Motion Planning and State Estimation in Robotics

Arun Kumar Singh

University of Tartu *arun.singh@ut.ee, aks1812@gmail.com*

October 5, 2021

Overview

Course Repo

Link to Course Repo
`https://github.com/arunkumar-singh/Motion_Planning_Lecture_Codes`

Kinematic Motion Model

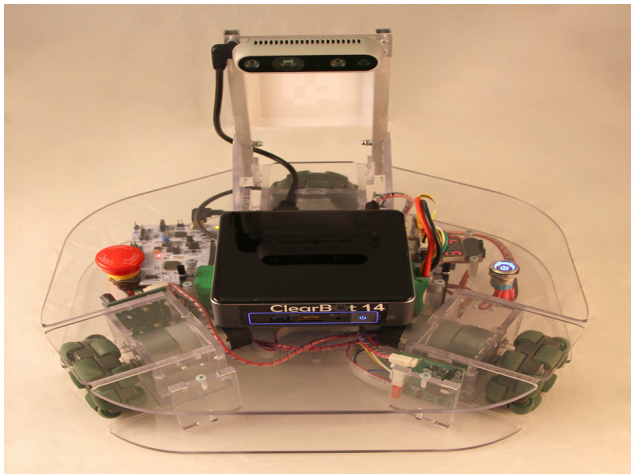
- A relation which allows you to map velocities, accelerations, jerks, snap to positions
- Simplest Relationship: Affine/Linear Relationships between velocities/accelerations/jerk and positions

Single Integrator System

Let us consider a mobile robot moving in the $X - Y$ plane and the robot is driven by velocities.

$$\dot{x} = v_x, \dot{y} = v_y \tag{1}$$

$$x(t) = v_x(t - t_0) + x_0, y(t) = v_y(t - t_0) + y_0 \tag{2}$$



(a)

Figure: Differential Drive Robot: P3DX

Motion Model Contd

Double Integrator Motion Model

Given a robot with position x_0, y_0, and velocity $(\dot{x}_0, \dot{y}_0)$

$$\ddot{x} = a_x, \ddot{y} = a_y \quad (3)$$

$$\dot{x} = \dot{x}_0 + a_x(t - t_0), \dot{y} = \dot{y}(t_0) + a_y(t - t_0) \quad (4)$$

$$x(t) = x(t_0) + \dot{x}_0(t - t_0) + \frac{1}{2}a_x(t - t_0)^2, y(t) = y(t_0) + \dot{y}_0(t - t_0) + \frac{1}{2}a_y(t - t_0)^2 \quad (5)$$

Motion Model Contd

Triple Integrator Motion Model

Given a robot with position x_0, y_0, and velocity $(\dot{x}_0, \dot{y}_0)$ and acceleration $(\ddot{x}_0, \ddot{y}_0)$

$$\dddot{x} = \mathrm{J}_x, \dddot{y} = \mathrm{J}_y \tag{6}$$

$$\ddot{x}(t) = \ddot{x}_0 + \mathrm{J}_x(t - t_0), \ddot{y} = \ddot{y}(t_0) + \mathrm{J}_y(t - t_0) \tag{7}$$

$$\dot{x}(t) = \dot{x}_0 + \ddot{x}_0(t - t_0) + \frac{1}{2}\mathrm{J}_x(t - t_0)^2, \dot{y}(t) = \dot{y}_0 + \ddot{y}_0(t - t_0) + \frac{1}{2}\mathrm{J}_y(t - t_0)^2 \tag{8}$$

$$x(t) = x_0 + \dot{x}_0 t + \frac{1}{2}\ddot{x}_0(t - t_0)^2 + \frac{1}{6}\mathrm{J}_x(t - t_0)^3 \tag{9}$$

$$y(t) = y_0 + \dot{y}_0 t + \frac{1}{2}\ddot{y}_0(t - t_0)^2 + \frac{1}{6}\mathrm{J}_y(t - t_0)^3 \tag{10}$$

Non-Holonomic Robots

A non-holonomic robot is driven by forward longitudinal velocity v and angular velocity (rate of change of heading angle) $\dot{\phi}$. Let the current position and heading angle be (x_0, y_0, ϕ_0). Then the velocity along $x - y$ direction is given by

$$\dot{x}(t) = v(\cos(\phi_0 + \dot{\phi}(t - t_0)), \dot{y}(t) = v(\sin(\phi_0 + \dot{\phi}(t - t_0)) \tag{11}$$

The position change is given by the following

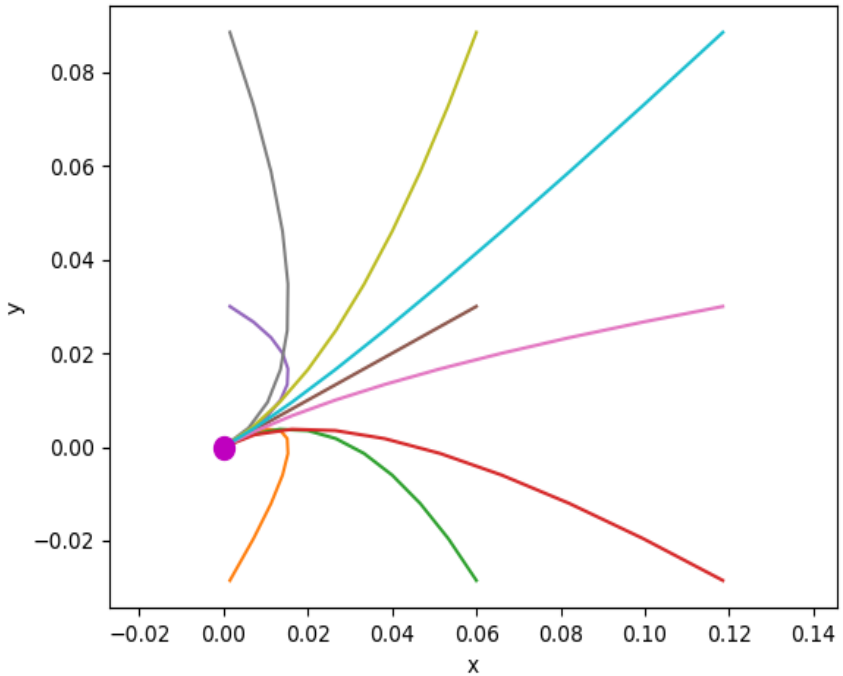
$$x(t) = x_0 + v(\cos(\phi_0 + \dot{\phi}(t - t_0))(t - t_0) \tag{12}$$

$$y(t) = y_0 + v(\sin(\phi_0 + \dot{\phi}(t - t_0))(t - t_0) \tag{13}$$

(a)

Motion Primitives

Double Integrator

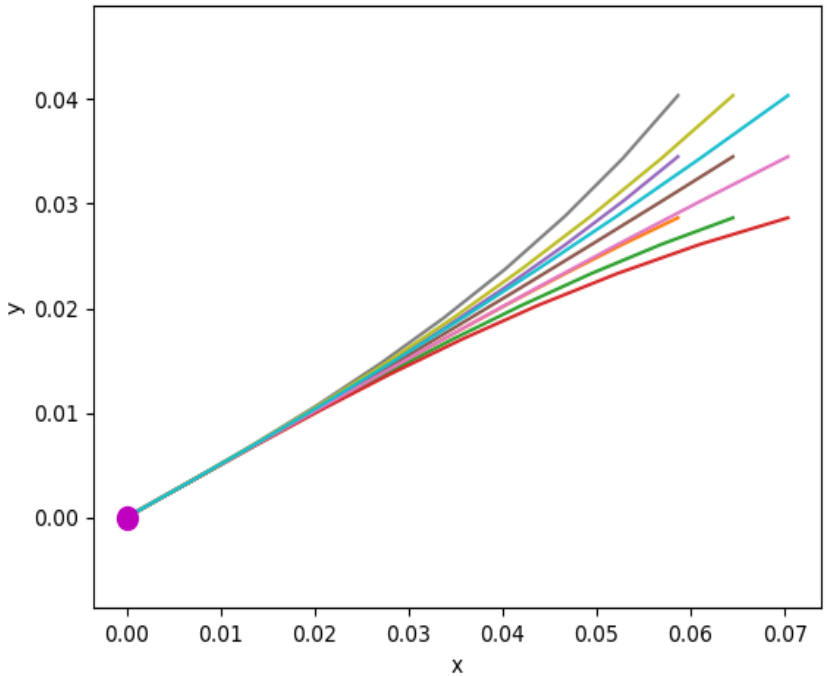


(a)

Figure: Motion primitives: Double Integrator System

Motion Primitives

Triple Integrator



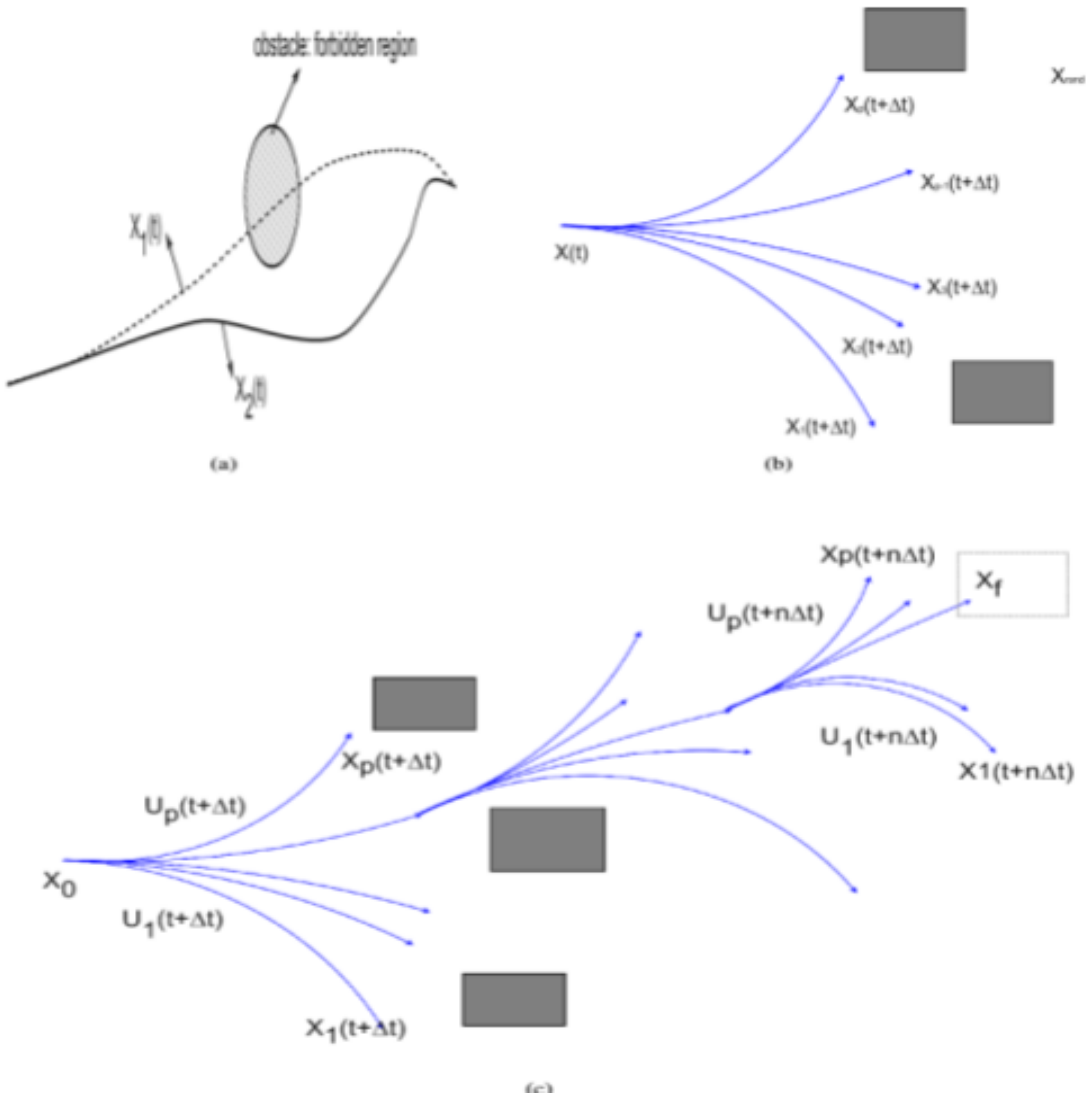
(a)

Figure: Motion primitives: Double Integrator System

Incremental/Reactive Planning

- Step 1. Use primitives to generate potential next state for the robot.
- Step 2: Identify the primitive with lowest (optimal) cost.
- Step 3: Use the lowest (optimal) cost primitive to move the robot.
- Redo (Re-plan) steps 1-3

A Cartoon for Incremental Planning



(a)

Notion of Cost

Kindly of common cost functions employed in motion planning

- Goal Reaching Cost: Measure of distance to the goal.

$$\text{Goal cost:}(x(t) - x_f)^2 + (y(t) - x_f)^2 \tag{14}$$

- Smoothness Cost: Can be defined in terms of norm of acceleration or jerk

$$\text{smoothness} = (a_x^2 + a_y^2) \tag{15}$$

$$\text{smoothness} = (j_x^2 + j_y^2) \tag{16}$$

$$\text{smoothness} = (v_x - v_x^{prev})^2 + (v_y - v_y^{prev})^2 \tag{17}$$

Common Cost Functions

- Collision Cost:

$$\text{Collision cost: } \max(0, f) \tag{18}$$

where f is a function that returns a positive number when in collision and negative otherwise.

- Desired Velocity Cost

$$(v_x - v_{des})^2 + (v_y - v_{des})^2 \tag{19}$$

- Tracking Cost

$$(x(t) - x_{des}(t))^2 + (y(t) - y_{des}(t))^2 \tag{20}$$

Reactive Planning for Goal Reaching in Free Space

Single Integrator Case

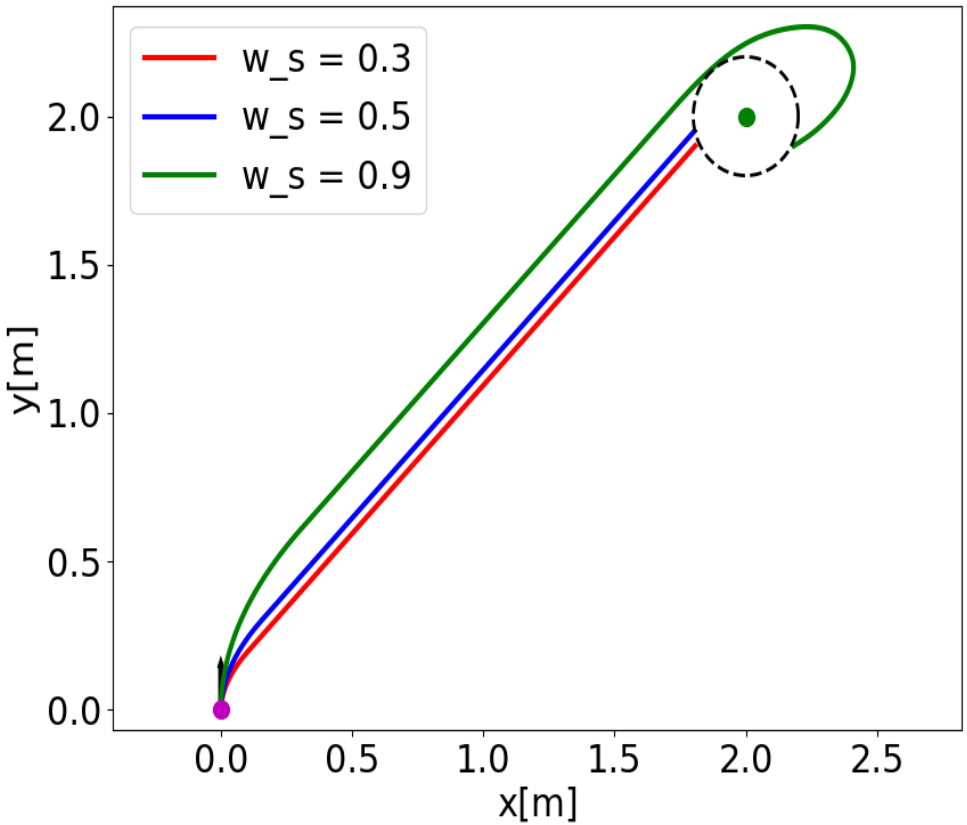
- Recall for single integrator system, we can directly change v_x, v_y
- Given a start position (x_0, y_0) and the final position (x_f, y_f), do the following

Generate n^2 samples of (v_x, v_y) from $[v_x^{min}, v_x^{max}] \times [v_y^{min}, v_y^{max}]$
for j,k = 1:n **do**
    $x(t) = x_0 + v_x(t - t_0), y(t) = y(t) + v_y(t - t_0))$
    Goal Cost: $\mathbf{C}_g = (x(t) - x_f)^2 + (y(t) - y_f)^2$

    Smoothness Cost: $\mathbf{C}_s = (v_x - v_x^{prev})^2 + (v_y - v_y^{prev})^2$
    $\mathbf{C}[j, k] = \mathbf{C}_g + w_s\mathbf{C}_s$

    $v_x, v_y = \arg\min \mathbf{C}[j, k]$
end for
Update initial position as $x_0 = x_0 + v_x(t - t_0), y_0 = y_0 + v_y(t - t_0)$
Repeat 1-7 till robot reaches goal i.e $(x_0 - x_f)^2 + (y_0 - y_f)^2 \leq \epsilon$

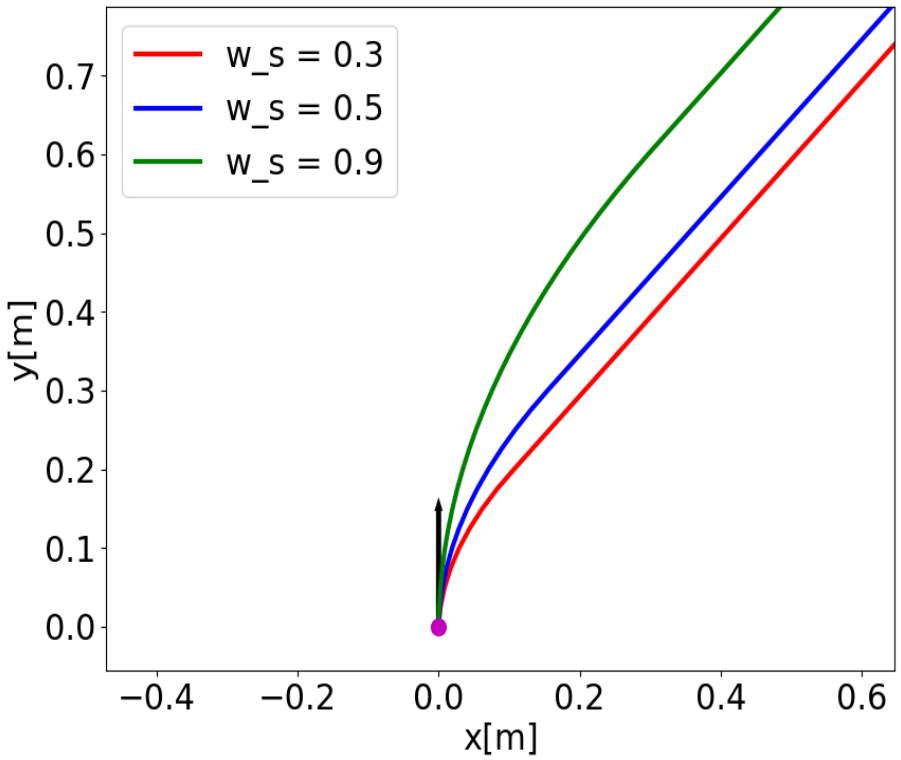
A Simple Example



(a)

Figure: Goal-reaching with different values of smoothness weight w_s.

A Simple Example



(a)

Figure: Goal-reaching with different values of smoothness weight w_s. With increase in w_s, the trajectories become smoother becoming more tangential to the initial velocity.

Reactive Planning for Trajectory Tracking in Free Space

Single Integrator Case

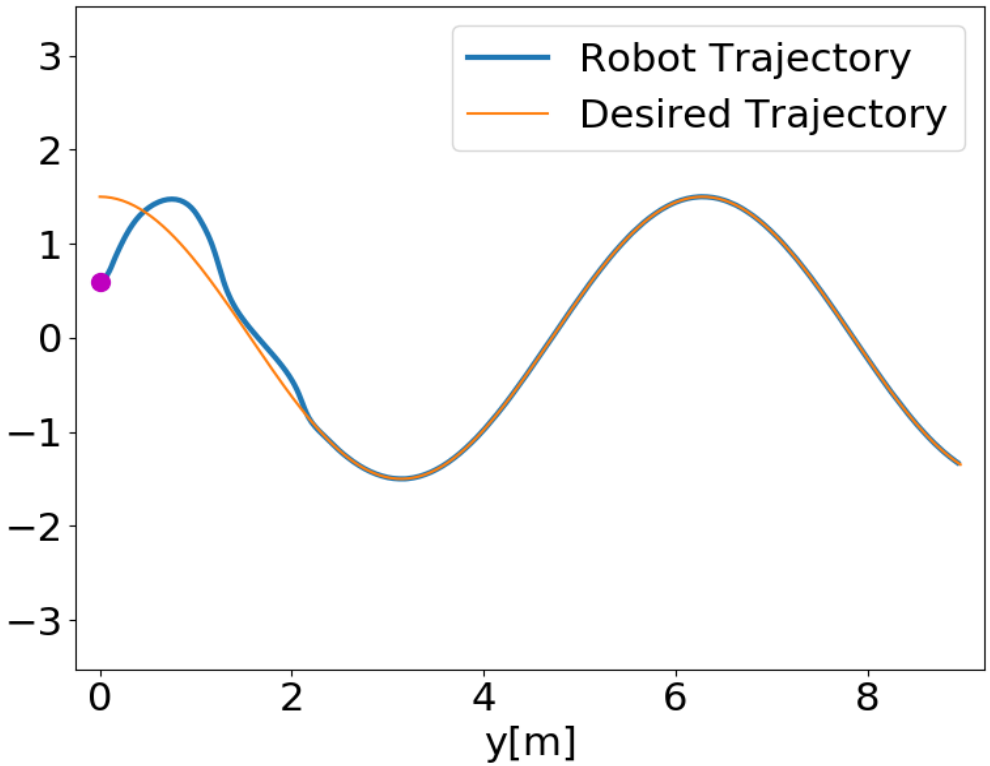
- Given a start position $x_0, y_0)$ and the final position (x_f, y_f), do the following.
- We need a time counter t

Generate n^2 samples of (v_x, v_y) from $[v_x^{min}, v_x^{max}] \times [v_y^{min}, v_y^{max}]$
Initialize a time counter t
for j,k = 1:n **do**
    Define $x_f = x_{des}(t), y_f = y_{des}(t)$
    $x(t) = x_0 + v_x(t - t_0), y(t_1) = y_0 + v_y(t - t_0)$
    Goal Cost: $\mathbf{C}_g = (x(t) - x_f)^2 + (y(t) - y_f)^2$

    Smoothness Cost: $\mathbf{C}_s = (v_x - v_x^{prev})^2 + (v_y - v_y^{prev})^2$
    $\mathbf{C}[j, k] = \mathbf{C}_g + w_s\mathbf{C}_s$

    $v_x, v_y = \arg\min \mathbf{C}[j, k]$
end for
Update initial position as $x_0 = x_0 + v_x(t - t_0), y_0 = y_0 + v_y(t - t_0)$
Update time counter $t = t + \Delta t$
Update $v_x^{prev} = v_x, v_y^{prev} = v_y$

Trajectory Tracking Example



(a)

Figure: A simple trajectory tracking example

Formalizing Incremental/Reactive Planning

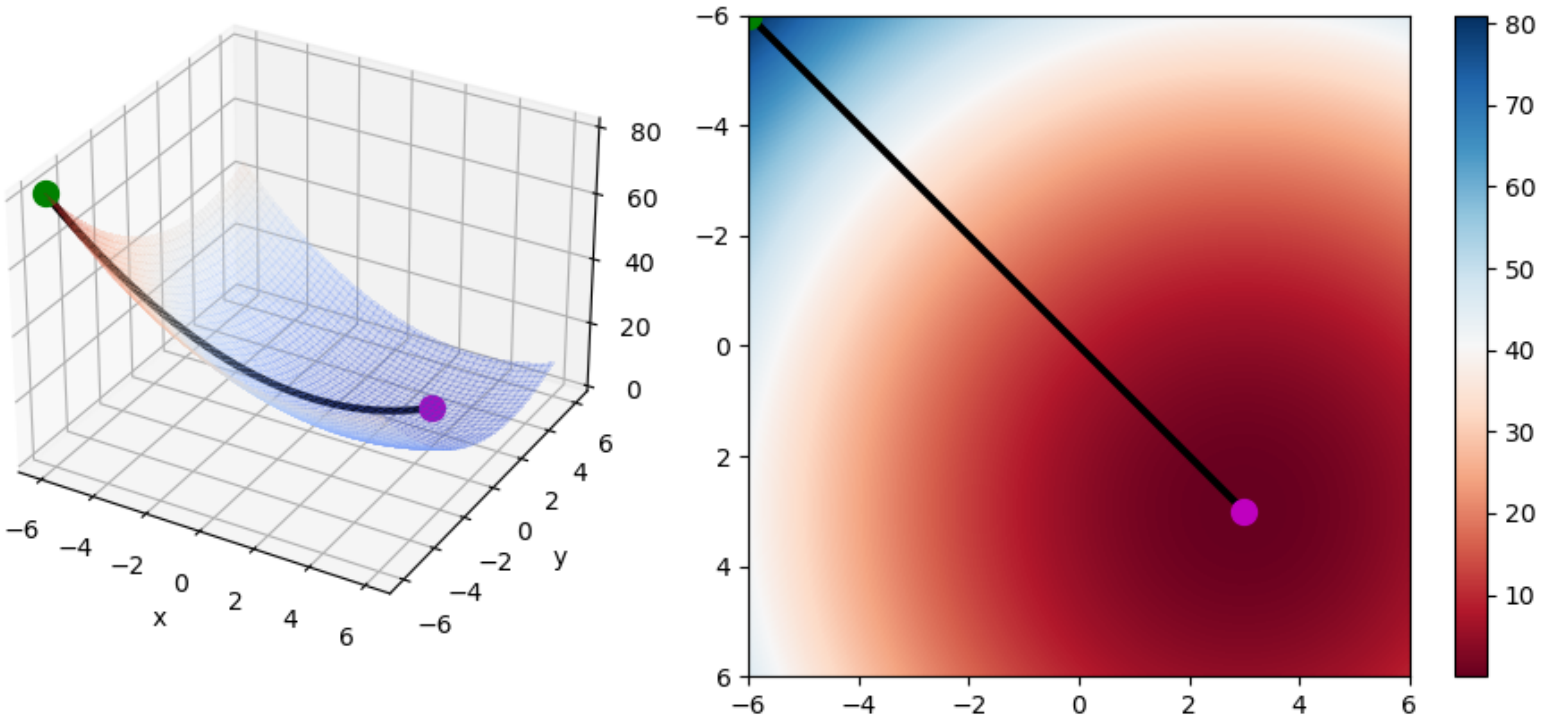
- In incremental planning, at each time instant, we are computing a velocity (or acceleration/jerk) that minimizes a cost function. For example, in simple goal reaching experiment, we are essentially solving the following optimization problem

$$\min_{x,y}(x - x_f)^2 + (y - y_f)^2 \tag{21}$$

$$\tag{22}$$

- In previous sections, we have just exhaustively tried several combination of v_x, v_y to compute the minimum of our cost function. Each (v_x, v_y) led to different values of (x, y) and we choose one which was gave the least value.
- We will now adopt a more sophisticated approach based on Gradient Descent.

Visualizing the Cost Surface



(a)

Figure: The goal-reaching cost has a bowl-like shape

Goal Reaching Through Gradient Descent

Define Goal Reaching Cost as

$$c(x, y) = (x - x_f)^2 + (y - y_f)^2 \tag{23}$$

Incremental/Reactive Planning through Gradient Descent

- Step 1. Compute velocity

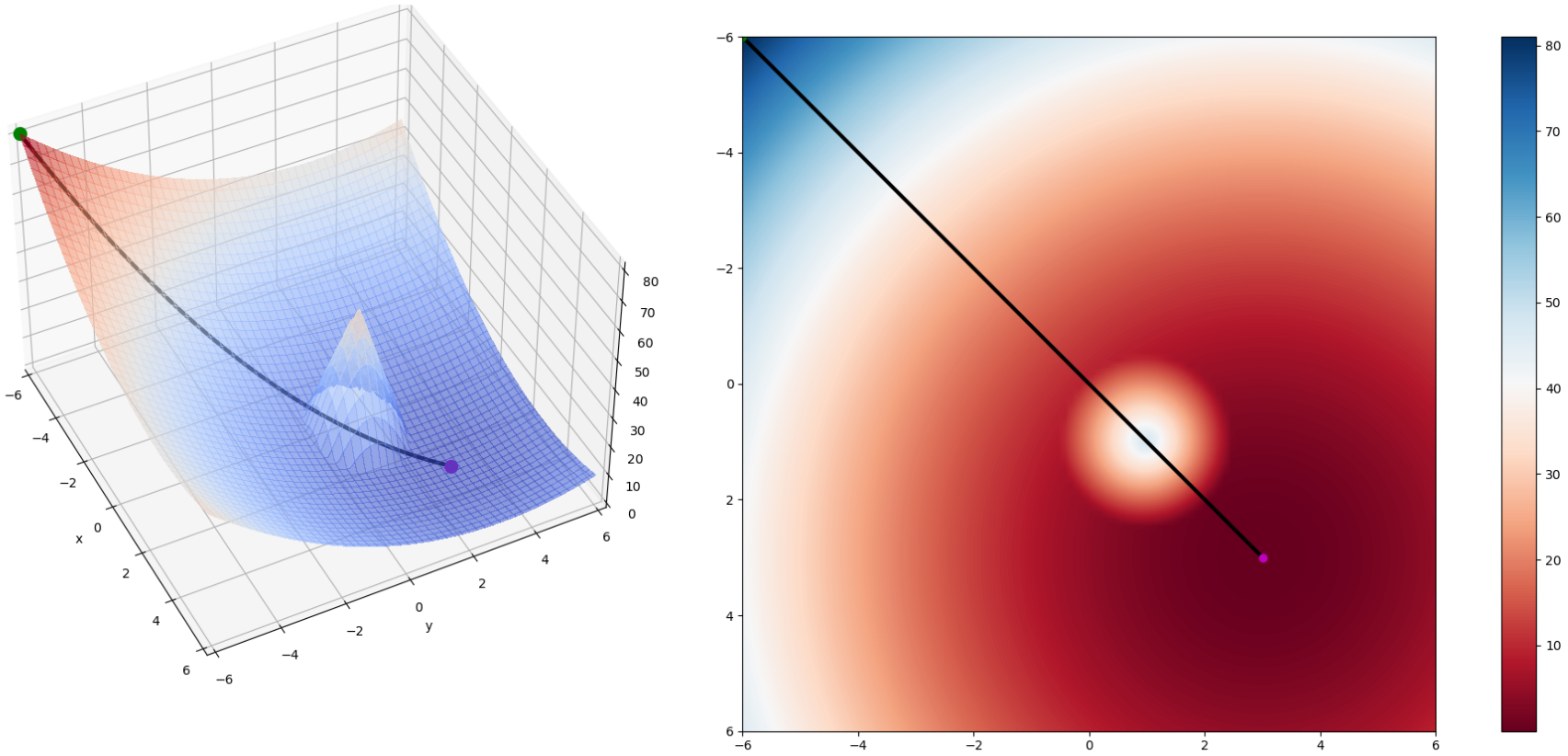
$$(v_x, v_y) = -\nabla c(x, y) \tag{24}$$

- Step 2. Update Position

$$x = x + v_x \Delta t, y = y + v_x \Delta t \tag{25}$$

The above method is known as Potential Field Method and the symbol ∇ means gradient

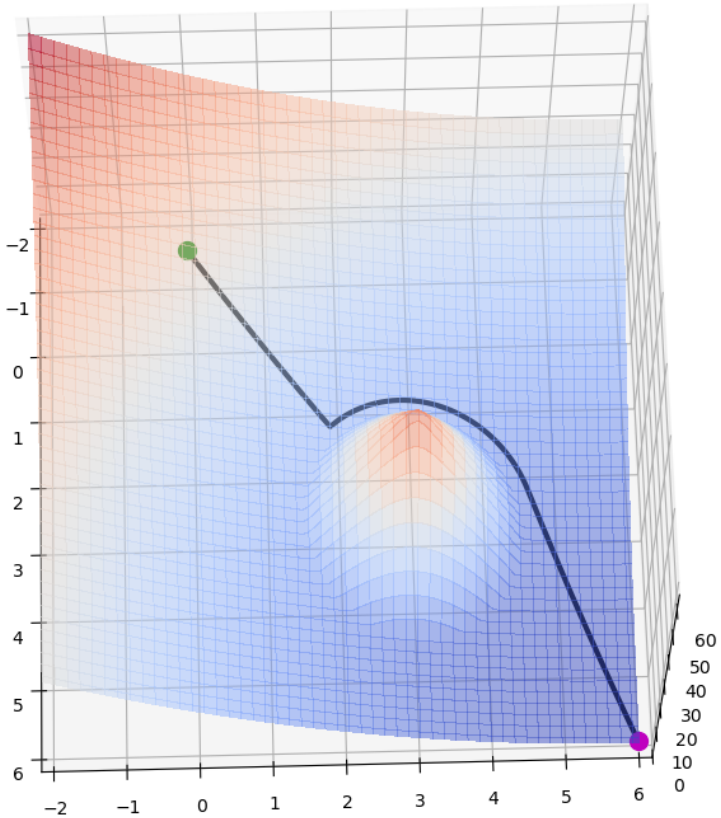
Bringing Obstacle Avoidance



(a)

Figure: Goal reaching cost overlaid with Obstacle Cost. Obstacles look like small hills and the trajectories are supposed to avoid passing through it.

Obstacle Avoidance: We want to avoid the hills on the cost surface



(a)

Figure: Goal reaching cost overlaid with Obstacle Cost. Obstacles look like small hills and the trajectories are supposed to avoid passing through it.

Obstacle Cost/Potential

- Obstacle Potential:

$$c_o(x,y) = \eta(\frac{1}{d(x,y)} - \frac{1}{d_0})^2, \text{if}, d(x,y) <= d_0 \tag{26}$$

$$c_o(x,y) = 0, \text{if}, d(x,y) > d_0 \tag{27}$$

$$d(x,y) = \sqrt{(x - x_o^i)^2 + (y - y_o^i)^2} \tag{28}$$

  where, (x_o^i, y_o^i) is the position of the i^{th} obstacle.

- Alternate obstacle potential

$$c_o(x,y) = max(0, -d(x,y) + d_0)) \tag{29}$$

Motion Planning based on Potential Field

- Combined Potential:

$$c(x, y) = c_g(x, y) + c_o(x, y) \tag{30}$$

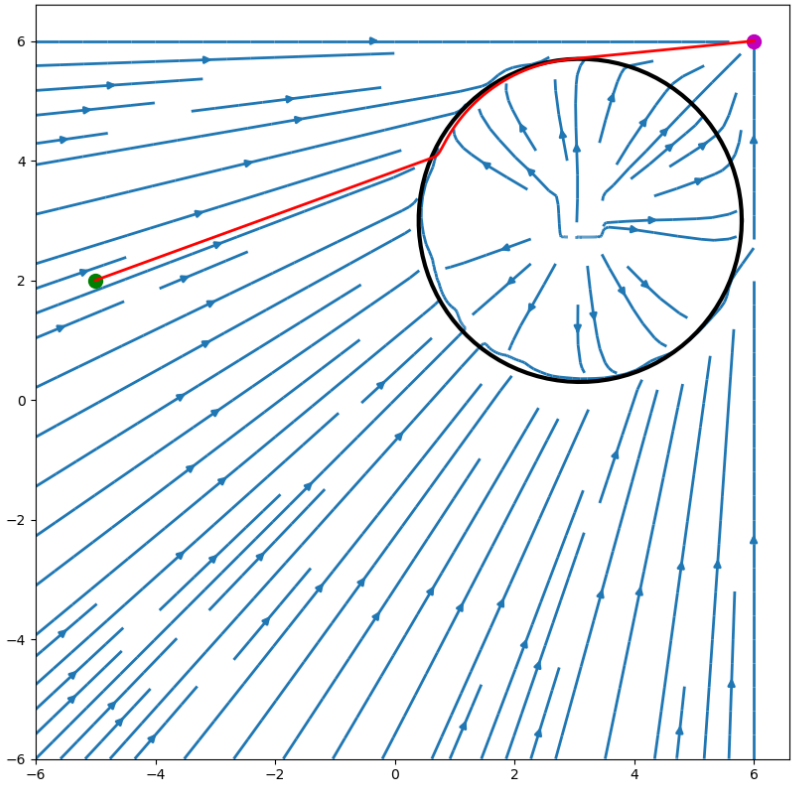
- Step 1. Compute velocity

$$(v_x, v_y) = -\nabla c(x, y) \tag{31}$$

- Step 2. Update Position

$$x = x + v_x \Delta t, y = y + v_x \Delta t \tag{32}$$

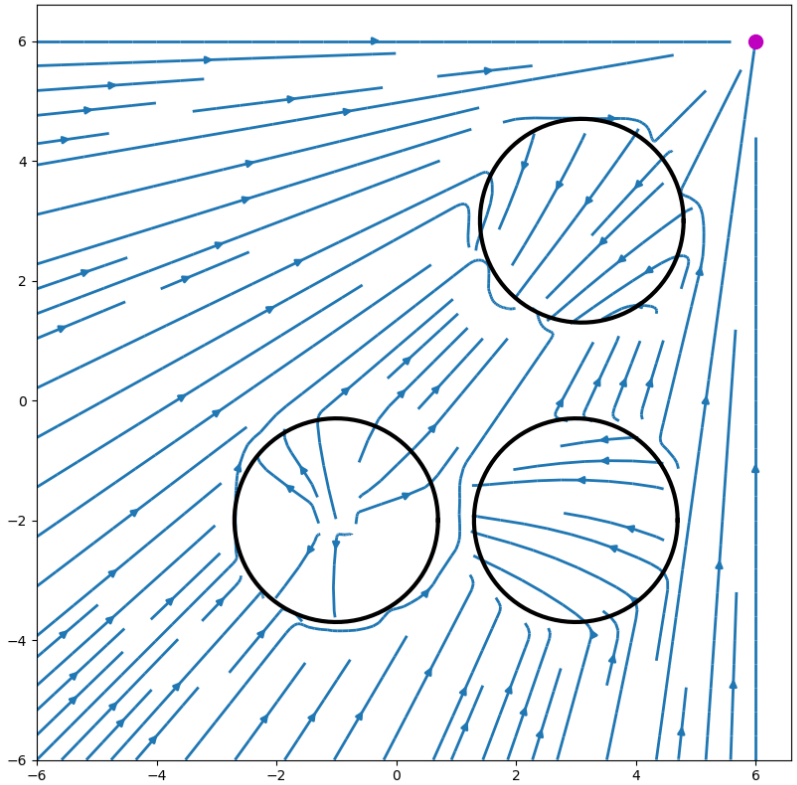
Gradient-Field Visualization of Potential Field



(a)

Figure: Gradient Field Visualization for the combined cost/potential function. As shown, the robot path follows the gradient direction.

Extension to Multiple Obstacles



(a)

Figure: Gradient Field Visualization for the combined cost/potential function for multiple obstacle case.

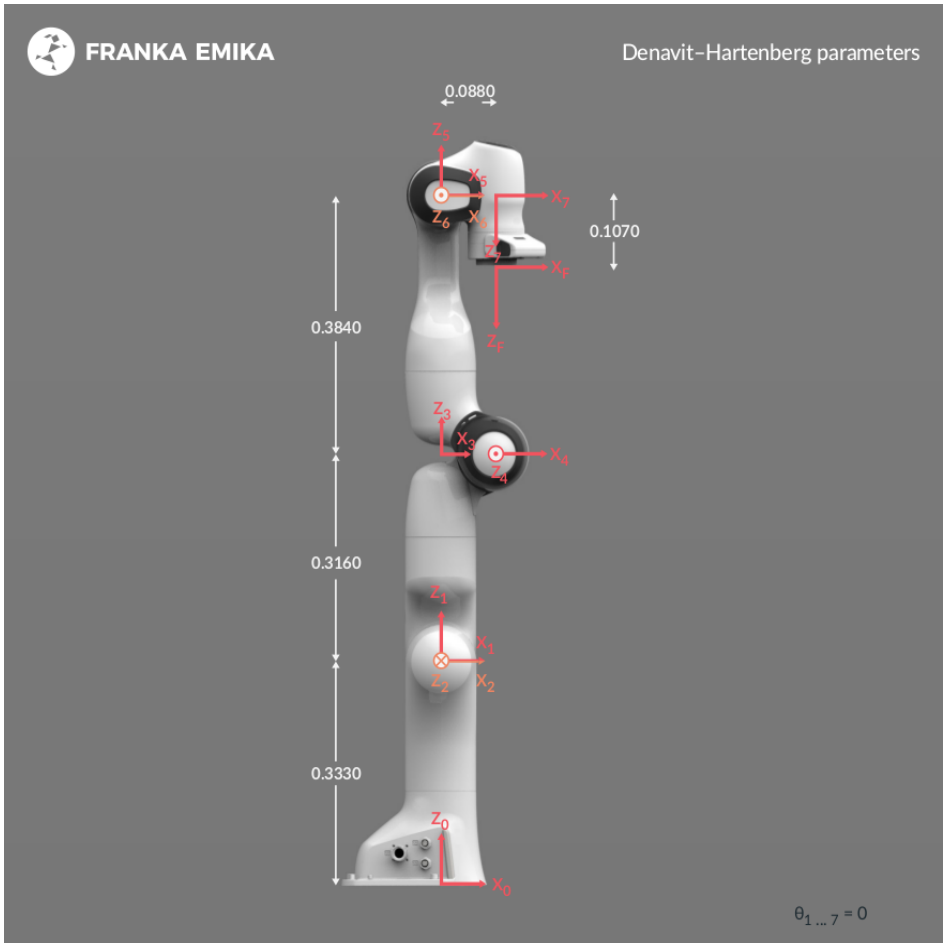
Gradient Based Approach for More General Cost Functions

- We can use gradient based method with more general class of explicitly dependent cost functions.
- The general concept is called Gradient Descent and is the most basic method for computing the minimum of a cost function.

Main Issues with Potential Field

- Finding the distance to the closest obstacle is tricky for complex obstacle shape.
- The gradient for each obstacle and goal can conflict with each other creating a local minima.

Manipulator Kinematics



(a)

Figure: Panda Arm from Franka Emica: Courtesy
`https://frankaemika.github.io/docs/control_parameters.html`

Joint Space Planning Vs Task Space planning

- Joint Space Planning: Planning a trajectory between $\theta_i(t_0)$ to $\theta_i(t_f)$
- Task Space Planning: Planning a trajectory between $x(t_0), y(t_0), z(t_0), \alpha(t_0), \beta(t_0), \gamma(t_0)$

Joint Space Planning is easy. However, task-space planning is difficult and still and active area of research.

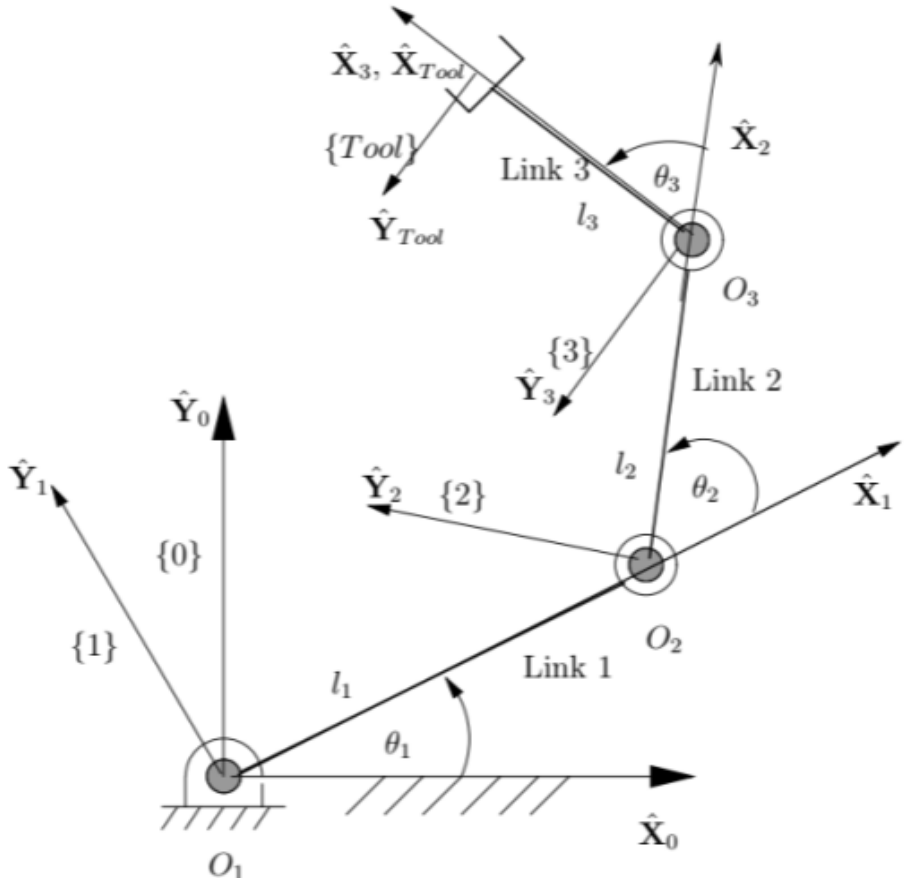
Manipulator Kinematics

- Let $\theta_1, \theta_2, \ldots, \theta_n$ be manipulator joints
- Let x, y, z be end effector position
- Let α, β, γ be roll, pitch, yaw of the end-effector

Forward Kinematics of the manipulator refers to the following mapping from θ_i to $x, y, z, \alpha, \beta, \gamma$

$$\begin{bmatrix} x \\ y \\ z \\ \alpha \\ \beta \\ \gamma \end{bmatrix} = \mathbf{f}_{fk}(\theta_1, \theta_2, \ldots, \theta_n) = \begin{bmatrix} f_{fk}^1(\theta_1, \theta_2, \ldots, \theta_n) \\ f_{fk}^2(\theta_1, \theta_2, \ldots, \theta_n) \\ f_{fk}^3(\theta_1, \theta_2, \ldots, \theta_n) \\ f_{fk}^4(\theta_1, \theta_2, \ldots, \theta_n) \\ f_{fk}^5(\theta_1, \theta_2, \ldots, \theta_n) \\ f_{fk}^6(\theta_1, \theta_2, \ldots, \theta_n) \end{bmatrix} \tag{33}$$

Forward Kinematics: Planar Manipulator



(a)

Figure: A 3-R planar manipulator. Image courtesy: Ashitava Ghosal's Robotic Course

Planar Manipulator Forward Kinematics

$$\overbrace{x(\boldsymbol{\theta}) = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3)}^{f_{fk}^1} \tag{34}$$

$$\overbrace{y(\boldsymbol{\theta}) = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3)}^{f_{fk}^2} \tag{35}$$

$$\overbrace{\gamma = \theta_1 + \theta_2 + \theta_3}^{f_{fk}^6} \tag{36}$$

where, $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$. The forward kinematics function has only three components for the planar manipulator.

Planar Manipulator Inverse Kinematics

IK Problem: For a given (x_f, y_f, γ_f), compute the associated joint angle $\theta_1, \theta_2, \theta_3$.

- Mathematically, this means function inversion
  For example,

$$\boldsymbol{\theta} = (f_{fk}^1)^{-1}(x) \tag{37}$$

- However, direct inversion can only be done in very special cases based on purely geometrical calculation.
- A more general approach is to do something called implicit function inversion in the form of an optimization problem. We already know, one of them called Potential Field based on Gradient Descent.

IK as an Optimization Problem: Potential Field Approach

Let us define the following

$$c_g = \frac{1}{2}((f_{fk}^1(\boldsymbol{\theta}) - x)^2 + (f_{fk}^2(\boldsymbol{\theta}) - y)^2 + (f_{fk}^3(\boldsymbol{\theta}) - \gamma_f)^2) \tag{38}$$

c_g as defined in (38) is nothing but the goal potential that we used earlier. Thus, we can solve the IK problem using the concepts from potential field

- Define

$$\dot{\boldsymbol{\theta}} = -\nabla c_g \tag{39}$$

-

$$\begin{aligned} \boldsymbol{\theta} &= \boldsymbol{\theta} + \dot{\boldsymbol{\theta}}\Delta t \\ \Rightarrow \boldsymbol{\theta} &= \boldsymbol{\theta} - \nabla c_g \Delta t \end{aligned} \tag{40}$$

Jacobian: Planar Manipulator

Differentiating our expressions for x, y, γ with respect to time, we get

$$\dot{x} = -l_1 \sin(\theta_1)\dot{\theta}_1 - l_2 \sin(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) - l_3 \sin(\theta_1 + \theta_2 + \theta_3)(\dot{\theta}_1 + \dot{\theta}_2 + \dot{\theta}_3) \quad (41)$$

$$\dot{y} = l_1 \cos(\theta_1)\dot{\theta}_1 + l_2 \cos(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3)(\dot{\theta}_1 + \dot{\theta}_2 + \dot{\theta}_3) \quad (42)$$

$$\dot{\gamma} = \dot{\theta}_1 + \dot{\theta}_2 + \dot{\theta}_3 \quad (43)$$

The above equations can be put in a matrix form

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\gamma} \end{bmatrix} = \mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \quad (44)$$

where as before, we have

$$\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3) \quad (45)$$

$$\dot{\boldsymbol{\theta}} = (\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3) \quad (46)$$

Jacobian: Planar Manipulator

$$\mathbf{J}(\theta) = \begin{bmatrix} -l_1 s_1 - l_2 s_{12} - l_3 s_{123} & -l_2 s_{12} - l_3 s_{123} & -l_3 s_{123} \\ l_1 c_1 + l_2 c_{12} + l_3 c_{123} & l_2 c_{12} + l_3 c_{123} & l_3 c_{123} \\ 1 & 1 & 1 \end{bmatrix} \tag{47}$$

where

$$s_{ijk} = \sin(i) + \sin(j) + \sin(k) \tag{48}$$

Task Space to Joint Space Mapping through Jacobian

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{-1}(\boldsymbol{\theta}) \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\gamma} \end{bmatrix} \tag{49}$$

The above equation tells us how to generate a particular joint motion to realize the given end-effector motion.

What if $\mathbf{J}$ is not square? One option is to go for pseudo inverse which will give the minimum norm solution.

IK through Jacobian Inverse

IK Problem: For a given (x_f, y_f, γ_f), compute the associated joint angle $\theta_1, \theta_2, \theta_3$.

- Step 1. Define $\dot{x} = k(x_f - x), \dot{y} = (y_f - y), \dot{\gamma} = (\gamma_f - \gamma)$
- Step 2: Compute

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{-1}(\boldsymbol{\theta}) \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\gamma} \end{bmatrix} \tag{50}$$

- Update joint position

$$Step3 : \boldsymbol{\theta} = \boldsymbol{\theta} + \dot{\boldsymbol{\theta}} \Delta t \tag{51}$$

- Repeat steps 1 to 3 till goal is reached.

Basics of 3D transformation

Consider two reference frames $\{0\}$ and $\{1\}$ and a vector p defined in $\{1\}$. The vector ${}^{1}\mathbf{p}$ in the reference frame $\{0\}$ would be

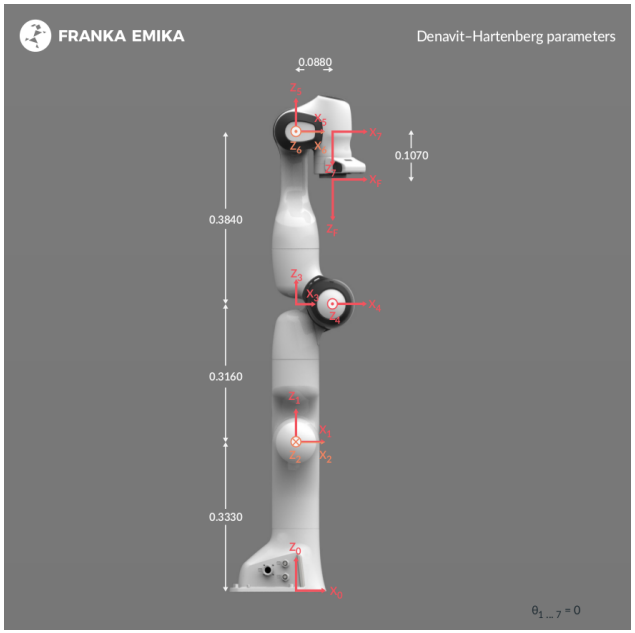
$$ {}^{0}\mathbf{p} = {}^{0}\mathbf{T}_{1}\,{}^{1}\mathbf{p} \tag{52} $$

where

$$ {}^{0}\mathbf{T}_{1}{}^{1} = \begin{bmatrix} {}^{0}\mathbf{R}_{1}{}^{1} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{53} $$

Task Space Planning: Forward Kinematics

The forward kinematics function is obtained by multiplying Transformation matrices



(a)

Figure: Panda Arm from Franka Emica: Courtesy
`https://frankaemika.github.io/docs/control_parameters.html`

$$^{0}\mathbf{T}_{n} = {}^{0}\mathbf{T}_{1}{}^{1}\mathbf{T}_{2}{}^{3}\mathbf{T}_{4}\ldots{}^{n-1}\mathbf{T}_{n} \tag{54}$$

General Form of Transformation Matrices

$$ {}^{i-1}\mathbf{T}_i = \begin{bmatrix} c(\theta_i) & -s(\theta_i) & 0 & a_{i-1} \\ s(\theta_i)c(\alpha_{i-1}) & c(\theta_i)c(\alpha_{i-1}) & -s(\alpha_{i-1}) & -s(\alpha_{i-1})d_i \\ s(\theta_i)s(\alpha_{i-1}) & c(\theta_i)s(\alpha_{i-1}) & c(\alpha_{i-1}) & c(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{55} $$

where $c(\theta_i) = \cos(\theta_i), s(\theta_i) = \sin(\theta_i), c(\alpha_i) = \cos(\alpha_i), s(\alpha_i) = \sin(\alpha_i)$

Joint	a (m)	d (m)	α (rad)	θ (rad)
Joint 1	0	0.333	0	θ_1
Joint 2	0	0	$-\frac{\pi}{2}$	θ_2
Joint 3	0	0.316	$\frac{\pi}{2}$	θ_3
Joint 4	0.0825	0	$\frac{\pi}{2}$	θ_4
Joint 5	-0.0825	0.384	$-\frac{\pi}{2}$	θ_5
Joint 6	0	0	$\frac{\pi}{2}$	θ_6
Joint 7	0.088	0	$\frac{\pi}{2}$	θ_7
Flange	0	0.107	0	0

(a)

Figure: DH Table Franka: Courtesy `https://frankaemika.github.io/docs/control_parameters.html`

Forward Kinematics

- The last three rows of ${}^0\mathbf{T}_n$ gives you the position of the end-effector. Lets call it $\mathbf{t} = (t_x, t_y, t_z)$
- The 3×3 sub-matrix of ${}^0\mathbf{T}_n$ define the orientation.
- Let $\mathbf{R}$ represent the 3×3 rotation sub-matrix of ${}^0\mathbf{T}_n$. Then the roll (α), pitch (β) and (γ)

$$\alpha = \arctan 2 \frac{\mathbf{R}_{3,2}}{\mathbf{R}_{3,3}} \tag{56}$$

$$\beta = -\arcsin(\mathbf{R}_{3,1}) \tag{57}$$

$$\gamma = -\arctan 2 (\frac{\mathbf{R}_{2,1}}{\mathbf{R}_{1,1}}) \tag{58}$$

IK for Franka Panda Manipulator

We can use the graident based technique of potential field to compute the IK for the Franka Panda Arm as well.

$$c_g(\boldsymbol{\theta}) = (t_x - x_f)^2 + (t_y - y_f)^2 + (t_z - z_f)^2 + (\alpha - \alpha_f) + (\beta - \beta_f) + (\gamma - \gamma_f) \tag{59}$$

We will use the following update equation to update $\boldsymbol{\theta}$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \nabla c_g \Delta t \tag{60}$$

Assignment 1

- Construct a potential field method for path-planning among 4 obstacles. You need to run your codes between given pairs of start and goal locations.
- Extend the potential field method to 3D obstacle avoidance. The obstacles are modeled as spheres and the coordinates of the obstacle and start and goal locations will be given.