Principles of Robot Autonomy I

Planning and Control: Trajectory following, feedback control

Principles of Robot Autonomy I

Announcements:

- Lab section 2 this week, in person
- HW 1 due today, Thurs Oct 9 at 5pm
- HW 2 out today, due Thurs, Oct 16 at 5pm
- Staff email list: **cs237a-aut2526-staff@lists.stanford.edu**
- If your local ROS install not working: Come to **any OHs or any lab section.** We'll have extra laptops with ROS and CAs to help.

Lecture 5:

- Spline trajectory optimization
- Trajectory following, feedback control
- Diff drive robot trajectory following
- Differential flatness
- LQR, gain scheduled LQR

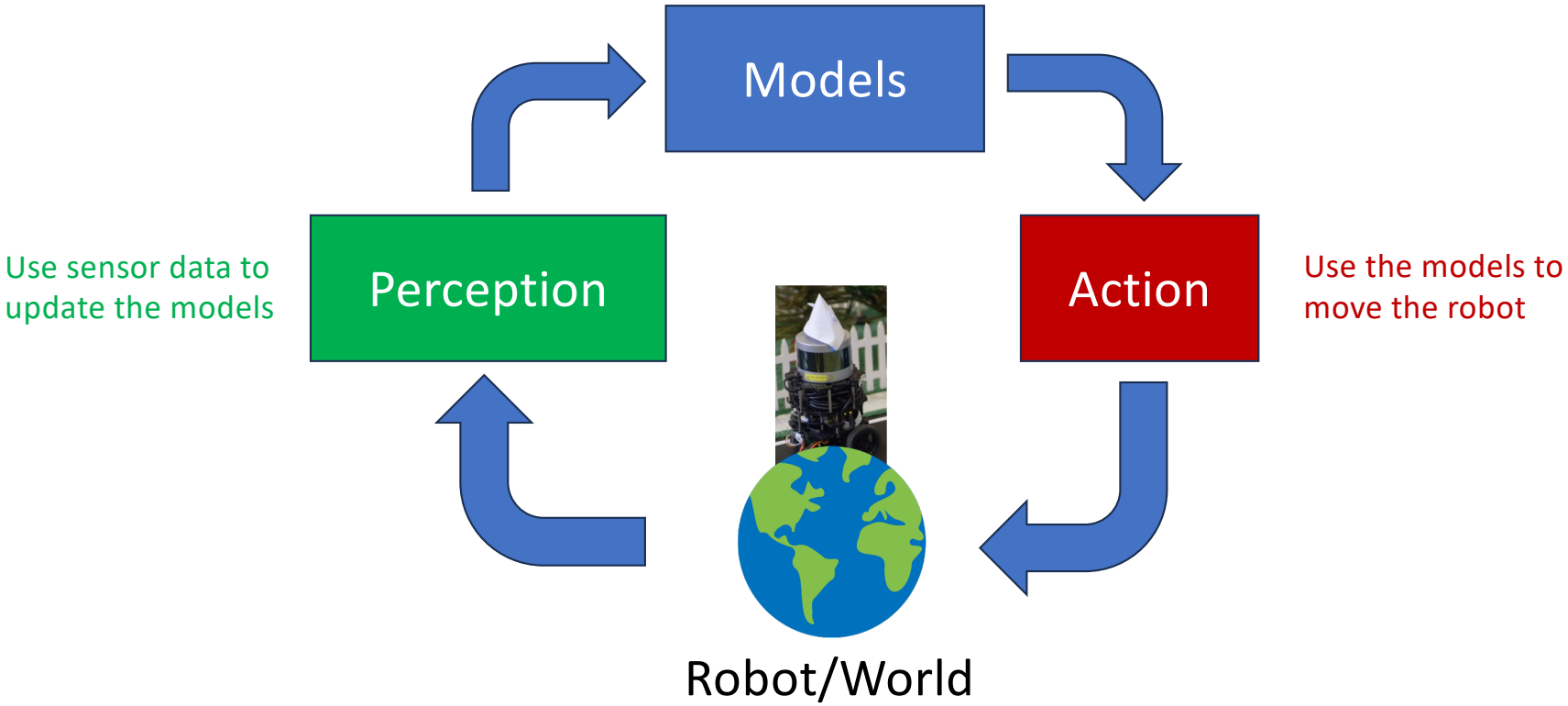
Stanford University

MSL

IPRL

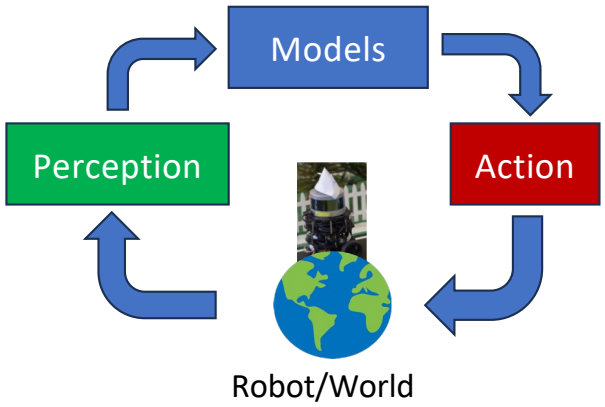
ASL Autonomous Systems Lab

Full Stack Autonomy: the Perception-Action Loop



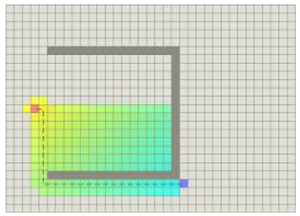
Action: Planning and Control Stack

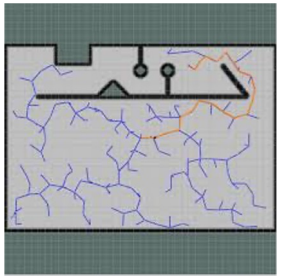
Use the models to move the robot



Path planning:

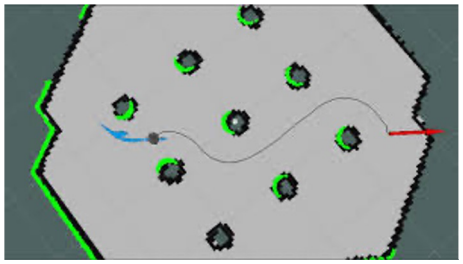
- A* planning on a graph
- RRT, RRT*





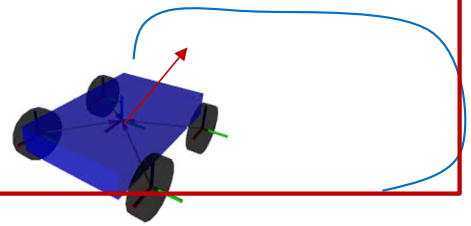
Trajectory optimization:

- Optimal control
- Numerical solvers

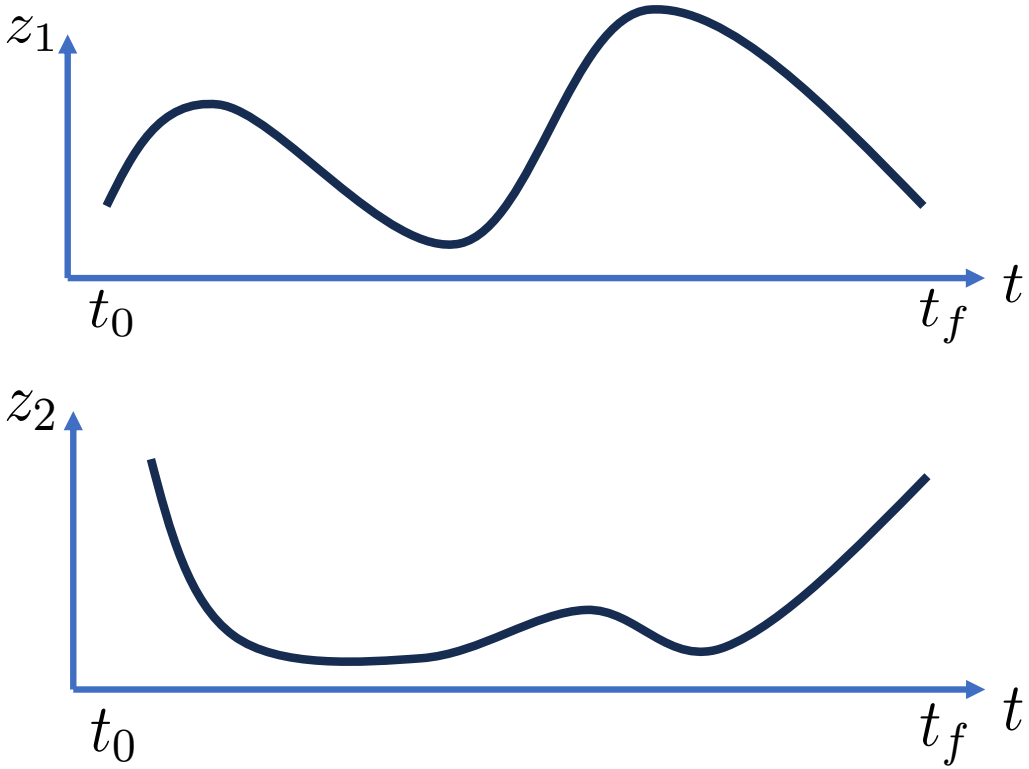


Trajectory following:

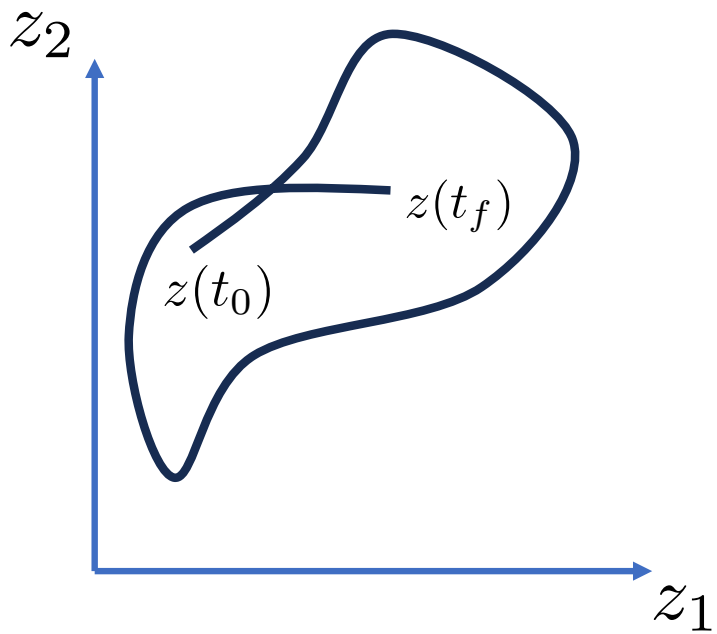
- Kinematic/dynamic models
- PID control
- LQR, gain scheduled LQR



Trajectory Splines: one for each pose dimension

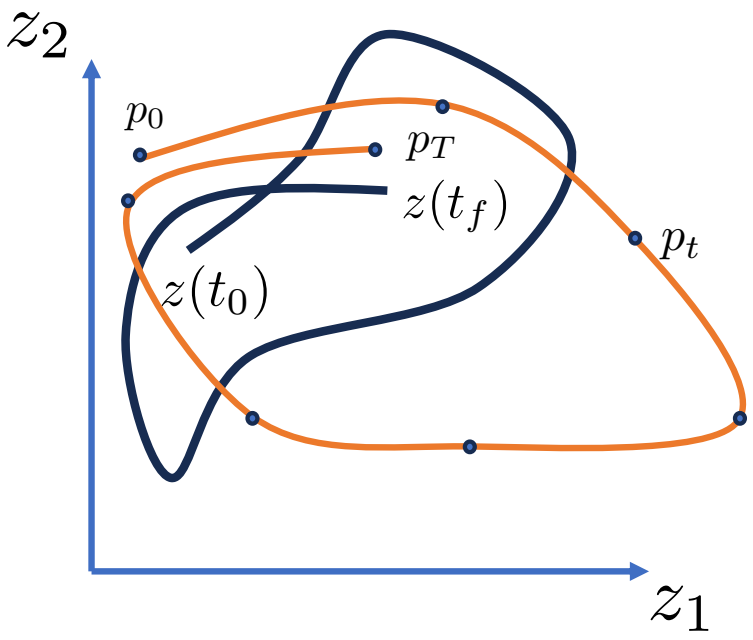


$$z(t) = \begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix}$$



Time-parameterized trajectory

Trajectory Splines Fitting



- Choose spline parameters to approximately pass through a set of waypoints
- E.g., waypoints from a path planner

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

For a given ways point p_t we get one eqn per dimension i:

$$p_t^i = \Psi(t)^T \alpha^i$$

See also:

- Bezier curves
- Chebyshev polynomials

both examples of polynomial splines.

Solve for the spline coefficients

Given desired waypoints: $p_0, p_{t_1}, p_{t_2}, \ldots, p_T$

Then solve linear system of equations with matrix inverse (least squares fit if over determined):

$$\begin{bmatrix} 0^N & 0^{N-1} & \cdots & 0^1 & 1 \\ t_1^N & t_1^{N-1} & \cdots & t_1^1 & 1 \\ t_2^N & t_2^{N-1} & \cdots & t_2^1 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ T^N & T^{N-1} & \cdots & T^1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_N^i \\ \alpha_{N-1}^i \\ \vdots \\ \alpha_1^i \\ \alpha_0^i \end{bmatrix} = \begin{bmatrix} p_0^i \\ p_1^i \\ p_2^i \\ \vdots \\ p_N^i \end{bmatrix}$$

Each row is a constraint:

$$\Psi(t)^T \alpha^i = p_t^i$$

Write as: $\Phi \alpha^i = P^i$

N + 1 waypoints: $\alpha^i = \Phi^{-1} P^i$

> N + 1 waypoints: $\alpha^i = (\Phi^T \Phi)^{-1} \Phi^T P^i$

< N + 1 waypoints: $\alpha^i = \Phi^T (\Phi \Phi^T)^{-1} P^i$

Min-Snap Planning

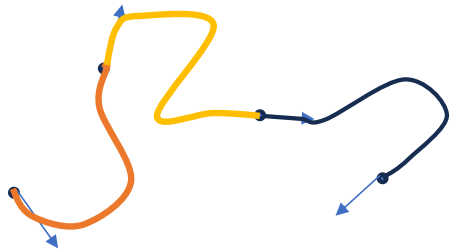
If you have more degrees of freedom in spline, matrix short-fat -> many solutions to system of equations.

Solve smooth optimization problem.

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

"snap" – 4th time derivative

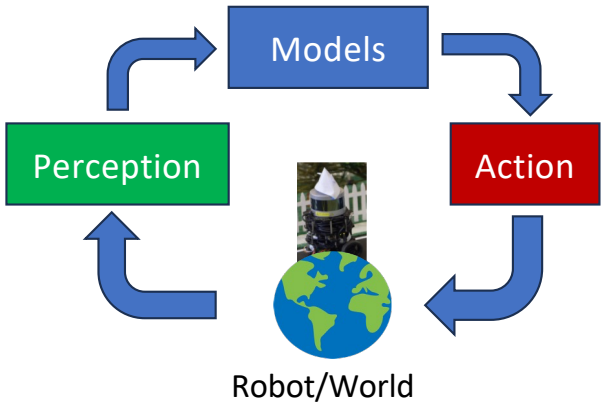
$$\text{subject to} \quad \begin{bmatrix} \psi_1(t_0) & \psi_2(t_0) & \dots & \psi_N(t_0) \\ \dot{\psi}_1(t_0) & \dot{\psi}_2(t_0) & \dots & \dot{\psi}_N(t_0) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(t_0) & \psi_2^{(q)}(t_0) & \dots & \psi_N^{(q)}(t_0) \\ \psi_1(t_f) & \psi_2(t_f) & \dots & \psi_N(t_f)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



Very common in drone trajectory planning

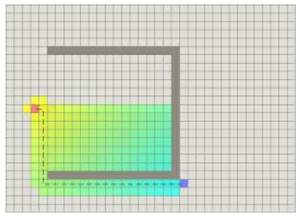
Action: Planning and Control Stack

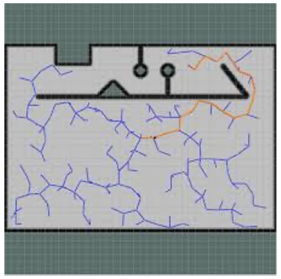
Use the models to move the robot



Path planning:

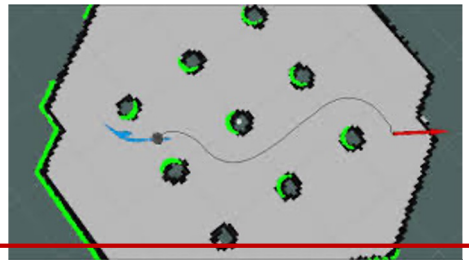
- A* planning on a graph
- RRT, RRT*





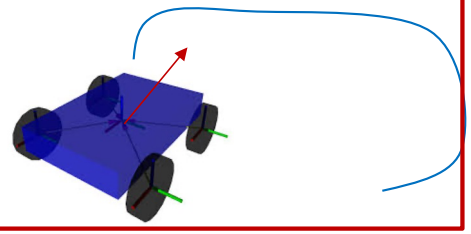
Trajectory optimization:

- Optimal control
- Numerical solvers

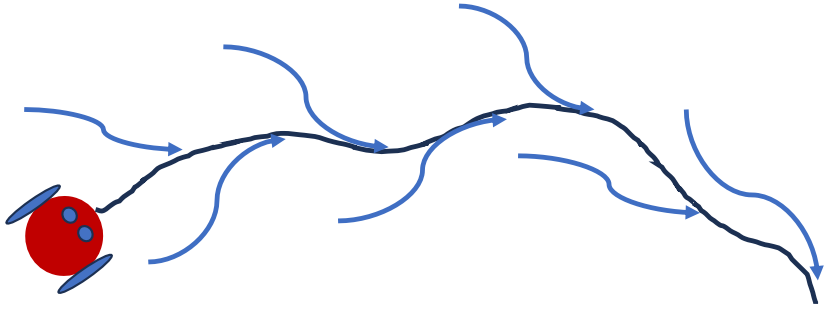


Trajectory following:

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



Trajectory Following



Motion model:

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

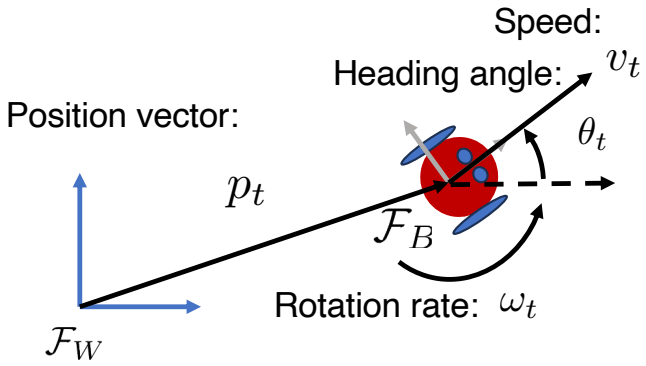
Feedback control law:

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

planned traj waypoints

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

Robot Kinematics and Dynamics



Robot pose: (p_t, θ_t)

Control inputs: (v_t, ω_t)

Differential Drive Robot (kinematic):

$$
\begin{aligned}
x^p_{t+1} &= x^p_t + v_t \cos(\theta_t)\delta t \\
y^p_{t+1} &= y^p_t + v_t \sin(\theta_t)\delta t \\
\theta_{t+1} &= \theta_t + \omega_t \delta t
\end{aligned}
$$

Nonholonomic: constraint on velocity vector
Nonlinear

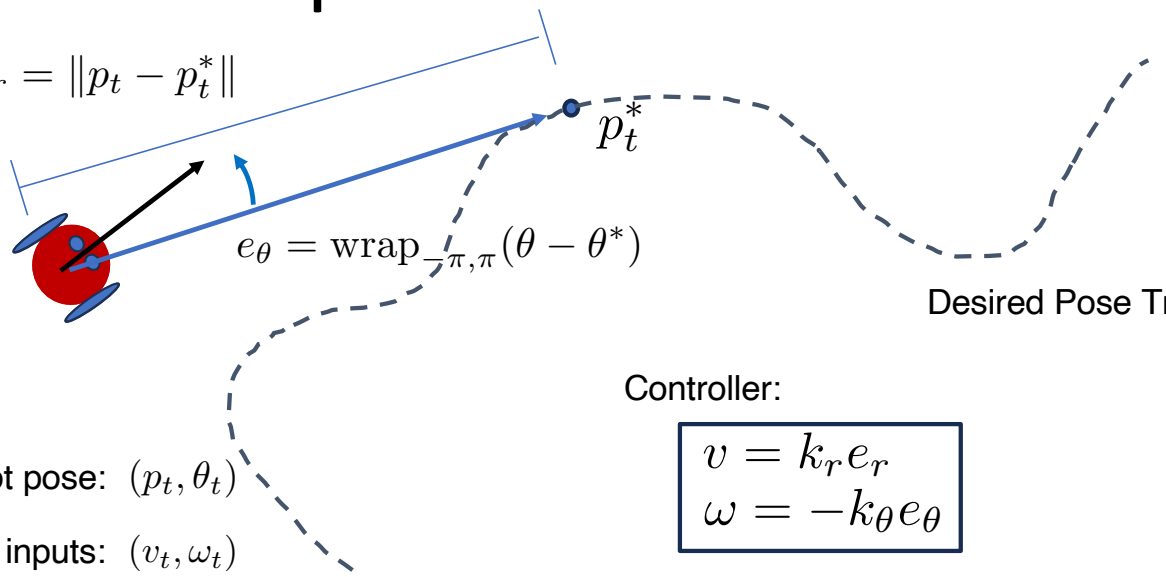
Robot dynamic state: $x_t = (p_t, \theta_t, v_t, \omega_t)$

Control inputs: $u_t = (a_t, \alpha_t)$

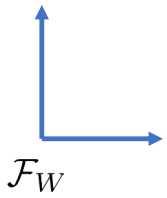
Differential Drive Robot (dynamic):

$$
\begin{aligned}
x^p_{t+1} &= x^p_t + v_t \cos(\theta_t)\delta t \\
y^p_{t+1} &= y^p_t + v_t \sin(\theta_t)\delta t \\
\theta_{t+1} &= \theta_t + \omega_t \delta t \\
v_{t+1} &= v_t + a_t \delta t \\
\omega_{t+1} &= \omega_t + \alpha_t \delta t
\end{aligned}
$$

Heuristic Proportional Controller



Position vector:



Robot pose: (p_t, θ_t)

Control inputs: (v_t, ω_t)

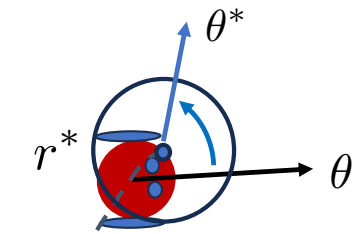
Controller:

$$v = k_r e_r$$
$$\omega = -k_\theta e_\theta$$

Differential Drive Robot (kinematic):

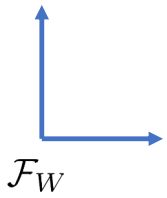
$$x_{t+1}^p = x_t^p + v_t \cos(\theta_t)\delta t$$
$$y_{t+1}^p = y_t^p + v_t \sin(\theta_t)\delta t$$
$$\theta_{t+1} = \theta_t + \omega_t \delta t$$

What about heading?



$$e_\theta = \text{wrap}_{[-\pi,\pi]}(|\theta - \theta^*|)$$

Position vector:



Robot pose: (p_t, θ_t)

Control inputs: (v_t, ω_t)

Differential Drive Robot (kinematic):

$$\begin{aligned} x^p_{t+1} &= x^p_t + v_t \cos(\theta_t)\delta t \\ y^p_{t+1} &= y^p_t + v_t \sin(\theta_t)\delta t \\ \theta_{t+1} &= \theta_t + \omega_t \delta t \end{aligned}$$

Controller:

$$\begin{aligned} v &= k_r e_r \\ \omega &= -k_\theta e_\theta \end{aligned}$$

Switching rule:

$$\begin{aligned} \text{if} \quad & \|p_t - p^*_t\| \leq r^* \\ & \theta^* = \text{desired heading} \\ & k_r = 0 \end{aligned}$$

Isn't there an easier way?

Unfortunately, no.

Brockett's Theorem (1983): There exists no **continuously differentiable** feedback control law to stabilize a nonholonomic system to a desired pose.

We can only achieve desired position and heading with a switching law.

Many other switching controllers for diff drive robot exist in literature.

Feedback Linearizing Controller

Differential Drive Robot (continuous time):

$$\begin{aligned} \dot{x}^p &= v_t \cos\theta \\ \dot{y}^p &= v_t \sin\theta \\ \dot{\theta} &= \omega \end{aligned}$$

Differentiate velocity wrt time:

$$\begin{aligned} \ddot{x}^p &= \dot{v}\cos(\theta) - v\sin(\theta)\dot{\theta} \\ \ddot{y}^p &= \dot{v}\sin(\theta) + v\cos(\theta)\dot{\theta} \\ \dot{v} &= a \\ \dot{\theta} &= \omega \end{aligned}$$

$$\begin{bmatrix} \ddot{x}^p \\ \ddot{y}^p \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\theta & -v\sin\theta \\ \sin\theta & v\cos\theta \end{bmatrix}}_{J(v,\theta)} \begin{bmatrix} a \\ \omega \end{bmatrix}$$

We design controller as:

$$\begin{bmatrix} a \\ \omega \end{bmatrix} = J^{-1} f_{des}(x^p, y^p, \dot{x}^p, \dot{y}^p)$$

Closed loop dynamics:

$$\begin{bmatrix} \ddot{x}^p \\ \ddot{y}^p \end{bmatrix} = f_{des}(x^p, y^p, \dot{x}^p, \dot{y}^p)$$

We choose f_{des} to give dynamics we like, e.g., linear.

Feedback Linearizing Controller

Dynamics:

$$\ddot{p} = Ju$$

Controller:

$$u = J^{-1}\underbrace{(\ddot{p}^* + k_d(\dot{p}^* - \dot{p}) + k_p(p^* - p))}_{f_{des}}$$

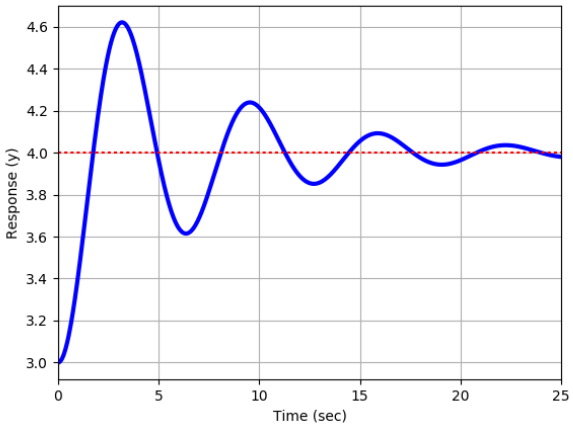
Closed loop dynamics:

$$\ddot{p} = \ddot{p}^* + k_d\underbrace{(\dot{p}^* - \dot{p})}_{-\delta\dot{p}} + k_p\underbrace{(p^* - p)}_{-\delta p}$$

Rewrite as:

$$\delta\ddot{p} + k_d(\delta\dot{p}) + k_p(\delta p) = 0$$

Exponential convergence: $p(t) \to p^*(t)$



Practical Details

Controller:

$$u = J^{-1}(\ddot{p}^* + k_d(\dot{p}^* - \dot{p}) + k_p(p^* - p))$$

where: $J = \begin{bmatrix} \cos\theta & -v\sin\theta \\ \sin\theta & v\cos\theta \end{bmatrix}$

Singularity at $v = 0$

- We couple this with the switching rule we saw before (Brockett's theorem is unavoidable)
- We building in safety checks to avoid $v \leq v_{\text{lower bound}}$
- Need to discretize in time to implement

LQR Based Controllers

The LQR problem:

$$\min_u \int_0^\infty x^T Q x + u^T R u \, dt$$

$$\text{subject to} \quad \dot{x} = Ax + Bu$$

$$x(0) = x_0$$

Quadratic state cost

Quadratic control cost

Linear dynamics

Initial conditions

LQR Solution

Open-loop system dynamics:

$$\dot{x} = Ax + Bu$$

Linear feedback controller:

$$u = -Kx$$

Solve Riccati equation for P:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0$$

Obtain K from:

$$K = R^{-1}B^T P$$

Closed loop system:

$$\dot{x} = Ax + B(-Kx) = (A - BK)x$$

Closed loop dynamics matrix determines convergence rate, response modes:

$$A_{cl} = (A - BK)$$

Advantages:

- Automatic control design
- Performs very well in practice
- Computationally very light
- Work for arbitrary dimensional systems

Challenges:

- Requires linear dynamics!
- Only regulates state to 0!
- How to adapt to nonlinear dynamics and tracking objectives?

Taylor Series Linearization

Open-loop nonlinear system dynamics:

$$\dot{x} = f(x, u)$$

Linearize about desired **feasible trajectory** from traj opt**:**

$$(x^*(t), u^*(t)) \quad \text{such that} \quad \dot{x}^*(t) = f(x^*(t), u^*(t))$$

1st order Taylor series expantion:

$$\dot{x} \approx f(x^*, u^*) + \underbrace{\nabla_x f(x^*, u^*)}_{A(x^*(t), u^*(t))}\underbrace{(x - x^*)}_{\delta x} + \underbrace{\nabla_u f(x^*, u^*)}_{B(x^*(t), u^*(t))}\underbrace{(u - u^*)}_{\delta u}$$

$$\underbrace{(\dot{x} - \dot{x}^*)}_{\delta \dot{x}}$$

We get: $\boxed{\delta \dot{x} = A(t)\delta x + B(t)\delta u}$

Recall Jacobians:

$$\nabla_x f(x^*, u^*) = \begin{bmatrix} \frac{\partial f_1(x,u)}{\partial x_1}\Big|_{x^*,u^*} & \cdots & \frac{\partial f_1(x,u)}{\partial x_n}\Big|_{x^*,u^*} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(x,u)}{\partial x_1}\Big|_{x^*,u^*} & \cdots & \frac{\partial f_n(x,u)}{\partial x_n}\Big|_{x^*,u^*} \end{bmatrix}$$

$$\nabla_u f(x^*, u^*) = \begin{bmatrix} \frac{\partial f_1(x,u)}{\partial u_1}\Big|_{x^*,u^*} & \cdots & \frac{\partial f_1(x,u)}{\partial u_m}\Big|_{x^*,u^*} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(x,u)}{\partial u_1}\Big|_{x^*,u^*} & \cdots & \frac{\partial f_n(x,u)}{\partial u_m}\Big|_{x^*,u^*} \end{bmatrix}$$

Gain Scheduled LQR

Choose N way points in desired trajectory:

$$(x^*, u^*) \rightarrow \{(x_i^*, u_i^*)\}_{i=1}^N$$

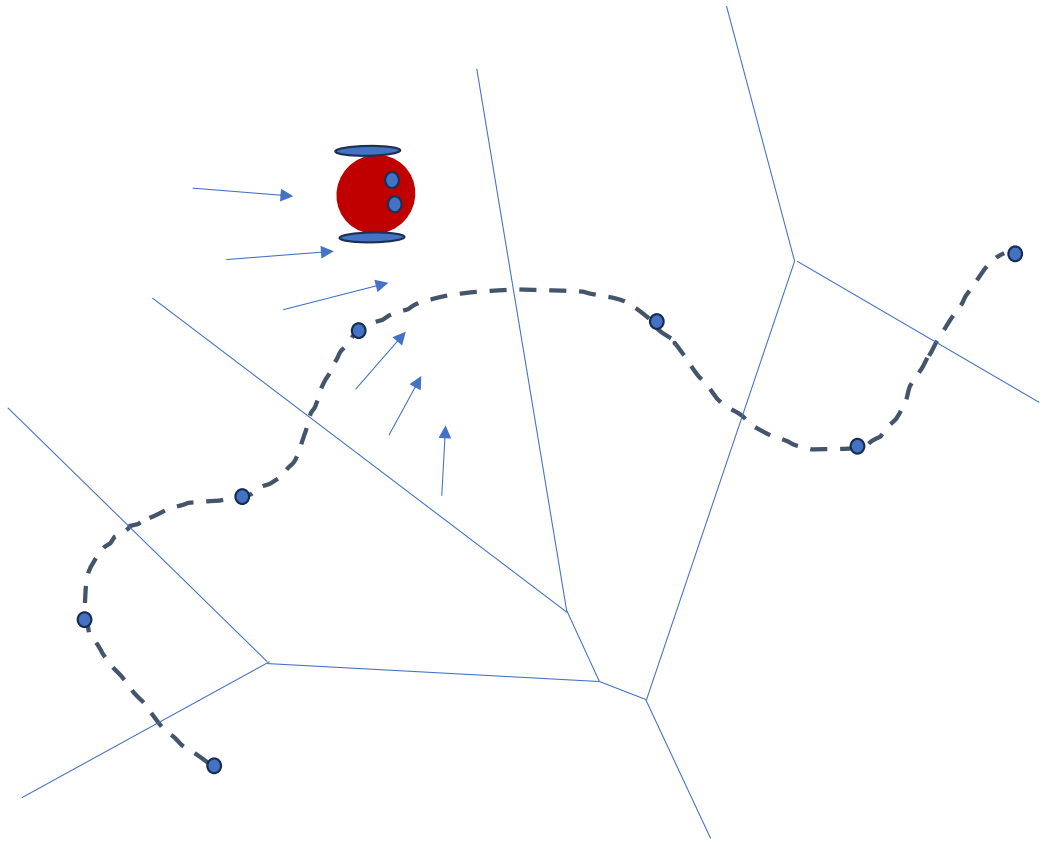
A collection of N LQR controllers:

$$\delta u = -K_i \delta x \quad \rightarrow \quad \boxed{u = u^* - K_i(x - x^*)}$$

$$K_i = R^{-1} B_i^T P_i$$

$$A_i^T P_i + P_i A_i - P_i B_i R^{-1} B_i^T P_i + Q = 0$$

Apply gain from closest point



Differential flatness

Differential flatness: A nonlinear system $\dot{\mathbf{x}} = \boldsymbol{f}(\mathbf{x}, \mathbf{u})$ is differentially flat if there exists a function α such that

$$\mathbf{z} = \alpha(\mathbf{x}, \mathbf{u}, \ldots, \mathbf{u}^{(p)})$$

and such that the solutions to the nonlinear system $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ can be written as functions of $\boldsymbol{z}$ and a finite number of derivatives

$$\mathbf{x} = \beta(\mathbf{z}, \dot{\mathbf{z}}, \ldots, \mathbf{z}^{(q)})$$

$$\mathbf{u} = \gamma(\mathbf{z}, \dot{\mathbf{z}}, \ldots, \mathbf{z}^{(q)})$$

In words, a system is differentially flat if we can find a set of outputs (equal in number to the number of inputs) such that all states and inputs can be determined from these outputs *without integration*

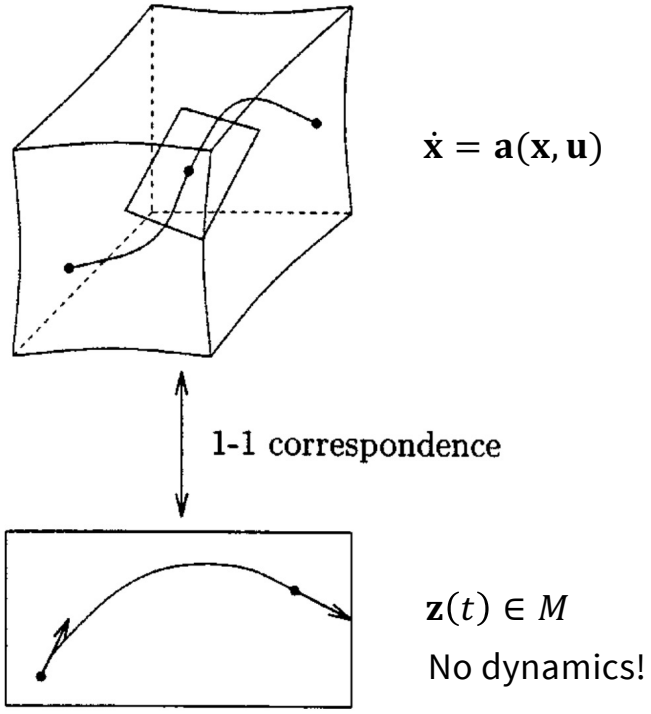
Differential flatness

- Implication for trajectory generation: to every sufficiently differentiable curve $t \to \mathbf{z}(t)$, there is a corresponding trajectory

$$t \to \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{u}(t) \end{pmatrix} = \begin{pmatrix} \beta(\mathbf{z}(t), \dot{\mathbf{z}}(t), \dots, \mathbf{z}^{(q)}(t)) \\ \gamma(\mathbf{z}(t), \dot{\mathbf{z}}(t), \dots, \mathbf{z}^{(q)}(t)) \end{pmatrix}$$

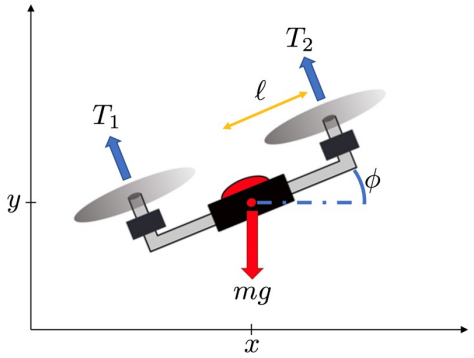
that identically satisfies the system equations

- The simple car is differentially flat with the position of the rear wheels as the flat output



From Nieuwstadt, Murray. 1998.

Example: planar quadrotor



$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\phi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ \frac{-(T_1+T_2)\sin\phi}{m} \\ v_y \\ \frac{(T_1+T_2)\cos\phi}{m} - g \\ \omega \\ \frac{(T_2-T_1)\ell}{I_{zz}} \end{bmatrix}$$

$$\mathbf{z} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\left.\begin{aligned} x &= x \\ v_x &= \dot{x} \\ y &= y \\ v_y &= \dot{y} \end{aligned} \qquad \begin{aligned} \phi &= \tan^{-1}\left(-\frac{\ddot{x}}{\ddot{y}+g}\right) \\ \omega &= \frac{\dddot{y}\ddot{x} - (\ddot{y}+g)\dddot{x}}{(\ddot{y}+g)^2 + \ddot{x}^2} \end{aligned}\right\} \beta$$

$$\mathbf{x} = \beta(\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \dddot{\mathbf{z}})$$

$$\mathbf{u} = \gamma(\mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \dddot{\mathbf{z}}, \ddddot{\mathbf{z}})$$

Practical implications

This leads to a simple, yet effective strategy for trajectory generation

1. Find the initial and final conditions for the flat output:

Given	**Find**
$(t_0, \mathbf{x}(t_0), \mathbf{u}(t_0))$	$(\mathbf{z}(t_0), \dot{\mathbf{z}}(t_0), \dots, \mathbf{z}^{(q)}(t_0))$
$(t_f, \mathbf{x}(t_f), \mathbf{u}(t_f))$	$(\mathbf{z}(t_f), \dot{\mathbf{z}}(t_f), \dots, \mathbf{z}^{(q)}(t_f))$

2. Build a smooth curve $t \to \mathbf{z}(t)$ for $t \in [t_0, t_f]$ by interpolation, possibly satisfying further constraints (e.g. following A* path)
3. Compute the corresponding trajectory $t \to (\mathbf{x}(t), \mathbf{u}(t))$

How do we get the smooth trajectory? Splines!

- As we saw last time
- We can parameterize the flat output trajectory using a set of smooth basis functions $\psi_i(t)$

$$z_j(t) = \sum_{i=1}^{N} \alpha_i^{[j]} \psi_i(t)$$

- and then solve

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

- Or solve min-snap optimization

When is a system differentially flat?

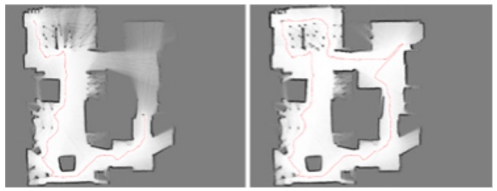
- The existence of a general, computable criterion to decide if the dynamical system $\dot{\mathbf{x}} = \boldsymbol{f}(\mathbf{x}, \mathbf{u})$ is differentially flat remains open
- Some results in this direction are, however, available
- Further readings:
  - Application to trajectory optimization:
    1. M. J. Van Nieuwstadt and R. M. Murray. Real-time trajectory generation for differentially flat systems. 1998
    2. R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. 1995
    3. B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Robotics: modelling, planning and control. 2010
    4. D. Mellinger. Trajectory Generation and Control for Quadrotors. 2012.
  - Theory:
    1. J. Levine. Analysis and control of nonlinear systems: A flatness-based approach. 2009
    2. G. G. Rigatos, Gerasimos. Nonlinear control and filtering using differential flatness approaches: applications to electromechanical systems. 2015

Perception Stack: Computer vision, filtering, SLAM

Use sensor data to update the models

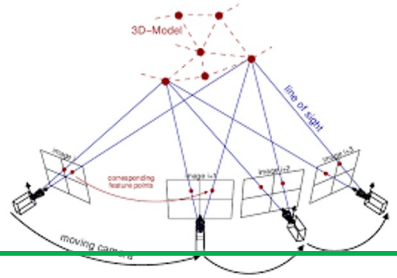
Localization, Mapping, Tracking:

- EKF/Monte Carlo localization
- Occupancy grid mapping
- Pose graph optimization
- Tracking (EKF and Particle Filter)



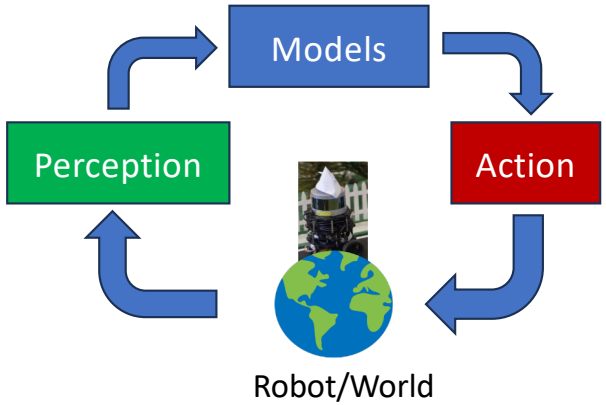
Information extraction

- Computer vision: features, correspondences, Structure from Motion (SfM), depth
- Lidar scan matching, ICP



Sensors:

- RGB Cameras, RGB-D/stereo cameras, Lidar
- IMU, GPS, wheel encoders



Next time: robotic sensors and introduction to computer vision