Robotics

Miao Li

Fall 2023, Wuhan University

WeChat: 15527576906

Email: limiao712@gmail.com

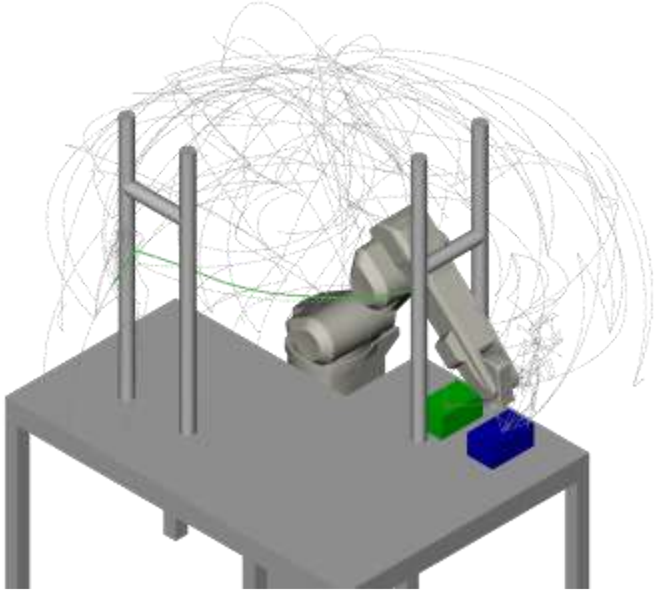
2023-10-9

Goal for this course

- **Design: soft hand design x1**
- **Perception: vision, point cloud, tactile, force/torque x1**
- **Planning: sampling-based, optimization-based, learning-based x3**
- **Control: feedback, multi-modal x2**
- **Learning: imitation learning, RL x2**
- **Simulation tool (pybullet, matlab, OpenRAVE, Issac Nvidia, Gazebo)**
- **How to get a robot moving!**

Today’s Agenda

- **Drawback of Sampling-based approach (~5)**
- **Potential field method (~15)**
  - **attractive, repulsive**
- **Gradient descent algorithm (~10)**
  - **vector field, velocity field, dynamic system**
- **Trajectory planning(~25)**
  - **Parameter, joint space, cartesian space**
- **Planning as optimization (~20)**
  - **Parameter, joint space, cartesian space**

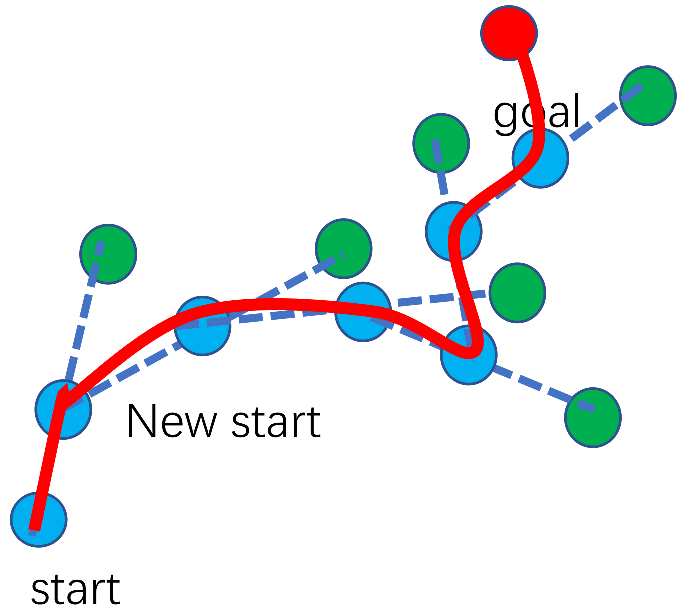


RRT Revisit

RRT Algorithm (x_{start}, x_{goal}, step, n)

G.initialize(x_{start})
for $i = 1$ **to** n **do**
  x_{rand} = Sample()
  x_{near} = near(x_{rand}, G)
  x_{new} = steer(x_{rand}, x_{near}, step_size)
  G.add_node(x_{new})
  G.add_edge(x_{new}, x_{near})
  if $x_{new} = x_{goal}$
    success()

S

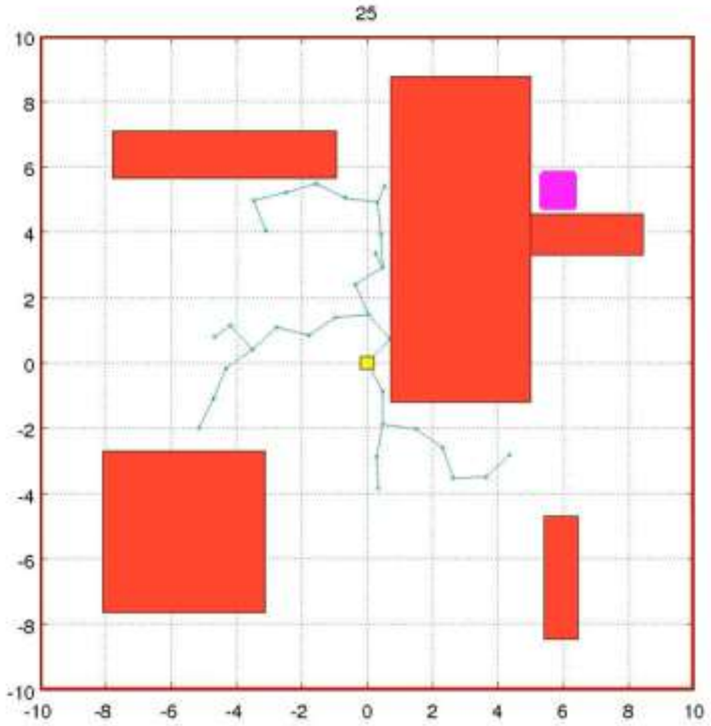


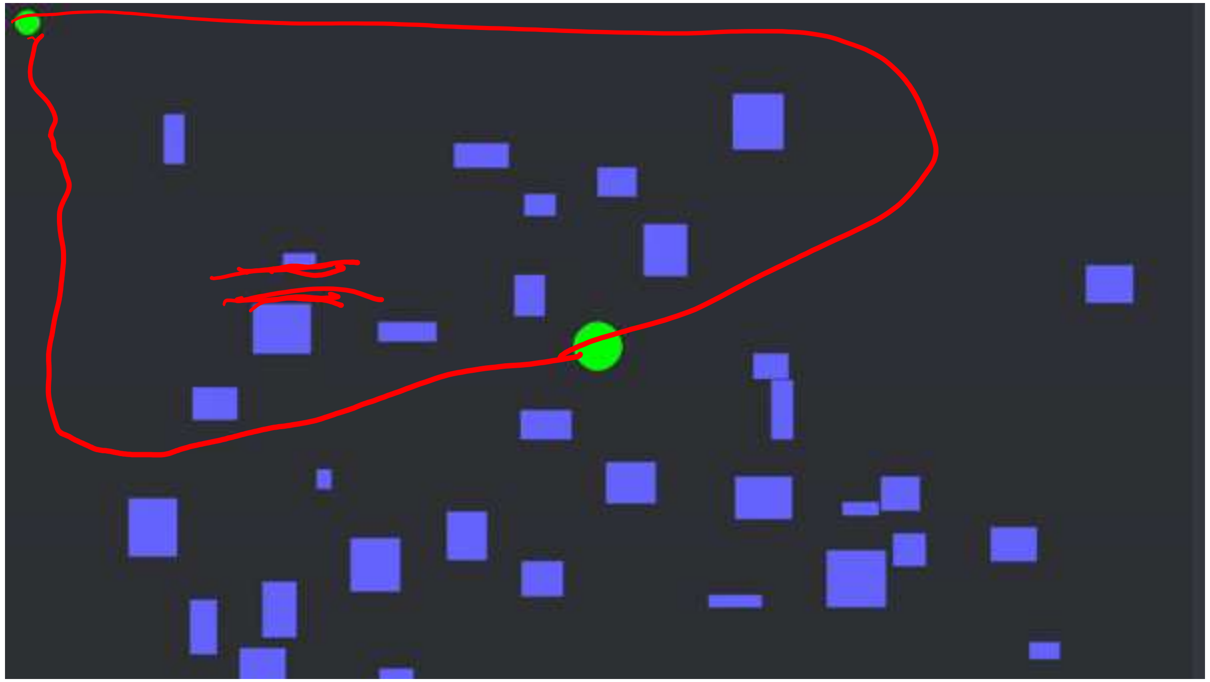
path

smoothing

– J-C. Latombe. Robot Motion Planning. Kluwer. 1991.
– S. Lavalle. Planning Algorithms. 2006. http://msl.cs.uiuc.edu/planning/
– H. Choset et al., Principles of Robot Motion: Theory, Algorithms, and Implementations. 2006.

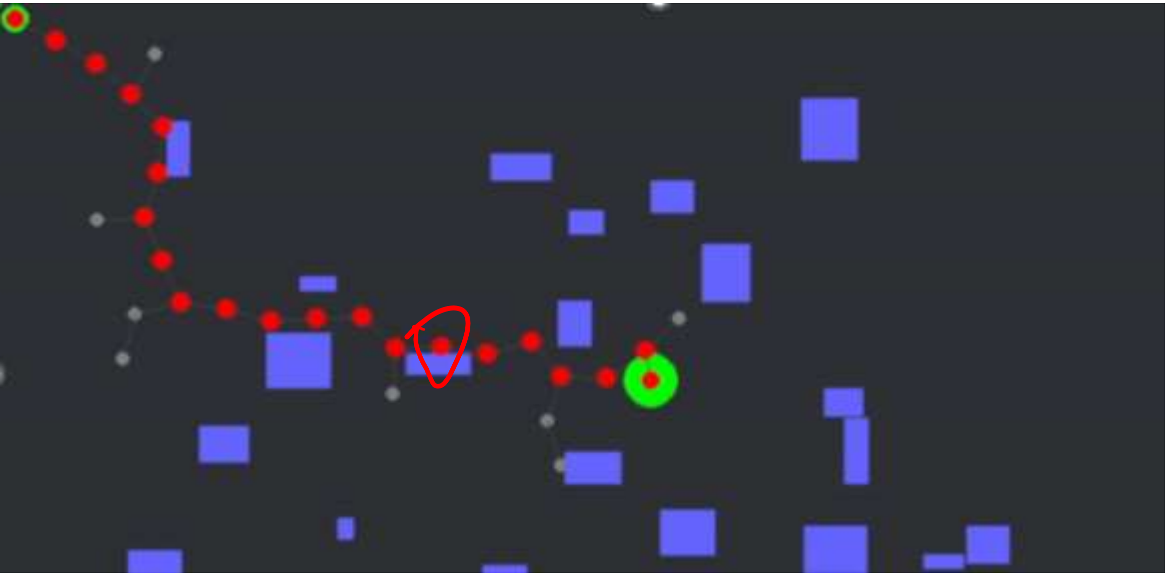
RRT revisit





What is the problem with this approach?

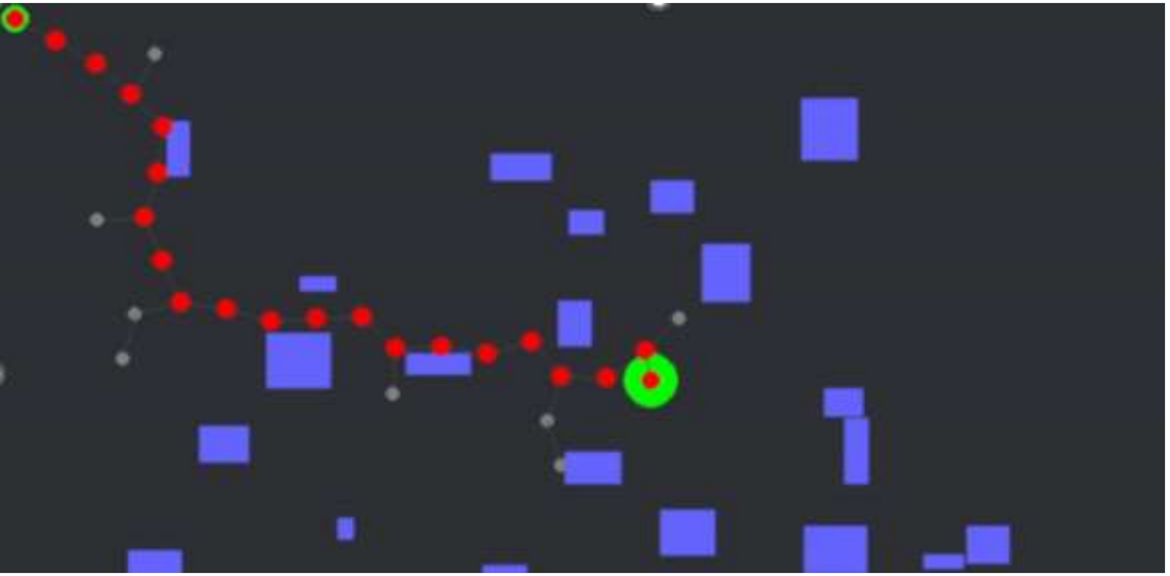
RRT revisit



- **Few control params of the solution**
- **Near to collisions**
- **Ignore trivial solution**
- **Path quality can be bad**
- **Quite different with different seeds**
- **Additional steps for collision checking**

What is the problem with this approach?

RRT revisit



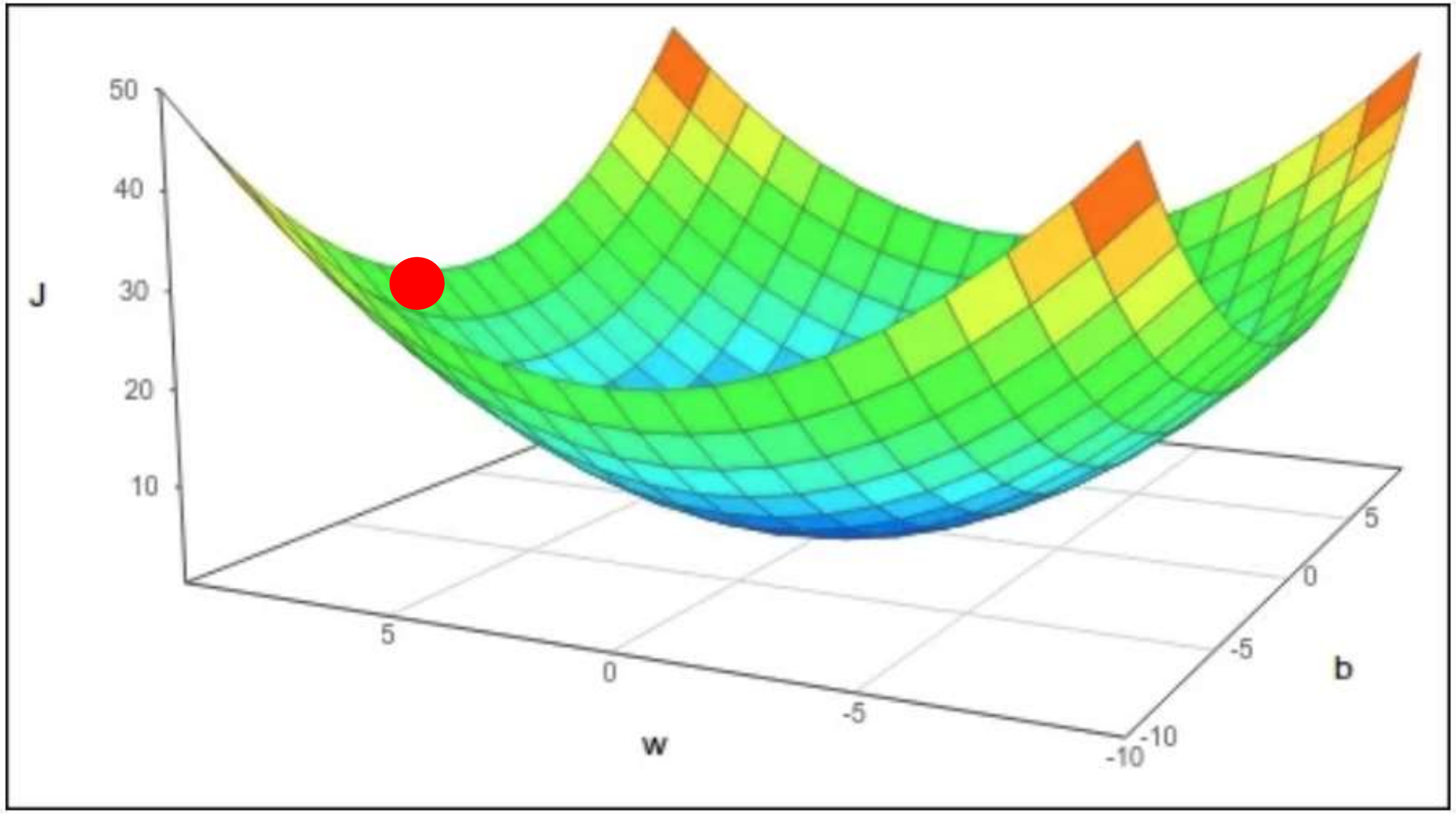
RRT is not optimal

What is the problem with this approach?

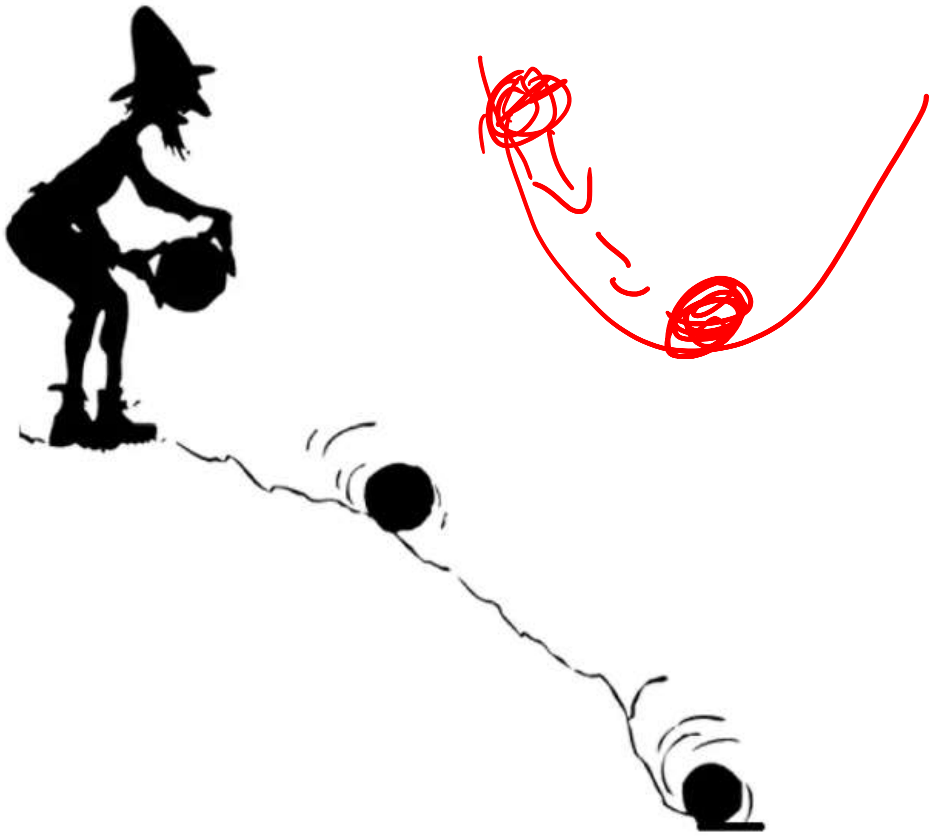
Motion planning as optimization

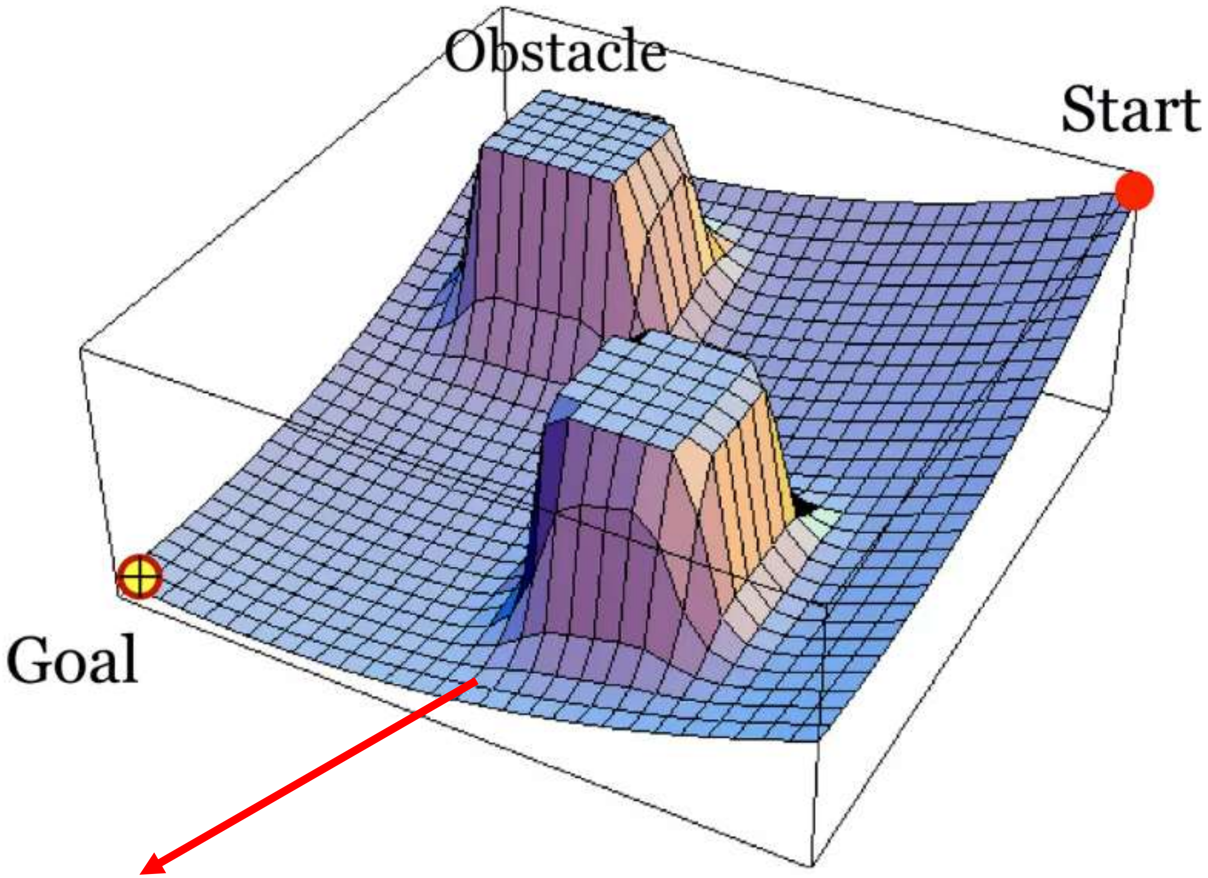
Can we develop a motion planner that relies on cost function instead?

Cost function in 2D



Potential field method

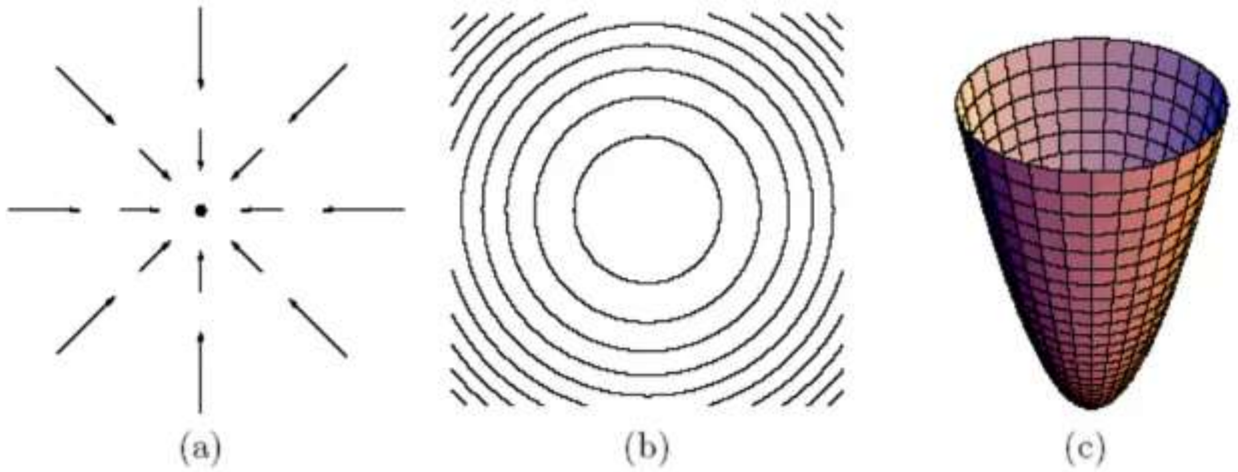




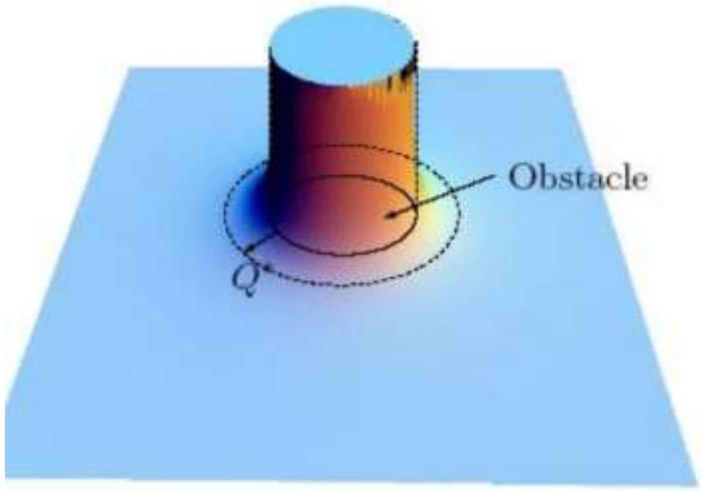
Can we create such a cost function?

Potential field method

Attraction



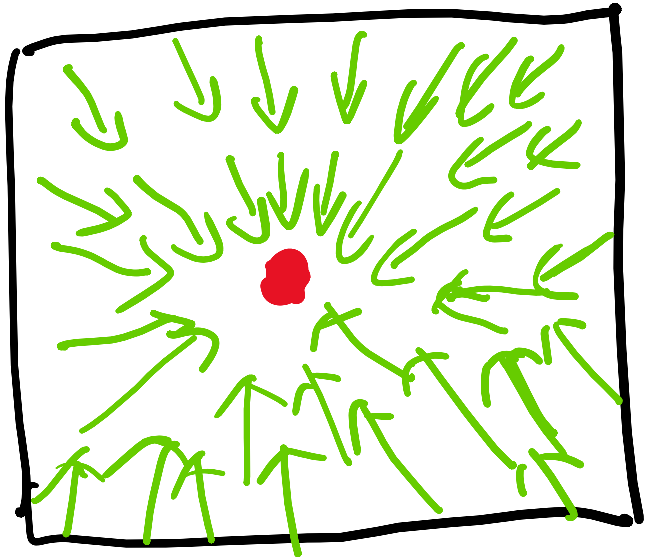
Repulsion



Minimize the cost function

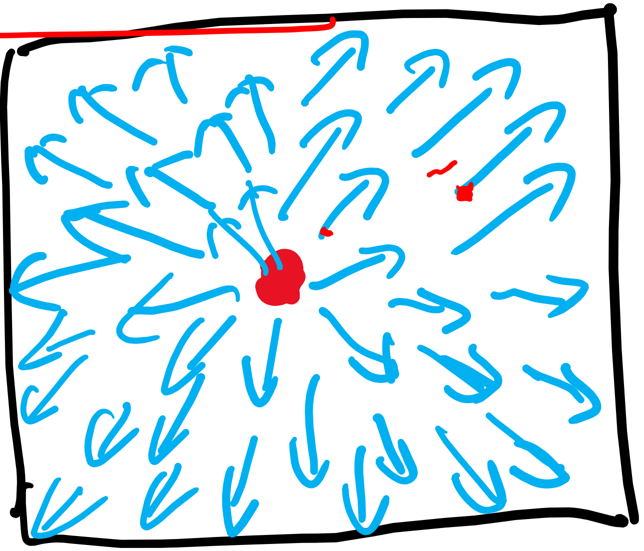
Potential field method

Attraction



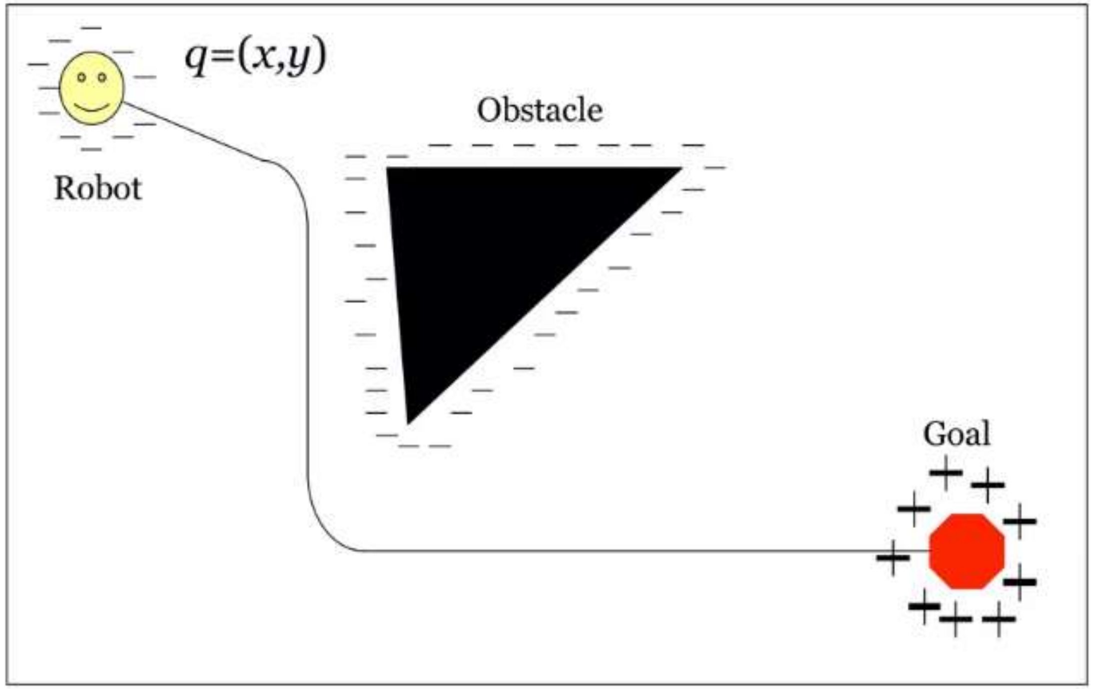
Repulsion

vector field



Gradient

Cost function as potential



differential potential:

$U(q)$

artificial force:

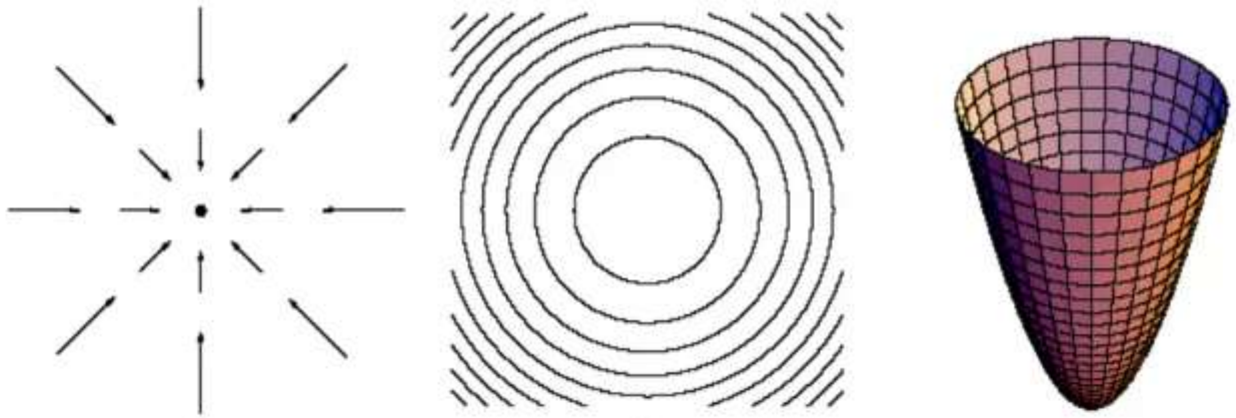
$F(q) = -\nabla U(q)$

gradient

$$\nabla U(q) = \begin{bmatrix} \frac{\partial U(q)}{\partial x} \\ \frac{\partial U(q)}{\partial y} \end{bmatrix}$$

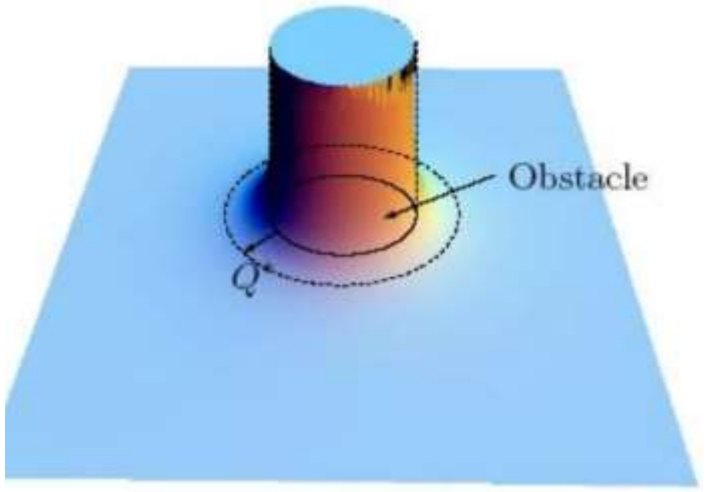
Potential field method

Attraction


(a) (b) (c)

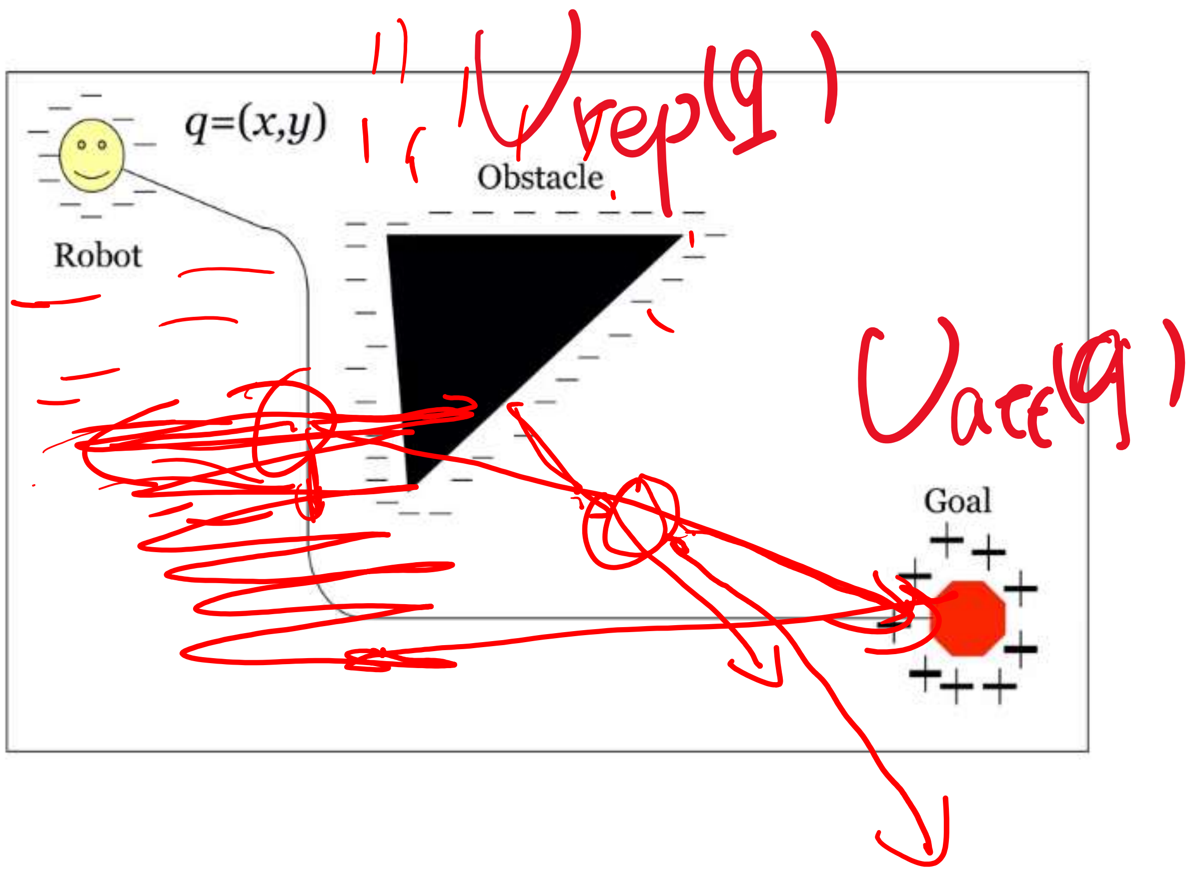
$U_{att}(q)$

Repulsion



$U_{rep}(q)$

Potential field method



$$U(q) = U_{att}(q) + U_{rep}(q)$$

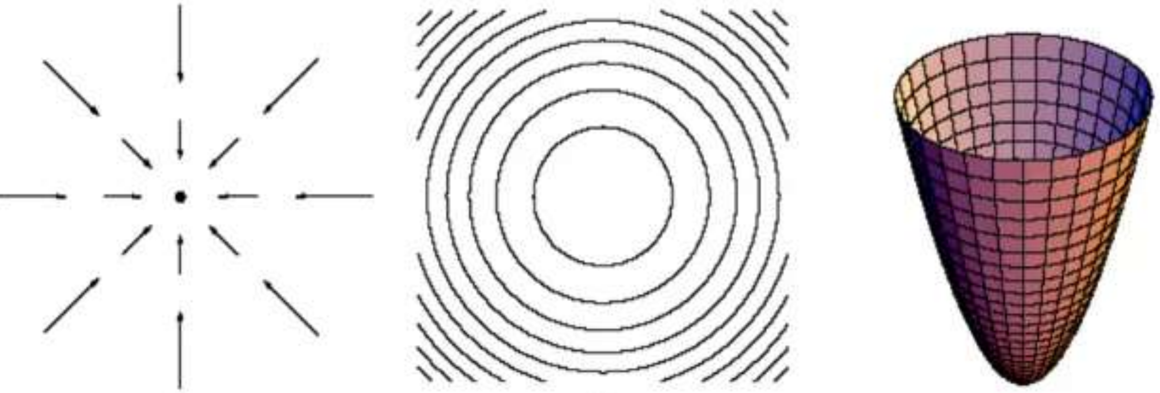
$U_{att}(q) \rightarrow$ move to the goal

$U_{rep}(q) \rightarrow$ avoid obstacles.

Potential field method

attractive potential.

examples:



(a) (b) (c)

quadratic potential:

$$U_{att}(\underline{q}) = \frac{1}{2} k_{att} \, d^2_{goal}(\underline{q})$$

$k_{att} \in R^+$, positive scaling param

$$d_{goal} = \| \underline{q} - \underline{q}_{goal} \|$$

Potential field method

attractive potential

★ differentiable

$$U_{att}(q) = \frac{1}{2} K_{att}\, d_{goal}^2(q)$$

force

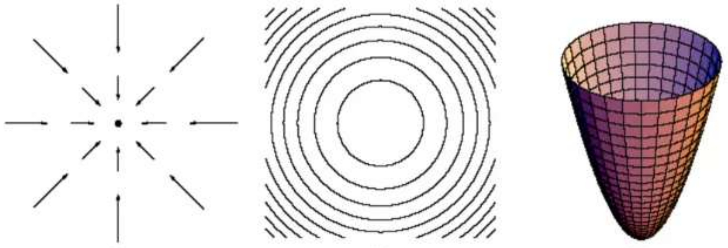
★ $F_{att}(q) = -\nabla U_{att}(q)$

$= -K_{att}\, d_{goal} \nabla d_{goal}$

$= -K_{att}(q - q_{goal})$

converge linear towards the goal

$$d_{goal} = \|q - q_{goal}\|$$



(a) (b) (c)

Potential field method

repulsive potential.

key idea: generate a force away from all known obstacles

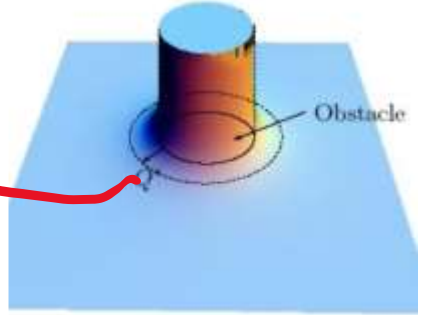
$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} \cdot \left(\frac{1}{d_{obj}(q)} - \frac{1}{Q^*} \right)^2 & d_{obj}(q) \le Q^* \\ 0 & d_{obj}(q) > Q^* \end{cases}$$

where

- k_{rep} is again a scaling factor,
- d_{obj} is the minimal distance from q to the object and
- Q^* is the distance of influence of the object.

① very strong: close

② zero: far away.



Potential field method

repulsive potential.

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} \cdot \left(\frac{1}{d_{obj}(q)} - \frac{1}{Q^*} \right)^2 & d_{obj}(q) \le Q^* \\ 0 & d_{obj}(q) > Q^* \end{cases}$$

where

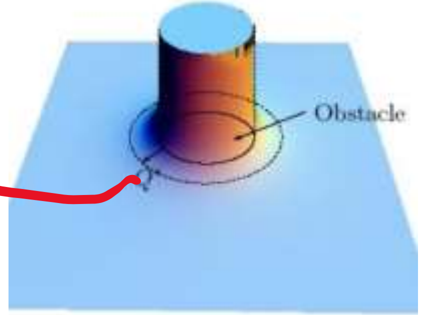
- k_{rep} is again a scaling factor,
- d_{obj} is the minimal distance from q to the object and
- Q^* is the distance of influence of the object.

$d_{obj}(q) \to 0$, $U_{rep} \to +\infty$

?

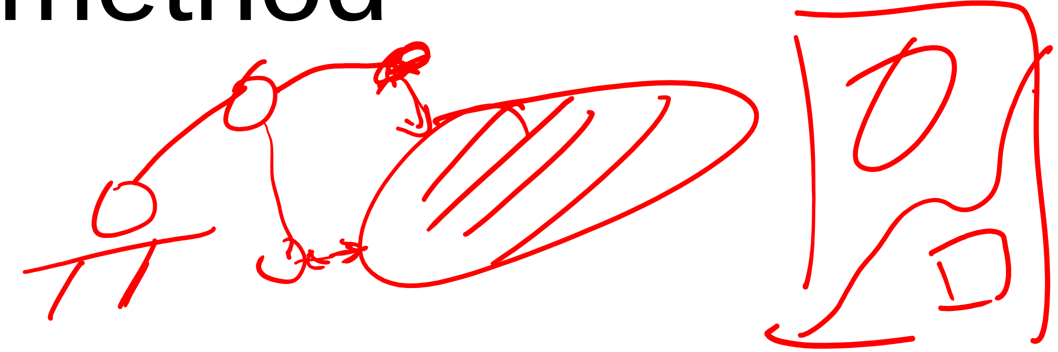
$d_{obj} = Q^*$, $U_{rep} \to 0$

$d_{obj} > Q^*$ $U_{rep} = 0$



Potential field method

repulsive force.



$$U_{rep}(q)=\begin{cases}\frac{1}{2}k_{rep}\cdot\left(\frac{1}{d_{obj}(q)}-\frac{1}{Q^*}\right)^2 & d_{obj}(q)\le Q^*\\ 0 & d_{obj}(q)>Q^*\end{cases} \Rightarrow F_{rep}(q)=-\nabla U_{rep}(q)=\begin{cases}k_{rep}\cdot\left(\frac{1}{d_{obj}(q)}-\frac{1}{Q^*}\right)\cdot\frac{1}{d_{obj}^2}.\nabla d_{obj} & d_{obj}(q)\le Q^*\\ 0 & d_{obj}(q)>Q^*\end{cases}$$

where

- ◇ k_{rep} is again a scaling factor,
- ◇ d_{obj} is the minimal distance from q to the object and
- ◇ Q^* is the distance of influence of the object.

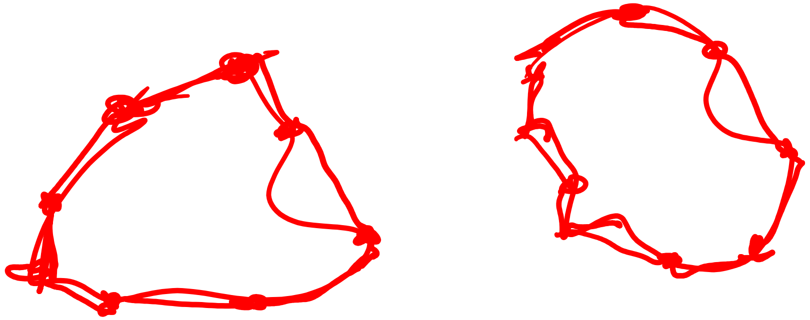
$$\nabla d_{obj}=\begin{bmatrix}\frac{\partial d_{obj}}{\partial x}\\ \frac{\partial d_{obj}}{\partial y}\end{bmatrix}$$

$F_{rep}(q) \nearrow$ when $d_{obj} \downarrow$

FCL mesh 1000 mesh 10000

Potential field method

repulsive force.



$$U_{rep}(q)=\begin{cases}\frac{1}{2}k_{rep}\cdot\left(\frac{1}{d_{obj}(q)}-\frac{1}{Q^*}\right)^2 & d_{obj}(q)\le Q^* \\ 0 & d_{obj}(q)>Q^*\end{cases} \Rightarrow F_{rep}(q)=-\nabla U_{rep}(q)=\begin{cases}k_{rep}\cdot\left(\frac{1}{d_{obj}(q)}-\frac{1}{Q^*}\right)\cdot\frac{1}{d_{obj}^2}.\nabla d_{obj} & d_{obj}(q)\le Q^* \\ 0 & d_{obj}(q)>Q^*\end{cases}$$

$\Downarrow$

How to compute d_{obj}?

geometry

where

- k_{rep} is again a scaling factor,
- d_{obj} is the minimal distance from q to the object and
- Q^* is the distance of influence of the object.

GJK

Potential field method

$$F(q) = F_{att}(q) + F_{rep}(q) = -\nabla U(q)$$

$-\nabla U(q)$

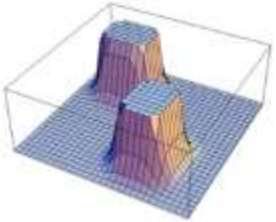
$\gamma \boxed{F(q)}$

$\gamma = 0.001$

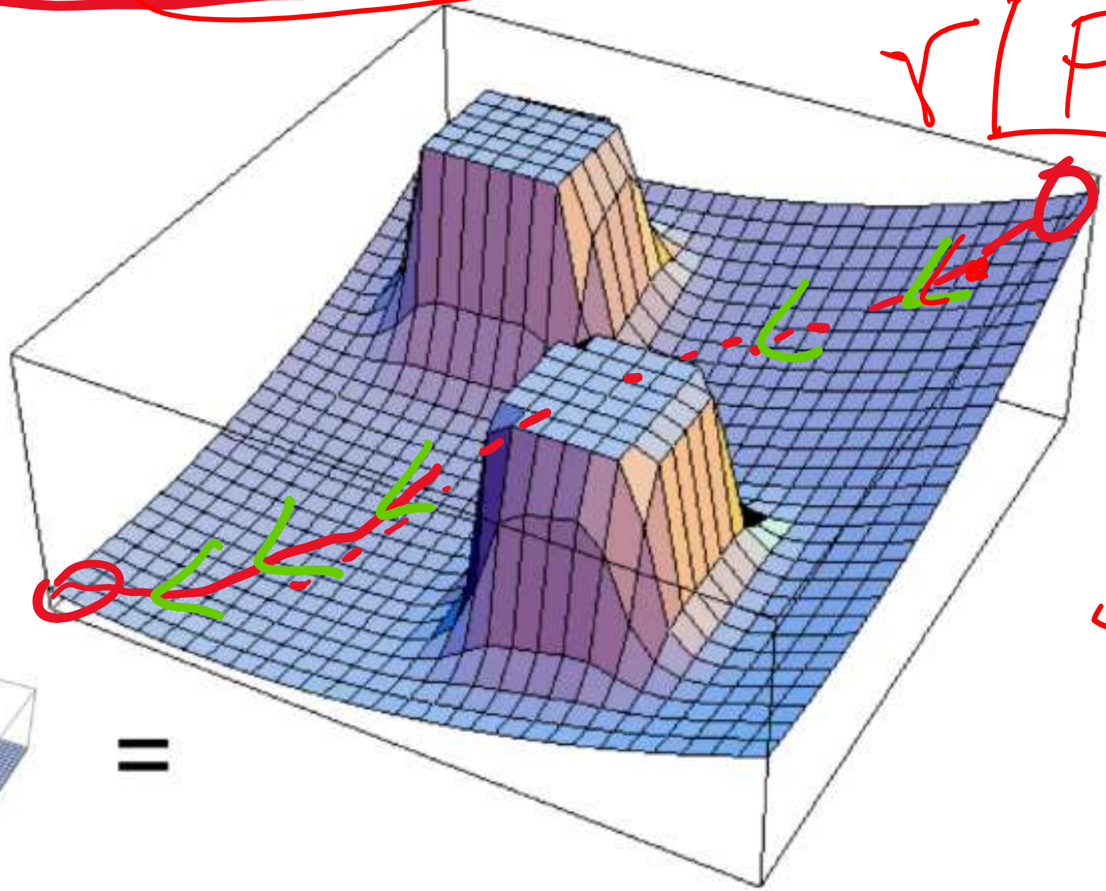
$\gamma = 100$

A first-order optimization algorithm such as **gradient descent** (also known as **steepest descent**) can be used to minimize this function by taking steps proportional to the negative of the gradient.

+



=



Potential field method

Gradient Descent or Steepest Descent

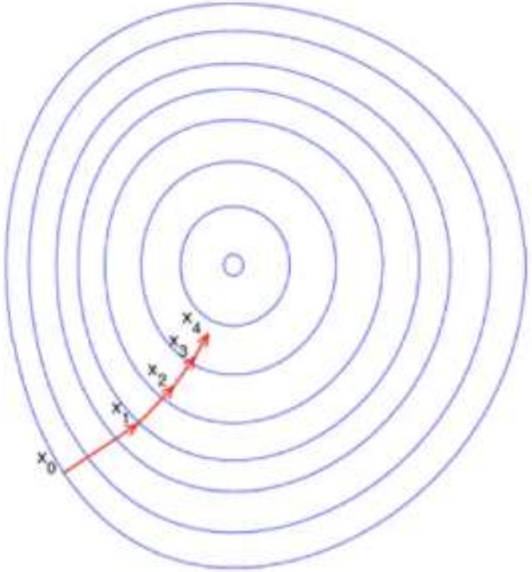
- ◇ Gradient descent is a first-order optimization algorithm.
- ◇ To find a local minimum of a function using gradient descent, one takes **steps proportional to the negative of the gradient** (or of the approximate gradient) of the function at the current point.

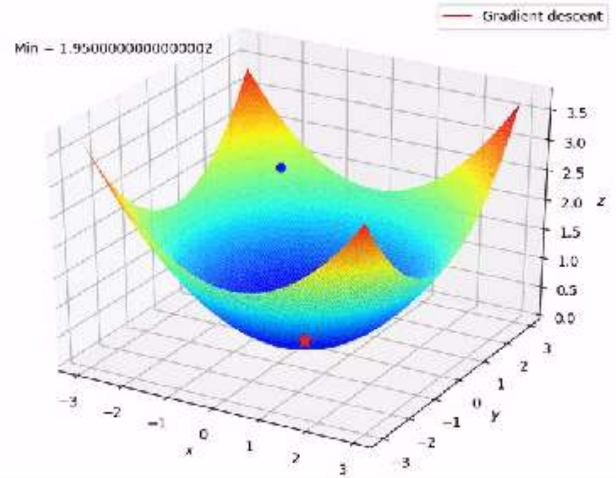
GradientDescent(x_{init}, x_{final}, $-\nabla f$)

while $x_{init} \neq x_{final}$

$$x_{n+1} = x_n - \gamma_n \nabla f(x_n), \quad n \geq 0$$

end





Potential field method

Gradient Descent or Steepest Descent

GradientDescent(x_o, x_{final}, - ∇f)

while $x_o \neq x_{final}$

$$x_{n+1} = x_n - \gamma_n \nabla f(x_n), \quad n \geq 0$$

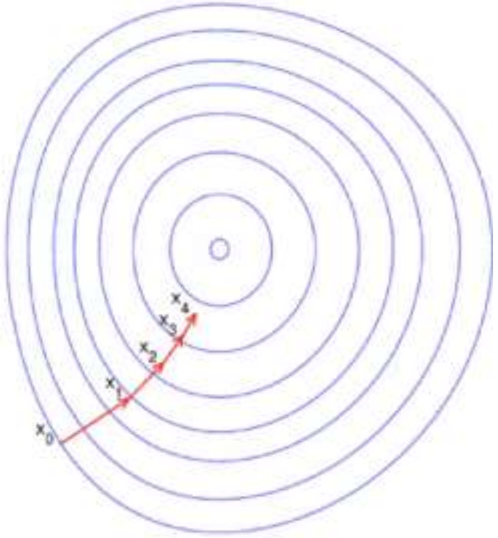
end

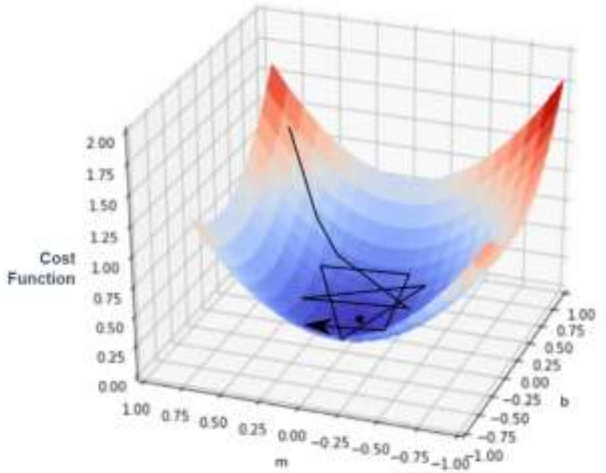
where

$\gamma > 0$ is a small enough number.

Note that the **step size** γ must be small enough to ensure that we do not collide with an obstacle or overshoot our goal position.

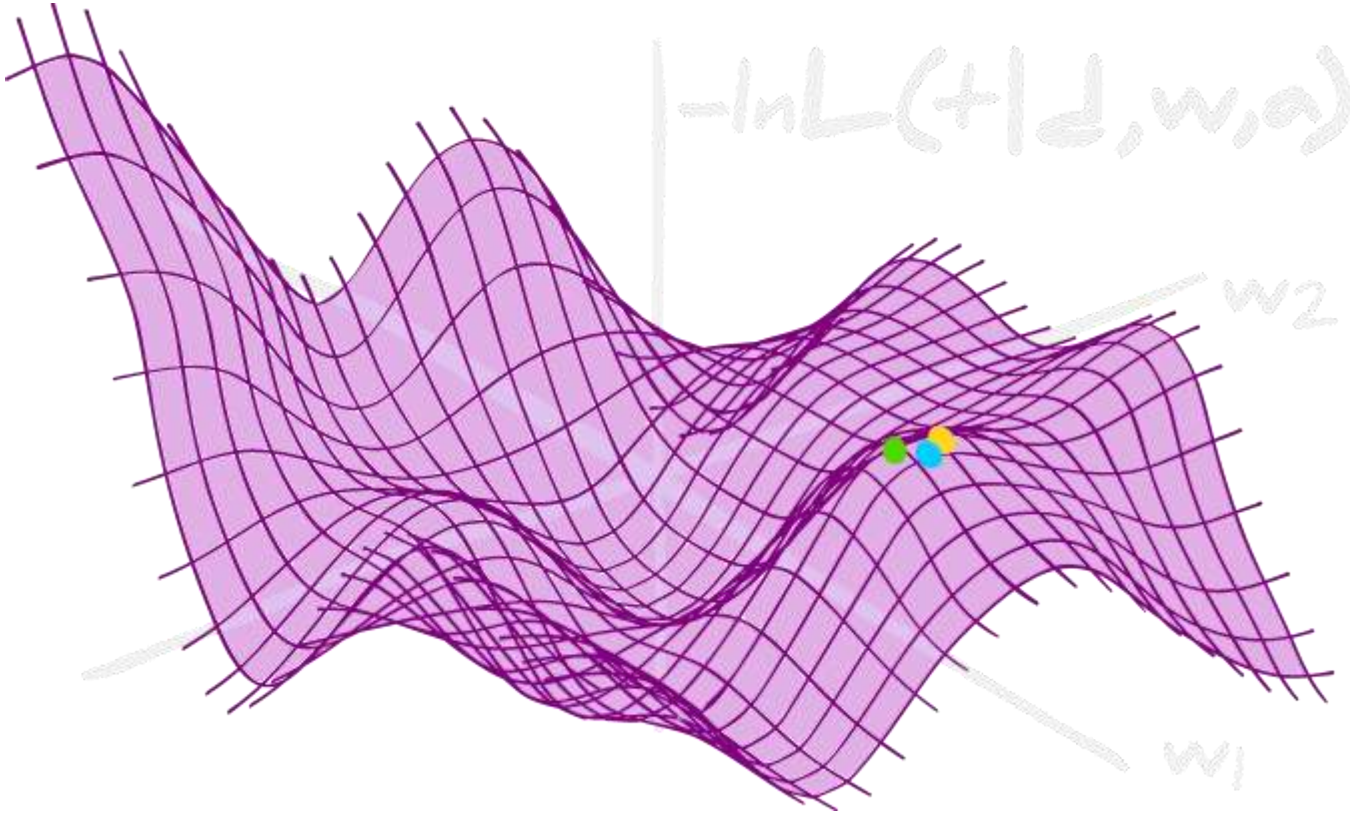
The value of the step size γ is allowed to change at every iteration.





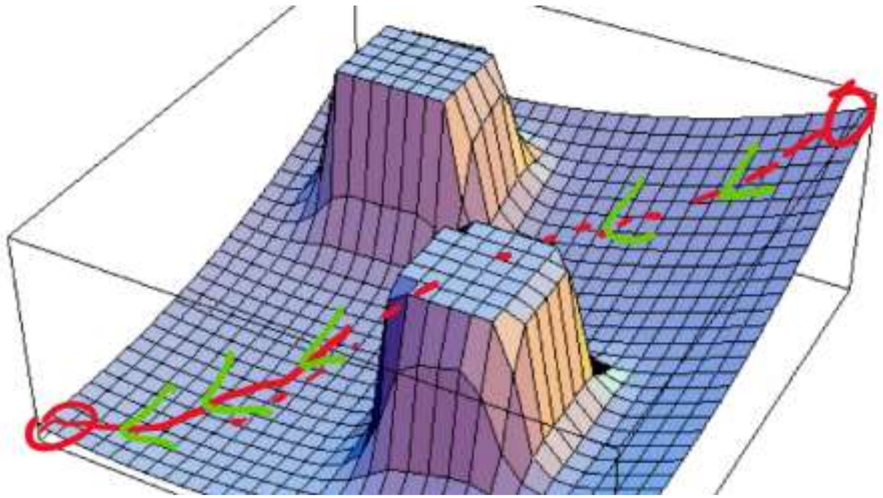
overshooting

Potential field method



gradient

Potential field method

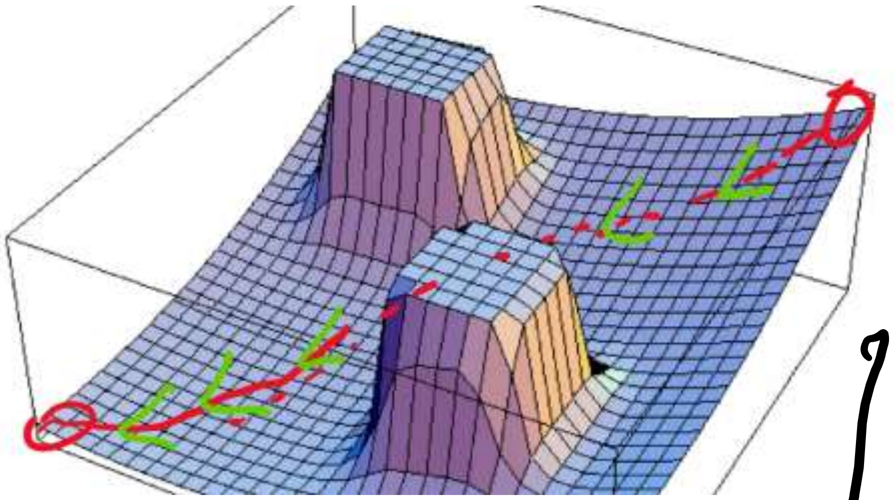


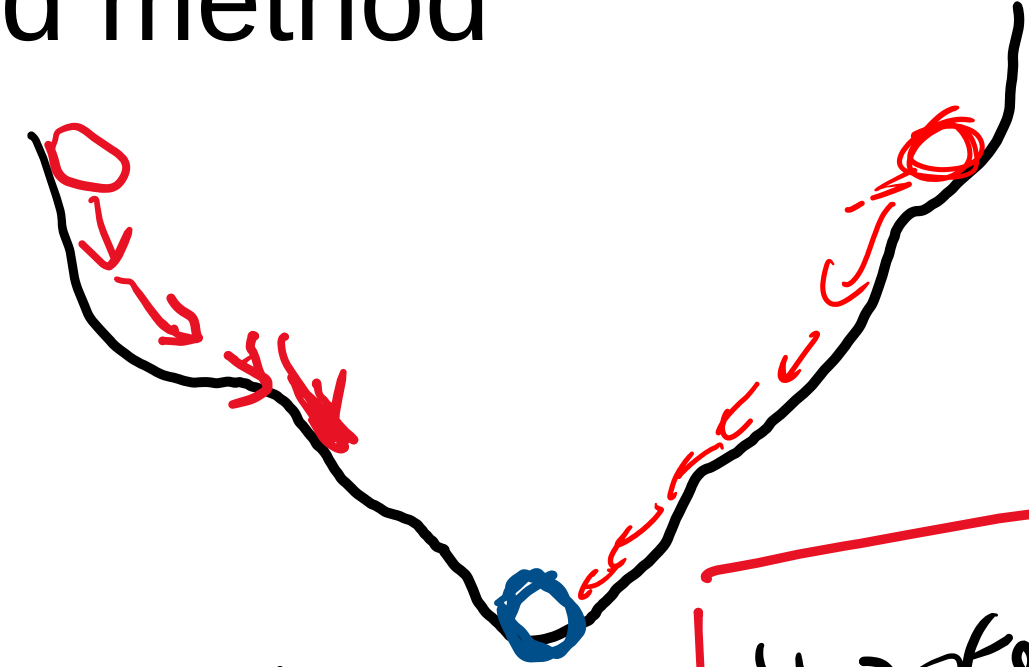
problems:

- **Local minima**
- **Hand crafted potential function**
- **Hard to compute distance**
- **Minimal distance may not be continuous**
- **No passage between closely spaces obstacles**
- **Oscillation**

Potential field method

Rective control:





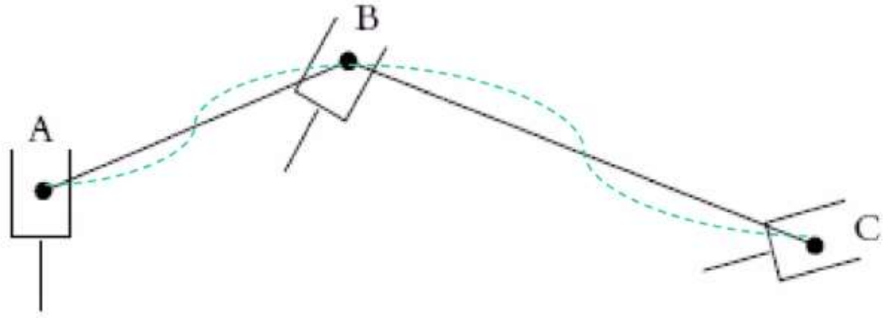
Open question:

Can we design a "potential function" that can globally coverge to a desired point?

Path planning notes

- **Until now, we have only discussed path: the collection of a sequence of robot configurations**
- **It is not clear how the robot can follow the planned path (*implementation)**
- **We don't care about the timing that the robot reaches these configurations**
- **Path is usually discrete and represented as key via-points**
- **Trajectory = path + timing law**

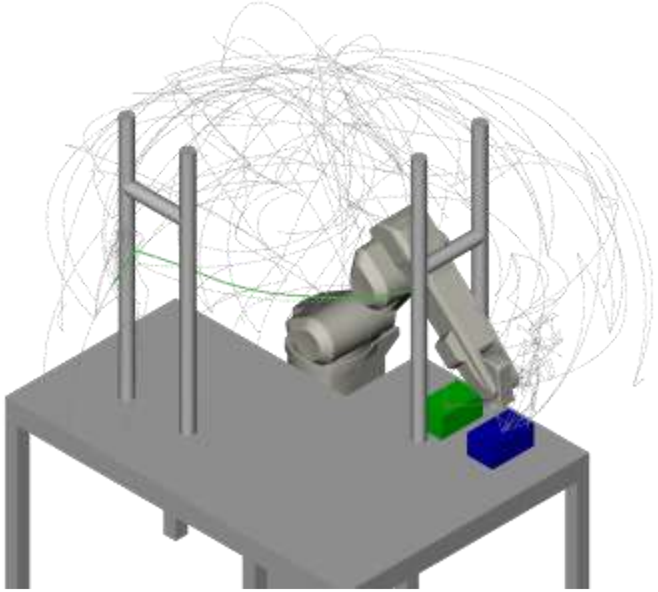
vel, acceleration



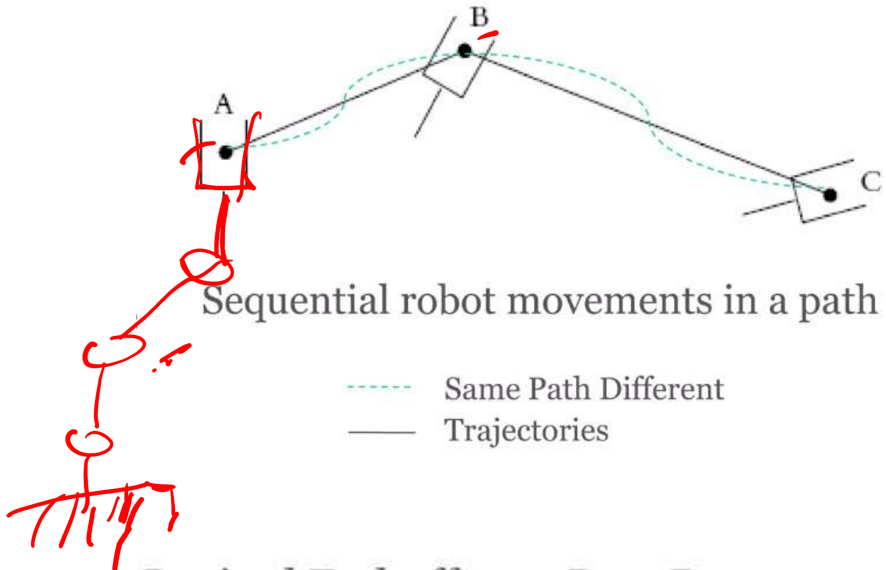
Sequential robot movements in a path

Today's Agenda

- **Drawback of Sampling-based approach (~5)**
- **Potential field method (~15)**
  - **attractive, repulsive**
- **Gradient descent algorithm (~10)**
  - **vector field, velocity field, dynamic system**
- **Trajectory planning (~25)**
  - **Parameter, joint space, cartesian space**
- **Planning as optimization (~20)**



Trajectory planning (joint space)



Sequential robot movements in a path

1. Robot at A pose.
2. compute θ_B (IK)
3. Send θ_B to robot controller

?

A

Desired End-effector Pose B

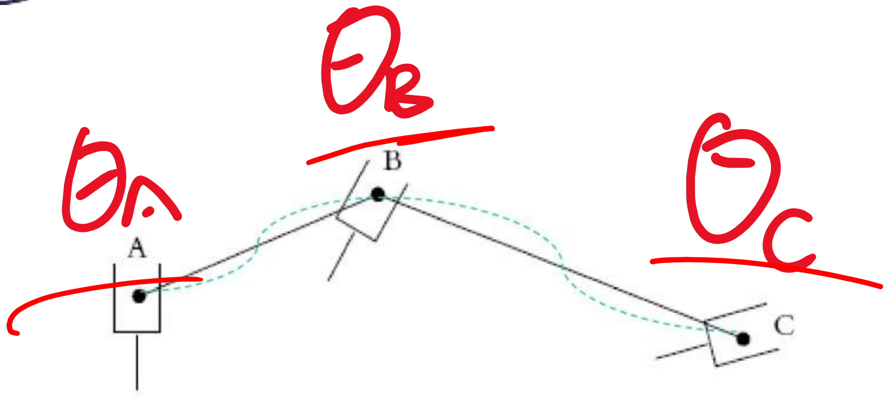
$$R \quad \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad pos$$

Inverse Kinematics → Joint Values θ_B $\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \end{bmatrix}$ → **Joint Controllers** → New Pose B

IK

6DOF.

Trajectory planning (joint space)



Sequential robot movements in a path

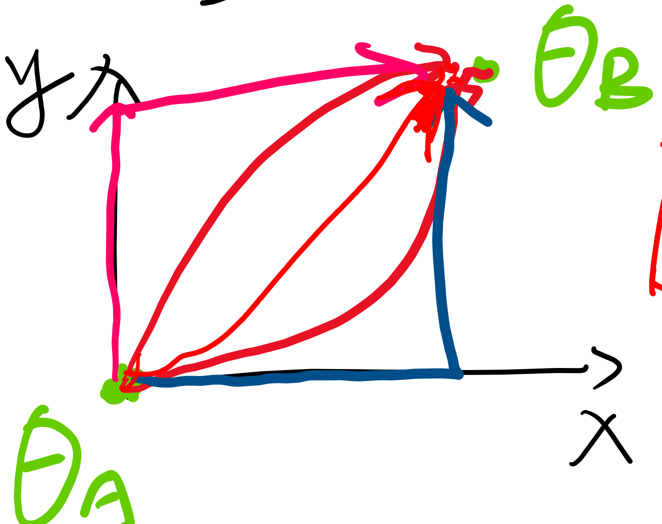
Same Path Different Trajectories

1. Robot at A pose.
2. compute θ_B
3. Send θ_B to robot controller

$$\theta_A \longrightarrow \theta_B \quad ?$$

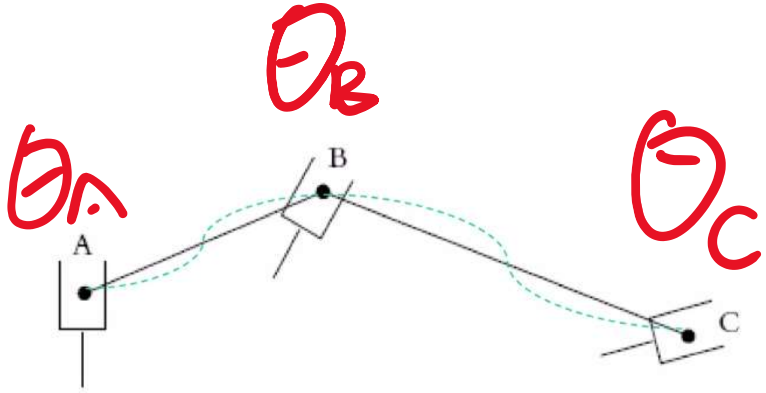
2D Example: $\theta_A = [0, 0]$

$\theta_B = [1, 1]$



infinite.

Trajectory planning (joint space)



Sequential robot movements in a path

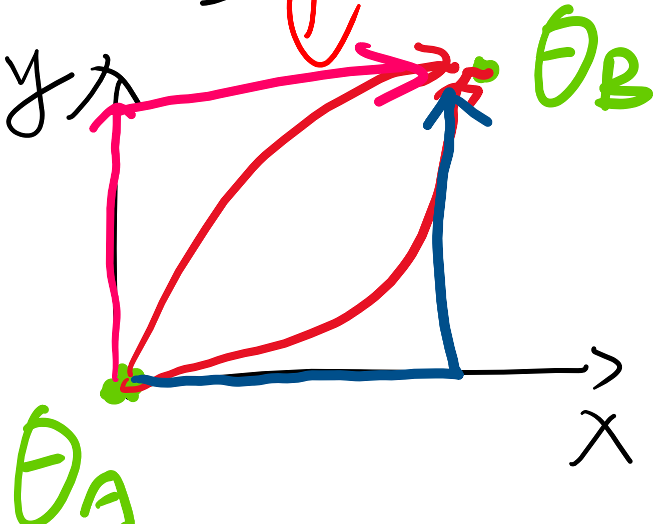
Same Path Different
Trajectories

No guarantee the trj of the robot.

$\theta_A \longrightarrow \theta_B$?

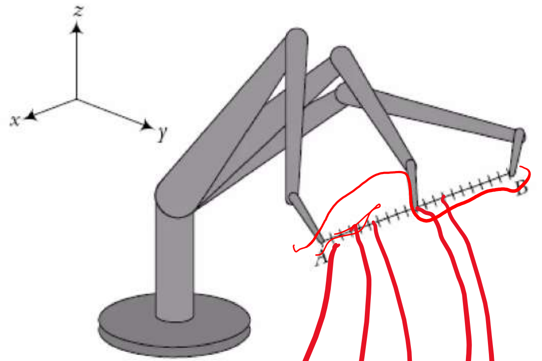
2D Example: $\theta_A = [0,0]$

$\theta_B = [1,1]$



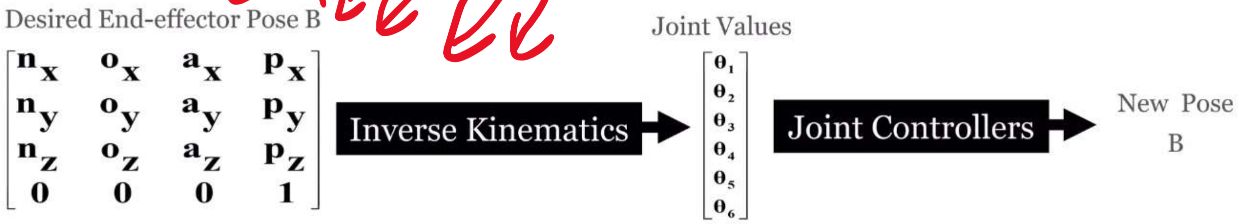
infinite:

Trajectory planning (Cartesian space)



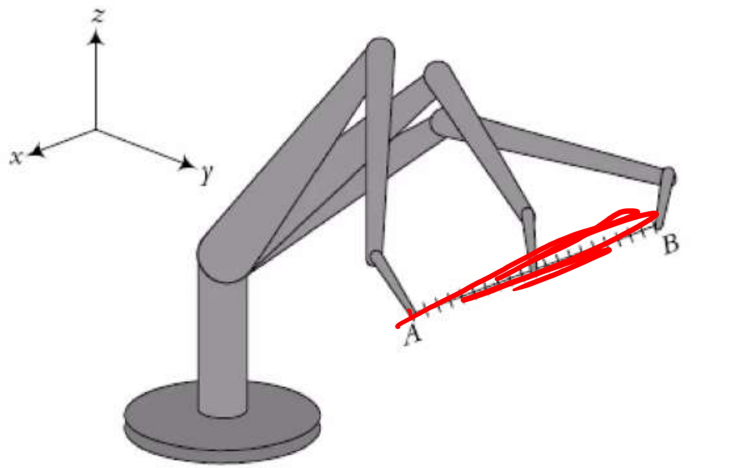
Sequential motions of a robot to follow a straight line

- Assume a straight line between pose A and pose B
- In this way, we force the robot to move from A to B through a straight line
- Then we divide the line into small segments and move the robots through all the intermediate points
- The Ik is computed at each intermediate point and send to the robot controller

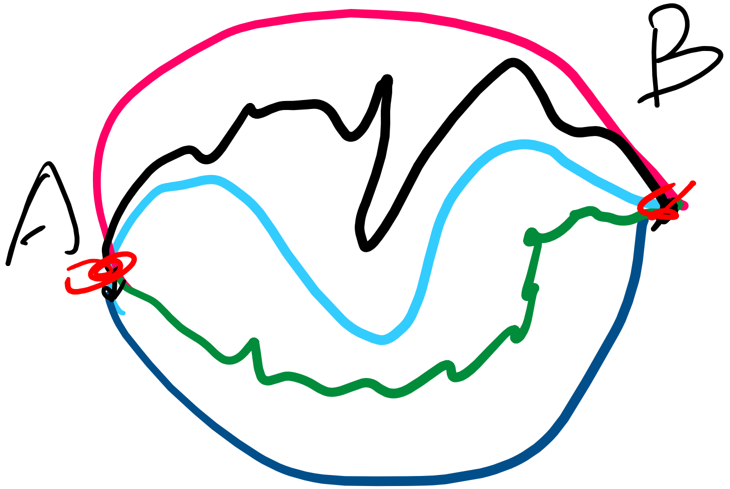


fast IK

Trajectory planning (Cartesian space)

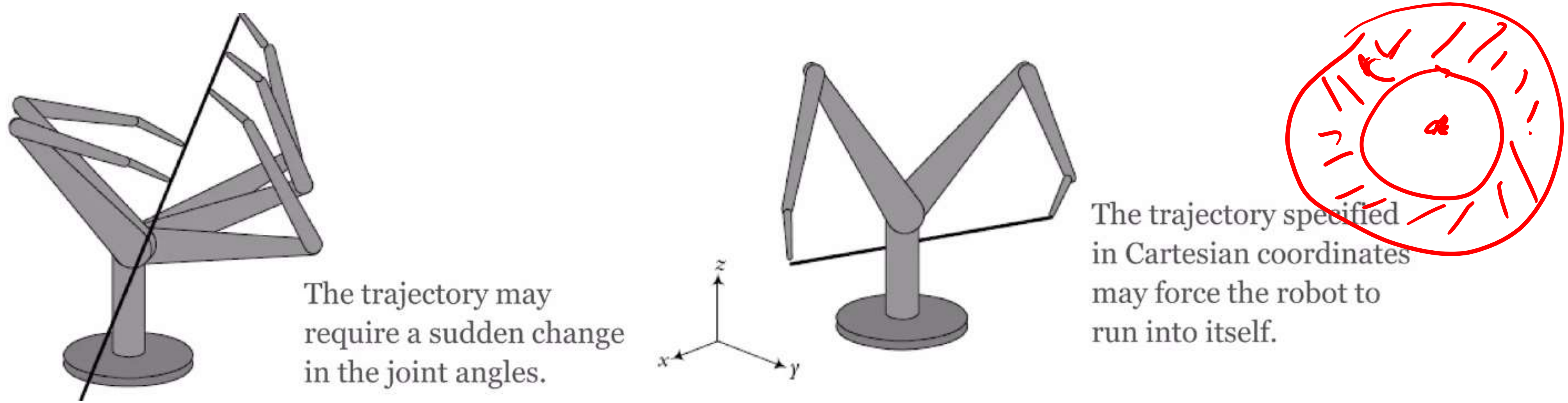


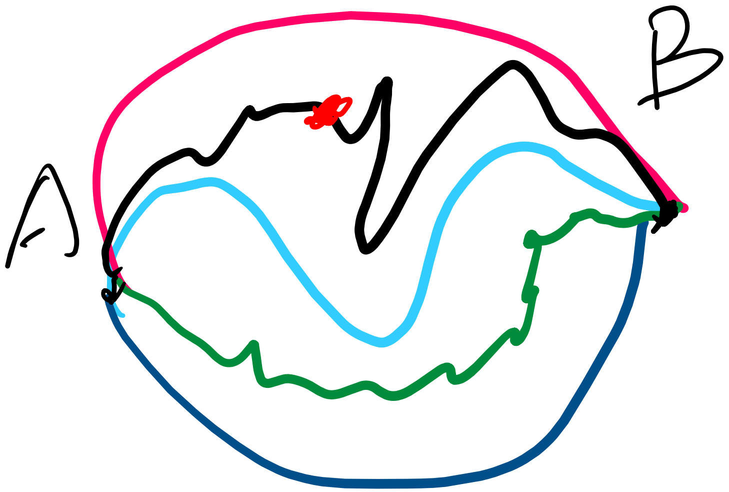
Sequential motions of a robot to follow a straight line



- **Cartesian space trajectories are very to visualize**
- **Computationally expensive: IK at each intermediate point**

Trajectory planning (Cartesian space)





- **Difficult to predict singularity** (奇异)
- **Self-collision**
- **Out of reach**
- **No IK solution along the path**

Trajectory planning (Cartesian space)

Given: a simple 2-DOF robot (mechanism)

Required:

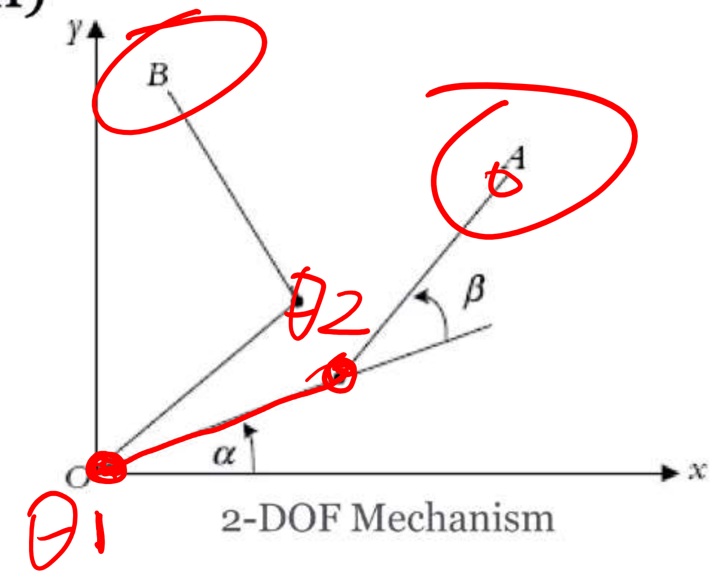
Move the robot from point A to point B.

Suppose that:

At initial point A: $\alpha=20^{o}$ & $\beta=30^{o}$.

At final point B: $\alpha=40^{o}$ & $\beta=80^{o}$.

Both joints of the robot can move at the maximum rate of 10 degrees/sec.



2-DOF Mechanism

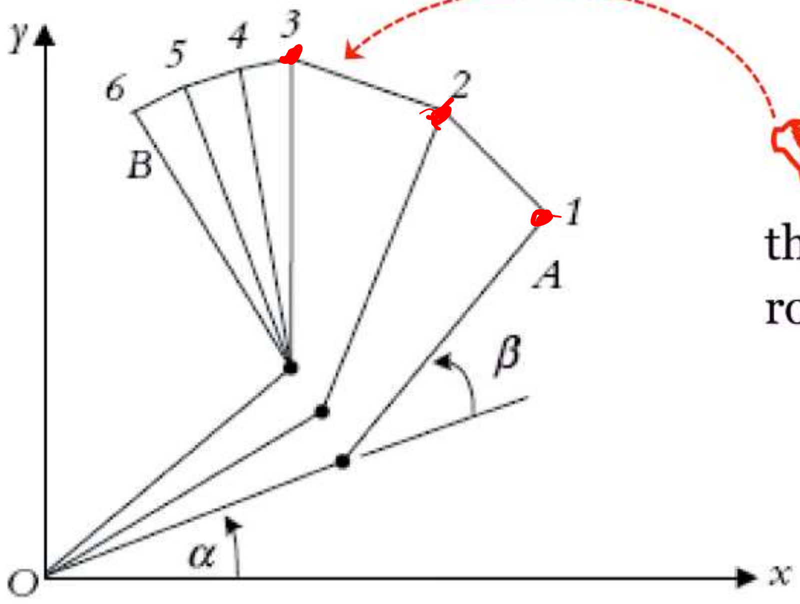
Example from: https://www.slideshare.net/AlaaKhamis/motion-planning

Trajectory planning (Cartesian space)

- **Joint-space, Non-normalized Movements:**

  One way to move the robot from point A to B is to run **both joints** at their **maximum angular velocities**. This means that at the end of the second time interval, the lower link of the robot will have finished its motion, while the upper link continues for another three seconds, as shown here:

Time (sec)	α	β
0	20	30
1	30	40
2	40	50
3	40	60
4	40	70
5	40	80



The path is **irregular**, and the distances traveled by the robot's end are **not uniform**.

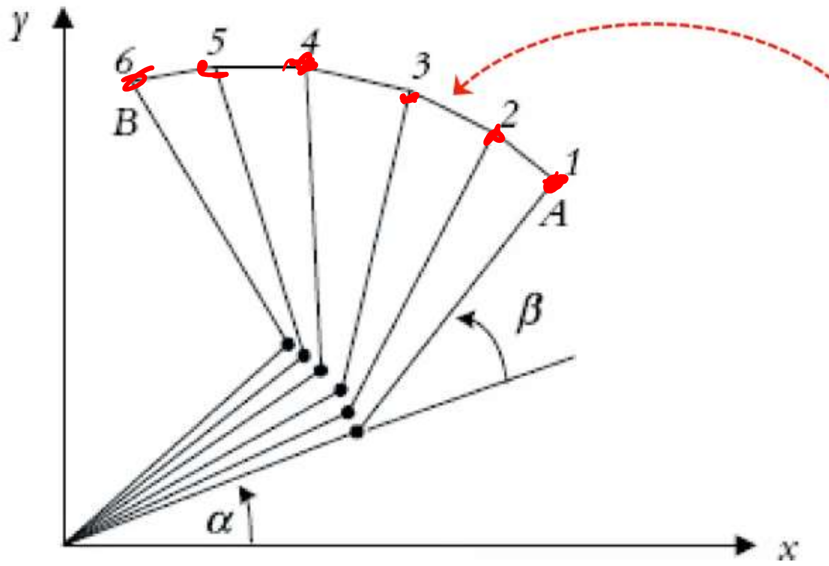
problem?

Trajectory planning (Cartesian space)

- **Basics: Joint-space, Normalized Movements**

  The motions of both joints of the robot are normalized such that the joint with smaller motion will move proportionally slower so that both joints will start and stop their motion simultaneously. In this case, both joints move at different speeds, but move continuously together. **α changes 4 degrees/second** while **β changes 10 degrees/second**.

Time (sec)	α	β
0	20	30
1	24	40
2	28	50
3	32	60
4	36	70
5	40	80



The **segments** of the movement are much **more similar** to each other than before, but the **path** is still **irregular** (and different from the previous case)

Trajectory planning (Cartesian space)

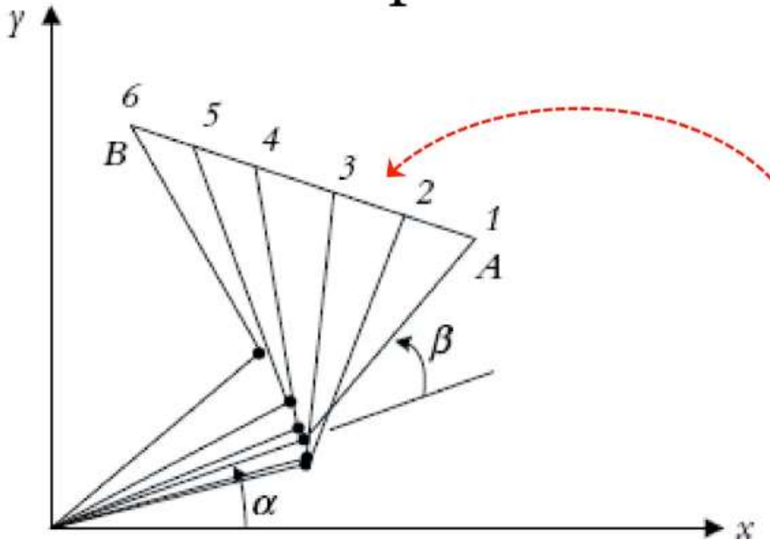
- **Basics: Cartesian-space Movements**

  Now suppose we want the robot's hand to follow a known path between points A and B, say, in a **straight line**.

  The simplest solution would be to draw a line between points A and B, **divide the line** into, say, **5 segments**, and solve for necessary **angles** α and β at each point. This is called **interpolation** between points A and B.

interpolation

Time (sec)	α	β
0	20	30
1	14	55
2	16	69
3	21	77
4	29	81
5	40	80

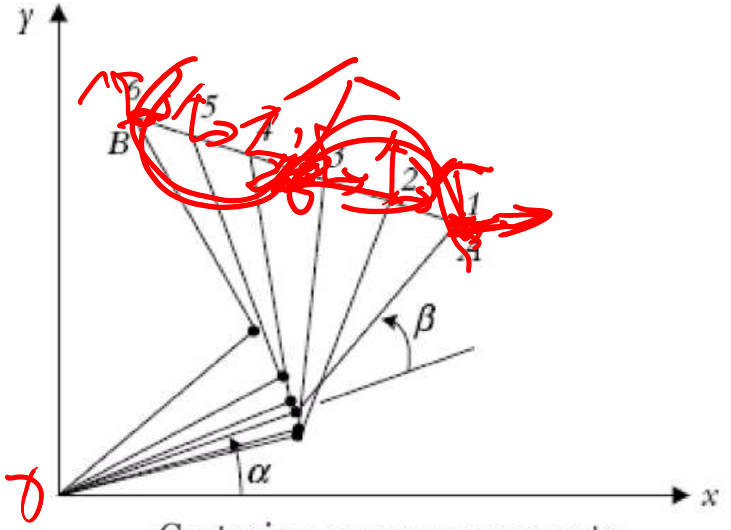


The **path** is a **straight line**, but the **joint angles** are **not uniformly changing**.

Trajectory planning (Cartesian space)

Basics: Cartesian-space Movements

◇ This trajectory is in **Cartesian-space** since all segments of the motion must be calculated based on the information expressed in a **Cartesian frame**.



Cartesian-space movements

◇ Although the resulting motion is a straight (and consequently, known) trajectory, it is necessary to **solve** for the **joint values at each point**.

◇ Obviously, many **more points** must be calculated for **better accuracy**; with so few segments the robot will not exactly follow the lines at each segment.

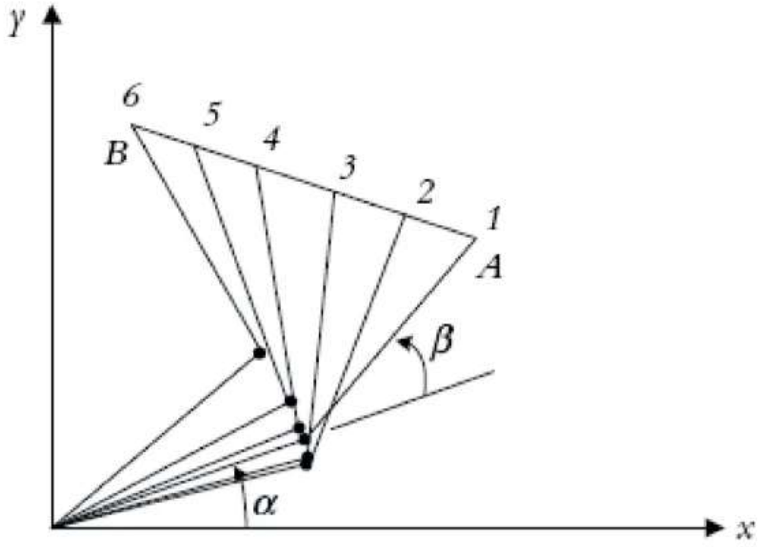
Trajectory planning (Cartesian space)

Basics: Cartesian-space Movements

◇ In this case, it is assumed that the **robot's actuators** are **strong** enough to provide the **large forces** necessary to **accelerate and decelerate** the joints as needed. For example, notice that we are assuming the arm will be instantaneously accelerated to have the desired velocity right at the beginning of the motion in segment 1.

◇ If this is **not true**, the robot will follow a **trajectory different** from our assumption; it will be slightly behind as it accelerates to the desired speed.

Time (sec)	α	β
0	20	30
1	14	55
2	16	69
3	21	77
4	29	81
5	40	80



Trajectory planning (Cartesian space)

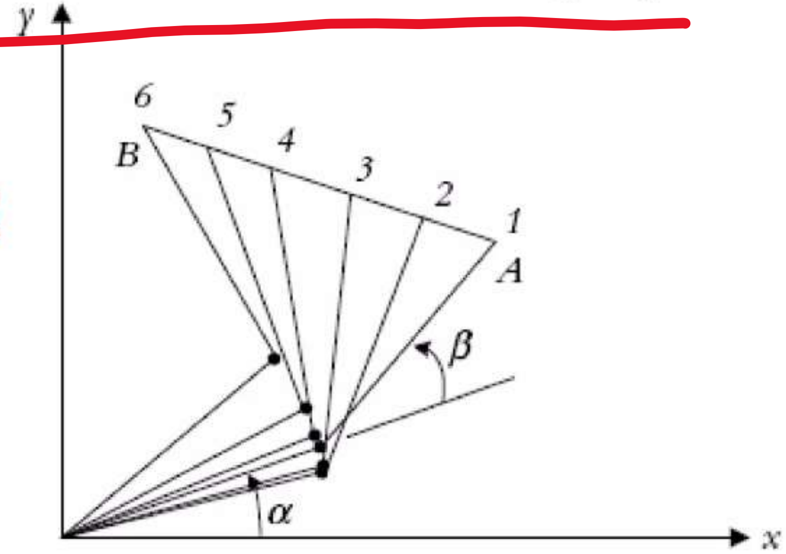
Basics: Cartesian-space Movements

- ◇ Note how the difference between two consecutive values is **larger than** the **maximum specified joint velocity** of **10 degrees/second** (e.g., between times 0 and 1, the joint must move **25 degrees**).
- ◇ Obviously, this is **not attainable**. Also note how, in this case, joint 1 moves downward first before moving up.

Time (sec)	α	β
0	20	30
1	14	55
2	16	69
3	21	77
4	29	81
5	40	80

!!

$11^\circ > 10$



A → B

A ? B

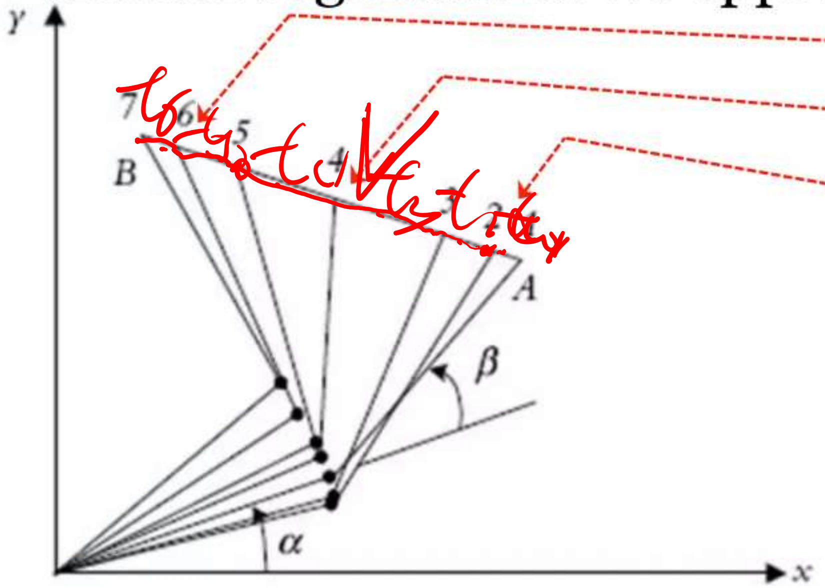
Trajectory planning (Cartesian space)

Basics: Cartesian-space Movements

◇ **Trajectory planning with an acceleration/deceleration regiment:**

Divide the segments differently by starting the arm with smaller segments and, as we **speed up** the arm, going at a **constant cruising rate** and finally **decelerating** with smaller segments as we approach point B.

profile



For each time interval **t**:
Acceleration: **a**
Segment: $\mathbf{x=(1/2).at^2}$
Cruising velocity of $\mathbf{v=at}$

Trajectory planning (Cartesian space)

Basics: Cartesian-space Movements

◇ **Trajectory planning with an acceleration/deceleration regiment:**

Of course, we still need to solve the inverse kinematic equations of the robot at each point, which is similar to the previous case.

However, in this case, instead of dividing the straight line AB into equal segments, we may divide it based on $\mathbf{x=(1/2).at^2}$ until such time t when we attain the cruising velocity of $\mathbf{v=at}$. Similarly, the end portion of the motion can be divided based on a decelerating regiment.

S

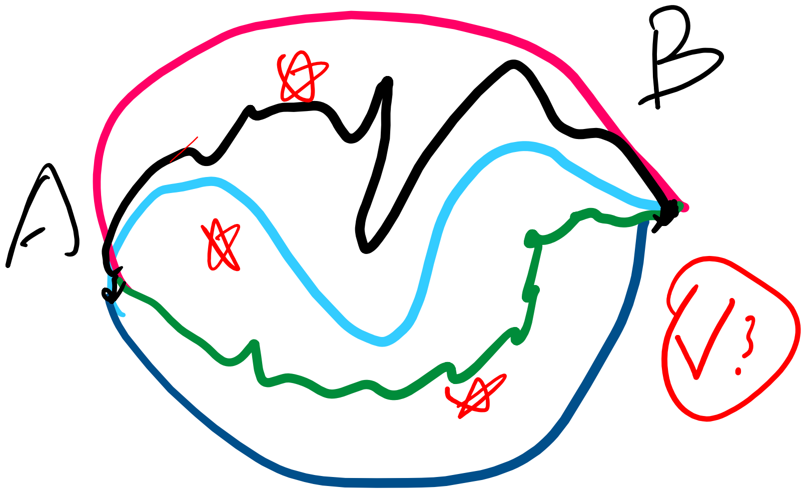
✓

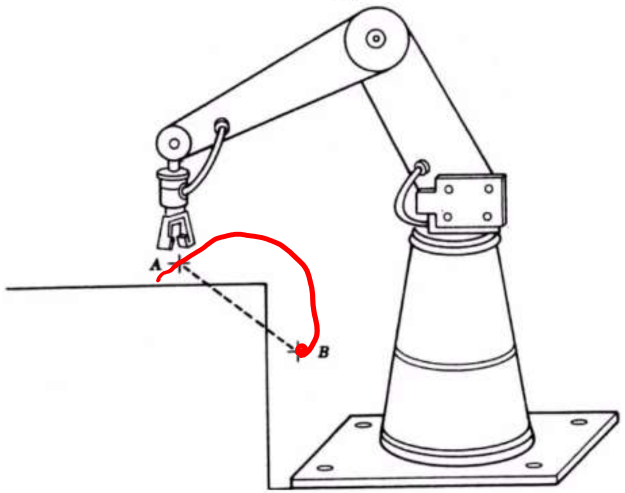
a

t

Trajectory planning (Cartesian space)

- **Basics: Cartesian-space Movements**
  - ◇ Another variation to this trajectory planning is to plan a path that is **not straight**, but one that follows some **desired path**, for example a **quadratic equation**.





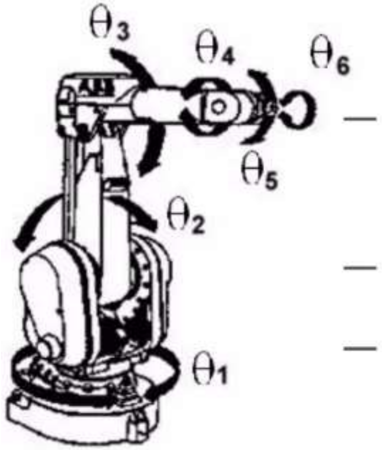
A case where straight line path is not recommended.

To do this, the coordinates of each segment are calculated based on the desired path and are used to calculate joint variables at each segment; therefore, the trajectory of the robot can be planned for any desired path.

Trajectory planning (Joint space)

pose

- **3rd Order Polynomial**
  - ◇ In this application, the initial location and orientation of the robot are known and, using the inverse kinematic equations, the final joint angles for the desired position and orientation are found.



– Initial Location and orientation of the robot

– Initial Joint Angles

– Desired Location and orientation of the robot.

Inverse Kinematics → Final Joint Angles

$t \Rightarrow \theta = [\theta_0, \theta_f]$

  - ◇ However, the motions of **each joint** of the robot must be **planned individually**.

Trajectory planning (Joint space)

- **3rd Order Polynomial**
  - ◇ Consider one of the joints,

    At the beginning of the motion segment at time $\mathbf{t_i}$

    The joint angle is $\boldsymbol{\theta_i}$

θ_i — following 3rd order polynomial trajectory → θ_f

$$\theta(t) = c_o + c_1 t + c_2 t^2 + c_3 t^3$$

4 Unknowns: $c_o, c_1, c_2 \;\&\; c_3$

4 pieces of information:
$$\theta(t_i) = \theta_i$$
$$\theta(t_f) = \theta_f$$
$$\dot{\theta}(t_i) = 0$$
$$\dot{\theta}(t_f) = 0$$

0, Vtf

Trajectory planning (Joint space)

- **3rd Order Polynomial**
  - ◇ Consider one of the joints,

θ_i → following 3rd order polynomial trajectory → θ_f

$$\theta(t) = c_o + c_1 t + c_2 t^2 + c_3 t^3$$

Taking the first derivative of the polynomial:

$$\dot{\theta}(t) = c_1 + 2c_2 t + 3c_3 t^2$$

Substituting the initial and final conditions:

$$\theta(t_i) = c_o = \theta_i$$

$$\theta(t_f) = c_o + c_1 t_f + c_2 t_f^2 + c_3 t_f^3 = \theta_f$$

$$\dot{\theta}(t_i) = c_1 = 0$$

$$\dot{\theta}(t_f) = c_1 + 2c_2 t_f + 3c_3 t_f^2 = 0$$

In matrix form

$$\begin{bmatrix} \theta_i \\ \theta_f \\ \dot{\theta}_i \\ \dot{\theta}_f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} c_o \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Trajectory planning (Joint space)

- **3rd Order Polynomial**

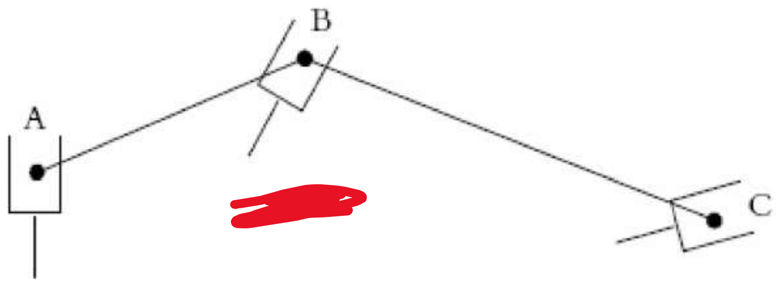
$$\begin{bmatrix} \theta_i \\ \theta_f \\ \dot{\theta}_i \\ \dot{\theta}_f \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} c_o \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

◇ By solving these four equations simultaneously, we get the necessary values for the constants. This allows us to calculate the **joint position** at **any interval of time**, which can be used by the controller to drive the joint to position. The same process must be used for each joint individually, but they are all driven together from start to finish.

◇ Applying this third-order polynomial to each joint motion creates a **motion profile** that can be used to drive each joint.

Trajectory planning (Joint space)

- **3rd Order Polynomial**
  - ◇ If more than two points are specified, such that the robot will go through the points **successively**, the **final velocities and positions** at the **conclusion of each segment** can be used as the **initial values for the next segments**.



Sequential robot movements in a path

  - ◇ Similar third-order polynomials can be used to plan each section. However, although positions and velocities are continuous, accelerations are not, which may cause problems.

Trajectory planning (Joint space)

- **3^{rd} Order Polynomial**

Exam

◇ ***Example:*** It is desired to have the first joint of a 6-axis robot go from initial angle of 30° to a final angle of 75° in 5 seconds. Using a third-order polynomial, calculate the joint angle at 1, 2, 3, and 4 seconds.

◇ ***Given:***

$$t_i = 0 \qquad \theta(t_i) = 30$$

$$t_f = 5 \qquad \theta(t_f) = 75$$

$$\dot{\theta}(t_i) = 0$$

$$\dot{\theta}(t_f) = 0$$

◇ ***Required:***

$$\theta \text{ at } t = 1,2,3 \text{ and } 4$$

Trajectory planning (Joint space)

- **3rd Order Polynomial**
  - ◇ ***Example (cont'd):***
  - ◇ ***Solution:*** Substituting the boundary conditions:

$$\begin{cases} \theta(t_i) = c_o = \theta_i \\ \theta(t_f) = c_o + c_1 t_f + c_2 t_f^2 + c_3 t_f^3 = \theta_f \\ \dot{\theta}(t_i) = c_1 = 0 \\ \dot{\theta}(t_f) = c_1 + 2c_2 t_f + 3c_3 t_f^2 = 0 \end{cases} \rightarrow \begin{cases} \theta(t_i) = c_o = 30 \\ \theta(t_f) = c_o + c_1(5) + c_2(5)^2 + c_3(5)^3 = 75 \\ \dot{\theta}(t_i) = c_1 = 0 \\ \dot{\theta}(t_f) = c_1 + 2c_2(5) + 3c_3(5)^2 = 0 \end{cases}$$

$$\downarrow$$

$$c_o = 30, c_1 = 0, c_2 = 5.4, c_3 = -0.72$$

Trajectory planning (Joint space)

- **3rd Order Polynomial**
  - ◇ ***Example (cont'd):*** This results in the following cubic polynomial equation for position as well as the velocity and acceleration equations for joint 1:

$$\theta(t) = 30 + 5.4t^2 - 0.72t^3$$

$$\dot{\theta}(t) = 10.84t - 2.16t^2$$

$$\ddot{\theta}(t) = 10.84 - 4.32t$$

Substituting the desired time intervals into the motion equation will result in:

$$\theta(1) = 34.68^o, \quad \theta(2) = 45.84^o, \quad \theta(3) = 59.16^o, \quad \theta(4) = 70.32^o$$

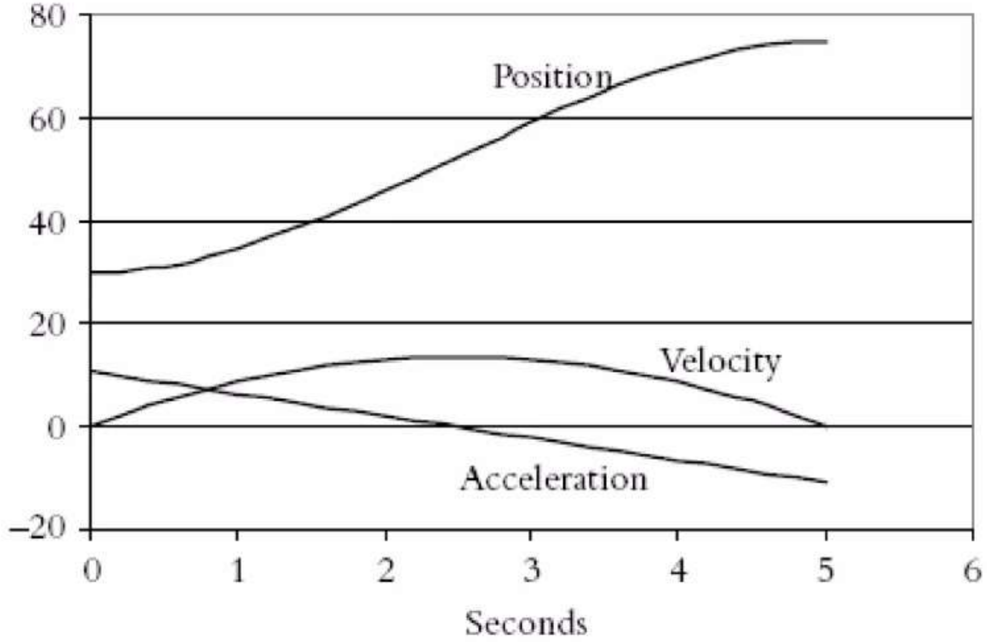
Trajectory planning (Joint space)

- **3rd Order Polynomial**
  - ***Example (cont'd):*** The joint angles, velocities, and accelerations are shown below. Notice that in this case, the acceleration needed at the beginning of the motion is 10.8°/sec² (as well as -10.8°/sec² deceleration at the conclusion of the motion).

$$\theta(t) = 30 + 5.4t^2 - 0.72t^3$$

$$\dot{\theta}(t) = 10.84t - 2.16t^2$$

$$\ddot{\theta}(t) = 10.84 - 4.32t$$



Joint positions, velocities, and accelerations

Trajectory planning (Joint space)

- **5th Order Polynomial**
  - ◇ Specifying the initial and ending positions, velocities, and accelerations of a segment yields **six pieces of information**, enabling us to use a **fifth-order polynomial** to plan a trajectory, as follows:

$$\theta(t) = c_o + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5$$

$$\dot{\theta}(t) = c_1 + 2c_2 t + 3c_3 t^2 + 4c_4 t^3 + 5c_5 t^4$$

$$\ddot{\theta}(t) = 2c_2 + 6c_3 t + 12c_4 t^2 + 20c_5 t^3$$

These equations allow us to calculate the coefficients of a fifth-order polynomial with position, velocity, and acceleration boundary conditions.

Trajectory planning (Joint space)

- **5th Order Polynomial**
  - ◇ ***Example:*** Repeat Example-1, but assume the initial acceleration and final deceleration will be $5^o/\sec^2$.
  - ◇ ***Solution:*** From Example-1 and the given accelerations, we have:

$$\theta_i = 30^o \quad \dot{\theta}_i = 0^o/\sec \quad \ddot{\theta}_i = 5^o/\sec^2$$

$$\theta_f = 75^o \quad \dot{\theta}_f = 0^o/\sec \quad \ddot{\theta}_f = -5^o/\sec^2$$

Substituting in the following equations will result in:

$$\begin{cases} \theta(t) = c_o + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5 \\ \dot{\theta}(t) = c_1 + 2c_2 t + 3c_3 t^2 + 4c_4 t^3 + 5c_5 t^4 \\ \ddot{\theta}(t) = 2c_2 + 6c_3 t + 12c_4 t^2 + 20c_5 t^3 \end{cases} \rightarrow \begin{matrix} c_o = 30 & c_1 = 0 & c_2 = 2.5 \\ c_3 = 1.6 & c_4 = -0.58 & c_5 = 0.0464 \end{matrix}$$

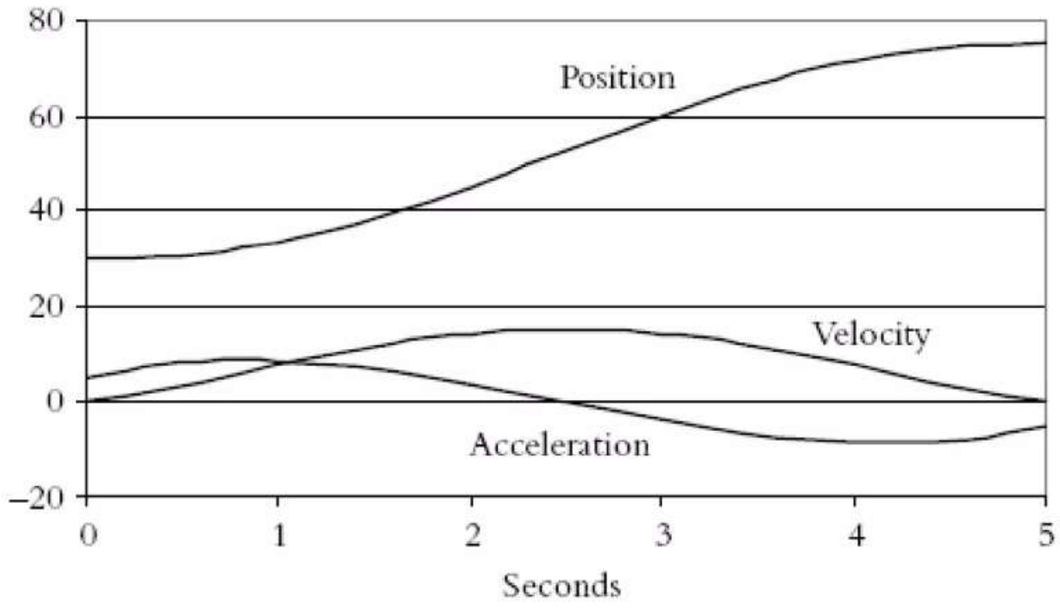
Trajectory planning (Joint space)

- **5th Order Polynomial**
  - ◊ ***Example (cont'd):*** This results in the following motion equations:

$$\theta(t) = 30 + 2.5t^2 + 1.6t^3 - 0.58t^4 + 0.0464\,t^5$$

$$\dot{\theta}(t) = 5t + 4.8t^2 - 2.32t^3 + 0.232\,t^4$$

$$\ddot{\theta}(t) = 5 + 9.6t - 6.9t^2 + 0.928t^3$$



Trajectory planning (Joint space)

- **5th Order Polynomial**
  - ◇ To ensure that the robot's accelerations will not exceed its capabilities, acceleration limits may be used to calculate the necessary time to reach the target.

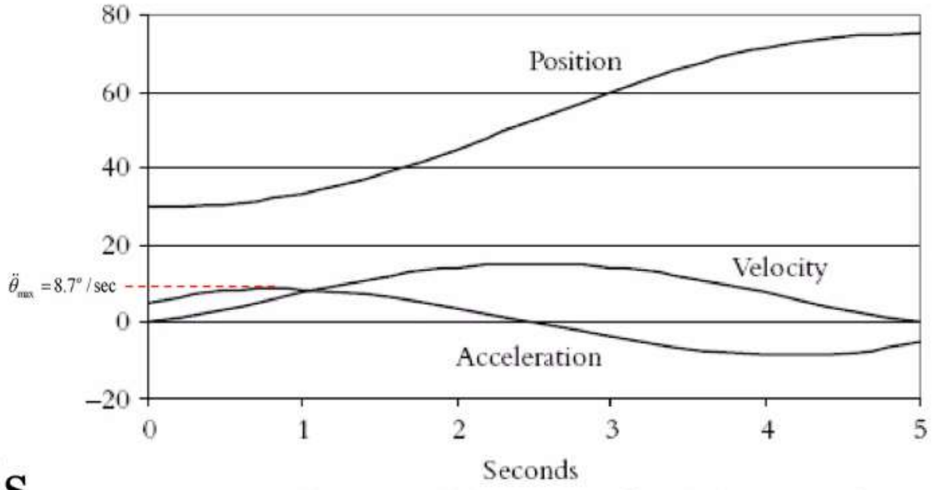
For $\dot{\theta}_i = 0$ and $\dot{\theta}_f = 0$

$$\left|\ddot{\theta}\right|_{\max} = \left|\frac{6(\theta_f - \theta_i)}{(t_f - t_i)^2}\right|$$

In example-2:

$$\left|\ddot{\theta}\right|_{\max} = \left|\frac{6(75-30)}{(5-0)^2}\right| = 10.8$$

The maximum acceleration is $8.7^{o}/\text{sec}^2 < \left|\ddot{\theta}\right|_{\max}$



Joint positions, velocities, and accelerations

Trajectory planning (Cartesian space)

- Cartesian-space trajectories relate to the motions of a robot relative to the Cartesian reference frame, as followed by the **position and orientation of the robot's hand**.
- In addition to simple **straight-line trajectories**, many other schemes may be deployed to drive the robot in its path between different points.
- In fact, **all of the schemes** used for joint-space trajectory planning can also be used for Cartesian-space trajectories.
- The basic difference is that for Cartesian-space, the joint values must be **repeatedly calculated** through the **inverse kinematic** equations of the robot.

fast IK.

Trajectory planning (Cartesian space)

- **Procedure**

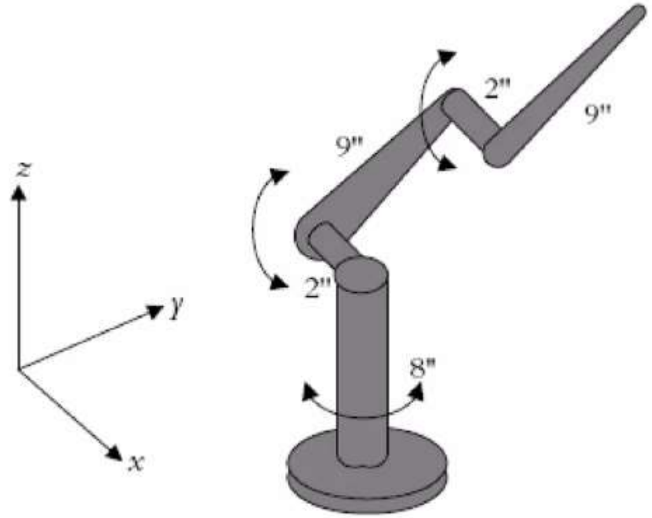
1. Increment the time by **t=t+Δt**.
2. Calculate the **position and orientation** of the hand based on the **selected function** for the trajectory.
3. Calculate the **joint values** for the position and orientation through the **inverse kinematic** equations of the robot.
4. Send the **joint information** to the **controller**.
5. Go to the beginning of the loop.

filter.

Trajectory planning (Cartesian space)

- **Example**

A 3-DOF robot designed for lab experimentation has two links, each 9 inches long. As shown in the figure, the coordinate frames of the joints are such that when all angles are zero, the arm is pointed upward.



The inverse kinematic equations of the robot are also given below.

We want to move the robot from point **(9,6,10)** to point **(3,5,8)** along a **straight line**.

Find the angles of the three joints for each intermediate point and plot the results.

Trajectory planning (Joint space)

- **Example (cont'd)**

Given:

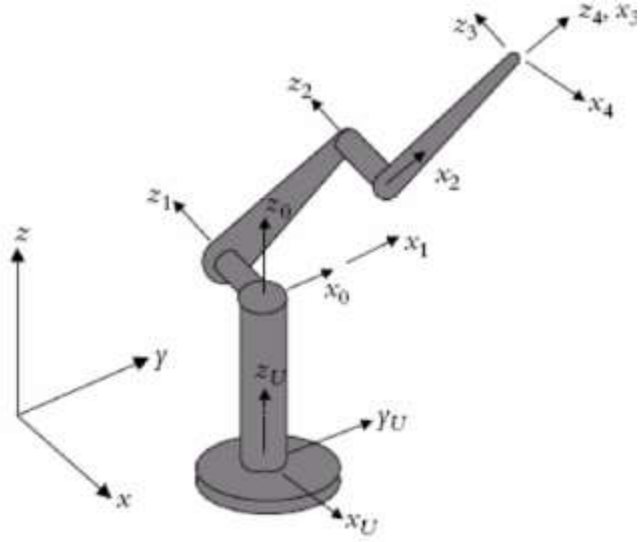
$$A(9,6,10) \xrightarrow{\text{straightline}} B(3,5,8)$$

Inverse Kinematics Solution

$$\theta_1 = \tan^{-1}(P_x / P_y)$$

$$\theta_3 = \cos^{-1}\left[\left((P_y / C_1)^2 + (P_z - 8)^2 - 162 \right) \Big/ 162 \right]$$

$$\theta_2 = \cos^{-1}\left[\left(C_1 (P_z - 8)(1 + C_3) + P_y S_3 \right) \Big/ \left(18(1 + C_3) C_1 \right) \right]$$



Required:

Angles of the three joints for each intermediate point and plot the results.

[8]

Trajectory planning (Joint space)

- **Example (cont'd)**

We **divide the distance** between the start and the end points into **10 segments**, although in reality, it is divided into many more sections. The coordinates of each intermediate point are found by dividing the distance between the initial and the end points into **10 equal parts**.

Straight line equation

between point (x_1,y_1,z_1) and point (x_2,y_2,z_2) is

$$\frac{x-x_1}{x_2-x_1}=\frac{y-y_1}{y_2-y_1}=\frac{z-z_1}{z_2-z_1}$$

$$\frac{x-9}{3-9}=\frac{y-6}{5-6}=\frac{z-10}{8-10}$$

$$(x-9)/6=y-6=0.5(z-10)$$

The Hand Frame Coordinates and Joint Angles for the Robot

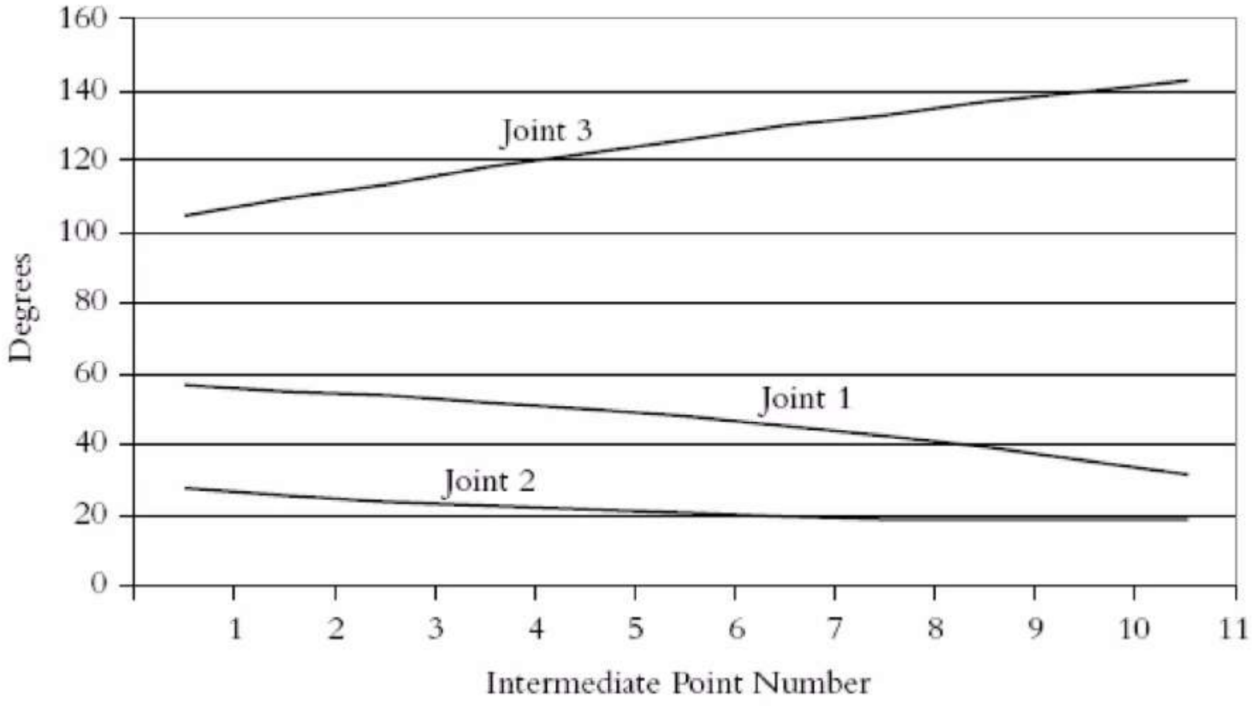
P_x	P_y	P_z	θ_1	θ_2	θ_3
9	6	10	56.3	27.2	104.7
8.4	5.9	9.8	54.9	25.4	109.2
7.8	5.8	9.6	53.4	23.8	113.6
7.2	5.7	9.4	51.6	22.4	117.9
6.6	5.6	9.2	49.7	21.2	121.9
6	5.5	9	47.5	20.1	125.8
5.4	5.4	8.8	45	19.3	129.5
4.8	5.3	8.6	42.2	18.7	133
4.2	5.2	8.4	38.9	18.4	136.3
3.6	5.1	8.2	35.2	18.5	139.4
3	5	8	31	18.9	142.2

Trajectory planning (Cartesian space)

- **Example (cont'd)**

The **inverse kinematic** equations are used to calculate the **joint angles** for **each intermediate point**, as shown in the table.

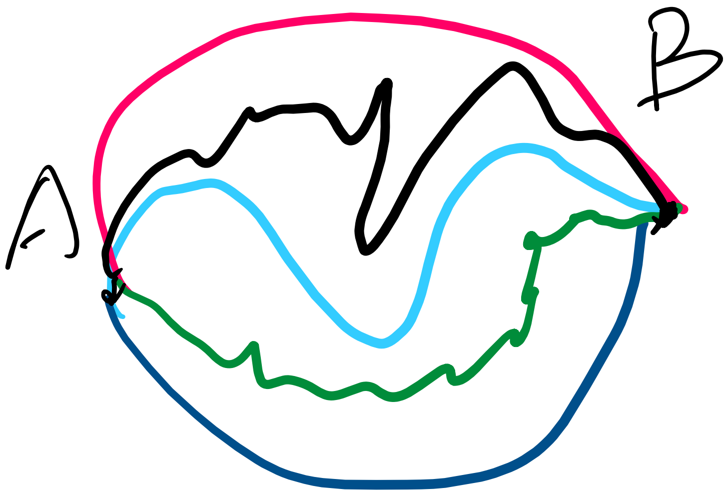
The **joint angles** are shown here.



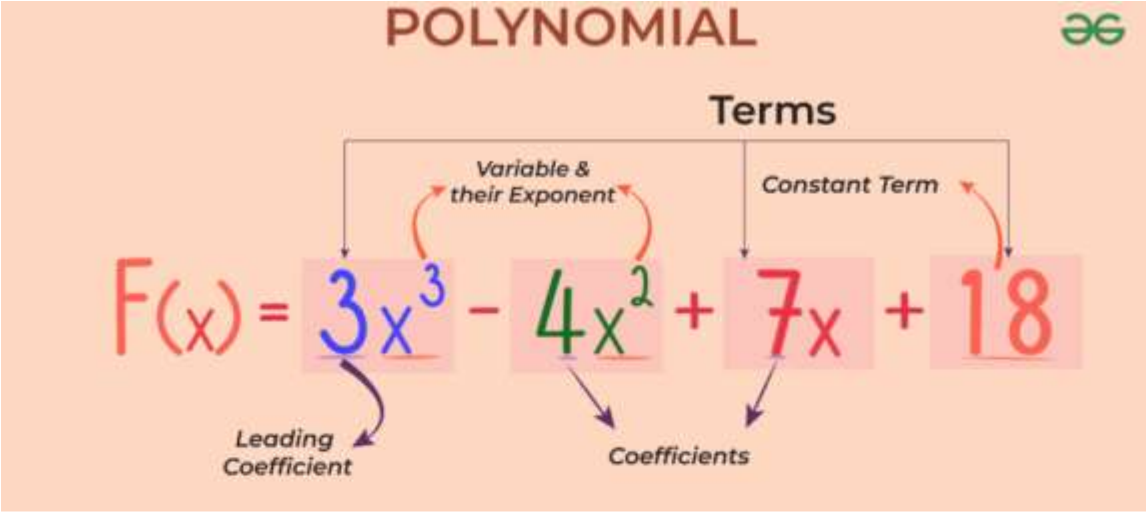
fast

robust

Trajectory planning (Cartesian space)



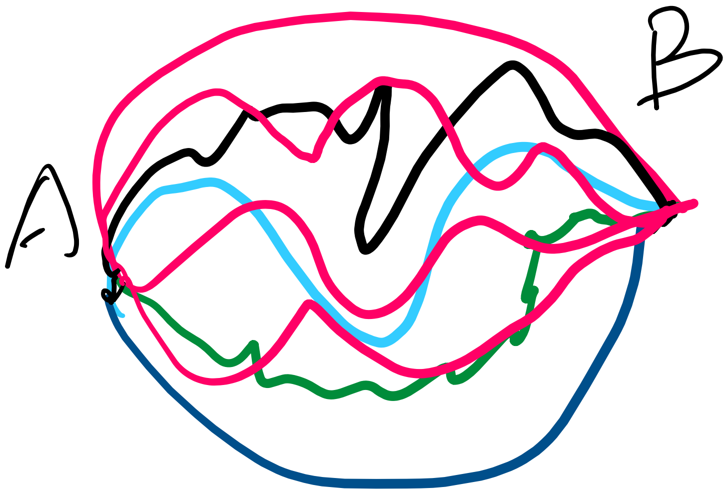
polynomial



any other form of trajectory? ✓

Trajectory planning

- **The key idea of trajectory planning is to use some form of trj representation to choose the proper trj profile (polynomial…)**
- **This process can be applied in both joint space and Cartesian space**
- **Have more flexibility than sampling-base methods**



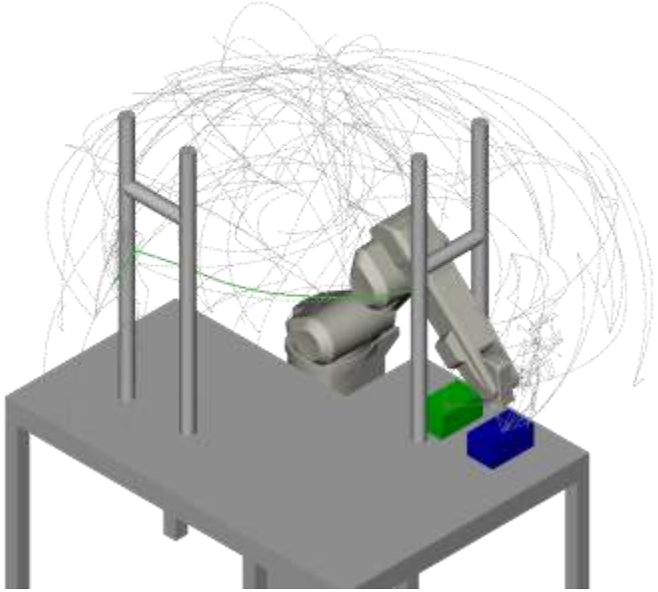
sepecial form

Trajectory planning

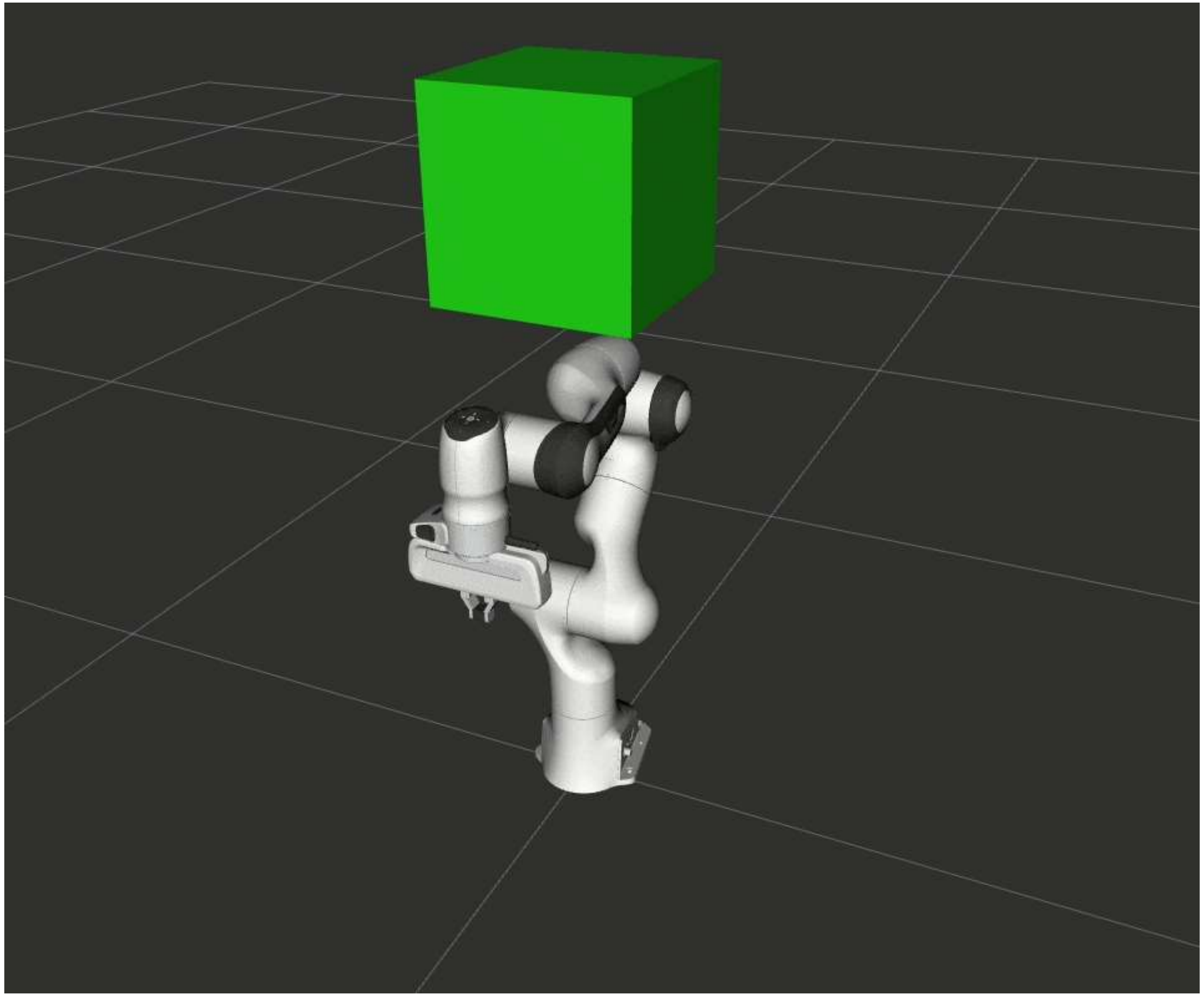
- **No optimality guaranteed**
- **Difficult to generalize to humanoids**
- **No dynamics is considered (The maximal acceleration is checked after planning)**
- **Other constraints such as collision avoidance are not considered.**
- **Restrictive and not human-like**
- **Still not connected with the sensor and actuator (almost only geometry)**

Today’s Agenda

- **Drawback of Sampling-based approach (~5)**
- **Potential field method (~15)**
  - **attractive, repulsive**
- **Gradient descent algorithm (~15)**
  - **vector field, velocity field, dynamic system**
- **Trajectory planning (~25)**
  - **trajectory and path**
  - **Parameter, joint space, cartesian space**
- **Planning as optimization (~20)**

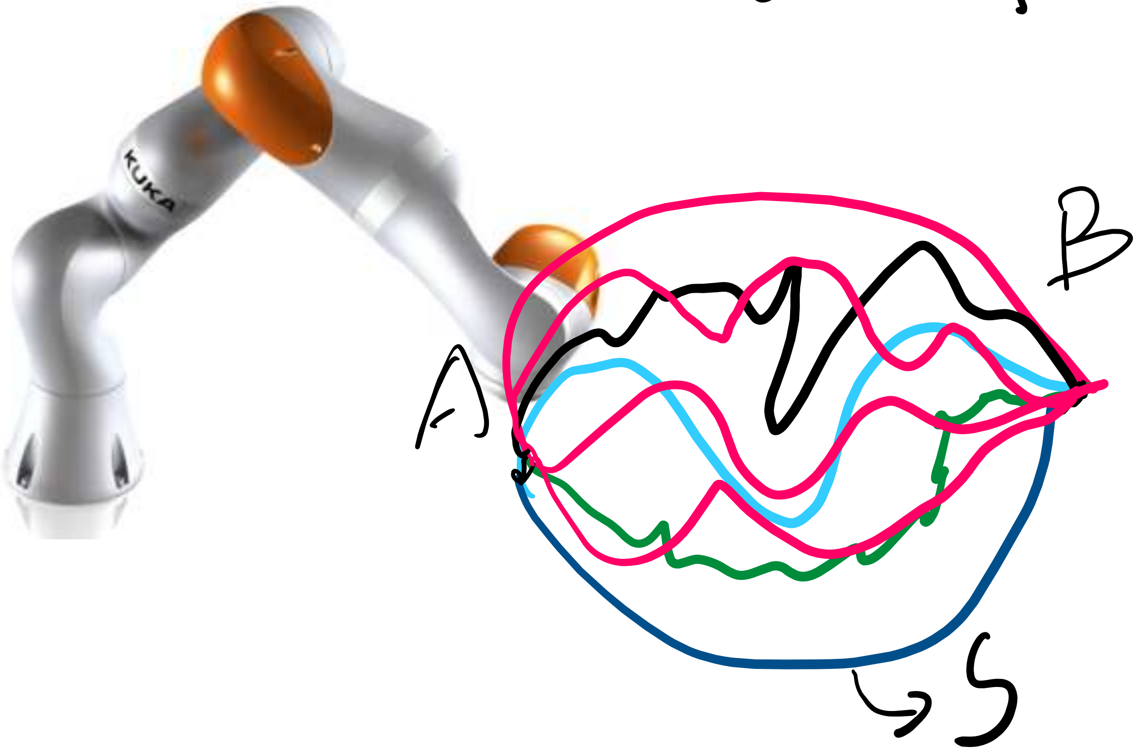


Trajectory optimization



Trajectory optimization

Cost function: $V: S \to R^+$



- path length
- efficiency
- obstacle avoidance
- uncertainty reduction
- predictability
- legibility / intent expression
- human comfort
- naturalness

difficult to represent.

Trajectory optimization

cost function: $U: S \rightarrow R^+$

traj optimization:

$$S^* = \arg\min_{S \in E} U[S]$$

$$\text{s.t.} \begin{cases} S(0) = q_s \\ S(T) = q_g \\ \text{other constraints} \end{cases}$$

- path length
- efficiency
- obstacle avoidance
- uncertainty reduction
- predictability
- legibility / intent expression
- human comfort
- naturalness

difficult to represent.

Trajectory optimization

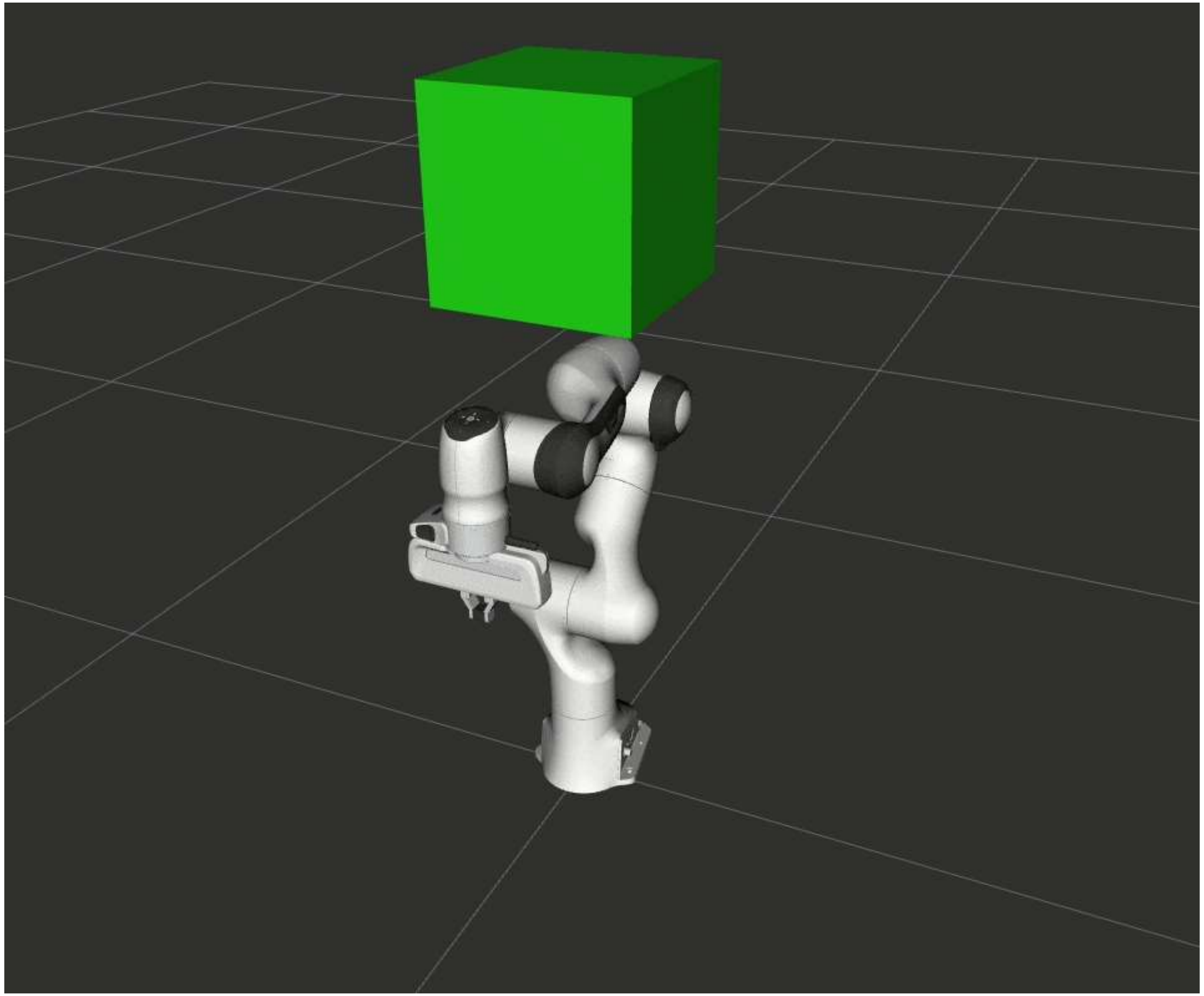
- **Optimization-based motion planning approaches, such as <u>Nonlinear Programming</u> (NLP) and <u>Mixed-Integer Programming</u> (MIP), solve optimization problems, and find solutions using gradient descent while satisfying constraints.**

- **For instance, CHOMP optimizes a cost functional using covariant gradient descent while TrajOpt solves a sequential convex optimization and performs convex collision checking.**

- **Various tasks including navigation, grasping, manipulation, collision-avoidance, running, cooking, and flying under various conditions.**

- **Local optimal (a general problem for nonlinear optimization)**

Trajectory optimization



Trajectory Optimization

$$\min_{\theta_{1:T}} \quad \sum_t \|\theta_{t+1} - \theta_t\|^2 + \text{other costs}$$

$$\text{subject to} \quad \theta_0 = \text{start state}, \quad \theta_T \text{ in goal set}$$

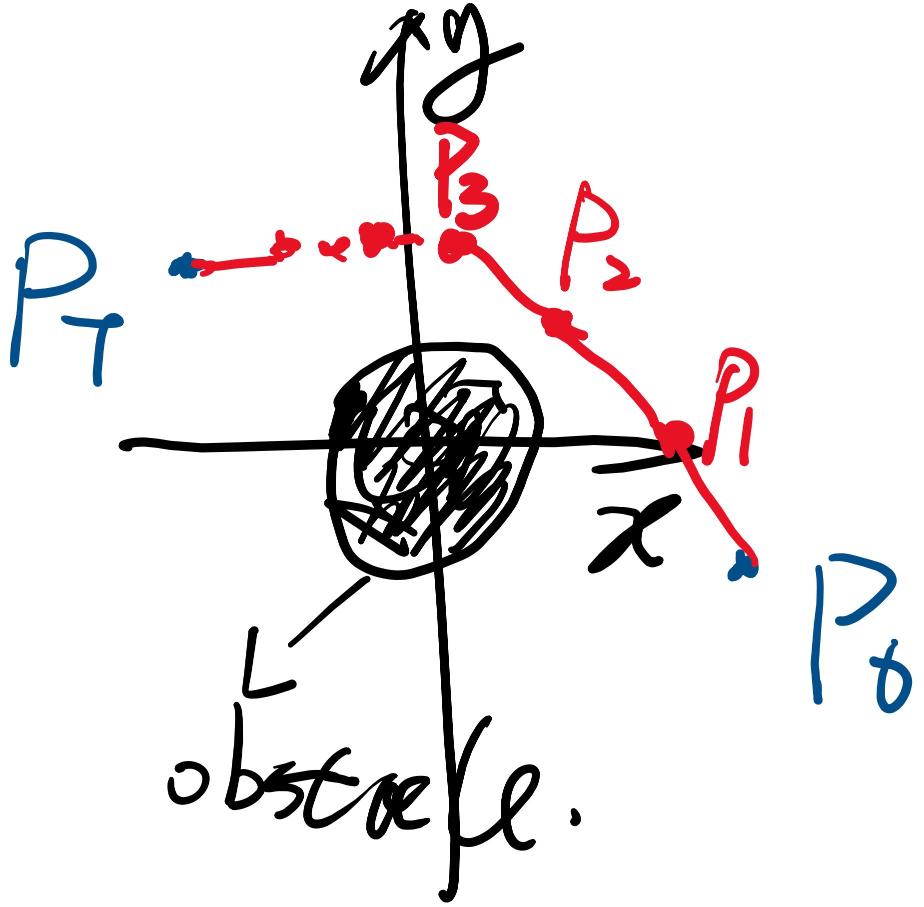
joint limits

for all robot parts, for all obstacles:
no collision $\longrightarrow$ ***non-convex***

Solution method: sequential convex optimization

https://people.eecs.berkeley.edu/~pabbeel/cs287-fa19/slides/Lec10-motion-planning.pdf

Trajectory optimization (Example)



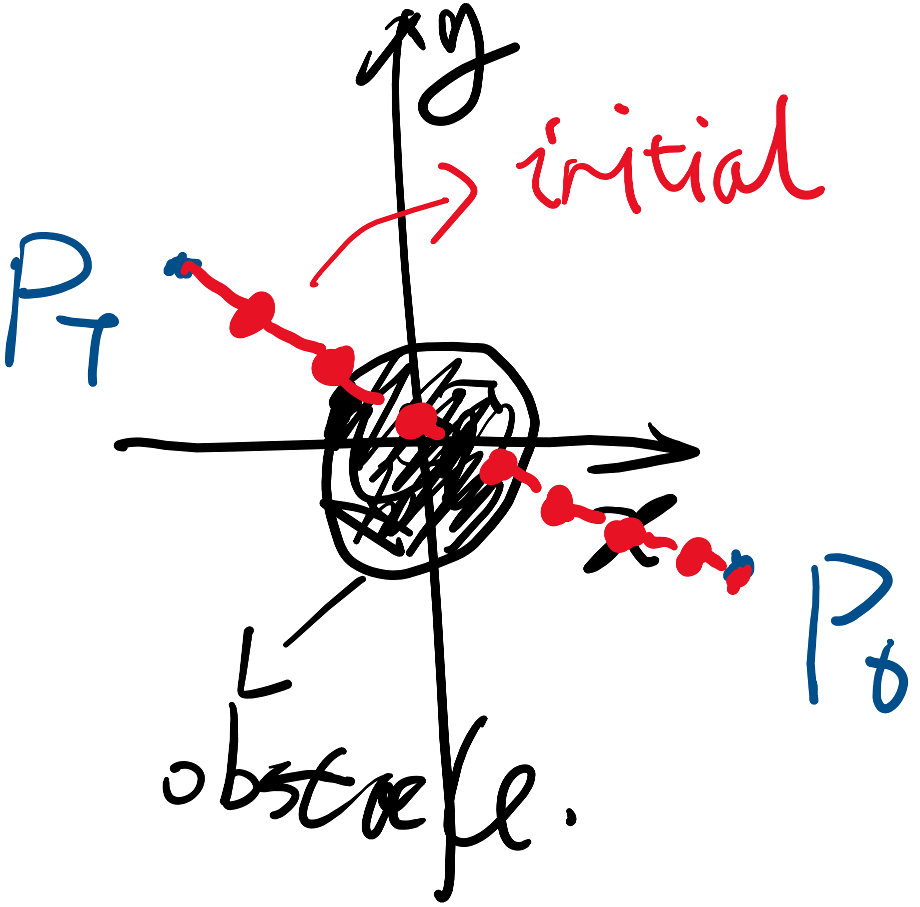
$x^2+y^2 \le 1$

$$\min_{P_{0\dots T}} : \sum_n \|P_n - P_{n-1}\|^2 + \text{other cost}$$

$$\text{s.t.}: \quad \|P_n\| \ge 1 + \delta_{safe}.$$

try to play with this example

Trajectory optimization (Example)

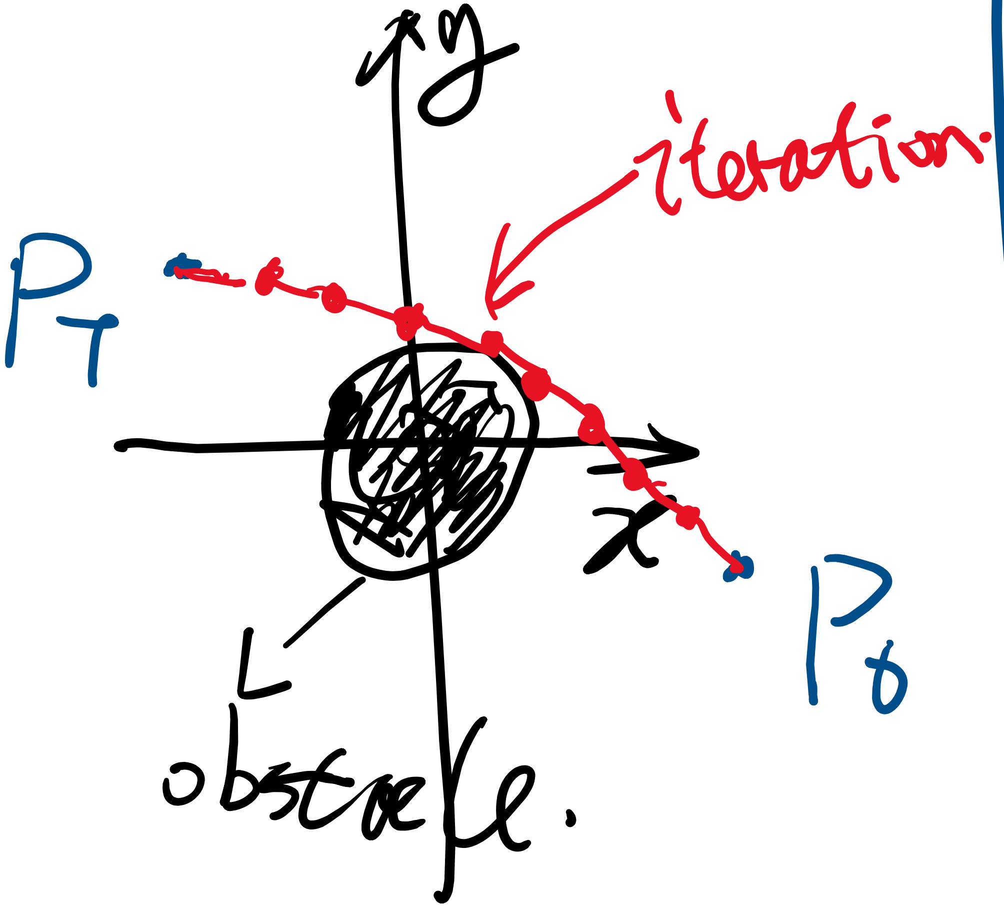


$$x^2+y^2\le 1$$

$$\min_{P_{0\cdots T}}: \sum_n \|P_n - P_{n-1}\|^2 + \text{other cost}$$

$$\text{s.t.}: \|P_n\| \ge 1 + \delta_{safe}.$$

Trajectory optimization (Example)



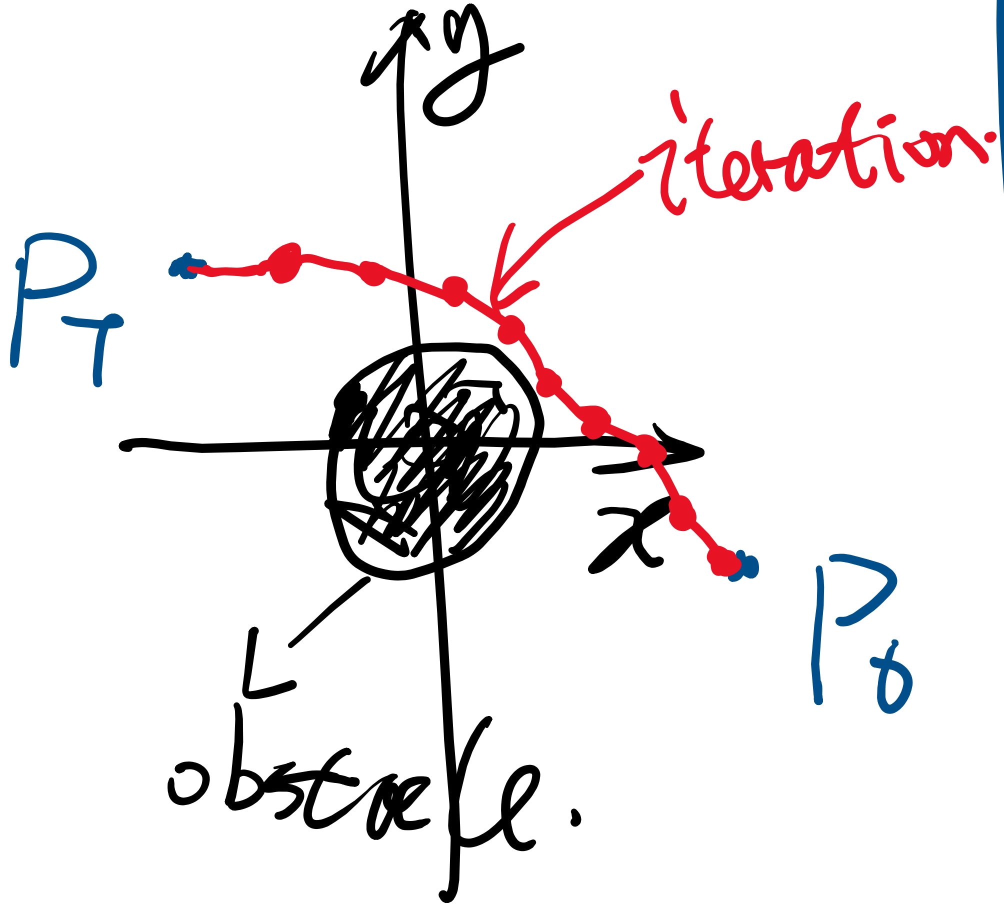
$x^2+y^2\leq 1$

$$\min_{P_{0\ldots T}} : \sum_n \|P_n - P_{n-1}\|^2 + \text{other cost}$$

s.t.:

$$\|P_n\| \geq 1 + \delta_{safe}.$$

Trajectory optimization (Example)

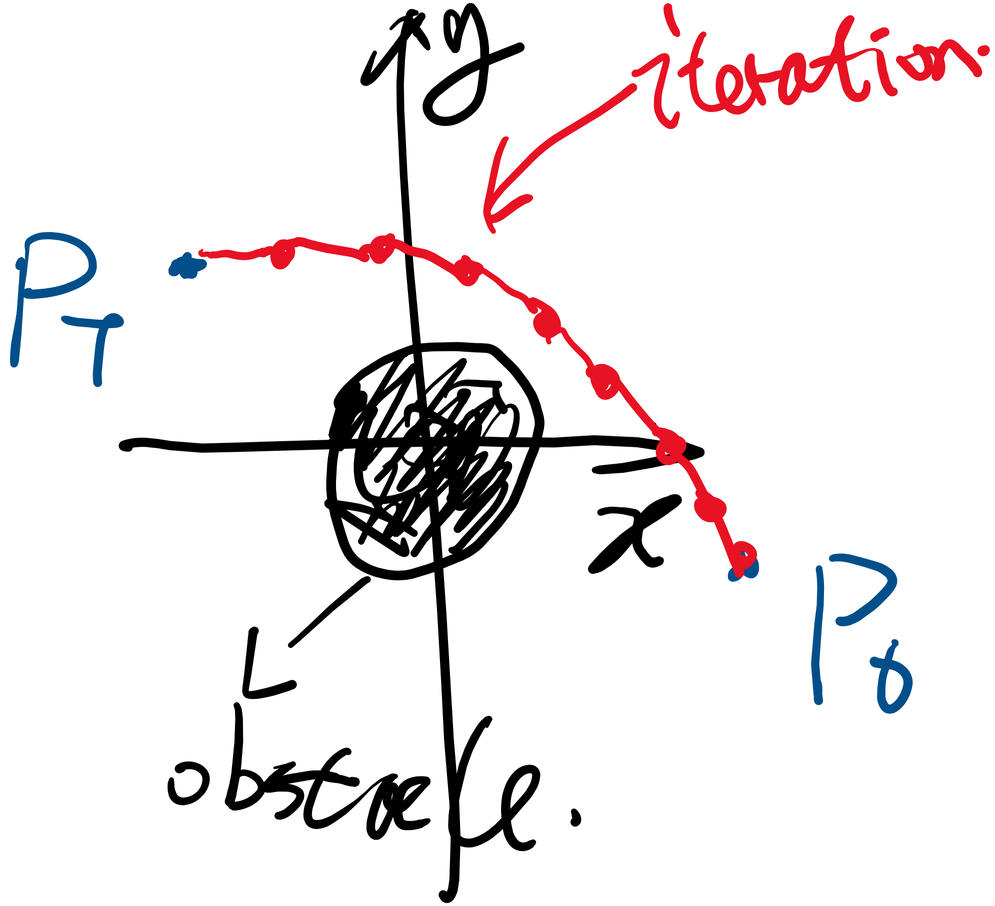


$$x^2+y^2\leq 1$$

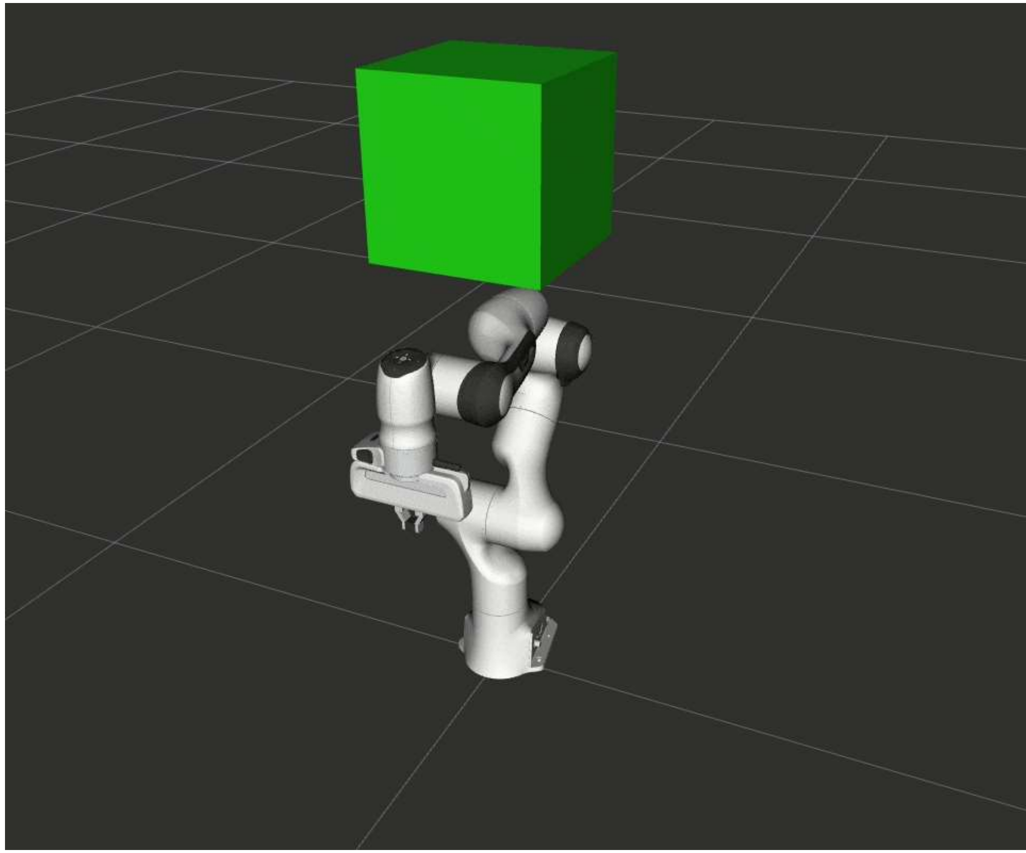
$$\min_{P_{0\dots T}} : \sum_n \|P_n - P_{n-1}\|^2 + \text{other cost}$$

$$\text{s.t.}: \quad \|P_n\| \geq 1 + \delta_{safe}.$$

Trajectory optimization (Example)



$x^2+y^2\leq 1$

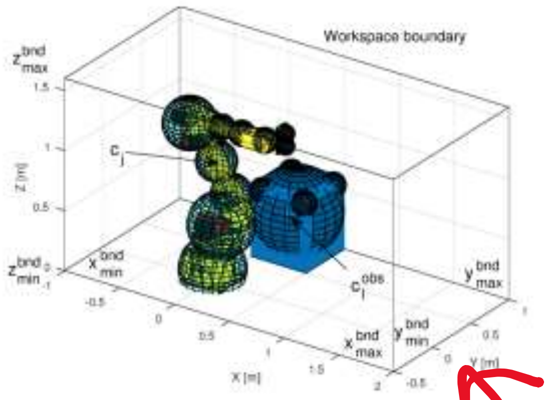


more complex constraints

Trajectory optimization

Efficient Trajectory Optimization for Robot Motion Planning

Yu Zhao, Hsien-Chung Lin, and Masayoshi Tomizuka



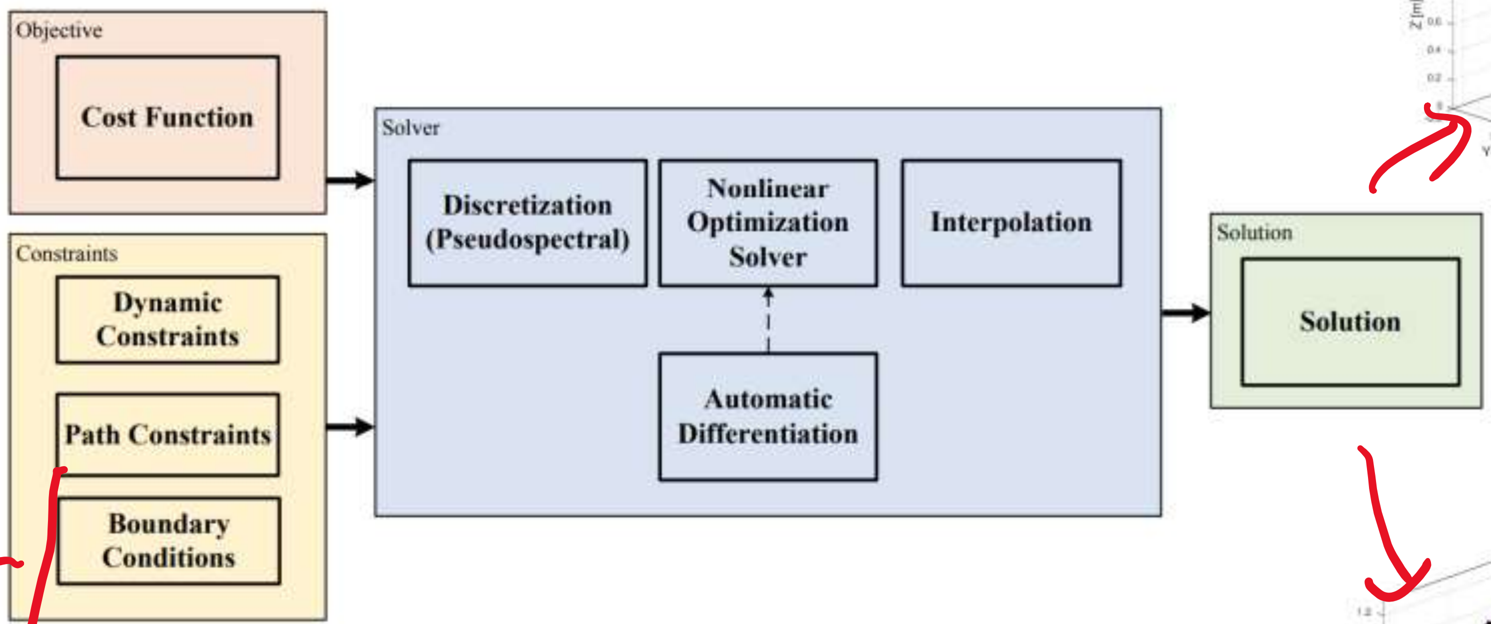
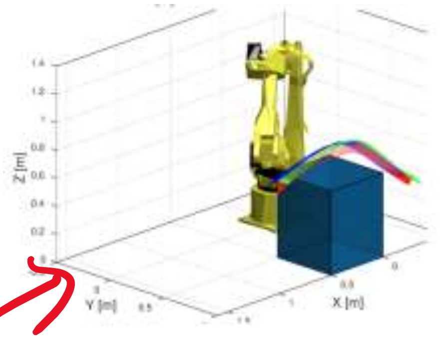
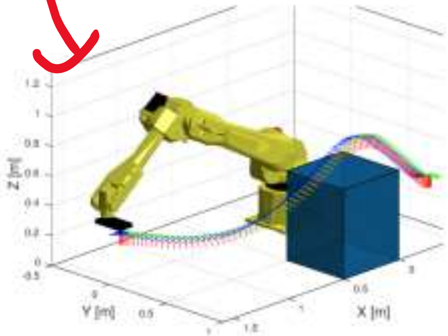


Fig. 2: Efficient numerical method for trajectory optimization





Position bounds:	$\boldsymbol{q}_{min} \leq \boldsymbol{q}(t) \leq \boldsymbol{q}_{max}$
Velocity bounds:	$\dot{\boldsymbol{q}}_{min} \leq \dot{\boldsymbol{q}}(t) \leq \dot{\boldsymbol{q}}_{max}$
Torque bounds:	$\boldsymbol{\tau}_{\min} \leq \boldsymbol{\tau}(t) \leq \boldsymbol{\tau}_{\max}$
Torque rate bounds:	$\dot{\boldsymbol{\tau}}_{\min} \leq \dot{\boldsymbol{\tau}}(t) \leq \dot{\boldsymbol{\tau}}_{\max}$

Trajectory optimization

STOMP: Stochastic Trajectory Optimization for Motion Planning

Mrinal Kalakrishnan[1] Sachin Chitta[2] Evangelos Theodorou[1] Peter Pastor[1] Stefan Schaal[1]

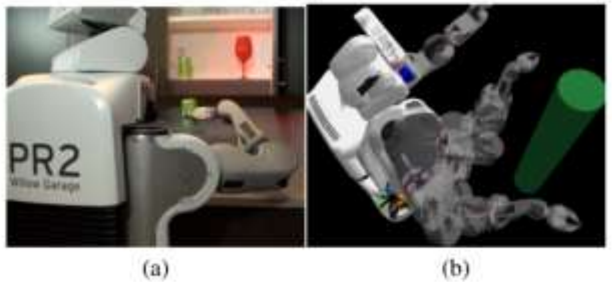


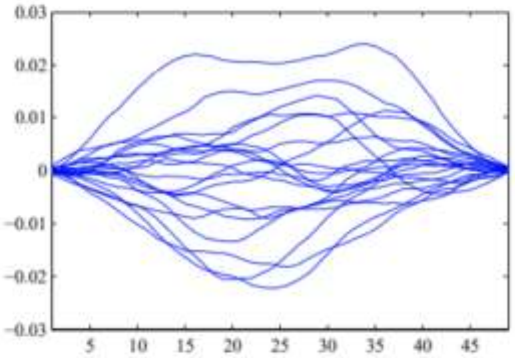
Fig. 1. (a) The Willow Garage PR2 robot manipulating objects in a household environment. (b) Simulation of the PR2 robot avoiding a pole in a torque-optimal fashion.

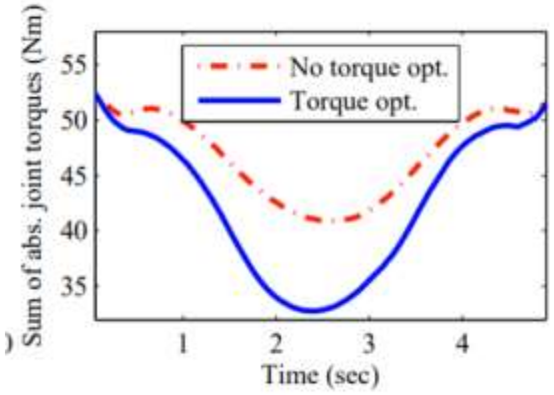
$$\min_{\tilde{\boldsymbol{\theta}}} \mathbb{E}\left[\sum_{i=1}^{N} q(\tilde{\boldsymbol{\theta}}_i) + \frac{1}{2}\tilde{\boldsymbol{\theta}}^{\mathsf{T}}\mathbf{R}\tilde{\boldsymbol{\theta}}\right]$$

STOMP is an algorithm that performs local optimization, i.e. it finds a locally optimum trajectory rather than a global one. Hence, performance will vary depending on the initial

sample locally

optimize. locally



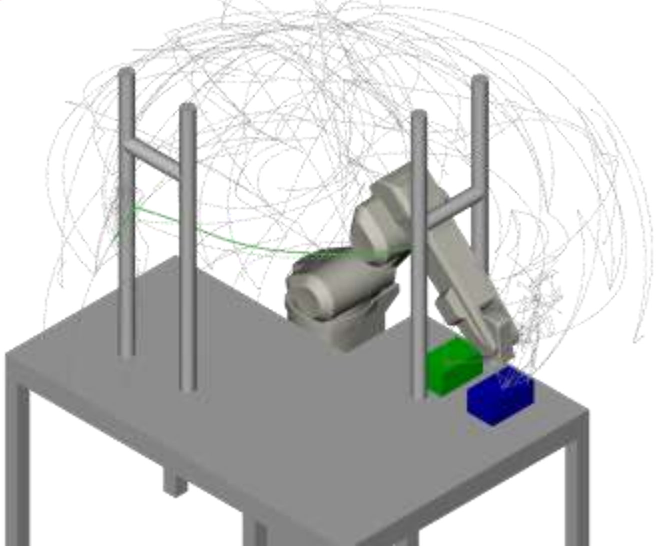


Trajectory optimization (reference)

- N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," 2009 IEEE International Conference on Robotics and Automation, May 2009, doi: 10.1109/robot.2009.5152817.
  < bib >
- J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization," Robotics: Science and Systems IX, Jun. 2013, doi: 10.15607/rss.2013.ix.031.
  < bib >
- S. Dai, S. Schaffert, A. Jasour, A. Hofmann, and B. Williams, "Chance Constrained Motion Planning for High-Dimensional Robots," 2019 International Conference on Robotics and Automation (ICRA), May 2019, doi: 10.1109/icra.2019.8793660.
  < bib >
- J. Carius, R. Ranftl, V. Koltun, and M. Hutter, "Trajectory Optimization With Implicit Hard Contacts," IEEE Robotics and Automation Letters, vol. 3, no. 4, pp. 3316–3323, Oct. 2018, doi: 10.1109/lra.2018.2852785.
  < bib >
- A. Wang, A. Jasour, and B. C. Williams, "Non-Gaussian Chance-Constrained Trajectory Planning for Autonomous Vehicles Under Agent Uncertainty," IEEE Robotics and Automation Letters, vol. 5, no. 4, pp. 6041–6048, Oct. 2020, doi: 10.1109/lra.2020.3010755.
  < bib >
- J. Tordesillas, B. T. Lopez, and J. P. How, "FASTER: Fast and Safe Trajectory Planner for Flights in Unknown Environments," 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nov. 2019, doi: 10.1109/iros40897.2019.8968021.
  < bib >
- R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "GuSTO: Guaranteed Sequential Trajectory optimization via Sequential Convex Programming," 2019 International Conference on Robotics and Automation (ICRA), May 2019, doi: 10.1109/icra.2019.8794205.
  < bib >
- R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-trees: Feedback Motion Planning via Sums-of-Squares Verification," The International Journal of Robotics Research, vol. 29, no. 8, pp. 1038–1052, Apr. 2010, doi: 10.1177/0278364910369189.
  < bib >
- Y. Shirai, X. Lin, Y. Tanaka, A. Mehta, and D. Hong, "Risk-Aware Motion Planning for a Limbed Robot with Stochastic Gripping Forces Using Nonlinear Programming," IEEE Robotics and Automation Letters, vol. 5, no. 4, pp. 4994–5001, Oct. 2020, doi: 10.1109/lra.2020.3001503.
  < bib >

Today's Summary

- **Drawback of Sampling-based approach (~5)**
- **Potential field method (~15)** ✓
  - **attractive, repulsive**
- **Gradient descent algorithm (~10)**
  - **vector field, velocity field, dynamic system** → Learning
- **Trajectory planning(~25)** ✓
  - **Parameter, joint space, cartesian space**
- **Planning as optimization (~20)** ✓
  - **Parameter, joint space, cartesian space**



Goal for this course

- **Design：soft hand design x1**
- **<u>Perception: vision, point cloud, tactile, force/torque x1</u>**
- **Planning: sampling-based, <u>optimization-based</u>, learning-based x3**
- **Control: feedback, multi-modal x2**
- **Learning: imitation learning, RL x2**
- **Simulation tool (pybullet, matlab, OpenRAVE, Issac Nvidia, Gazebo)**
- **How to get a robot moving!**

Goal for this course

- **Next course: course project kick-off**
- **Please download and play with pyBullet or any other robot simulator**
- **Choose one of the projects and find your group**
- **Make a detailed pipeline and the key milestones**
- **Please contact us directly if you have any questions**