conditions

$$x_t \in \mathcal{X}_{free}$$

Safety, collision avoidance constraints

Ref states



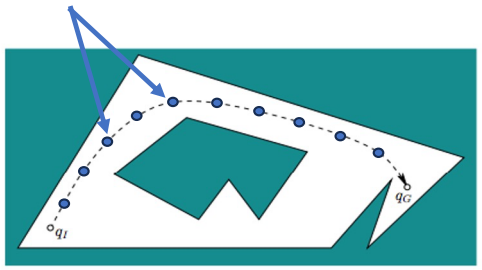
- Attempts to track reference state $x(t) \rightarrow x_{ref}(t)$
- Usually discretize time
- Solve for open **loop control** with numerical optimizer

Iterative LQR

Iterate k:

Given previous iterate's solution: $x_{1:T}^{k-1}, u_{0:T-1}^{k-1}$

Ref state



$$\min_{\delta u_{0:T-1}} \sum_{t=0}^{T-1} (\delta x_t^T Q \delta x_t + \delta u_t^T R \delta u_t) + \delta x_T^T Q_T \delta x_T$$

Linearized dynamics about k-1 solution:

$$\text{subject to} \quad \delta x_{t+1} = A_t \delta x_t + B_t \delta u_t$$

$$x_0 = x(0)$$

State variation: $\delta x_t = x_t - x_t^{k-1}$

Control variation: $\delta u_t = u_t - u_t^{k-1}$

Specialized solution techniques

- Iterative LQR (iLQR)/Differential Dynamic Programming (DDP) – Guess solution, take linear-quadratic approx about guess, solve LQR analytically, iterate to convergence. Only if no collision constraints.
- Sequential Quadratic Programming (SQP) – Guess solution, linear-quadratic approx. about guess, solve QP numerically with linear constraints. Iterate about new solution to convergence.
- Augmented Lagrangian Traj Opt (ALTRO) – Guess solutions, linear-quadratic approx. about guess, form Augmented Lagrangian with linear constraints, solve LQR analytically, iterate with dual-ascent.

Software packages:

Traj opt software packages:

- DIDO: http://www.elissarglobal.com/academic/products/
- PROPT: http://tomopt.com/tomlab/products/propt/
- GPOPS: http://www.gpops2.com/
- CasADi: https://github.com/casadi/casadi/wiki
- ACADO: http://acado.github.io/
- Trajax: https://github.com/google/trajax

General Constrained Optimization software packages often used for Traj Opt:

- Scipy.optimize: https://docs.scipy.org/doc/scipy/reference/optimize.html
- SNOPT: https://en.wikipedia.org/wiki/SNOPT
- IPOPT: https://en.wikipedia.org/wiki/IPOPT
- GUROBI: https://www.gurobi.com/

In addition to implementing efficient trajectory optimization algorithms, many of these tools provide easier-to-use modeling languages for problem specification.

Other solution methods

Sampling-based planning can be re-tooled for traj opt!

Often called kinodynamic (sampling based) motion planning.

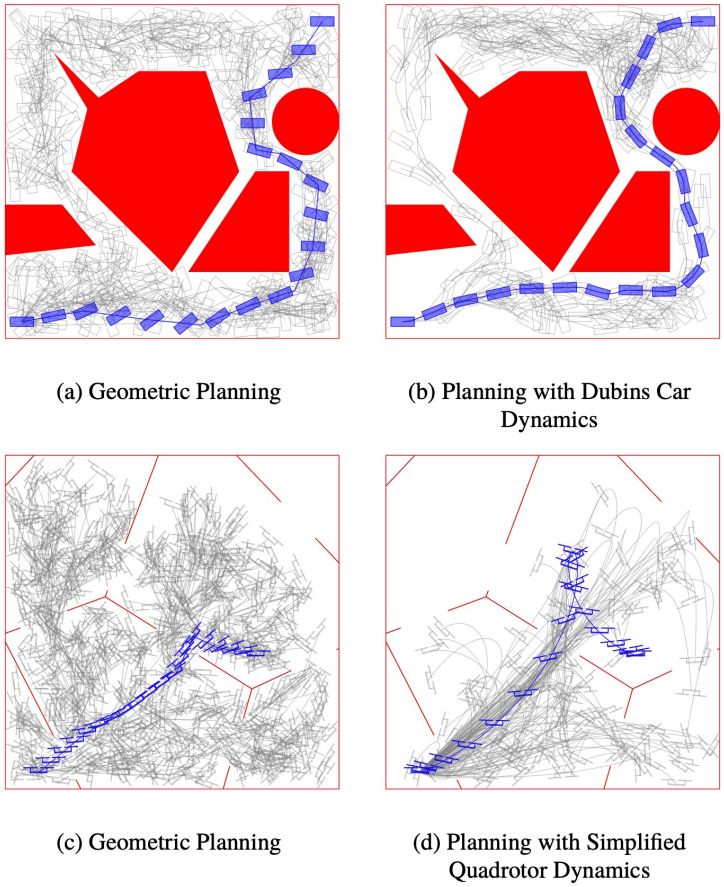
Kinodynamic planning

Kinodynamic motion planning problem: in addition to obstacle avoidance, paths are subject to differential constraints

- Path planning in the **state space** X, poses and velocities
- To move the robot applies control $u \in U$
- Motion needs to satisfy the system's constraints:

$$\dot{x} = f(x, u) \text{ for } x \in X, u \in U$$

Reference: Schmerling and Pavone. Kinodynamic Planning. 2019



(a) Geometric Planning

(b) Planning with Dubins Car Dynamics

(c) Geometric Planning

(d) Planning with Simplified Quadrotor Dynamics

Kinodynamic RRT

RRT can be extended to the kinodynamic case in a relatively easy way:

1. Draw a random state and find its nearest neighbor x_{near}
2. Sample a random control $u \in U$ and random duration t
3. Forward propagate the control u for t time from x_{near}

1: $\mathcal{T}$.init(x_{init})
2: **for** $i = 1$ to k **do**
3: $x_{\text{rand}} \leftarrow$ RANDOM_STATE()
4: $x_{\text{near}} \leftarrow$ NEAREST_NEIGHBOR($x_{\text{rand}}, \mathcal{T}$)
5: $t \leftarrow$ SAMPLE_DURATION($0, T_{\text{prop}}$)
6: $u \leftarrow$ SAMPLE_CONTROL_INPUT($\mathbb{U}$)
7: $x_{\text{new}} \leftarrow$ PROPAGATE(x_{near}, u, t)
8: **if** COLLISION_FREE($x_{\text{near}}, x_{\text{new}}$) **then**
9: $\mathcal{T}$.add_vertex(x_{new})
10: $\mathcal{T}$.add_edge($x_{\text{near}}, x_{\text{new}}$)
11: **return** $\mathcal{T}$

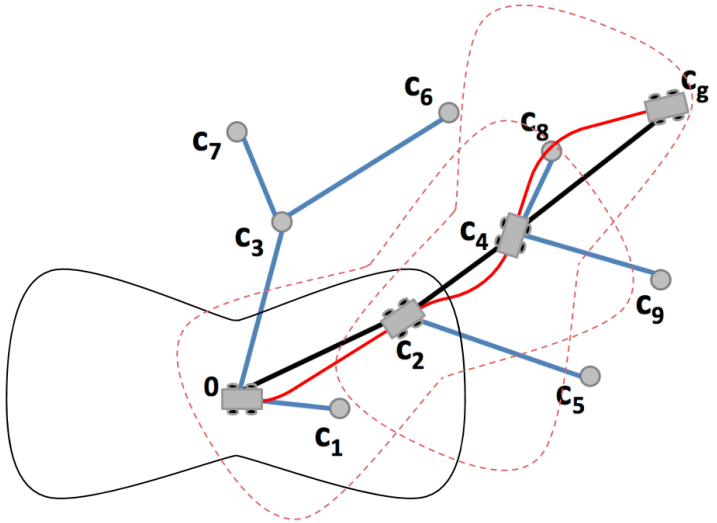
Reference: Kleinbort et al. Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation. 2018.

RRT* and PRM not so easy!

- Embeds trajectory optimization inside of sampling-based planning
- Must solve a “small” optimal control/traj opt problem to link up two desired states with a dynamically feasible, collision free trajectory

1. Connect samples in state space by using an optimal trajectory (steering problem)
2. Use reachable sets to find nearest neighbors
3. Con: unless we have a very fast traj opt solve, these algos are impractical

Reference: E. Schmerling et al. Optimal Sampling-Based Motion Planning under Differential Constraints: the Driftless Case. 2015



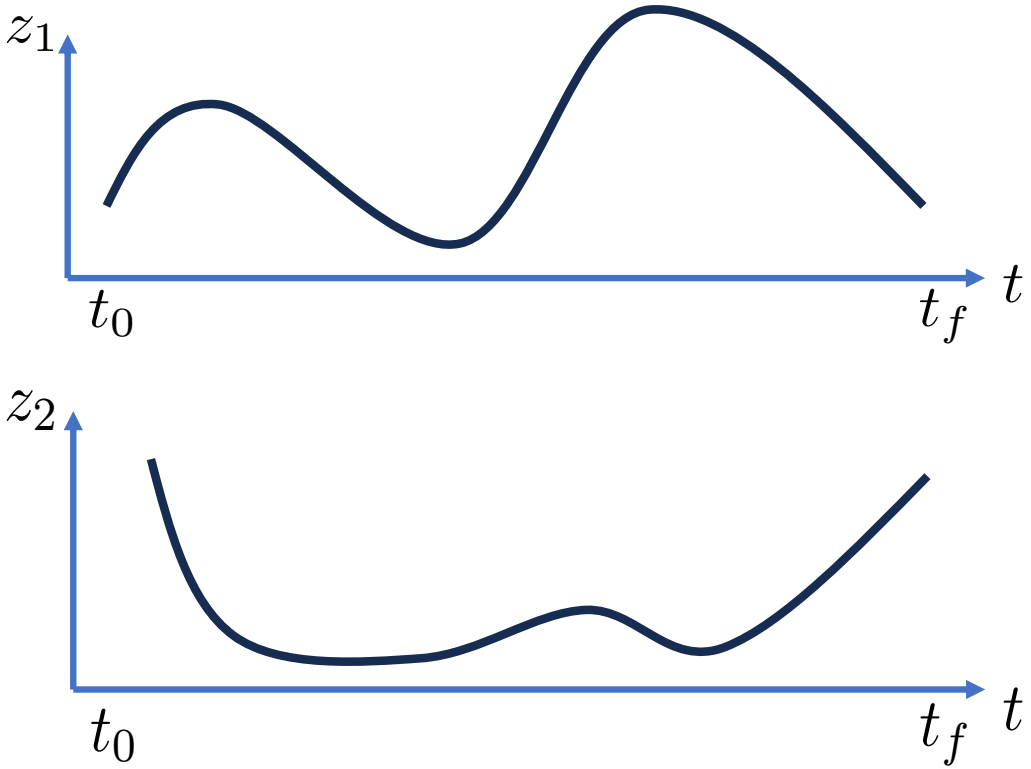
Other solution methods

We can ignore dynamics and instead “smooth” a discrete path with splines.

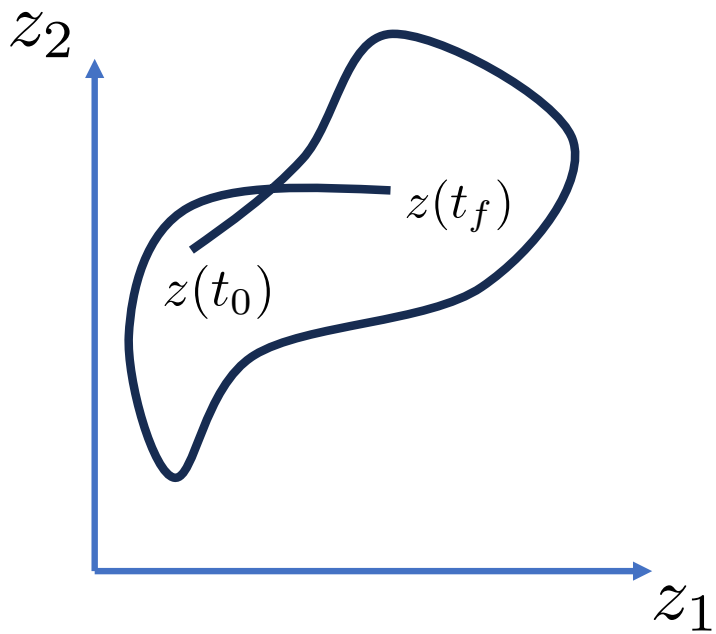
Rely on feedback control to follow smoothed trajectory

Spline-based trajectory planning

Trajectory Splines: one for each pose dimension



$$z(t) = \begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix}$$



Time-parameterized trajectory

Polynomial Splines

$$z_i(t) = \alpha_N^i t^N + \alpha_{N-1}^i t^{N-1} + \ldots + \alpha_1^i t + \alpha_0^i =$$

$$\begin{bmatrix} t^N & t^{N-1} & \cdots & t & 1 \end{bmatrix} \begin{bmatrix} \alpha_N^i \\ \alpha_{N-1}^i \\ \vdots \\ \alpha_1^i \\ \alpha_0^i \end{bmatrix} = \Psi(t)^T \alpha^i$$

Coefficients define the shape

See also:

- Bezier curves
- Chebyshev polynomials

both examples of polynomial splines.

Solve for the spline coefficients

Given desired initial and final states: $z(t_0), \dot{z}(t_0) \quad z(t_f), \dot{z}(t_f)$

Then solve linear system of equations with matrix inverse:

$$\begin{bmatrix} \psi_1(t_0) & \psi_2(t_0) & \dots & \psi_N(t_0) \\ \dot{\psi}_1(t_0) & \dot{\psi}_2(t_0) & \dots & \dot{\psi}_N(t_0) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_0) & \psi_2^{(q)}(t_0) & \dots & \psi_N^{(q)}(t_0) \\ \psi_1(t_f) & \psi_2(t_f) & \dots & \psi_N(t_f) \\ \dot{\psi}_1(t_f) & \dot{\psi}_2(t_f) & \dots & \dot{\psi}_N(t_f) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_f) & \psi_2^{(q)}(t_f) & \dots & \psi_N^{(q)}(t_f) \end{bmatrix} \begin{bmatrix} \alpha_1^{[j]} \\ \alpha_2^{[j]} \\ \vdots \\ \alpha_N^{[j]} \end{bmatrix} = \begin{bmatrix} z_j(t_0) \\ \dot{z}_j(t_0) \\ \vdots \\ z_j^{(q)}(t_0) \\ z_j(t_f) \\ \dot{z}_j(t_f) \\ \vdots \\ z_j^{(q)}(t_f) \end{bmatrix}$$

Notice: $\dot{z}_i(t) = \begin{bmatrix} Nt^{N-1} & (N-1)t^{N-2} & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha_N^i \\ \alpha_{N-1}^i \\ \vdots \\ \alpha_1^i \\ \alpha_0^i \end{bmatrix} = \dot{\Psi}(t)^T \alpha^i$

Min-Snap Planning

If you have more degrees of freedom in spline, matrix short-fat -> many solutions to system of equations.

Solve smooth optimization problem.

$$\min_{\alpha_{1:N}} \int_{t_0}^{t_f} \|\dddot{z}(t)\|^2 \, dt$$

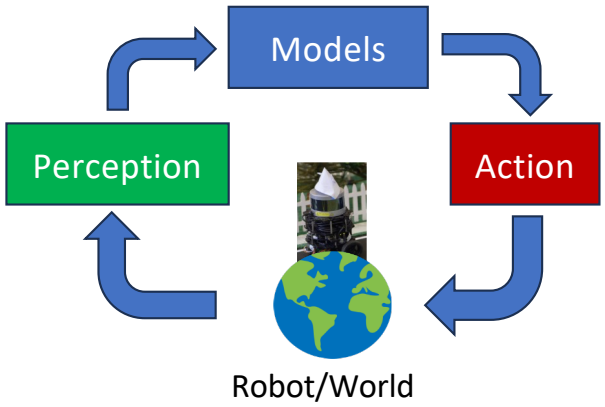
"snap" – 4th time derivative

$$\text{subject to} \quad \begin{bmatrix} \psi_1(t_0) & \psi_2(t_0) & \dots & \psi_N(t_0) \\ \dot{\psi}_1(t_0) & \dot{\psi}_2(t_0) & \dots & \dot{\psi}_N(t_0) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_0) & \psi_2^{(q)}(t_0) & \dots & \psi_N^{(q)}(t_0) \\ \psi_1(t_f) & \psi_2(t_f) & \dots & \psi_N(t_f) \\ \dot{\psi}_1(t_f) & \dot{\psi}_2(t_f) & \dots & \dot{\psi}_N(t_f) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_f) & \psi_2^{(q)}(t_f) & \dots & \psi_N^{(q)}(t_f) \end{bmatrix} \begin{bmatrix} \alpha_1^{[j]} \\ \alpha_2^{[j]} \\ \vdots \\ \alpha_N^{[j]} \end{bmatrix} = \begin{bmatrix} z_j(t_0) \\ \dot{z}_j(t_0) \\ \vdots \\ z_j^{(q)}(t_0) \\ z_j(t_f) \\ \dot{z}_j(t_f) \\ \vdots \\ z_j^{(q)}(t_f) \end{bmatrix}$$

Very common in drone trajectory planning

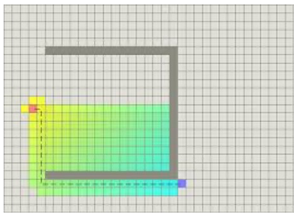
Action: Planning and Control Stack

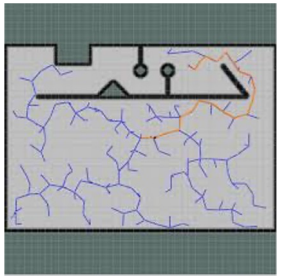
Use the models to move the robot



Path planning:

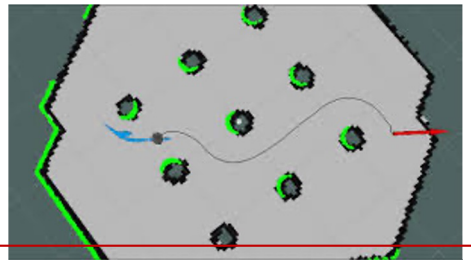
- A* planning on a graph
- RRT, RRT*





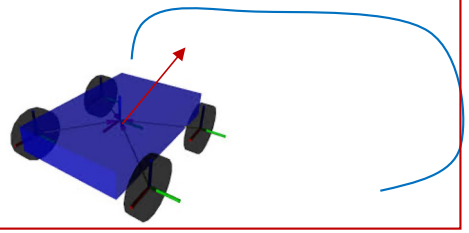
Trajectory optimization:

- Optimal control
- Numerical solvers

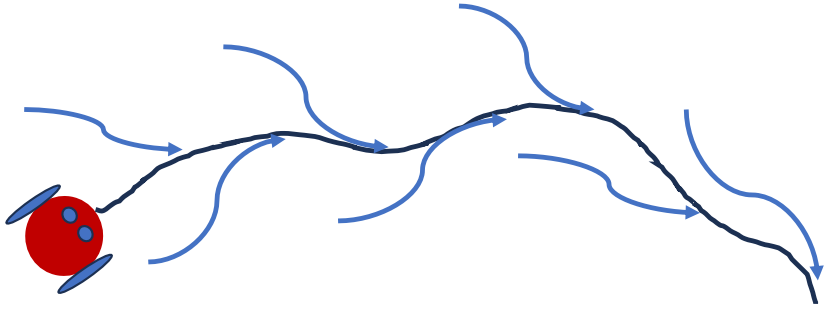


Trajectory following:

- Kinematic/dynamic models
- PID control
- LQR, gain scheduled LQR



Next time: Trajectory Following



Motion model:

$$x_{t+1} = f(x_t, u_t)$$

Feedback control law:

$$u_t = h(x_t, x_t^*)$$

planned traj waypoints

Goal: Use closed loop feedback control to track desired trajectory, rejecting disturbances and being robust to model errors.