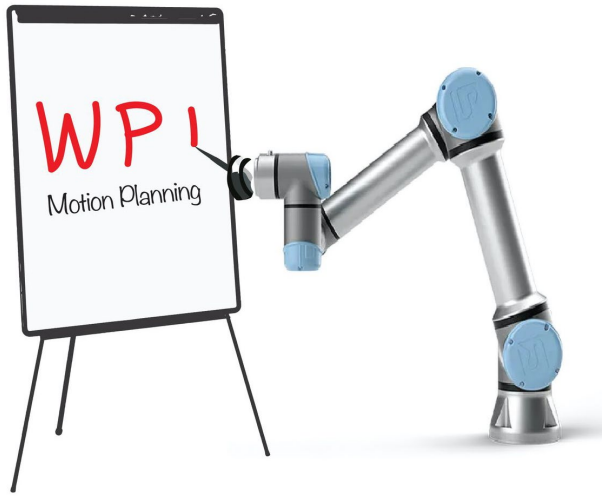
RBE550
Motion Planning
Kinodynamic Formulation

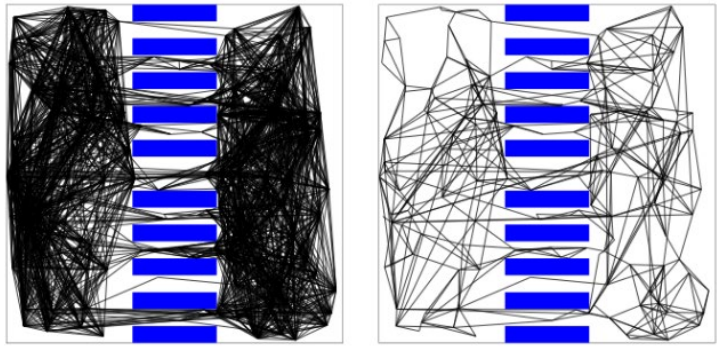


Constantinos Chamzas
www.cchamzas.com
www.elpislab.org

Kinodynamic Motion Planning

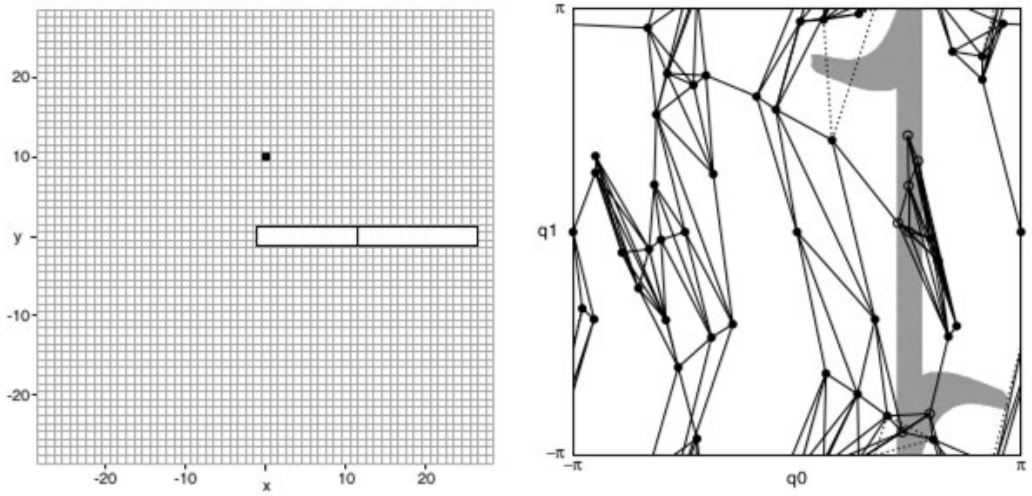
- *The slides are a compilation of work based on notes and slides mainly from from Erion Plaku, but also Constantinos Chamzas, Lydia Kavraki, Howie Choset, Morteza Lahijanian, David Hsu, Greg Hager, Mark Moll, G. Ayorkor Mills-Tetty, Hyungpil Moon, Zack Dodds, Zak Kingston, Nancy Amato, Steven Lavalle, Seth Hutchinson, George Kantor, Dieter Fox, Vincent Lee-Shue Jr., Prasad Narendra Atkar, Kevin Tantiseviand, Bernice Ma, David Conner, and students taking comp450/comp550 at Rice University.*

Last Time



(a) Roadmap with 2346 edges (b) Spanner with 470 edges

- SPARS/SPARS2



- Dynamic Roadmaps

Overview

- Midterm Overview
- Kinematic constraints
- Dynamics constraints
- Integration of the dynamics
- Planning with PRM for Kino-dynamic systems
- Planning with RRT for Kino-dynamic systems

Midterm is next week Oct 10.

5. (4 points) **Example Question** Describe a fundamental difference between FMT* and RRT*.

7. (4 points) **Example Question:** A manipulator arm that has 5 degrees-of-freedom (DOF) picks an object from a table. The object is a 3D rigid Body and is rigidly grasped by the robot e.g., it cannot move freely after being grasped. How many degrees of freedom does the combined (arm+object) have?

12. Answer YES or NO for the following questions, and explain your answer.

    (a) (1 point) is this a valid 2D rotation matrix? $\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$

6. (4 points) **Example Question** Give an explanation of the separating axis theorem, Either in words or equations

7. (4 points) **Example Question** Sketch a separating axis and corresponding separating line to detect if the following objects are not in collision

Motivation

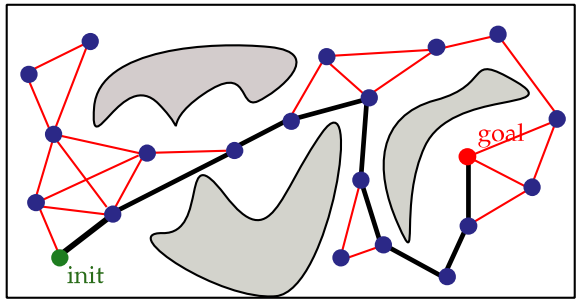
- **Sampling-based methods:**

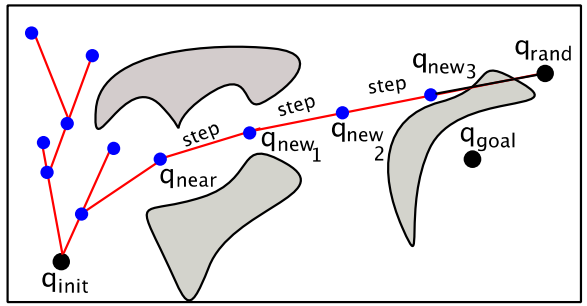
  Advantages

  - Computationally efficient
  - Solves high-dimensional problems
  - Easy to implement

  Disadvantages

  - Not complete:
    - cannot report when no solution exists
  - Probabilistically complete:
    - If a path exists, the probability of not finding it → 0 as number of samples → ∞

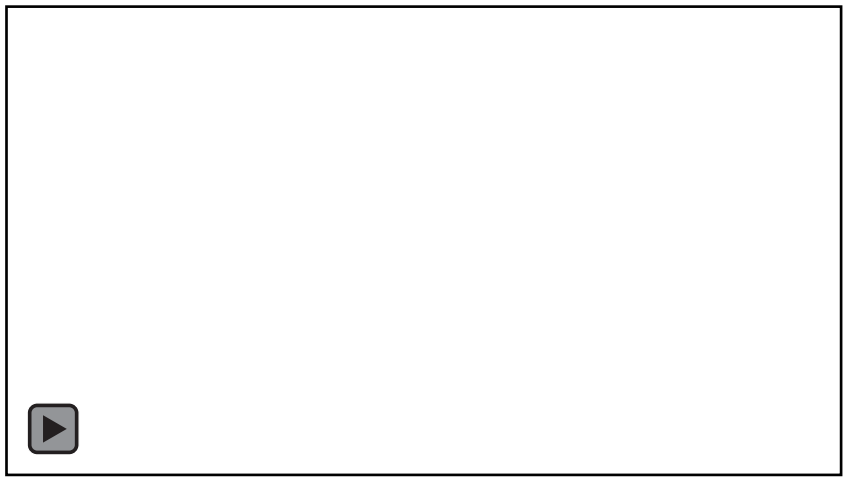




Geometric Planning

- Geometric (path) planning refers to planning problems that consider only geometric constraints (e.g., collision avoidance).
  - Geometric planning does not take into account the constraints on the motion of the robot.

- Sampling-based motion planners are powerful tools for geometric planning



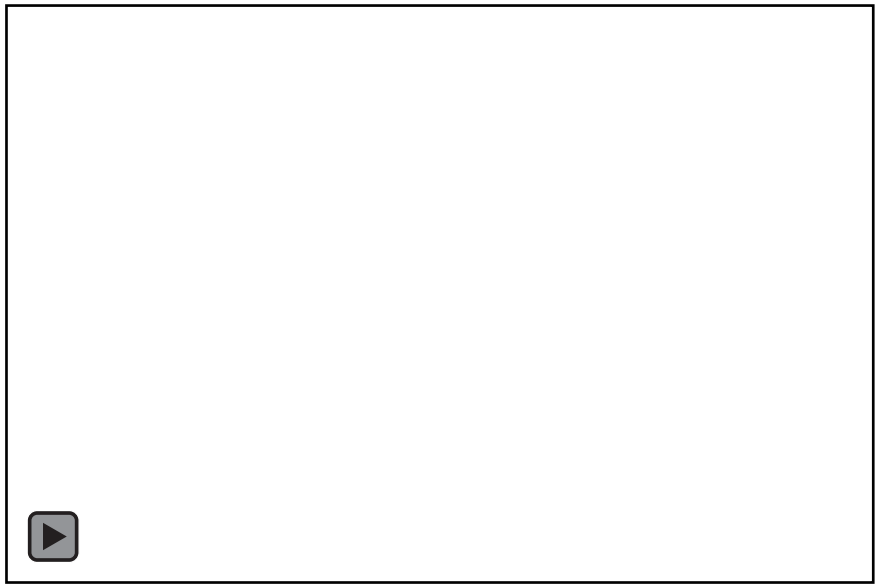
PR2 performing manipulation

Motion Planning with Kinematics and Dynamics

- Geometric constraints are generally not sufficient to adequately express robot motion

- Constraints on velocity, forces, torques, accelerations are needed for better representations

Geometric planning

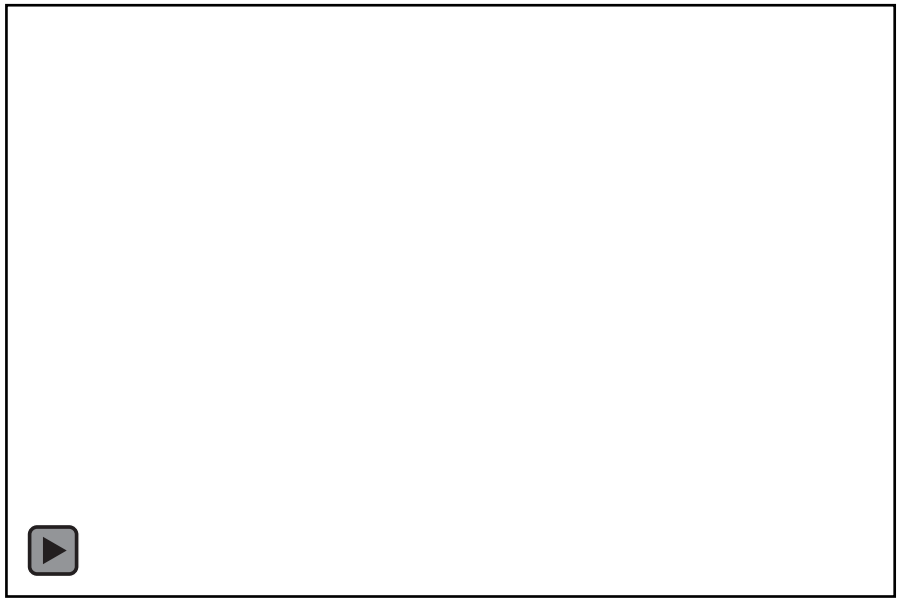


Motion Planning with Kinematics and Dynamics

- Geometric constraints are generally not sufficient to adequately express robot motion

- Constraints on velocity, forces, torques, accelerations are needed for better representations

Kinematic planning

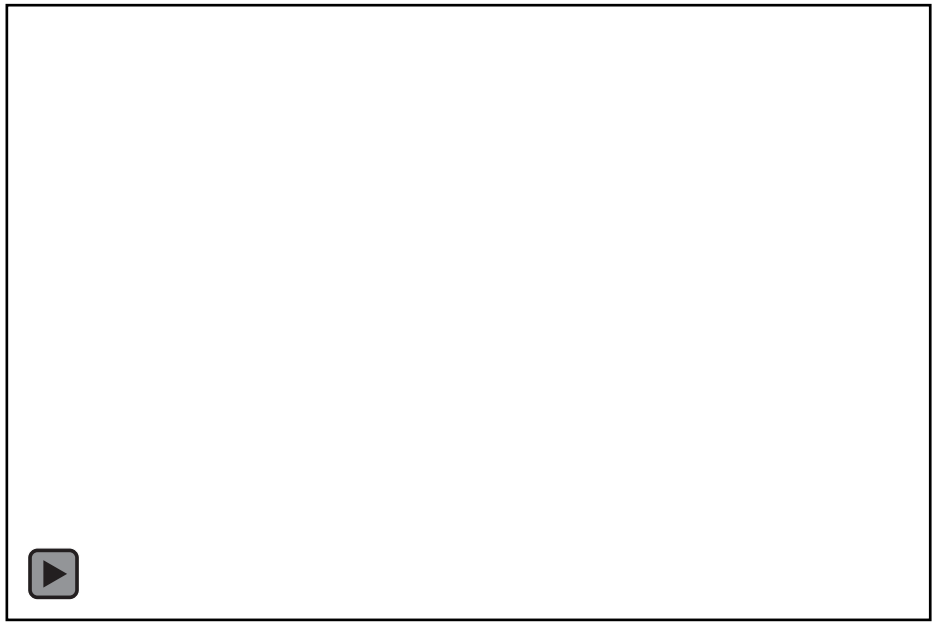


Motion Planning with Kinematics and Dynamics

- Geometric constraints are generally not sufficient to adequately express robot motion

- Constraints on velocity, forces, torques, accelerations are needed for better representations

Dynamic planning



Robot Motion

- Robot motion is typically described by a differential equation

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

  Where:

  - $\boldsymbol{x} \in X$ is the state and X is the state space
  - $f(\cdot,\cdot)$ is a continuous integrable function
  - $\boldsymbol{u} \in U$ is the input control, and U is the control space

- In other words, robot motion is subject to constraints defined by the differential equation.

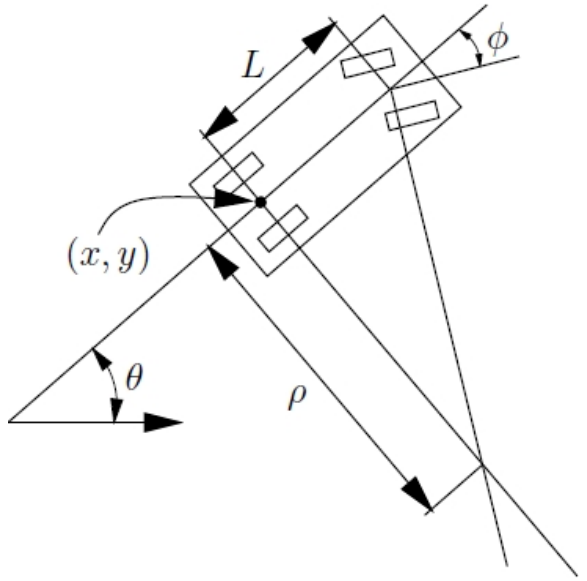
- If $X = C$ (configuration space, $\boldsymbol{x} = \boldsymbol{q}$) and the control input $\boldsymbol{u}$ includes velocity, then the system is called first-order and subject to kinematic constraints

- If X includes C in addition to derivatives of some of the configuration parameters and $\boldsymbol{u}$ includes acceleration, then the system is called second-order and subject to dynamics constraints

Kinematics for Wheeled System – Simple Car

Simple car

- Configuration: $q = (x, y, \theta) \in \mathbb{R}^2 \times S^1$
- Body frame:
  - Origin is at the center of rear axle
  - x-axis point along main axis of the car
- Velocity: v
- Steering angle: ϕ



Kinematics for Wheeled System – Simple Car

Simple car

- How should we control the car?
  - Setting the speed v, i.e.,

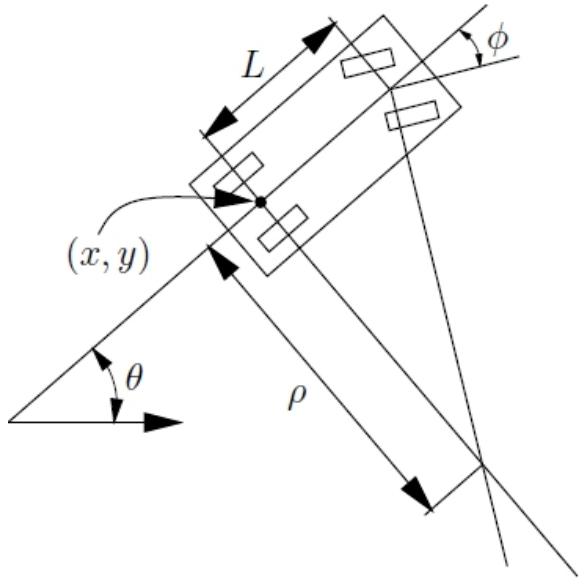
$$u_v = v$$

  - Setting the steering angle ϕ, i.e.,

$$u_\phi = \phi$$

  - Putting it all together:
    - Input controls: u_v (speed) and u_ϕ (steering angle)
    - Equation of motion:

$$\dot{x} = u_v \cos\theta$$
$$\dot{y} = u_v \sin\theta$$
$$\dot{\theta} = \frac{u_v}{L} \tan u_\phi$$



What is the state-space X of the first order simple-car model?

2

x, y, θ

$x, y, \theta, u_\varphi, u_v$

$\dot{x}, \dot{y}, \dot{\theta}$

$\dot{x}, \dot{y}, \dot{\theta}, x, y, \theta$

What is the state-space X of the first order simple-car model?

2

What is the state-space X of the first order simple-car model?

2

Variations of the Simple Car Model

- Equation of motion: $\dot{x} = u_v \cos\theta \qquad \dot{y} = u_v \sin\theta \qquad \dot{\theta} = \frac{u_v}{L}\tan u_\phi$

- Questions
  - What are the bounds on the steering angle?
  - What are the bounds on the speed?

- **Tricycle**
  - $u_v \in [-1, 1]$ and $u_\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$
  - Can it rotate in place?

- **Standard simple car**
  - $u_v \in [-1,1]$
  - $u_\phi \in (-\phi_{\max}, \phi_{\max})$ for some $\phi_{\max} < \frac{\pi}{2}$

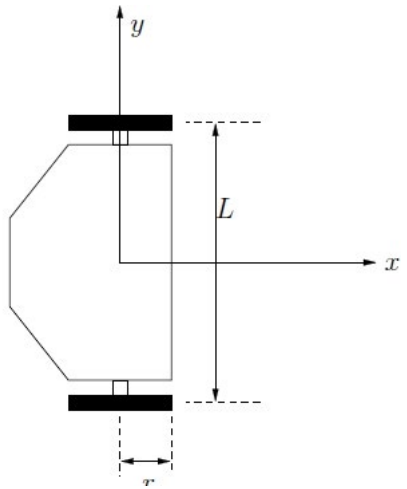
- **Reeds-Shepp car**
  - $u_v \in \{-1, 0, 1\}$ (i.e., “reverse”, “park”, “forward”)
  - u_ϕ same as in the standard simple car

- **Dubins car**
  - $u_v \in \{0, 1\}$ (i.e., “park”, “forward”)
  - u_ϕ same as in the standard simple car

Kinematics for Wheeled System – Differential Drive

Differential drive

- Input controls: $u = (u_l, u_r)$
  - u_l: left wheel angular velocity
  - u_r: right wheel angular velocity

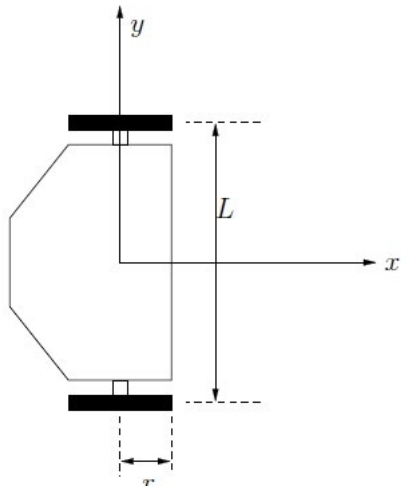


- How does it move?
  - $u_l = u_r$
    - moves forward in the direction the wheels are pointing
    - Speed proportional to wheel radius r
  - $u_l = -u_r$
    - Rotates clockwise because wheels are turning in opposite directions

Kinematics for Wheeled System – Differential Drive

Differential drive

- Where is the body frame placed?
  - Origin at the center of the axle between the wheels

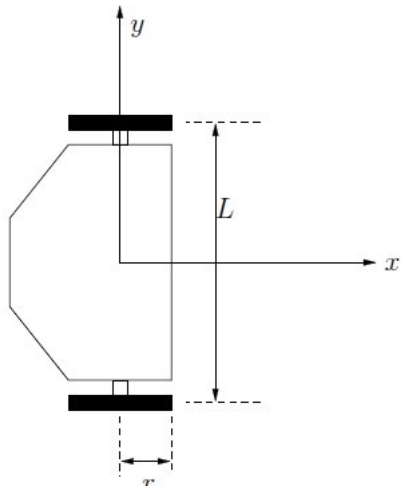


- Equations of motion
  - $\dot{x} = \frac{r}{2}(u_l + u_r)\cos\theta$
  - $\dot{y} = \frac{r}{2}(u_l + u_r)\sin\theta$
  - $\dot{\theta} = \frac{r}{L}(u_r - u_l)$

Kinematics for Wheeled System – Differential Drive

Differential drive

- Where is the body frame placed?
  - Origin at the center of the axle between the wheels



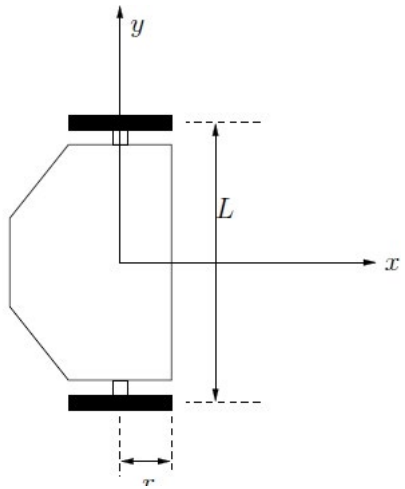
- Different way of representing equation of motion
  - Translation: $u_\omega = \frac{1}{2}(u_l + u_r)$
  - Rotation: $u_\psi = (u_l - u_r)$

- Equations of motion
  - $\dot{x} = r\, u_\omega \cos\theta$
  - $\dot{y} = r\, u_\omega \sin\theta$
  - $\dot{\theta} = \frac{r}{L}\, u_\psi$

Kinematics for Wheeled System – Differential Drive

Differential drive

- Where is the body frame placed?
  - Origin at the center of the axle between the wheels
  - Equations of motion 1
    - $\dot{x} = \frac{r}{2}(u_l + u_r)\cos\theta$
    - $\dot{y} = \frac{r}{2}(u_l + u_r)\sin\theta$
    - $\dot{\theta} = \frac{r}{L}(u_r - u_l)$

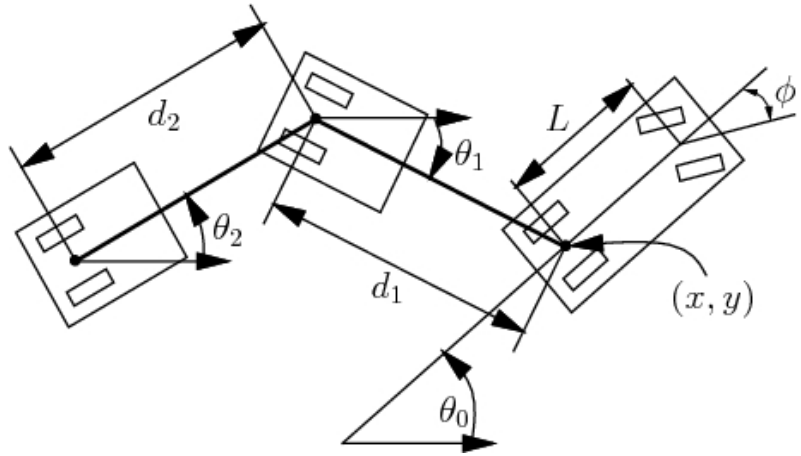


- Equations of motion 2
  - $\dot{x} = r\,u_\omega \cos\theta$
  - $\dot{y} = r\,u_\omega \sin\theta$
  - $\dot{\theta} = \frac{r}{L}\,u_\psi$

Can the differential drive move between any two configurations? YES

Kinematics for Wheeled System – Tractor Trailer

Tractor Trailer

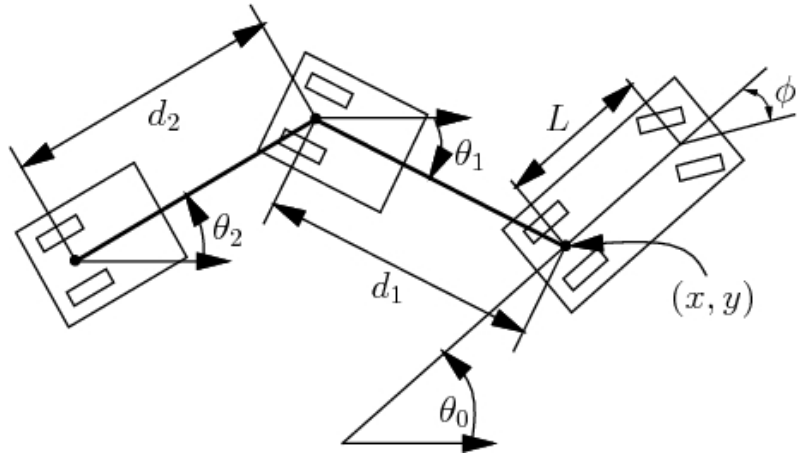


- Simple car pulling k trailers
- Each trailer is attached to rear axle of body in front of it.
- d_i : hitch length
  - the distance from the center of the rear axle of trailer i to the point at which the trailer is hitched to next body.
- C-space:
  - Car itself: $\mathbb{R}^2 \times S^1$
  - Each trailer: S^1
  - Overall: $\mathbb{R}^2 \underbrace{\times S^1 \times \ldots \times S^1}_{k+1}$

Kinematics for Wheeled System – Tractor Trailer

Tractor Trailer

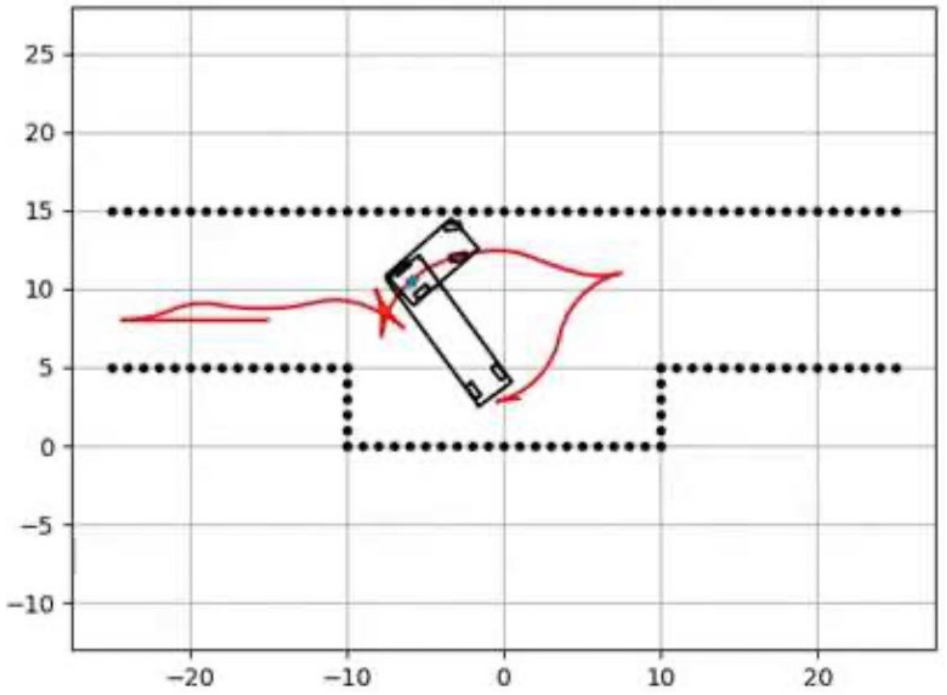
"The configuration transition equation is somewhat of an art to get right. The one here is adapted from [Murray, Sastry, IEE Trans. Autom. Control, 1993]."



- Equations of motion:
  - $\dot{x} = u_v \cos\theta_0$
  - $\dot{y} = u_v \sin\theta_0$
  - $\dot{\theta}_0 = \frac{u_v}{L} \tan u_\phi$
  - $\dot{\theta}_1 = \frac{u_v}{d_1} \sin(\theta_0 - \theta_1)$
  - …
  - $\dot{\theta}_i = \frac{u_v}{d_i} \left(\Pi_{j=1}^{i-1} \cos(\theta_{j-1} - \theta_j)\right) \sin(\theta_{i-1} - \theta_i)$

Kinematics for Wheeled System – Tractor Trailer

Tractor Trailer



- How about acceleration?

Dynamical Systems

- Involve acceleration $\ddot{q}$ in addition to velocity $\dot{q}$ and configuration q

- Implicit constraints $g(\ddot{q}, \dot{q}, q) = 0$
  - Easier to derive, but difficult to use

- Parametric constraints $\ddot{q} = f(\dot{q}, q, u)$
  - Easier to work with, control is similar to actions

Phase Space: Reducing Degree by Increasing Dimension

- Can turn high-order dynamics constraints to velocity constraints by increasing the dimension of the state space

- Example

$$\ddot{y} - 3\dot{y} + y = 0$$

- Let $\boldsymbol{x} = (x_1, x_2)$ denote a state vector, where
  - $x_1 = y$
  - $x_2 = \dot{y}$
- Then,

$$\dot{x}_2 - 3x_2 + x_1 = 0$$

- Thus, the equation of motion becomes

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} x_2 \\ 3\, x_2 - x_1 \end{pmatrix}$$

Kinematics vs Dynamics: Car

Kinematic (first-order) model

- State $\boldsymbol{x} = (x, y, \theta)$
  - Position $(x, y) \in \mathbb{R}^2$
  - Orientation $\theta \in S^1$
- Control inputs: $u = (u_v, u_\phi)$
  - Translational velocity: $u_v \in \mathbb{R}$
  - Steering angle: $u_\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$
- Equation of motion

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} u_v \cos\theta \\ u_v \sin\theta \\ \dfrac{u_v \tan u_\phi}{L} \end{pmatrix}$$

Dynamics (second-order) model

- State $\boldsymbol{x} = (x, y, \theta, v, \phi)$
  - Position $(x, y) \in \mathbb{R}^2$
  - Orientation $\theta \in S^1$
  - Linear speed $v \in \mathbb{R}$
  - Steering angle: $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$
- Control inputs: $u = (u_1, u_2)$
  - Translational acceleration: $u_1 \in \mathbb{R}$
  - Steering rotational velocity: $u_2 \in \mathbb{R}$
- Equation of motion

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} v \cos\theta \\ v \sin\theta \\ (v/L)\tan\phi \\ u_1 \\ u_2 \end{pmatrix}$$

What is the state-space of the second-order simple Car

x, y, θ

$\dot{x}, \dot{y}, \dot{\theta}$

$\dot{x}, \dot{y}, \dot{\theta}, v, \varphi$

x, y, θ, v, φ

x, y, θ, u_1, u_2

What is the state-space of the second-order simple Car

What is the state-space of the second-order simple Car

Kinematics vs Dynamics: Differential Drive

Kinematic (first-order) model

- State $\boldsymbol{x} = (x, y, \theta)$
  - Position $(x, y) \in \mathbb{R}^2$
  - Orientation $\theta \in S^1$
- Control inputs: $u = (u_l, u_r)$
  - Angular velocities: $\mathrm{u}_l, u_r \in \mathbb{R}$
- Equation of motion

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} (r/2)(u_l + u_r)\cos\theta \\ (r/2)(u_l + u_r)\sin\theta \\ (r/2)(u_l - u_r) \end{pmatrix}$$

Dynamics (second-order) model

- State $\boldsymbol{x} = (x, y, \theta, \omega_l, \omega_r)$
  - Position $(x, y) \in \mathbb{R}^2$
  - Orientation $\theta \in S^1$
  - angular velocities $\omega_l, \omega_r \in \mathbb{R}$
- Control inputs: $u = (u_1, u_2)$
  - Angular accelerations: $\mathrm{u}_1, u_2 \in \mathbb{R}$
- Equation of motion

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\omega}_l \\ \dot{\omega}_r \end{pmatrix} = \begin{pmatrix} (r/2)(\omega_l + \omega_r)\cos\theta \\ (r/2)(\omega_l + \omega_r)\sin\theta \\ (r/2)(\omega_l - \omega_r) \\ u_1 \\ u_2 \end{pmatrix}$$

Kinematics vs Dynamics: Unicycle

Kinematic (first-order) model

- State $\boldsymbol{x} = (x, y, \theta)$
  - Position $(x, y) \in \mathbb{R}^2$
  - Orientation $\theta \in S^1$
- Control inputs: $u = (u_\sigma, u_\omega)$
  - Translational velocity: $\mathrm{u}_\sigma \in \mathbb{R}$
  - Rotational speed: $\mathrm{u}_\omega \in \mathbb{R}$
- Equation of motion

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} u_\sigma\, r\, \cos\theta \\ u_\sigma\, r\, \sin\theta \\ u_\omega \end{pmatrix}$$

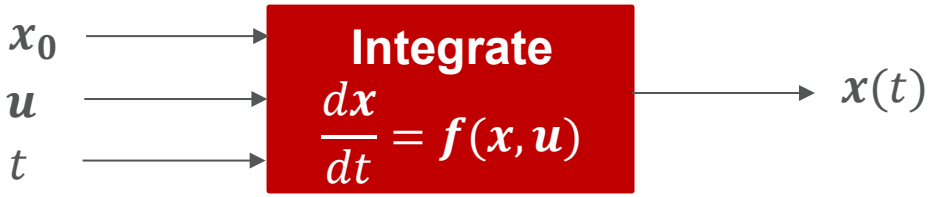
Dynamics (second-order) model

- State $\boldsymbol{x} = (x, y, \theta, \sigma, \omega)$
  - Position $(x, y) \in \mathbb{R}^2$
  - Orientation $\theta \in S^1$
  - Transitional velocity $\sigma \in \mathbb{R}$
  - Rotational speed $\omega \in \mathbb{R}$
- Control inputs: $u = (u_1, u_2)$
  - Translational acceleration: $\mathrm{u}_1 \in \mathbb{R}$
  - Rotational acceleration: $u_2 \in \mathbb{R}$
- Equation of motion

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\sigma} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \sigma\, r\, \cos\theta \\ \sigma\, r\, \sin\theta \\ \omega \\ u_1 \\ u_2 \end{pmatrix}$$

Generating Motions with Forward Propagation

- Robot motions obtained by applying input controls and integrating equations of motions



- Consider
  - A starting state $\boldsymbol{x_0}$
  - An input control $\boldsymbol{u}$
  - Motion equation: $\dot{x} = f(x, u)$
- Then,

$$x(t) = x_0 + \int_0^t f(x(\tau), u)\, d_\tau$$

- Computation can be carried out by:
  - Closed-form integration when available or
  - Numerical integration

Forward Numerical Propagation– Euler Method

Euler Method:

- Let Δt denote a small-time step. We would like to compute $\boldsymbol{x}(\Delta t)$ as

$$\boldsymbol{x}(\Delta t) = \boldsymbol{x}(0) + \int_0^{\Delta t} f(\boldsymbol{x}(\tau), \boldsymbol{u})\, d_\tau$$

- Euler approximation

$$\boldsymbol{f}(\boldsymbol{x}(t), u) = \dot{\boldsymbol{x}}(t) = \frac{d\boldsymbol{x}(t)}{dt} \approx \frac{\boldsymbol{x}(\Delta \mathrm{t}) - \boldsymbol{x}(0)}{\Delta \mathrm{t}}$$

- Therefore,

$$\boldsymbol{x}(\Delta t) \approx \boldsymbol{x}(0) + \Delta \mathrm{t}\, f(\boldsymbol{x}(t), \boldsymbol{u})$$

Forward Numerical Propagation– Euler Method

Euler Method:

$$\boldsymbol{x}(\Delta t) \approx \boldsymbol{x}(0) + \Delta \mathrm{t}\, f(\boldsymbol{x}(t), \boldsymbol{u})$$

- Example
  - Euler integration of the kinematic model of unicycle yields

$$\boldsymbol{x}(\Delta t) \approx \begin{pmatrix} x_0 \\ y_0 \\ \theta_0 \end{pmatrix} + \Delta t \begin{pmatrix} u_\sigma\, r\, \cos\theta \\ u_\sigma\, r\, \sin\theta \\ u_\omega \end{pmatrix}$$

- Advantage:
  - Simple and efficient
- Disadvantage:
  - Not very accurate (first-order approximation)

Forward Numerical Propagation– Runge-Kutta Method

Runge-Kutta Method:

- Let Δt denote a small-time step. We would like to compute $\boldsymbol{x}(\Delta t)$ as

$$\boldsymbol{x}(\Delta t) = \boldsymbol{x}(0) + \int_0^{\Delta t} f(\boldsymbol{x}(\tau), \boldsymbol{u})\, d_\tau$$

- Fourth-order Runge-Kutta integration (derived by performing a Taylor expansion at $\boldsymbol{x}(\frac{1}{2}\Delta t)$):

$$\boldsymbol{x}(\Delta t) \approx \boldsymbol{x}(0) + \frac{\Delta t}{6}\,(w_1 + 2w_2 + 2w_3 + w_4),$$

Where:

$$w_1 = f(\boldsymbol{x}(0), u),$$

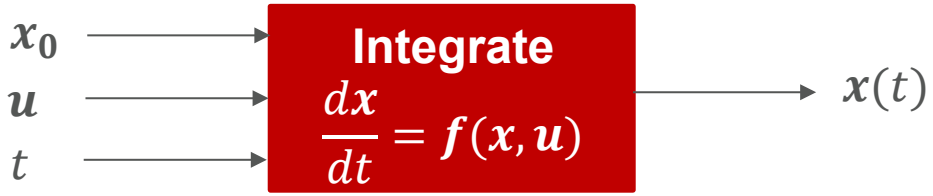
$$w_2 = f\left(\boldsymbol{x}(0) + \frac{\Delta t}{2} w_1, u\right),$$

$$w_3 = f\left(\boldsymbol{x}(0) + \frac{\Delta t}{2} w_2, u\right),$$

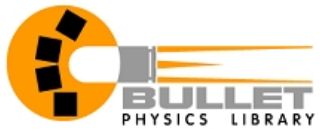
$$w_4 = f(\boldsymbol{x}(0) + \Delta t\, w_3, u),$$

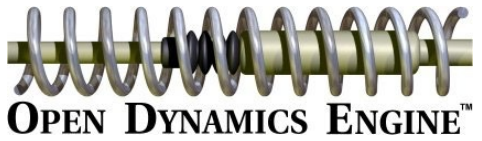
Forward Numerical propagation with Physics-based Simulations

- Tree-based approaches require only the ability to simulate robot motions



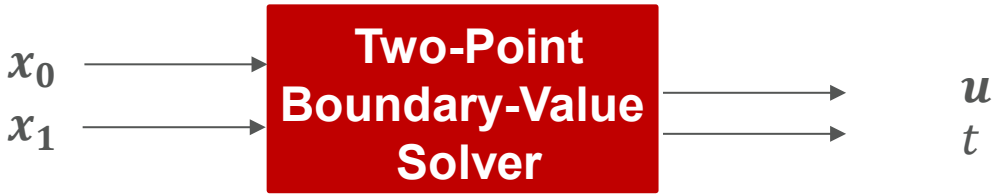
- Physics engines can be used to simulate robot motions
- Physics engines provide greater simulation accuracy
- Physics engines can take into account friction, gravity, and interactions of the robot with objects in the environment
- Existing tools





Two-Point Boundary Value Problem (BVP)

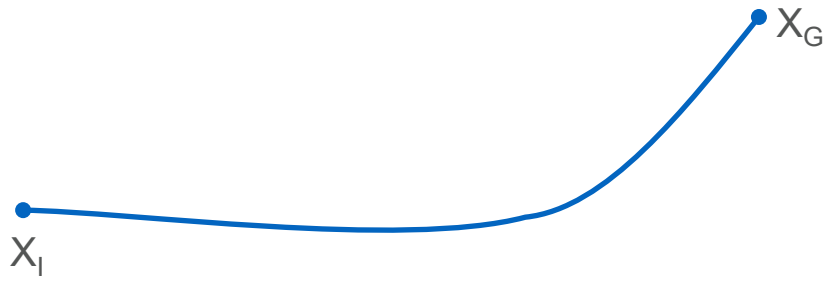
- The 2 point boundary problem arises when we need to compute the control u to move from point $\boldsymbol{x_0}$ to x_1



- In general quite challenging problem, akin to solving a full optimal control problem.

Moving between two states (without obstacles)

- **Two-Point Boundary Value Problem (BVP):**
  - Find a control sequence to take system from state X_I to state X_G while obeying kinematic constraints.
  - How to solve this problem?



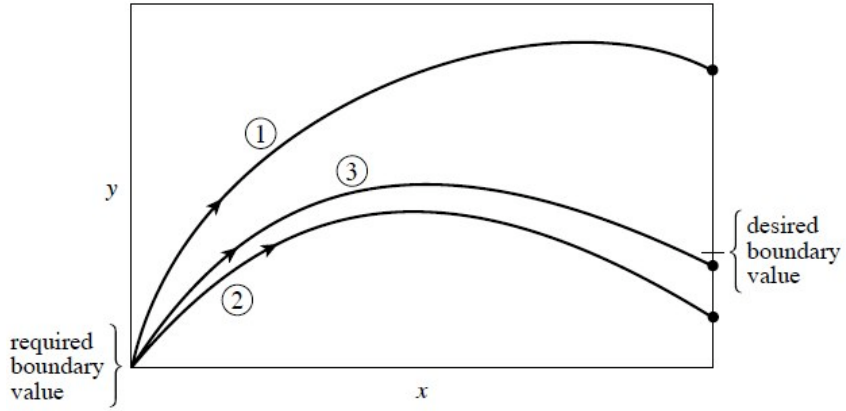
Shooting Method for solving a BVP problem

- "Shoot" out trajectories in different directions until a trajectory of the desired boundary value is found.
  - System

$$\frac{d\mathbf{y}}{dx} + \mathbf{f}(x, \mathbf{y}) = 0$$

  - Boundary condition

$$y(0) = 0,\ y(1) = 1$$

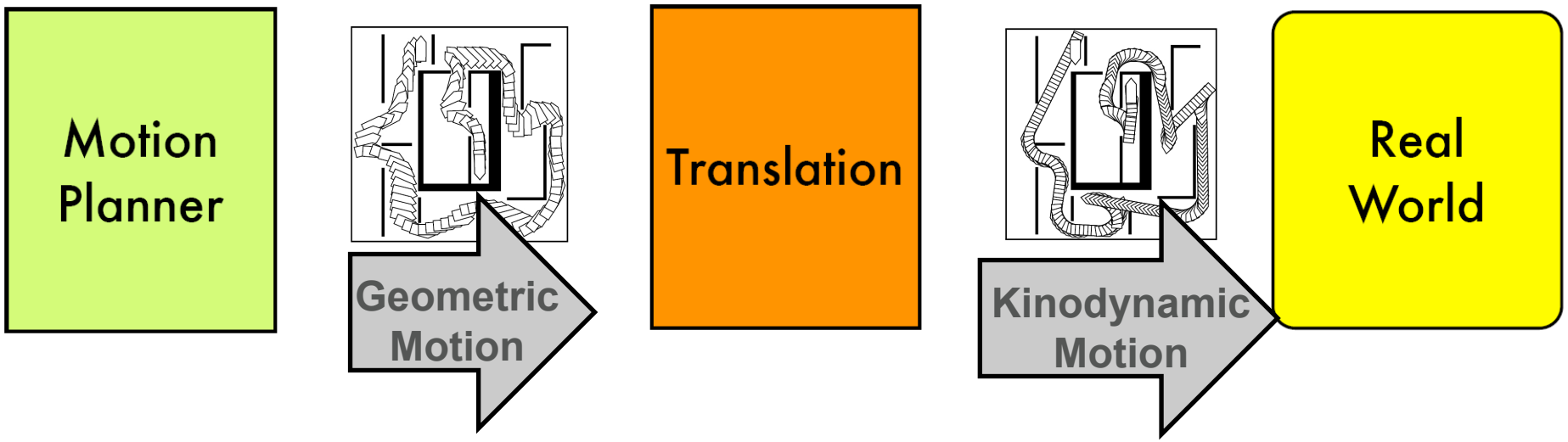


  - Convert to an initial boundary problem: Pick initial guess, iteratively get closer to goal

Motion Planning with Kinodynamic Constraints

- **Decoupled approach**
  1. Compute a geometric solution path ignoring differential constraints
  2. Transform the geometric path into a trajectory that satisfies the differential constraints
- **Native approach**
  - Take the differential constraints into account during motion planning
    - Sampling-based techniques:
      - Roadmap approaches
      - Tree approaches

Decoupled Motion Planning Paradigm

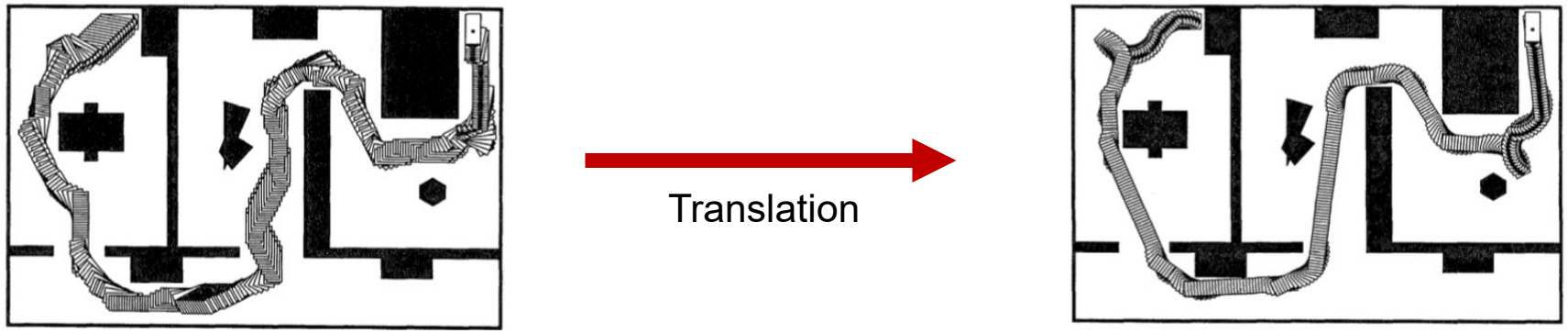


- Finds a geometrically feasible plan with a planner that is approximately correct
- Satisfy any remaining physical constraints using a translation step
- Execute in the real world

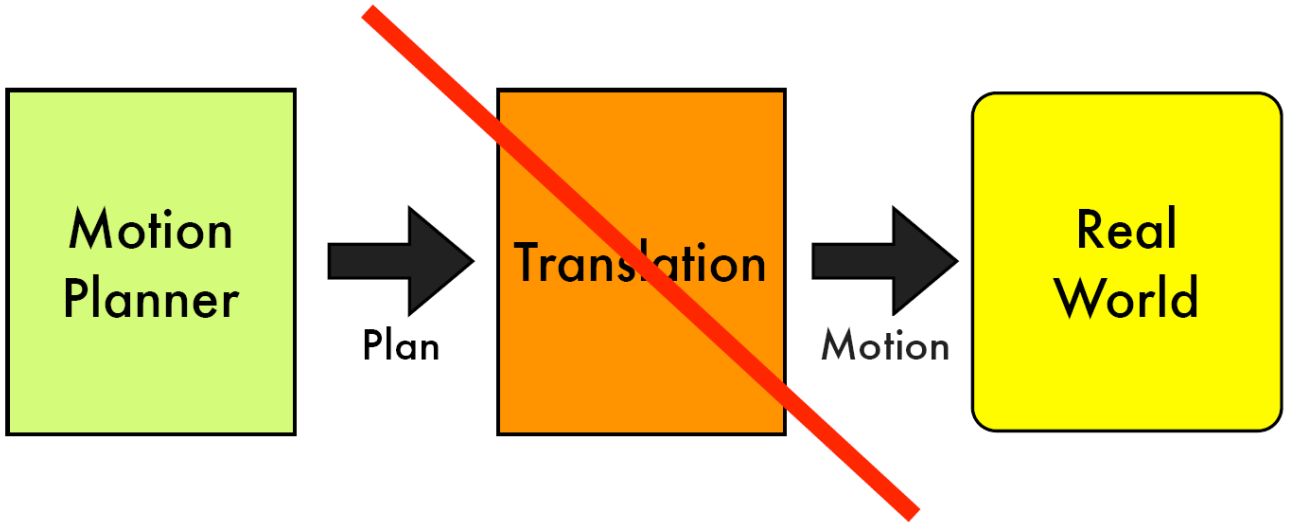
Motion Planning with Kinodynamic Constraints

- **Decoupled approach**

  1. Compute a geometric solution path ignoring differential constraints
  2. Transform the geometric path into a trajectory that satisfies the differential constraints

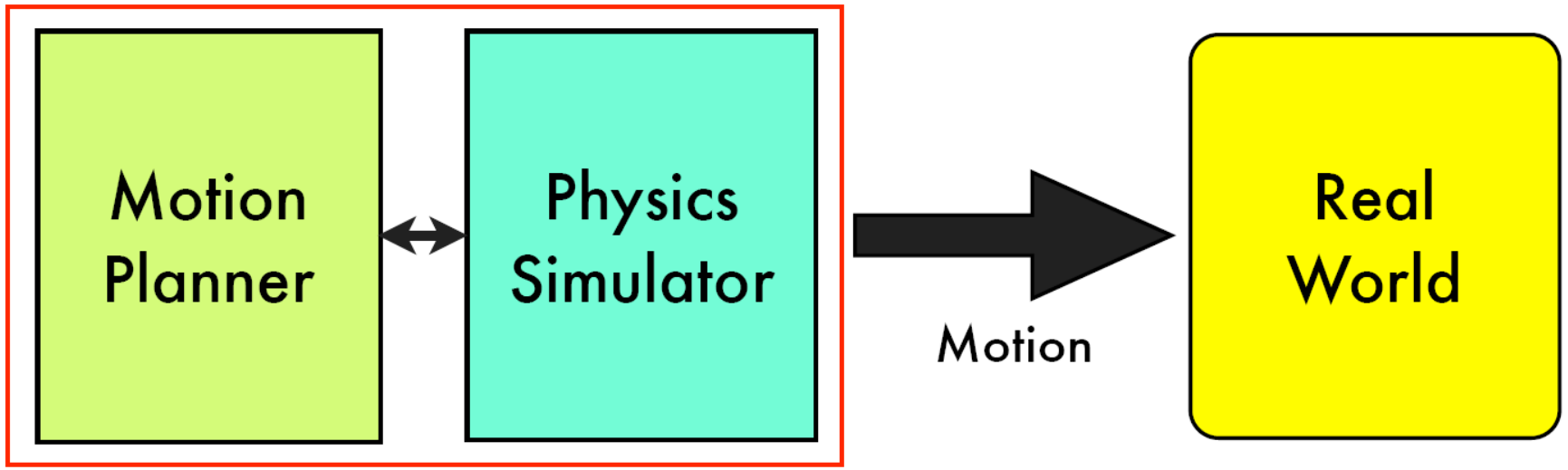


Shortcomings of Decoupled Motion Planning



- Motions are often low quality or inadmissible
- Translating “collision-free” paths to physical motion is hard

Another approach (Native Approach)



- Integrate physical/differential constraints in the motion planner
- How can we do that?

Motion Planning with Kinodynamical Constraints

- Planning Problem, given:
  - State space X
  - Control space U
  - Equations of motion as differential equations: $f: X \times U \to \dot{X}$
  - State-validity function $\text{valid}: X \to \{\text{true}, \text{false}\}$, e.g., check collision
  - Goal function $\text{goal}: X \to \{\text{true}, \text{false}\}$
  - Initial state $\boldsymbol{x}_0$
- Compute
  - a control trajectory $u: [0, T] \to U$ such that the resulting state trajectory $\boldsymbol{x}: [0, T] \to X$ obtained by integration is valid and reaches the goal:

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \int_0^t f(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau))\, d_\tau \qquad \text{s.t.} \qquad \forall t \in [0, T]: \text{valid}\big(x(t)\big) = \text{true}$$

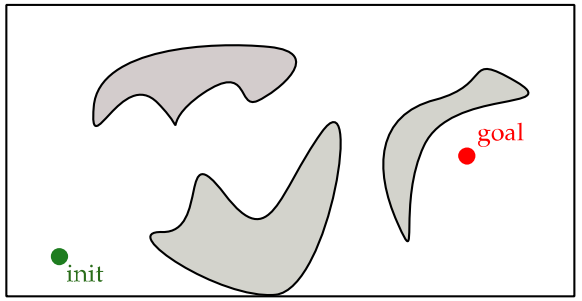
$$\exists t \in [0, T]: \text{goal}\big(x(t)\big) = \text{true}$$

Sampling-based Motion Planning with Kinodynamical Constraints

Roadmap Approaches

0. Initialization

- Add $\boldsymbol{x}_0$ and $\boldsymbol{x}_{\text{goal}}$ to roadmap vertex set V



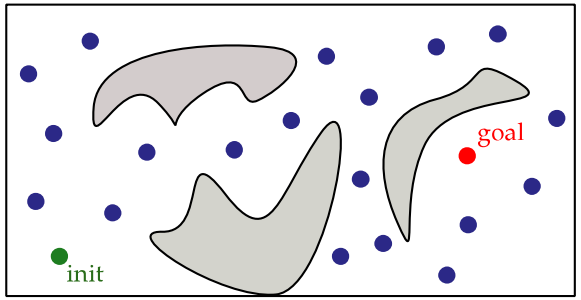
1. Sampling

- Repeat several times

  $\boldsymbol{x} \leftarrow \text{StateSample}()$

  If $\text{IsStateValid}(\boldsymbol{x})$ = true

    add $\boldsymbol{x}$ to roadmap vertex set V



Sampling-based Motion Planning with Kinodynamical Constraints

Roadmap Approaches

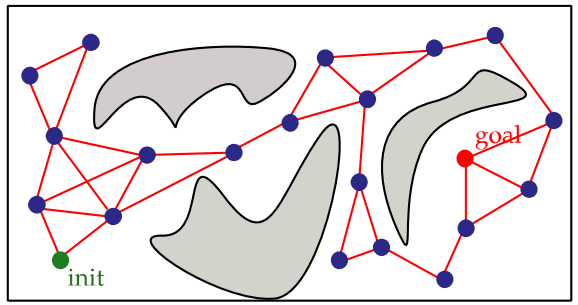
2. Connect Samples

- for each pair of neighboring samples $(x_a, x_b) \in V \times V$

$\lambda \leftarrow \text{GenerateLocalTrajectory}(x_a, x_b)$

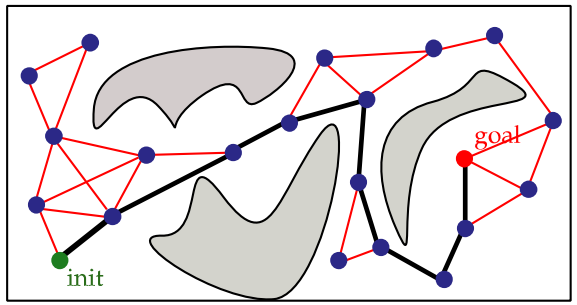
If $\text{IsTrajectoryValid}(\lambda) = \text{true}$

add (x_a, x_b) to roadmap edge set E



3. Graph Search

- Search graph (V, E) for path from x_0 to x_{goal}



Sampling-based Motion Planning with Kinodynamical Constraints

Implementation of Roadmap Approaches

- $x \leftarrow \text{StateSample}()$
  - Generate random values for all state components

- $\text{IsStateValid}(\boldsymbol{x})$
  - place the robot in the configuration specified by the position and orientation components of the state
  - check if the robot collides with the obstacles
  - check if velocity and other state components are within desired bounds

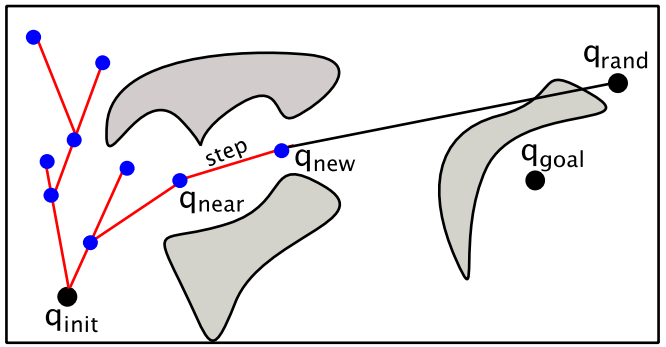
Sampling-based Motion Planning with Kinodynamical Constraints

Implementation of Roadmap Approaches

- $\lambda \leftarrow \text{GenerateLocalTrajectory}(x_a, x_b)$
  - linear interpolation between x_a and x_b does NOT work as it does not respect underlying differential constraints
  - need to find control function $u : [0, T] \rightarrow U$ such that trajectory obtained by applying u to x_a for T time units ends at x_b
  - known as two-point boundary value problem
  - cannot always be solved analytically, and numerical solutions increase computational cost
- $\text{IsTrajectoryValid}(\lambda)$
  - use subdivision or incremental approach to check if intermediate states are valid

Sampling-based Motion Planning with Kinodynamical Constraints

Tree-based Approaches



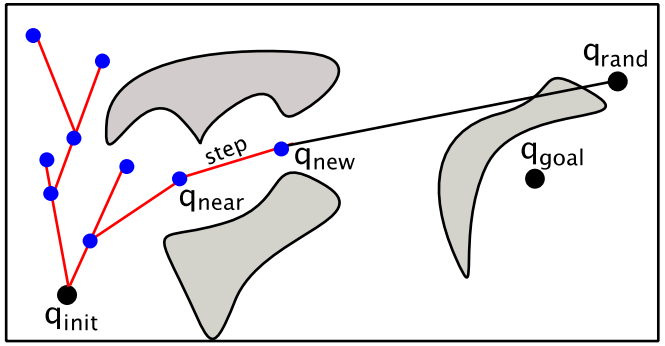
RRT

1: $\mathrm{T} \leftarrow$ create tree rooted at x_0
2: **While** solution not found **do**
\\ select state from tree
3: $x_{rand} \leftarrow \mathrm{StateSample}()$
4: $x_{near} \leftarrow$ nearest state in T to x_{rand} according to distance ρ
\\ add new branch to tree from selected state
5: $\lambda \leftarrow \mathrm{GenerateLocalTrajectory}(x_{near}, x_{rand})$
6: **if** $\mathrm{IsSubTrajectoryValid}(\lambda, 0, step)$ **then**
7: $x_{new} \leftarrow \lambda(step)$
8: add configuration x_{new} and edge (x_{near}, x_{new}) to T
\\ check if a solution is found
9: **if** $\rho(x_{new}, x_{goal}) \approx 0$ **then**

10: **return** solution trajectory from root to x_{new}

Sampling-based Motion Planning with Kinodynamical Constraints

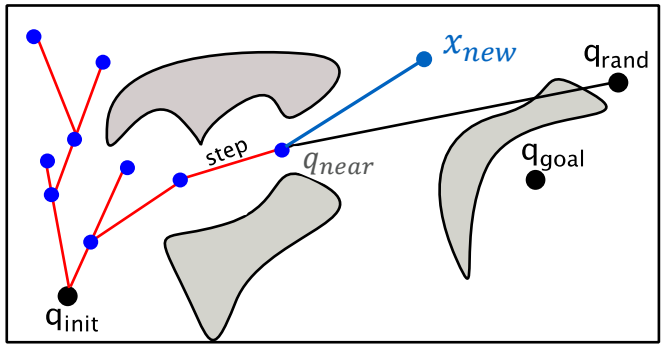
Implementation of Tree-based Approaches



- SampleState()
  - random values for state components
- $\rho: X \times X \rightarrow \mathbb{R}^{\geq 0}$
  - Distance metric between states
- $\lambda \leftarrow \text{GenerateLocalTrajectory}(x_{near}, x_{rand})$
  - does it not create the same two-boundary value problems as in PRM?
  - is it necessary to connect to x_{rand}?
  - does it suffice to just come close to x_{rand}?
- $\text{IsSubTrajectoryValid}(\lambda, 0, steps)$
  - Incremental approach

Avoiding Two-Boundary Value Problem

Idea: Rather than computing a trajectory from x_{near} to x_{rand} compute a trajectory that starts at x_{near} and extends toward x_{rand}

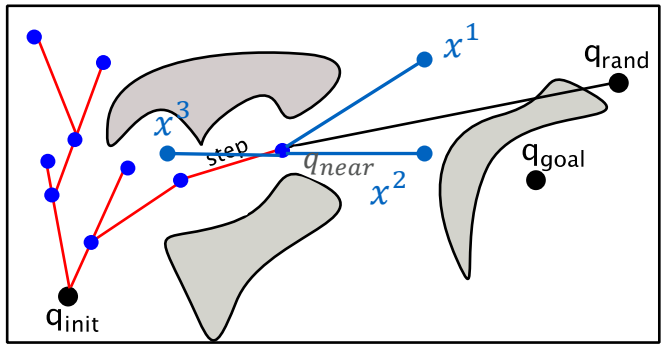


- Approach 1 – extend according to random control
  - Sample random control u in U
  - Integrate equations of motions when applying u to x_{near} for Δt units of time, i.e.,

$$\lambda \leftarrow x(t) = x_{near} + \int_0^{\Delta t} f(x(\tau), u)\, d\tau$$

Avoiding Two-Boundary Value Problem

Idea: Rather than computing a trajectory from x_{near} to x_{rand} compute a trajectory that starts at x_{near} and extends toward x_{rand}

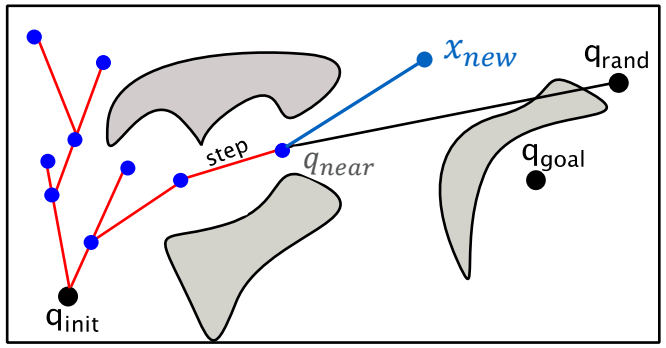


- Approach 2 – find the best-out-of-many random controls

1: **for** $i = 1, \ldots, m$ **do**
2: $u \leftarrow$ sample random control in U
3: $\lambda \leftarrow x(t) = x_{near} + \int_0^{\Delta t} f(x(\tau), u)\, d\tau$
4: $\mathrm{d_i} \leftarrow \rho(x_{rand}, \lambda_i(\Delta t)$
5: **return** λ_i with minimum d_i

Avoiding Two-Boundary Value Problem

Idea: Rather than computing a trajectory from x_{near} to x_{rand} compute a trajectory that starts at x_{near} and extends toward x_{rand}



- Approach 2 – find the best-out-of-many random controls

1: **for** $i = 1, \ldots, m$ **do**
2: $u \leftarrow$ sample random control in U
3: $\lambda \leftarrow x(t) = x_{near} + \int_0^{\Delta t} f(x(\tau), u)\, d\tau$
4: $\mathrm{d_i} \leftarrow \rho(x_{rand}, \lambda_i(\Delta t)$
5: **return** λ_i with minimum d_i

Overview

- Kinematic constraints
- Dynamics constraints
- Integration of the dynamics
- Planning with PRM for Kino-dynamic systems
- Planning with RRT for Kino-dynamic systems