Intro to Robotics: Review for Exam 3

Neeloy Chakraborty

neeloyc2@illinois.edu

Agenda

- Common Mistakes on Last Exam
- Trajectory Generation
- Motion Planning
- Manipulation

Common Mistakes on Exam 2

- Writing [S] instead of S when we asked for the screw
- Missing negative sign when computing v = -w x q
- Forgetting to multiply by the pitch when computing v = - w x q + h*w
- Correctly performing calculations but then referring to wrong variable name, leading to wrong answer
- Formatted answer wrong (row vs. column vector; missing commas in input)
- e^(A+B) does not equal (e^A)(e^B) for matrices A & B!

Normalized Trajectories

A trajectory is a mapping from continuous time to a robot state

$$\theta: [0, \mathrm{T}] \rightarrow \mathbb{R}^n$$

Where T is the time at the end of the trajectory & n is the state dim

We define normalized trajectories with an additional time-scaling function to map continuous time to a completion percentage along the trajectory

$$\mathrm{s}[0, \mathrm{T}] \rightarrow [0,1]; \qquad \theta: [0,1] \rightarrow \mathbb{R}^n$$

Normalized Trajectories

$$s[0, T] \rightarrow [0,1]; \quad \theta: [0,1] \rightarrow \mathbb{R}^n$$

Different people might define their time-scaling function differently

Suppose we are working with a straight-line path:

$$\theta(s) = \theta_0 + s(t) * (\theta_1 - \theta_0)$$

$$\dot{\theta}(s) = \frac{ds}{dt} * (\theta_1 - \theta_0)$$

$$\ddot{\theta}(s) = \frac{d^2 s}{dt} * (\theta_1 - \theta_0)$$

s and T can be chosen to meet robot and task constraints

Trajectory generation

View...

We want a robot to move from one location to another location in a straight line with time $T = 2$. Let's parameterize the path as follows:

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

where time $t \in [0, T]$. We need the robot to start with zero velocity, and reach the destination with zero velocity. Find a_0, a_1, a_2, a_3.

$a_0 =$	number (rtol=0.01, atol=1e-08)	?	$a_1 =$	number (rtol=0.01, atol=1e-08)	?
$a_2 =$	number (rtol=0.01, atol=1e-08)	?	$a_3 =$	number (rtol=0.01, atol=1e-08)	?

$$s(0) = s'(0) = s'(T) = 0$$

The initial completion %, and initial/end velocity should be 0

$$s(T) = 1$$

The final completion % should be 1

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \qquad s'(t) = a_1 + 2a_2 t + 3a_3 t^2$$

$$\boxed{s(0) = a_0 = 0} \qquad \boxed{s'(0) = a_1 = 0}$$

Using the fact that $s(T) = 1$ & $s'(T) = 0$:

$$a_2 = \frac{1}{T^2} - a_3 T \qquad \frac{2}{T} + T^2 a_3 = 0$$

$$\boxed{a_3 = -\frac{2}{T^3}} \qquad \boxed{a_2 = \frac{3}{T^2}}$$

Then, we can plug in $T = 2$

Trajectory generation

A cylindrical path in $X = (x, y, z)$ is given by:

$$x = cos(2\pi s)$$

$$y = sin(2\pi s)$$

$$z = 2s$$

where $s \in [0, 1]$.

Suppose we have a time-scaling given as $s(t) = 0.25t + 0.125t^2$, $t \in [0, 2]$. Find $\dot{X}$ and $\ddot{X}$ when $t = 0.9$.

$$x(s) = \cos(2\pi s) \qquad x'(s) = -2\pi s' * \sin(2\pi s)$$

$$x''(s) = (-2\pi s')(2\pi s' * \cos(2\pi s)) + (\sin(2\pi s))(-2\pi s'')$$

$$y(s) = \sin(2\pi s) \qquad y'(s) = 2\pi s' * \cos(2\pi s)$$

$$y''(s) = (2\pi s')(-2\pi s' * \sin(2\pi s)) + (\cos(2\pi s))(2\pi s'')$$

$$z(s) = 2s \qquad z'(s) = 2s' \qquad z''(s) = 2s''$$

$$s(t) = \frac{1}{4}t + \frac{1}{8}t^2 \qquad s'(t) = \frac{1}{4} + \frac{1}{4}t \qquad s''(t) = \frac{1}{4}$$

Plugging in $t = 0.9$ gives us the position, velocity, and acceleration of the body at the given time

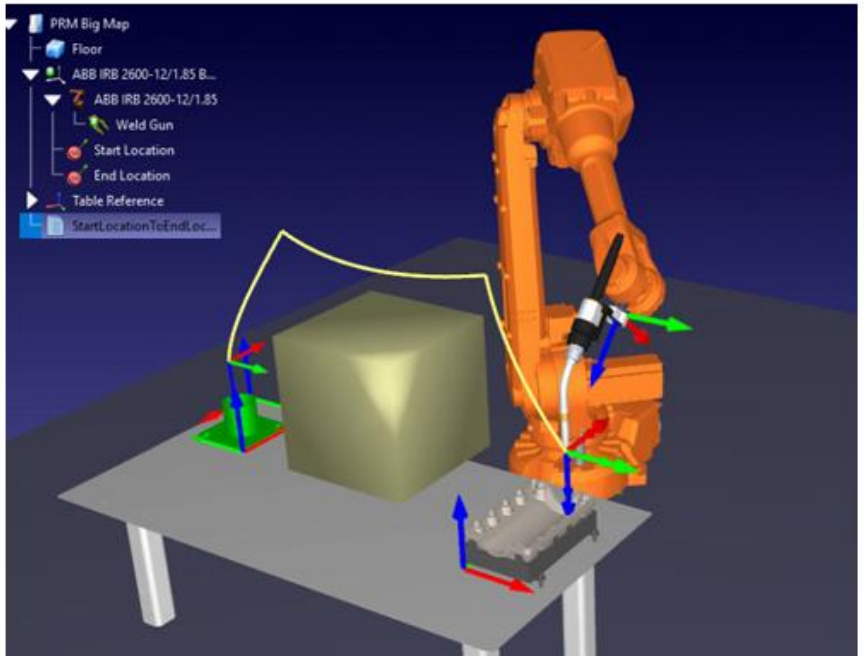
Suppose we have a time-scaling given as $s(t) = 0.25t + 0.125t^2$, $t \in [0, 2]$. Find $\dot{X}$ and $\ddot{X}$ when $t = 0.9$.

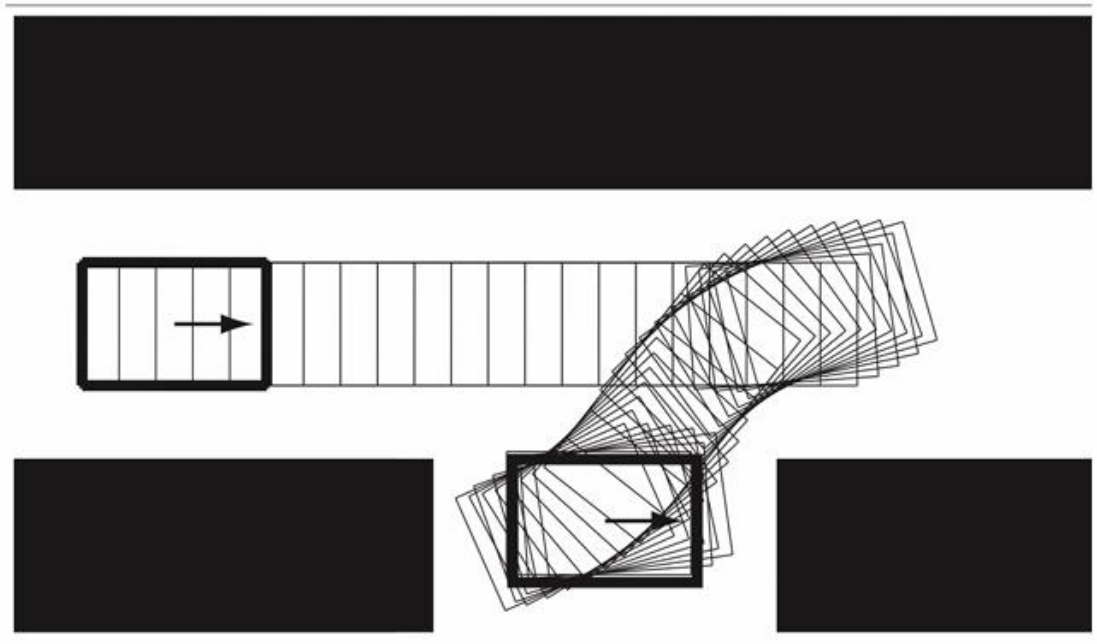
$\dot{x} =$	-2.6484969876793265	?	✓ 100%
$\dot{y} =$	-1.3757840231070713	?	✓ 100%
$\dot{z} =$	0.95	?	✓ 100%
$\ddot{x} =$	2.7120995479342964	?	✓ 100%
$\ddot{y} =$	-8.628570599837545	?	✓ 100%
$\ddot{z} =$	0.5	?	✓ 100%

Note that t ranges from 0 to 2. We can plot the trajectory, velocity, and acceleration of the body for all valid times to ensure we meet actuator constraints!

Overview of Motion Planning

- **Motion planning** is the problem of finding a robot motion from start state to a goal state that avoids obstacles in the environment
- Recall the **configuration space or C-space:** every point in the C-space $\mathcal{C} \subset \mathbb{R}^n$ corresponds to a unique configuration q of the robot
  - E.g., configuration of a robot arm is $q = (\theta_1, \theta_2, \ldots, \theta_n)$
- The **free C-space** $\mathcal{C}_{\text{free}}$ consists of the configurations where the robot neither collides with obstacles nor violates constraints





Motion Planning

Given an initial state $x(0) = x_{start}$ and a desired final state x_{goal}, find a time T and a set of controls u: $[0, T] \rightarrow \mathcal{U}$ such that the motion satisfies $x(T) = x_{goal}$ and $q\big(x(t)\big) \in \mathcal{C}_{\text{free}}$ for all $t \in [0, T]$

Assumptions:

1. A feedback controller can ensure that the planned motion is followed closely
2. An accurate model of the robot and environment will evaluate $\mathcal{C}_{\text{free}}$ during motion planning

Motion Planning Methods

- **Complete methods:** exact representations of the geometry of the problem and space
- **Grid methods:** discretize $\mathcal{C}_{\text{free}}$ and search the grid from q_{start} to goal
- **Sampling Methods:** randomly sample from the C-space, evaluate if the sample is in $\mathcal{X}_{\text{free}}$, and add new sample to previous samples
- **Virtual potential fields:** create forces on the robot that pull it toward goal and away from obstacles
- **Nonlinear optimization:** minimize some cost subject to constraints on the controls, obstacles, and goal
- **Smoothing:** given some guess or motion planning output, improve the smoothness while avoiding collisions

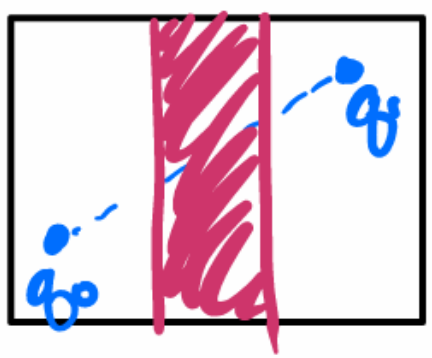
Configuration Space Obstacles

- We want to partition our C-space into parts:

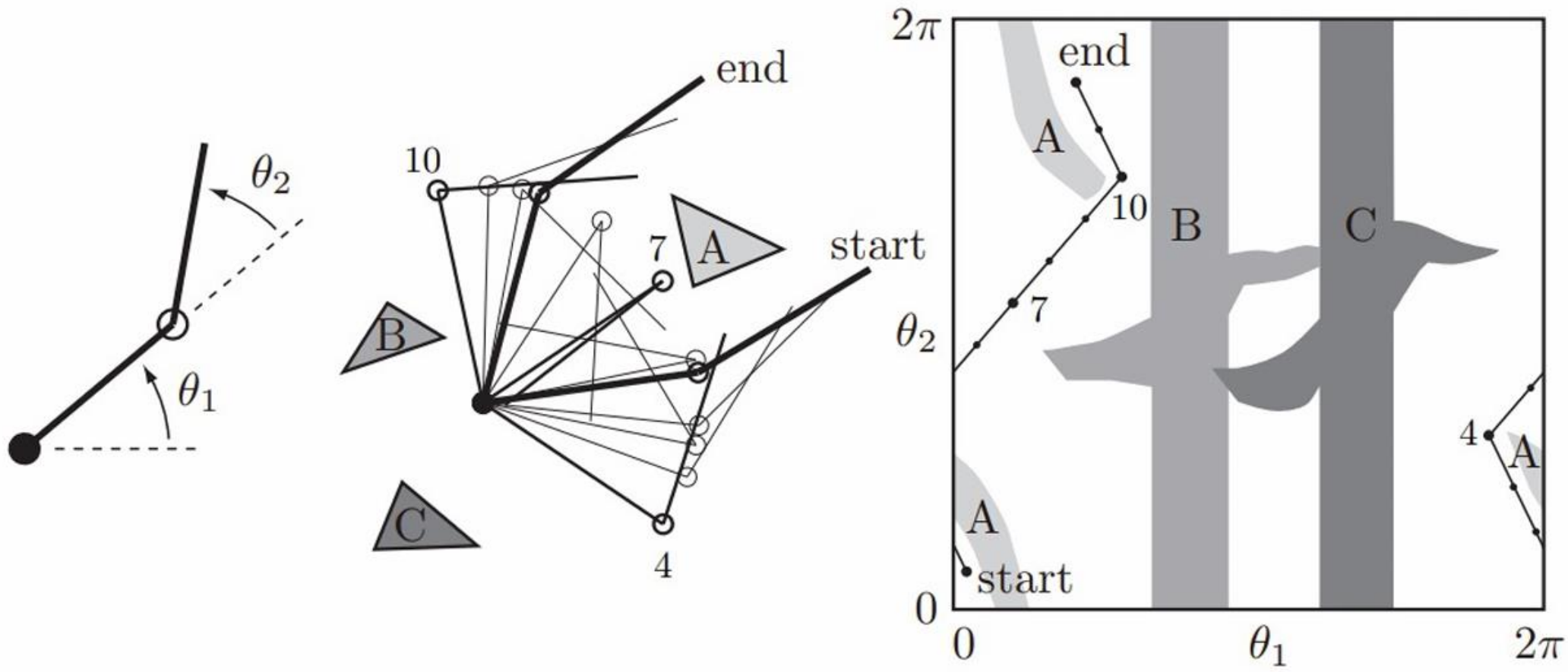
$C = C_{free} \cup C_{obs}$

obstacles
joint limits

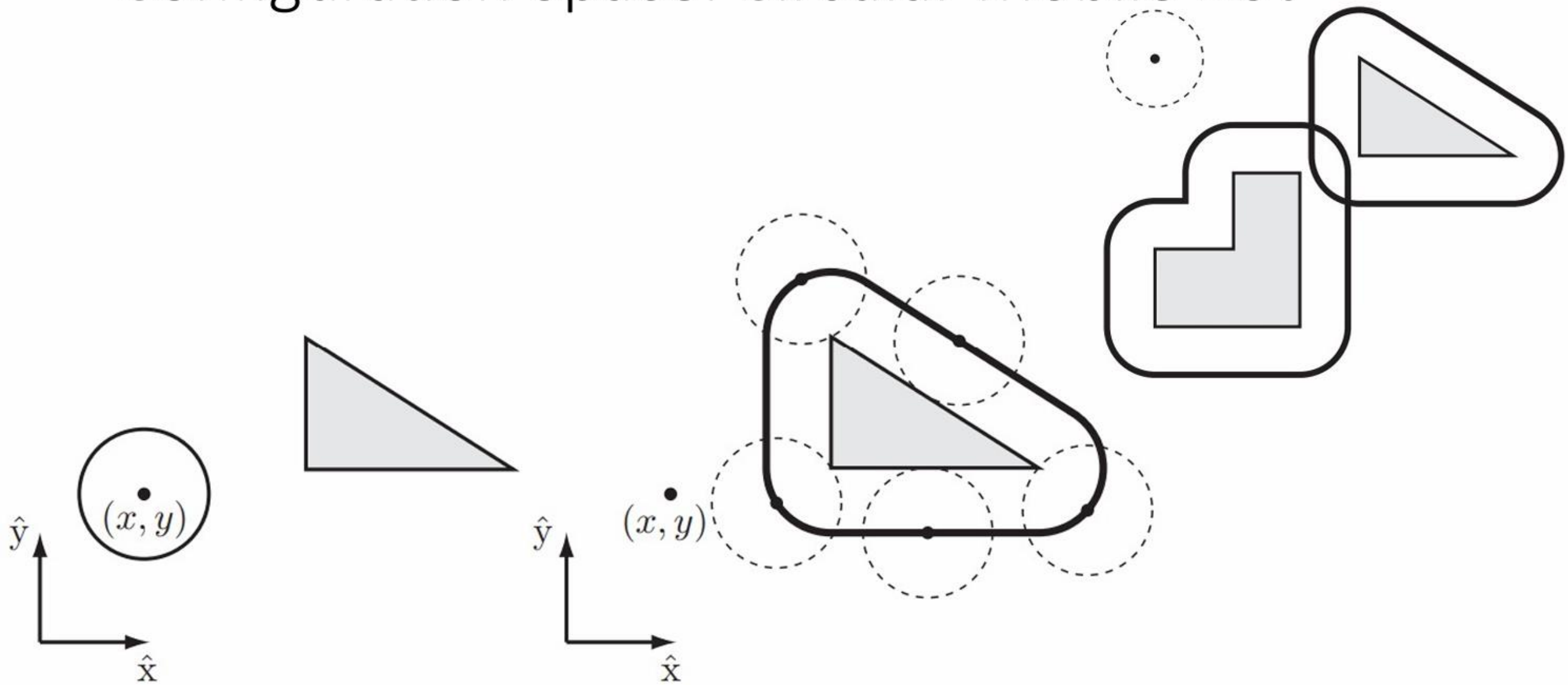
- If obstacles break C_{free} into separate components and q_0 and q_1 are not in the same connected components, then there is no collision free path



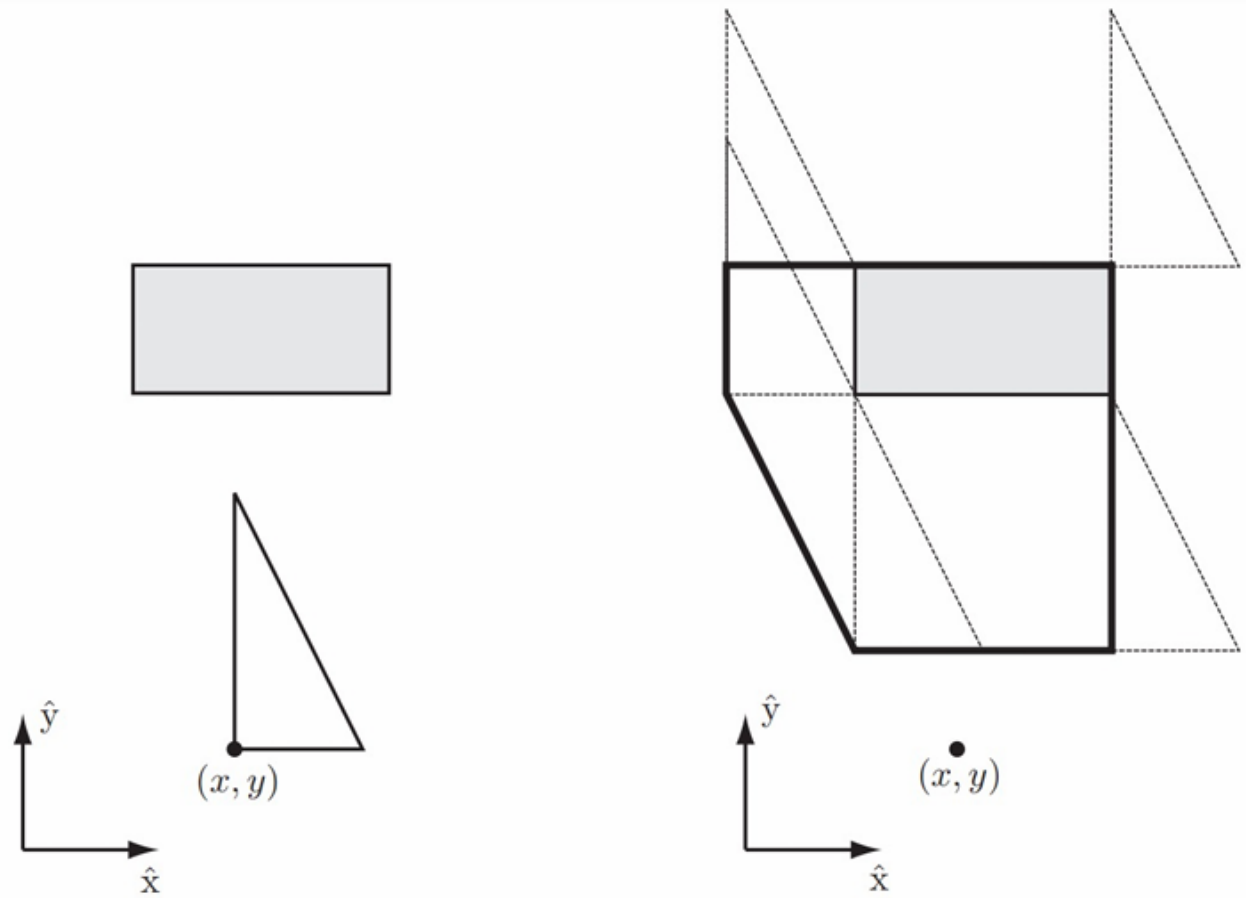
Configuration Space: 2R Planar Arm



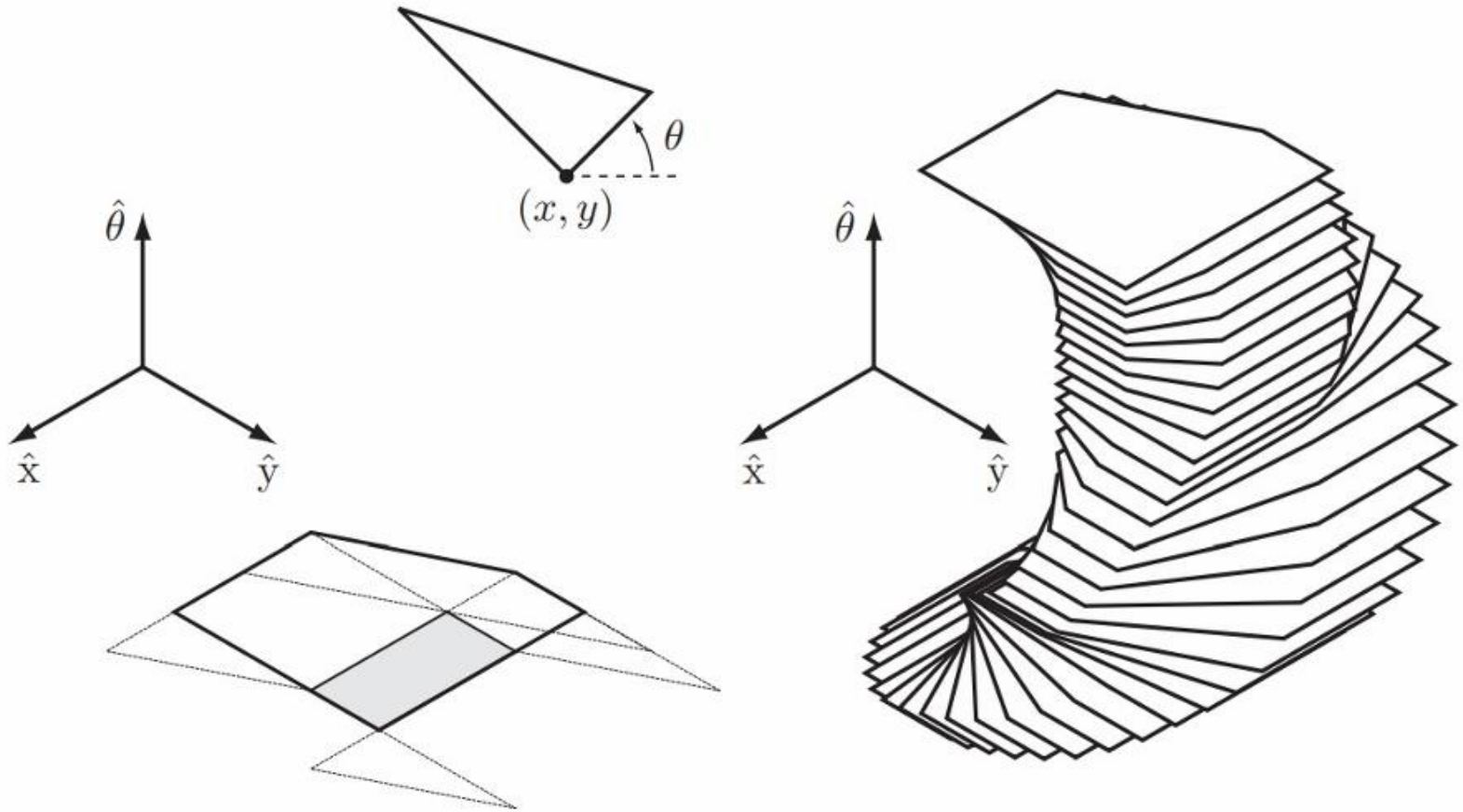
Configuration Space: Circular Mobile Bot



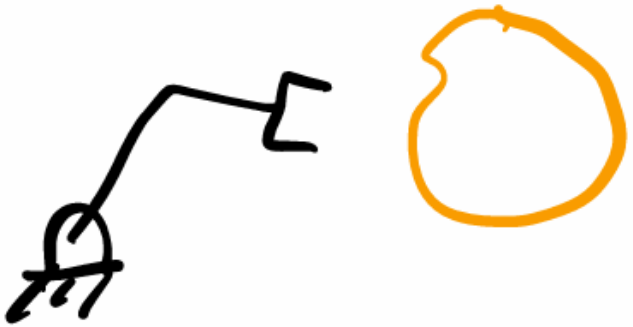
Configuration Space: bot that translates + rotates



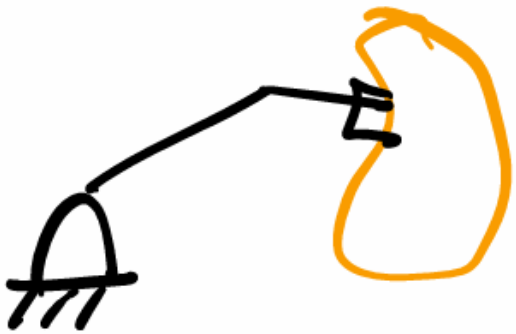
Configuration Space: bot that translates + rotates



Collision Detection



No contact



Contact

Penetration

We use a function $d(q, \mathcal{B})$ to measure the distance between the robot and the obstacle

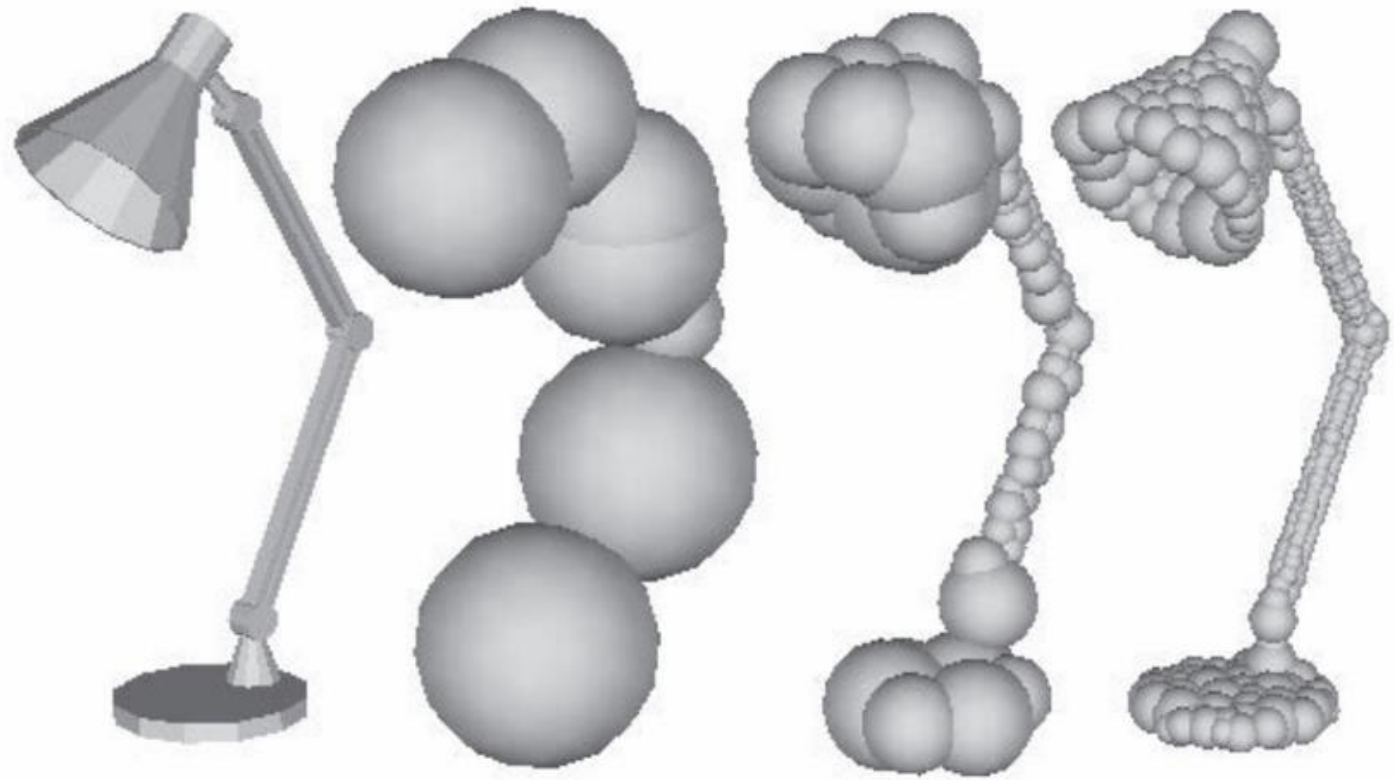
$d(q, \mathcal{B}) > 0$ -> No contact

$d(q, \mathcal{B}) = 0$ -> Contact

$d(q, \mathcal{B}) < 0$ -> Penetration

Spherical Approximations

One simple method is to approximate the robot and obstacles as unions of overlapping spheres



Distance Measures

Given a robot at q represented by k spheres of radius R_i centered at $r_i(q)$, and an obstacle $\mathcal{B}$ represented by l spheres of radius B_j centered at b_j, the distance can be calculated as:

$$d(q, \mathcal{B}) = \min_{i,j} \| r_i(q) - b_j \| - R_i - B_j$$

$$\forall\ i = 1, \ldots, k$$
$$j = 1, \ldots, l$$

Don't forget to check for robot self-collisions!

Are two spheres in collision

View...

Two spheres of radius r_1 and r_2 have centers at points p_1 and p_2 (both written in the coordinates of the same frame), respectively. Are these spheres in collision?

matlab | python

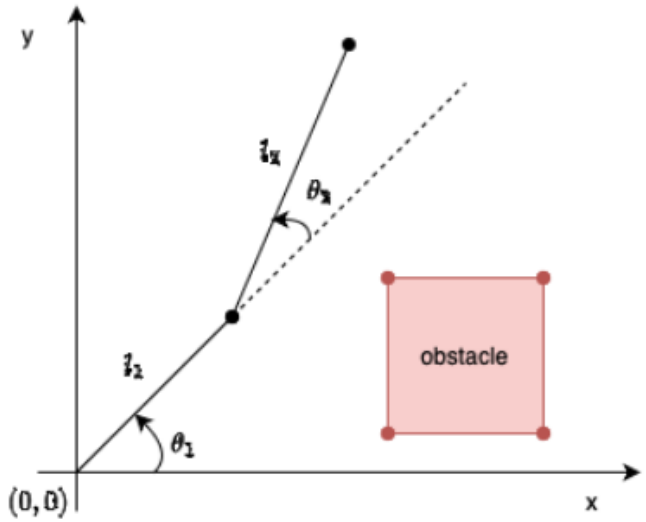
```
import numpy as np

p_1 = np.array([[-1.9706], [-1.2766], [-4.7440]])
p_2 = np.array([[-1.3391], [-2.0459], [-1.9077]])
r_1 = 1.9808
r_2 = 1.7493
```

Is $||p_2 - p_1|| > r_1 + r_2$?

No! Thus, there is a collision.

Robot Arm and Obstacles Colliding Sphere

View...



Consider the robot configuration shown above for a two link robot with link lengths l_1, l_2 and joint angles θ_1, θ_2 and a rectangular obstacle.

The robot can be represented by 20 spheres given by the following list of spheres, $robot\_spheres = [(x_i, y_i, r_i)]_{i=1}^{20}$. The obstacle can be represented by 40 spheres given by the list, $obstacle\_spheres = [(x_i, y_i, r_i)]_{i=1}^{40}$.

For the ith sphere in both lists: (x_i, y_i, r_i); the center of the sphere is (x_i, y_i) and the radius of the sphere is r_i.

matlab | python

```
import numpy as np

N_robot = 20.00000000
N_obstacles = 40.00000000
robot_spheres = np.array([[0.00000000, 0.00000000, 1.09776243], [0.46914896, -0.17291401, 1.09776243], [0.93829791,
obstacle_spheres = np.array([[4.88022208, 8.63180587, 0.17393494], [4.19093438, 7.02163727, 0.41332998], [3.8607081
```

Now, we can check collision between each robot-obstacle sphere pair

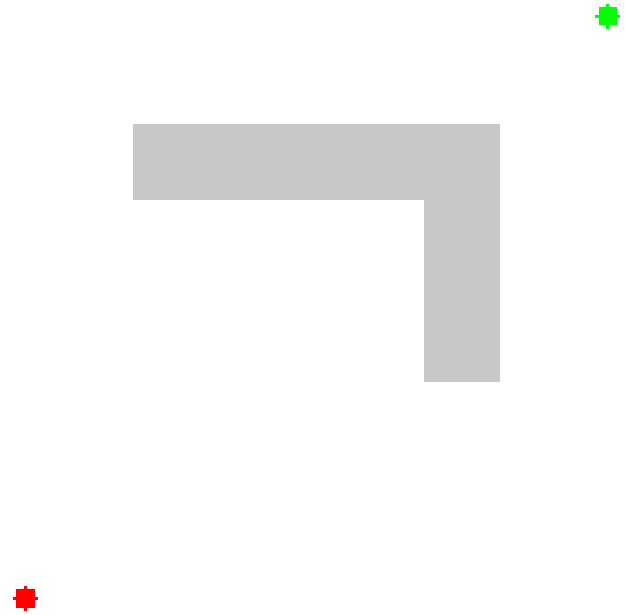
Key Properties of Planners

Complete:	If there is a solution, the planner is guaranteed to find it
Probabilistically Complete:	With more samples, the planner will converge to a solution if one exists
Optimal:	The solution found is the shortest path
Satisficing:	The solution found may be suboptimal

Graph-based Methods

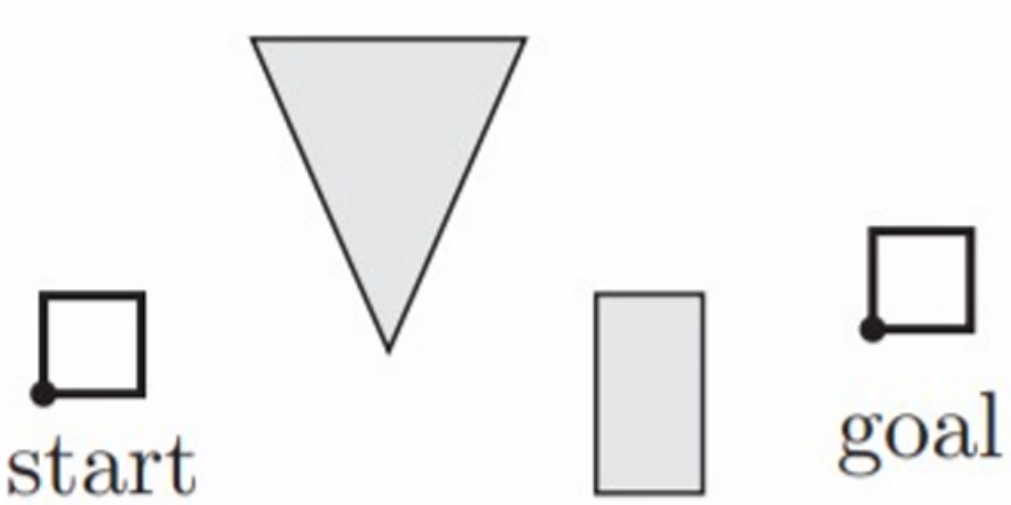
A*

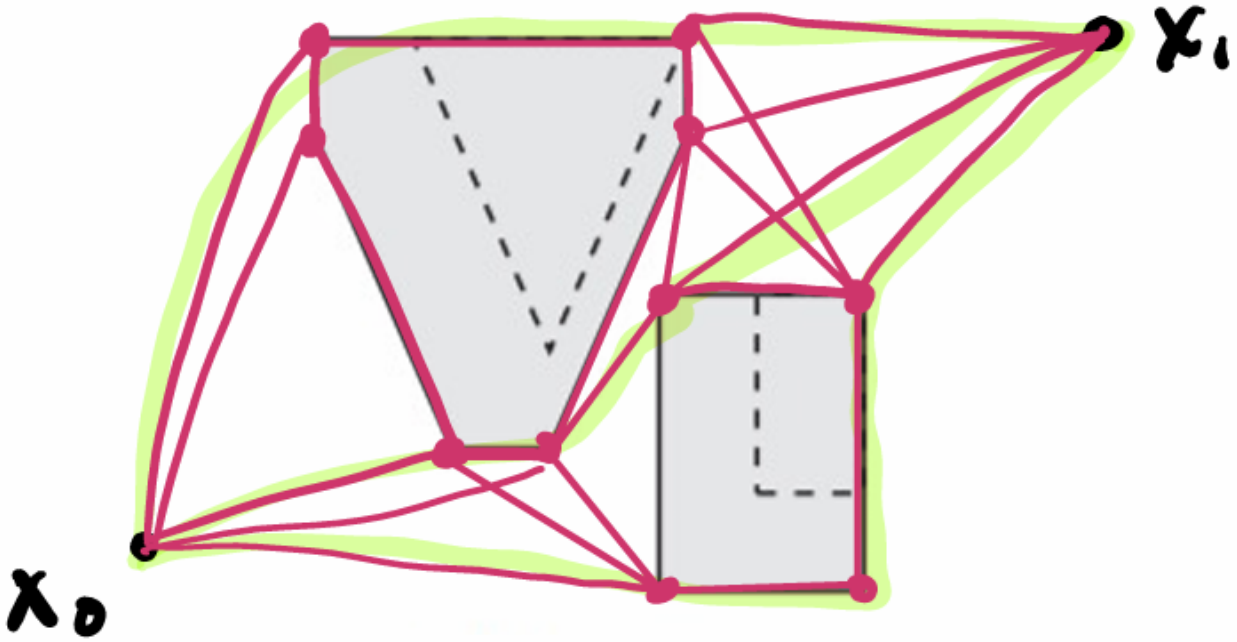
Dijkstra's



Complete and optimal

A simple roadmap: visibility graph





Properties of the planner depend on the construction method of the visibility graph

Sampling methods

- A random or deterministic function to choose a sample from the C-space or state space
- A function to evaluate whether a sample is in $\mathcal{C}_{\text{free}}$
- A simple local planner to try to connect to the new sample
- These functions are used to build up a graph or tree representing feasible motions of the robot

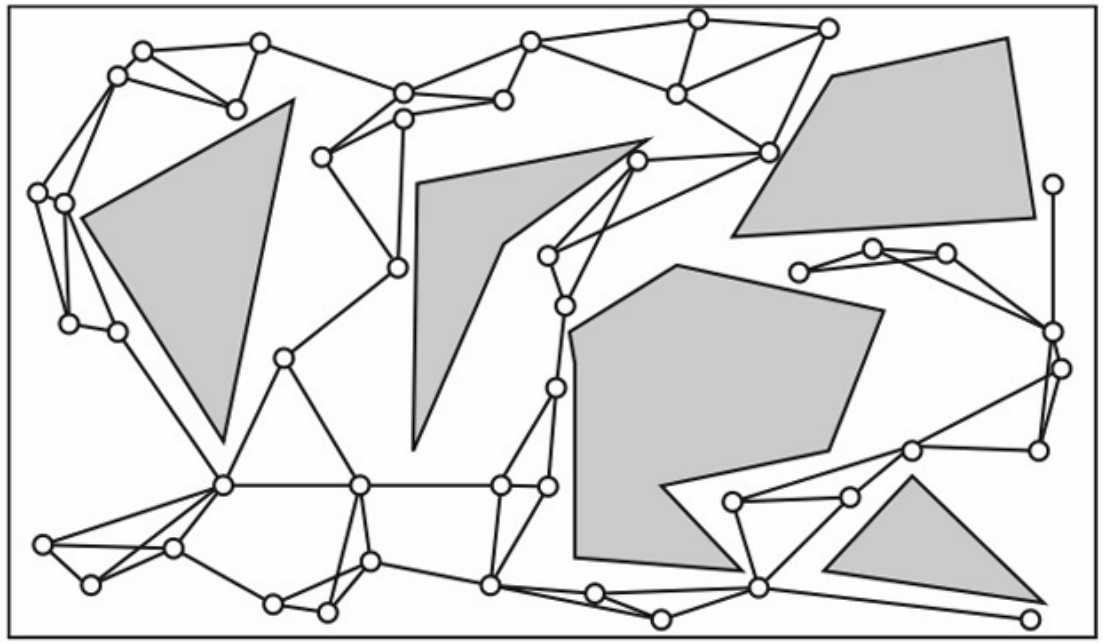
Sampling methods

- Compared to grid search methods, sampling methods can quickly find *satisficing* solutions in high-dimensional spaces
  - Fewer nodes will be used
  - Node number of grid points increases exponentially with the dimension of the space
- Most sampling methods are probabilistically complete
  - The probability of finding a solution, when one exists, approaches 100% as the number of samples goes to infinity

Sampling Based Planners: Probabilistic Roadmaps (PRMs) Kavraki and Latombe, 1994

- **Idea:** build (offline) a graph (i.e., the roadmap) representing the "connectivity" of the environment

First planner ever to demonstrate the ability to solve general planning problems in > 4-5 dimensions!



Rapidly Exploring Random Trees (RRT)

Algorithm 10.3 RRT algorithm.

1: initialize search tree T with x_{start}
2: **while** T is less than the maximum tree size **do**
3: $x_{\text{samp}} \leftarrow$ sample from $\mathcal{X}$
4: $x_{\text{nearest}} \leftarrow$ nearest node in T to x_{samp}
5: employ a local planner to find a motion from x_{nearest} to x_{new} in the direction of x_{samp}
6: **if** the motion is collision-free **then**
7: add x_{new} to T with an edge from x_{nearest} to x_{new}
8: **if** x_{new} is in $\mathcal{X}_{\text{goal}}$ **then**
9: **return** SUCCESS and the motion to x_{new}
10: **end if**
11: **end if**
12: **end while**
13: **return** FAILURE

https://github.com/smitdumore/f110-rrt

Spatial Forces as Wrenches

Twists were introduced as a compact way to capture relevant velocity information (linear and angular)

→ Can we do the same with forces?

Consider a frame $\{a\}$ with a linear force $f_a \in \mathbb{R}^3$ acting on a rigid body at point $p_a \in \mathbb{R}^3$

This force creates a **torque** or a **moment** $m_a \in \mathbb{R}^3$

$$m_a = p_a \times f_a$$

All this information for a spatial force can be represented as a **wrench:**

$$\mathcal{F}_a = \begin{bmatrix} m_a \\ f_a \end{bmatrix} \in \mathbb{R}^6$$

Got more than one wrench?
The total wrench on the body is the vector sum of the individual wrenches (in the same frame).

Contact Dynamics

Write distance between two points of contact:

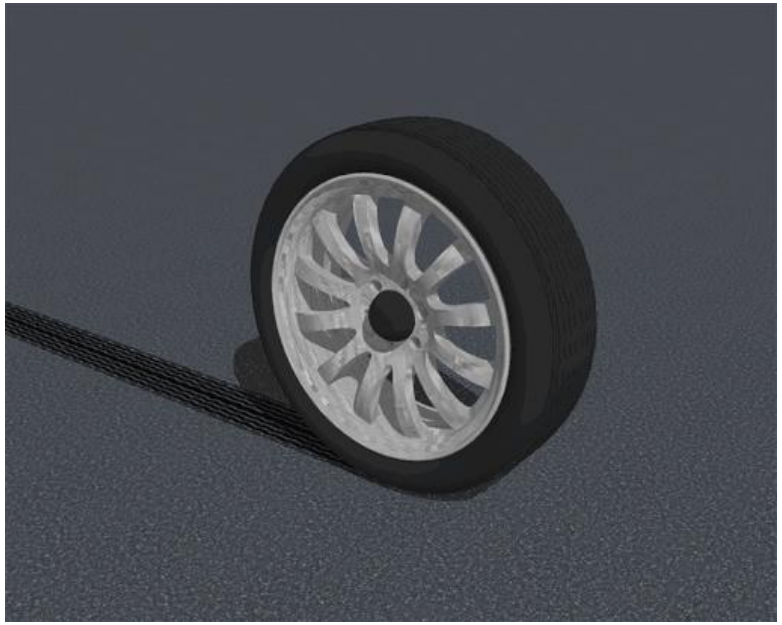
$$d(q = (q_1, q_2))$$

Consider a first order analysis:

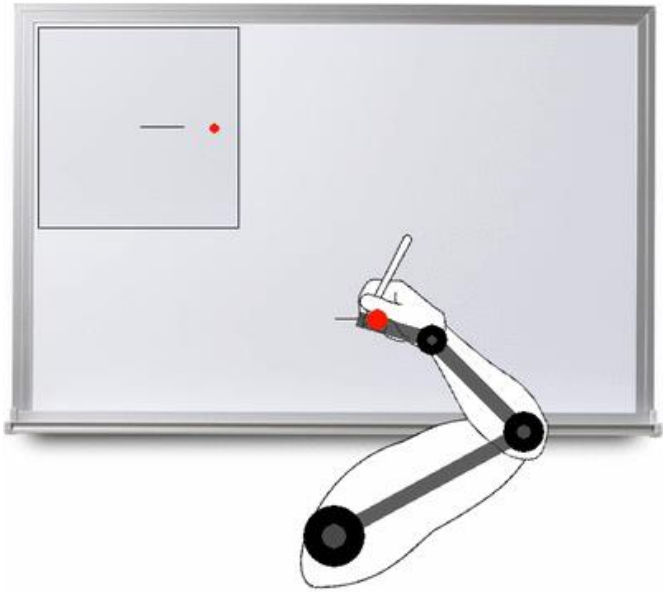
d	$\dot{d}$	situation
> 0		No contact
< 0		Infeasible
$= 0$	> 0	Contact, but **breaking free**
$= 0$	< 0	Contact, but infeasible
$= 0$	$= 0$	In contact

Contact is maintained iff all time derivatives are 0

Examples of Motions



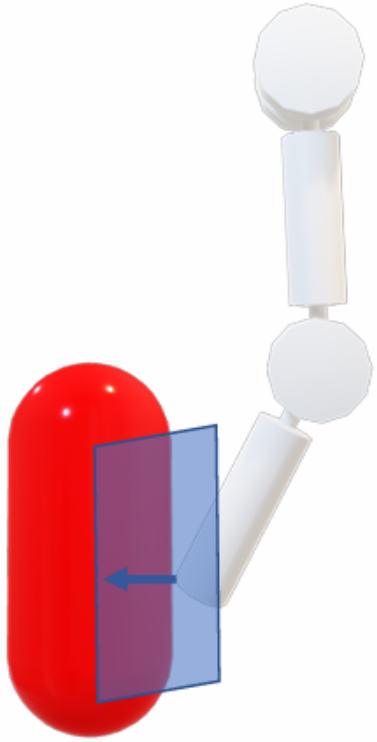
Roll



Slide

Roll-Slide

Contact Dynamics



Write distance between two points of contact:

$$d(q = (q_1, q_2))$$

Consider a first order analysis:

$$\dot{d}(q) = \frac{\partial d}{\partial q}\dot{q}$$

This derivative gives the **contact normal** $\hat{n}$

$\frac{\partial d}{\partial q}$ carries information on the separation direction, or contact normal

Contact Normal

- For contact, we write the velocity condition as a constraint:

$$\dot{d}(q) = \frac{\partial d}{\partial q}\dot{q} \geq 0$$

- We can rewrite this equation as follows:

$$\hat{n}^{\top}(\dot{p}_A - \dot{p}_B) \geq 0$$

where p_A is a point on body A and p_B is a point on body B

$\hat{n}$ is the unit contact normal vector

p_A and p_B are equal, but $\dot{p_A}$ and $\dot{p_B}$ might be different

Back to Twists!

- What is $\dot{p}_A$? Recall twist derivation from previous lectures:

$$\dot{p}_A = v_A + \omega_A \times p_A = v_A + [\omega_A] p_A$$
$$\dot{p}_B = v_B + \omega_B \times p_B = v_B + [\omega_B] p_B$$

- Can we bring in a wrench $\mathcal{F} = (m, f)$?
- If we apply a unit force to along the contact normal:

$$\mathcal{F} = \begin{bmatrix} p_A \times \hat{n} \\ \hat{n} \end{bmatrix} = \begin{bmatrix} [p_A] \hat{n} \\ \hat{n} \end{bmatrix}$$

Contact Constraints with Twists and Wrenches

Rewrite: $\hat{n}^{\mathsf{T}}(\dot{p}_A - \dot{p}_B)$ as two constraints:

Impenetrability constraint: $\mathcal{F}^{\mathsf{T}}(\mathcal{V}_A - \mathcal{V}_B) \geq 0$

Active contact: $\mathcal{F}^{\mathsf{T}}(\mathcal{V}_A - \mathcal{V}_B) = 0$

If body B is a static object, we drop $\mathcal{V}_B$

Roll-slide contacts satisfy the active contact constraint

- Rolling contacts: no motion relative to each other $\dot{p}_A = \dot{p}_B$
- Sliding contacts: all other satisfying solutions

Derive the symbolic expression for a roll-slide constraint

Consider two bodies A and B in contact at point p. The contact normal direction is n.

We know that the impenetrability constraint tells us that:

$$\mathcal{F}^T (\mathcal{V}_A - \mathcal{V}_B) \geq 0$$

Find the symbolic expression c such that $c = 0$ represents valid roll-slide twists.

For example, ω_{A_x}, ω_{B_y}, and v_{A_z} can be referenced symbolically as wax, wby, and vaz, respectively.

Click on the question mark next to the answer input for a list of available variables and operators.

matlab | python

```
import numpy as np

p = np.array([[ 1, -6, -9]])
n = np.array([[0, 0, 1]])
```

Rewriting our active constraint:

$$\mathcal{F}^T(V_A - V_B) = 0$$

$$[(p \times \hat{n})^T \quad \hat{n}^T] \begin{bmatrix} \omega_A - \omega_B \\ v_A - v_B \end{bmatrix} = [(p \times \hat{n})^T \quad \hat{n}^T] \begin{bmatrix} \omega_{Ax} - \omega_{Bx} \\ \omega_{Ay} - \omega_{By} \\ \omega_{Az} - \omega_{Bz} \\ v_{Ax} - v_{Bx} \\ v_{Ay} - v_{By} \\ v_{Az} - v_{Bz} \end{bmatrix} = 0$$

Plugging in values for p and $\hat{n}$ yields:

$$v_{Az} - v_{Bz} - 6\omega_{Ax} - \omega_{Ay} + 6\omega_{Bx} + \omega_{By} = 0$$

We like working with planar systems because the space of twists is 3D, and easier to visualize

A planar twist is represented by:

$$\mathcal{V} = \begin{bmatrix} \omega_z \\ v_x \\ v_y \end{bmatrix}$$

The **CoR** is the point in the projective plane that **stays stationary under the motion**

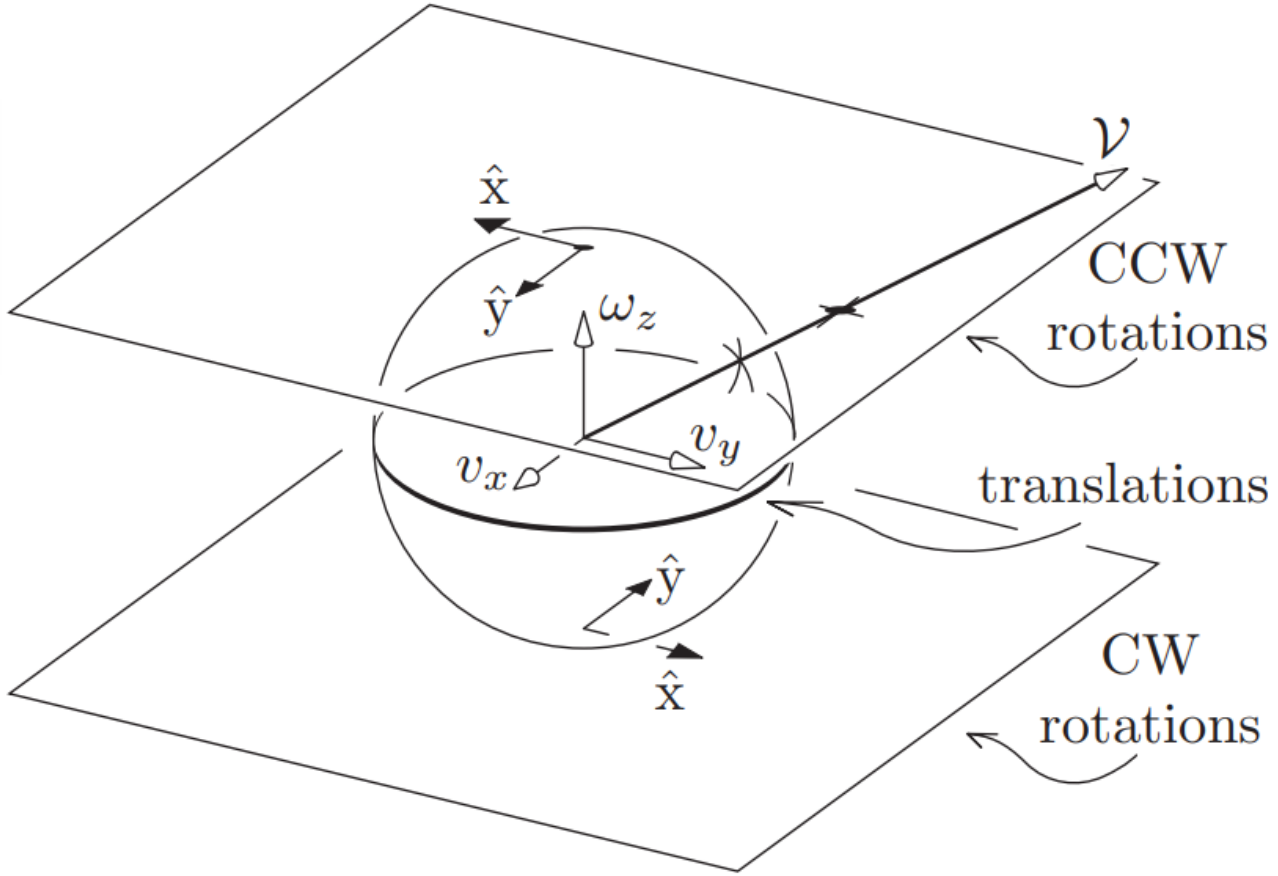
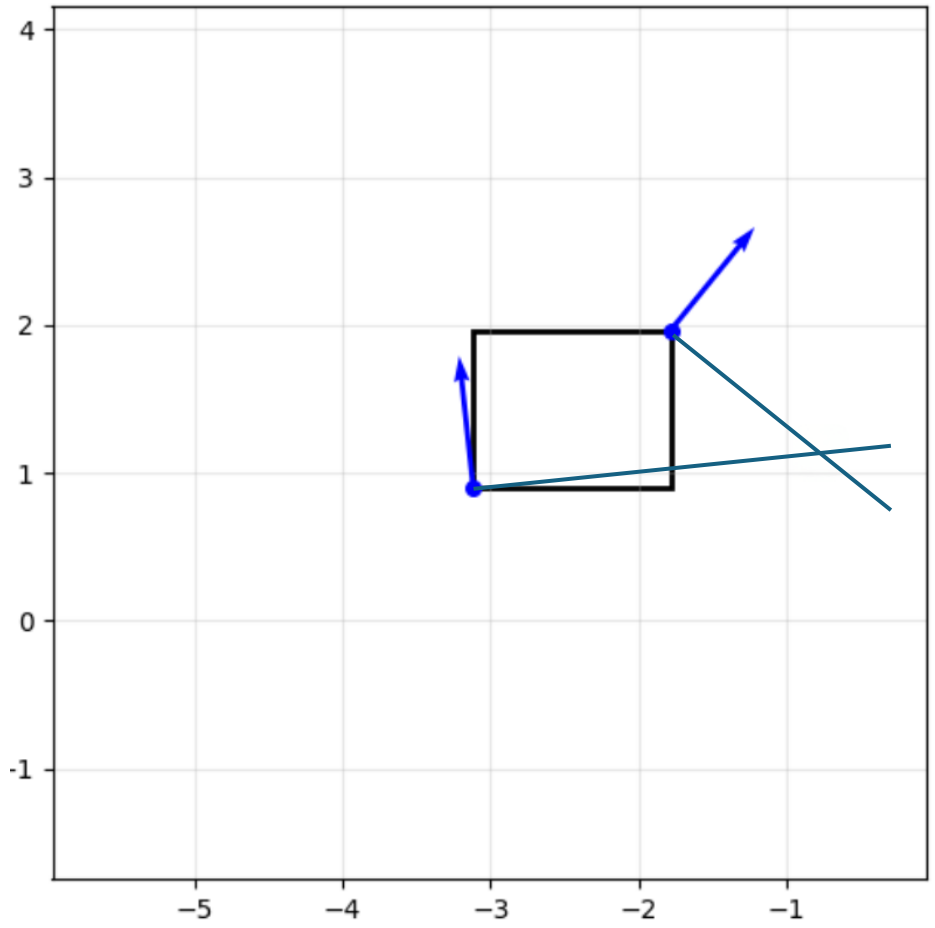


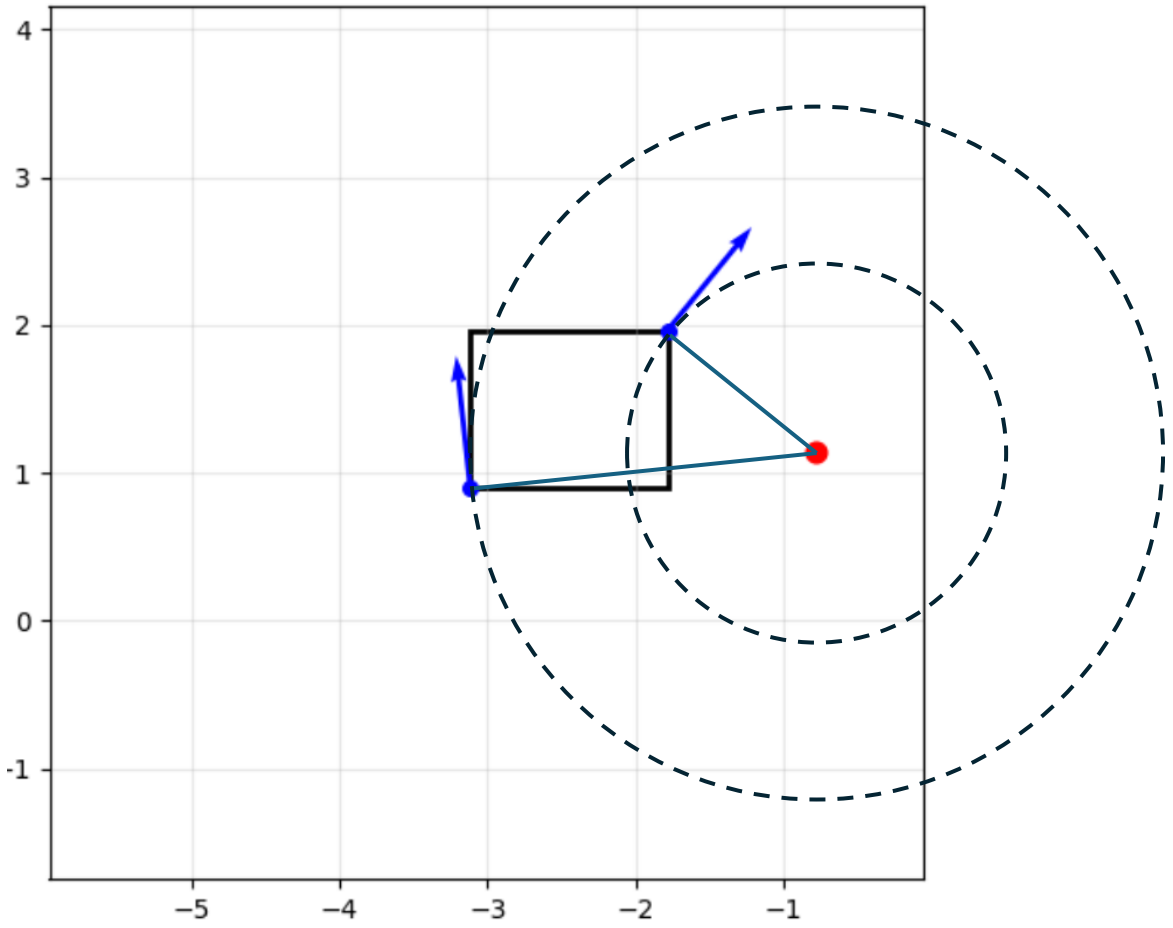
Figure 12.8: Mapping a planar twist $\mathcal{V}$ to a CoR. The ray containing the vector $\mathcal{V}$ intersects the plane of '+' CoRs at $\omega_z = 1$, or the plane of '−' CoRs at $\omega_z = -1$, or the circle of translation directions.

Finding the center of rotation of a body

View...

A rectangular planar body is shown with velocity vectors drawn on two of its corners. Select the picture with the correct corresponding center of rotation drawn as a red point.





Multiple points of contact

- The possible twists of a body A with multiple contacts, can be written:

$$V = \{\mathcal{V}_A \mid \mathcal{F}_i^\top (\mathcal{V}_A - \mathcal{V}_i) \geq 0 \; \forall i\}$$

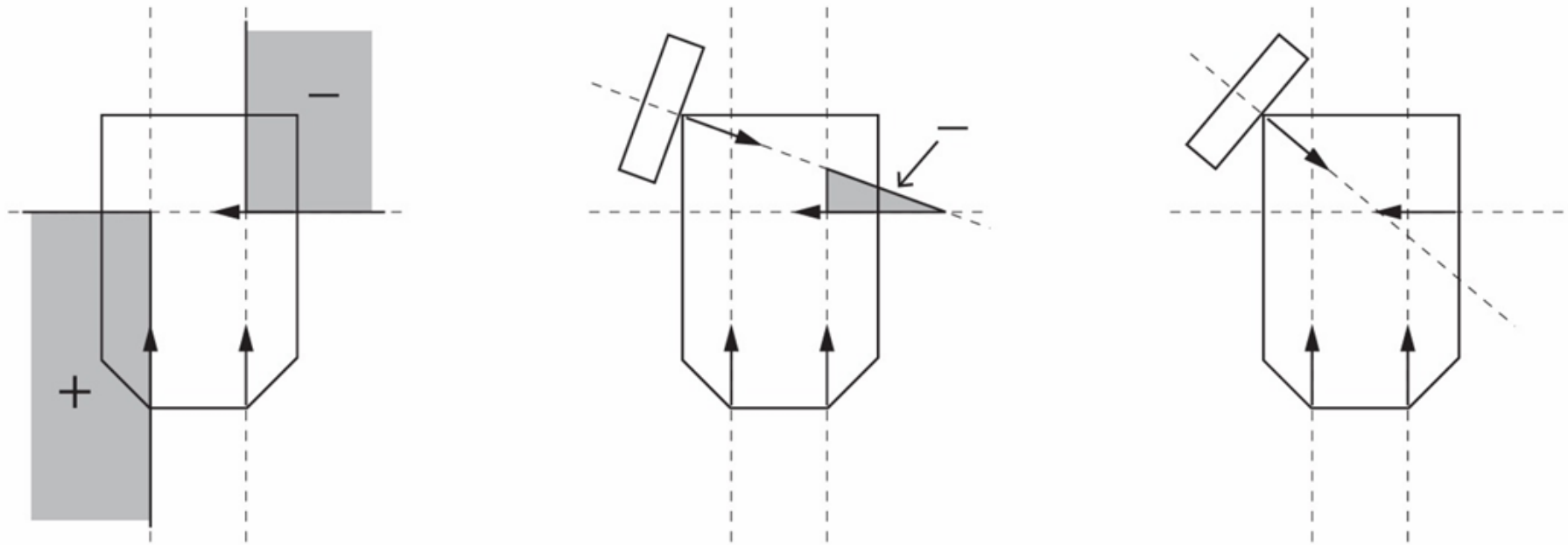
Suppose our figures are not moving ($\mathcal{V}_i = 0$)

The above set becomes a polyhedral convex cone:

$$V = \{\mathcal{V}_A \mid \mathcal{F}_i^\top \mathcal{V}_A \geq 0 \; \forall i\}$$

- If the $\mathcal{F}_i$ positively span the wrench space or the convex span of the $\mathcal{F}_i$ contains the origin then the contacts **constrain the motion of the body**
- We call this **form closure**

What if we want no motion?



Form Closure Grasps

- **Form closure** of a body is when a set of stationary constraints prevent *all* motions
- Recall our stationary contact conditions:

$$\mathcal{F}^{\mathsf{T}}\mathcal{V} \geq 0$$

- If the only twist $\mathcal{V}$ that satisfies this constraint is a zero vector, we are in form closure. Said in different way:

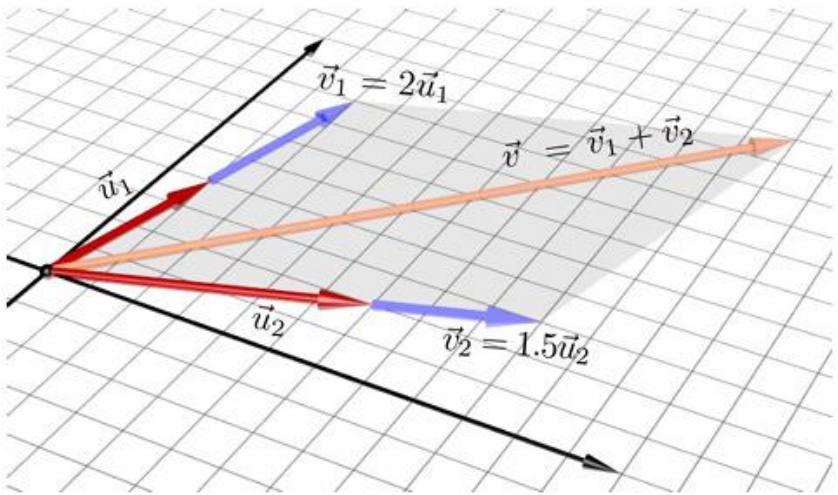
$$\text{positive span}\left(\left\{\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_j\right\}\right) = \mathbb{R}^6$$

Positive Span?

- The positive span of vectors is defined as:

$$\text{positive span}(\mathcal{A}) = \left\{ \sum_{i=1}^{j} k_i a_i \mid k_i \geq 0 \right\}$$

- The space $\mathbb{R}^n$ can be spanned by $n + 1$ vectors, but no fewer



Contacts needed for Form Closure

- For a planar body, at least four point contacts are needed for first-order form closure
- For a spatial body, at least seven point contacts are needed for first-order form closure
- Fine print:
  - First order analysis is a simplified estimate! If a body is in form closure by first-order analysis, it is in form closure for higher-order analysis. If it is not in form closure, it could be with higher-order reasoning.
  - **Exceptional objects** are objects that can always spin about their center, for any number of contacts

Form Closure Checker -- Linear Program

We can formulate this as a linear optimization program:

$$\begin{aligned} \underset{k}{\text{minimize}} \quad & \mathbf{1}^\top k \\ \text{subject to} \quad & Fk = 0 \\ & k_i \geq 1, i = 1, \ldots, j \end{aligned}$$

What's up with k_i?

Linear program solvers need to take in constraints in a particular form: $Ax \leq b$, so we re-write our constraints:

$$-\mathbf{I}k \leq -\mathbf{1}$$

For solving this, use a linear program solver:
numpy `np.linalg.solve`, scipy `scipy.optimize.linprog`, or matlab `linprog`