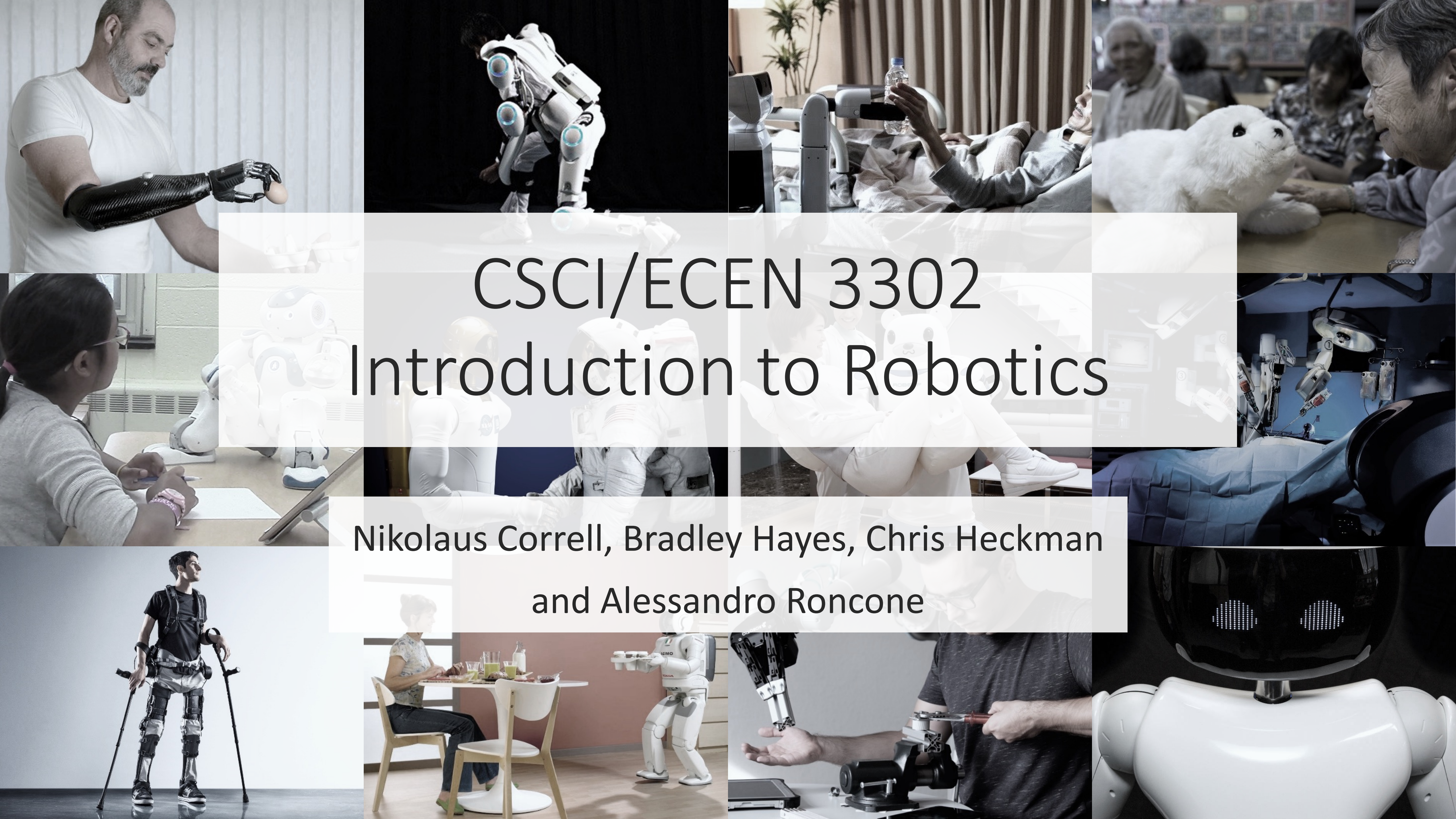


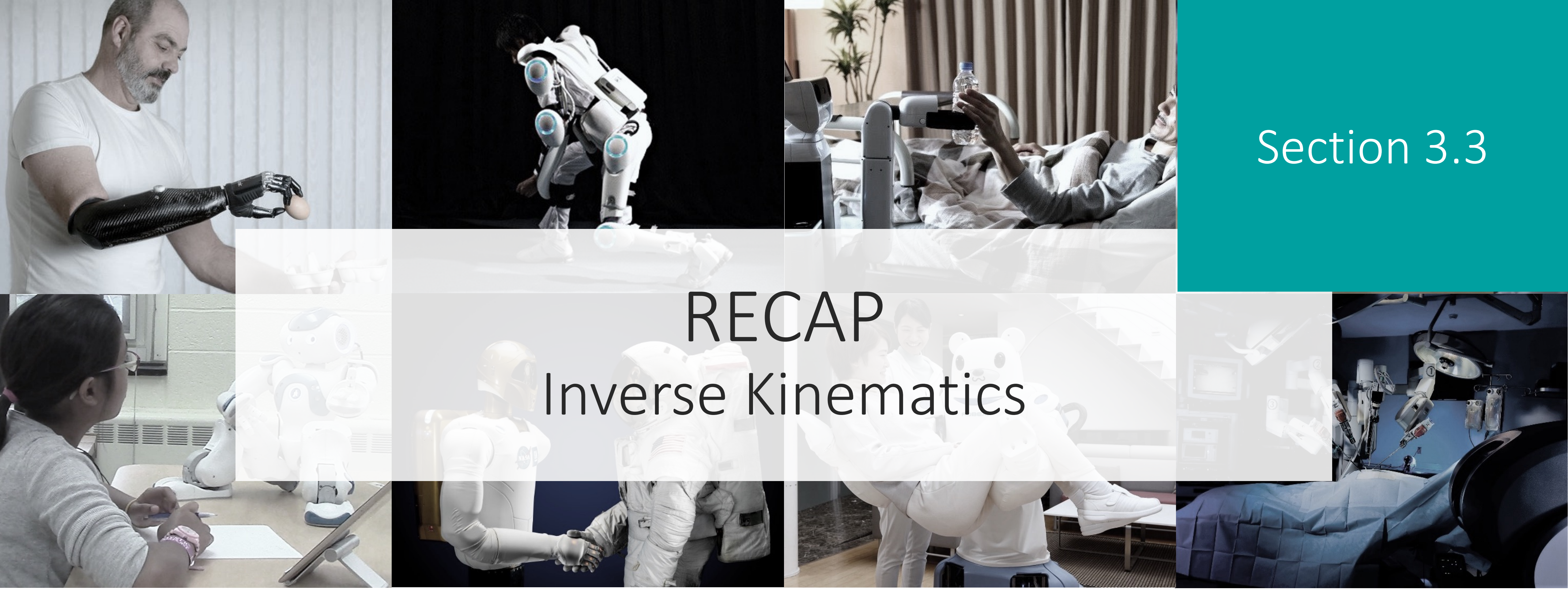
CSCI/ECEN 3302 Introduction to Robotics

Nikolaus Correll, Bradley Hayes, Chris Heckman
and Alessandro Roncone

Administrivia

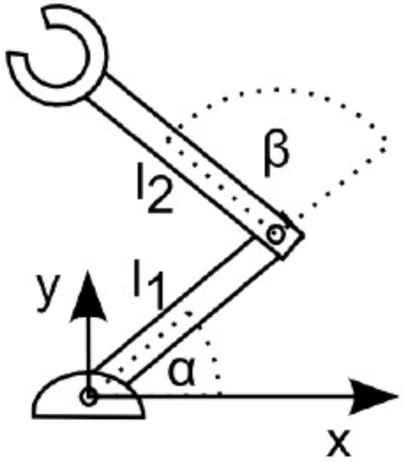
- Lab 2 deadline tonight
- Quiz 5
- Tomorrow (Wednesday):
  - Lab 3: inverse kinematics

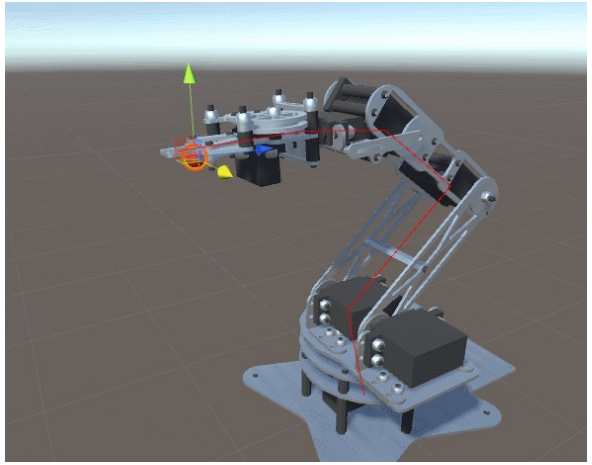
Questions?



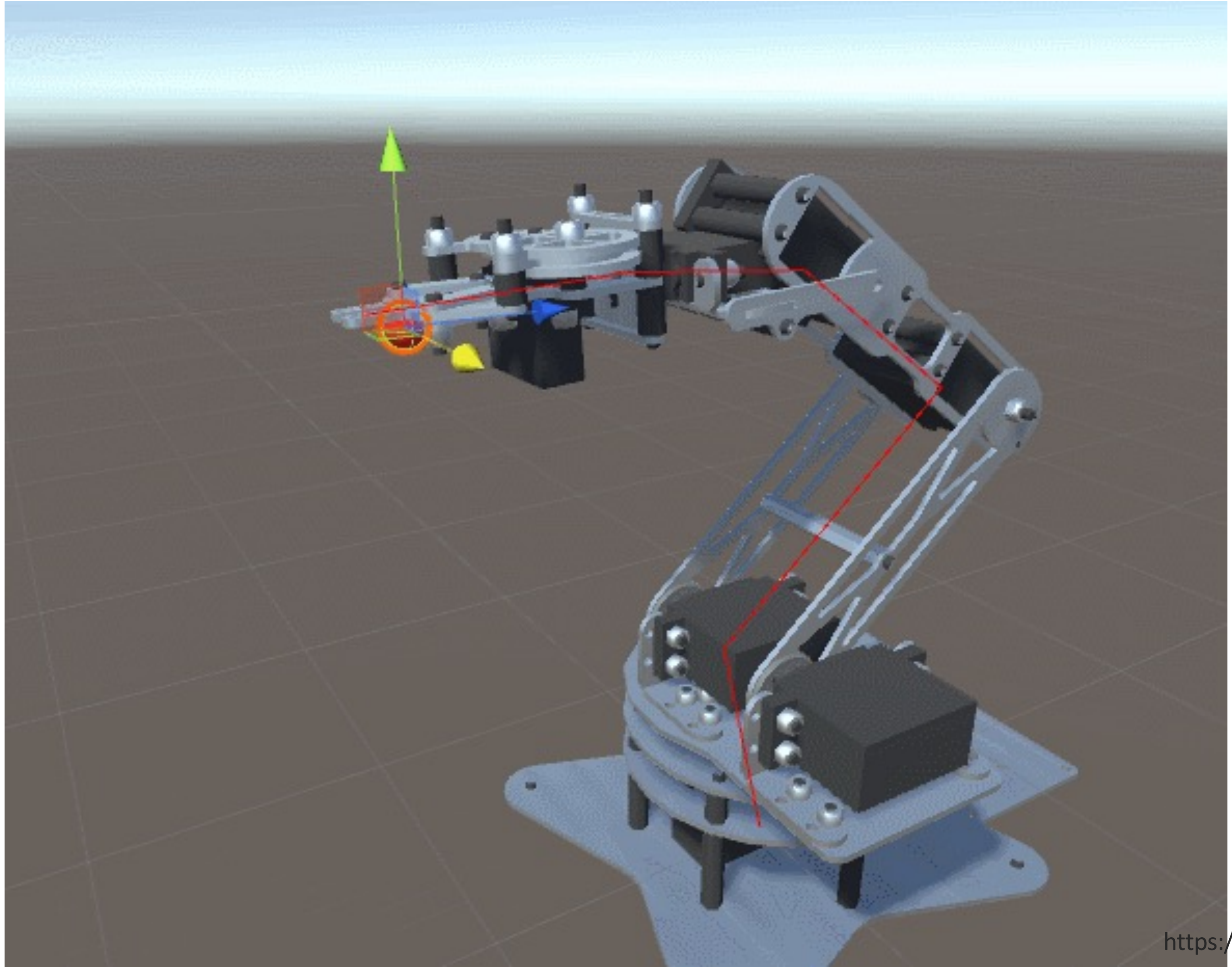
Section 3.3

RECAP
Inverse Kinematics



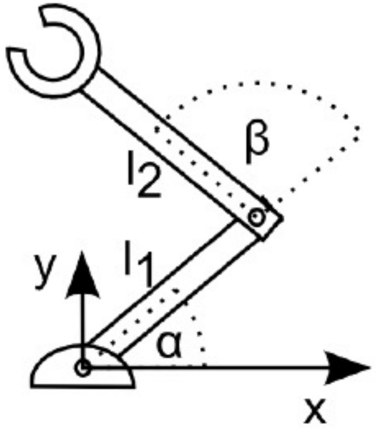


End-effector Position Control

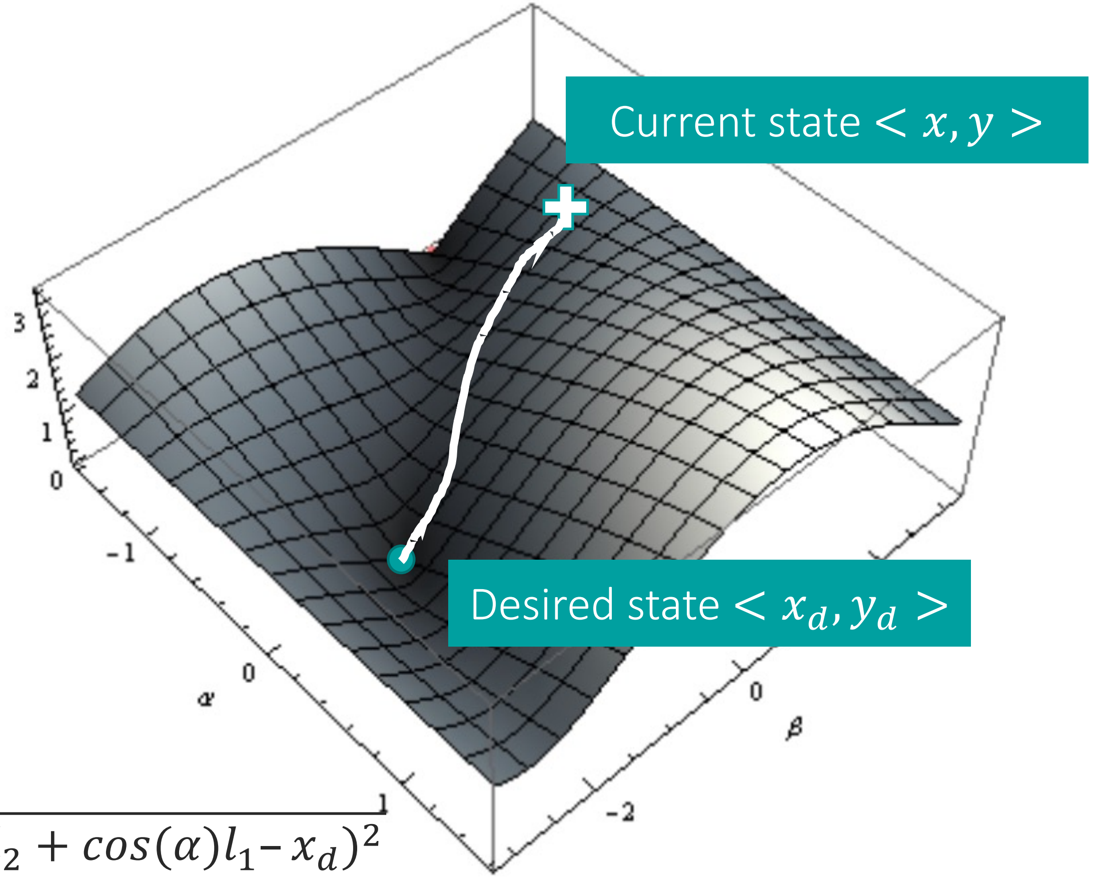


https://www.alanzucconi.com/2017/04/10/robotic-arms/

Motion Planning in EE Space



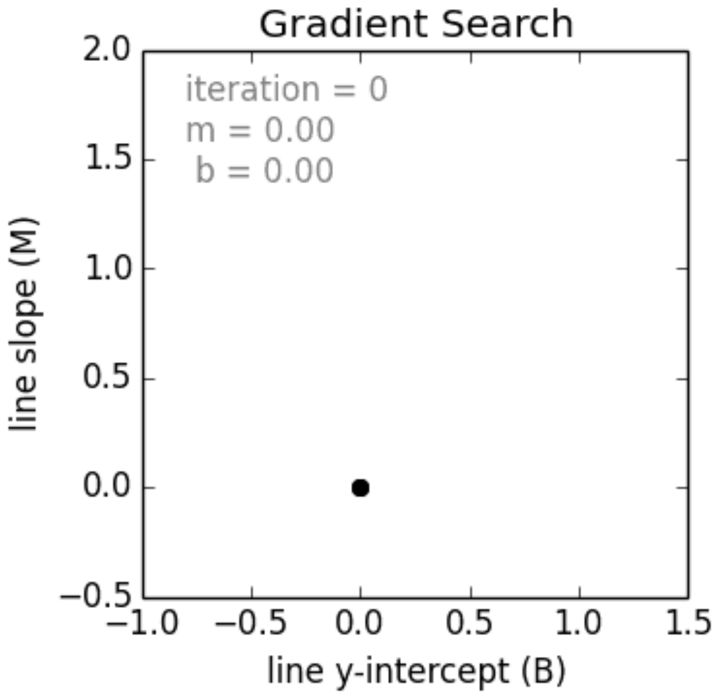
$$x = l_1 \cos(\alpha) + l_2 \cos(\alpha + \beta)$$
$$y = l_1 \sin(\alpha) + l_2 \sin(\alpha + \beta)$$



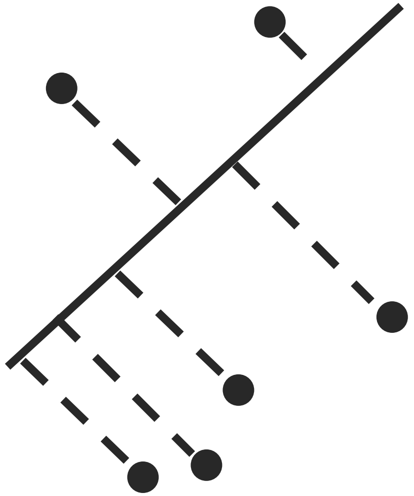
Just the Euclidean distance between two vectors!

$$f_{x,y}(\alpha, \beta) = \sqrt{(\sin(\alpha+\beta) l_2 + \sin(\alpha) l_1 - y_d)^2 + (\cos(\alpha+\beta) l_2 + cos(\alpha) l_1 - x_d)^2}$$

A primer on gradient descent

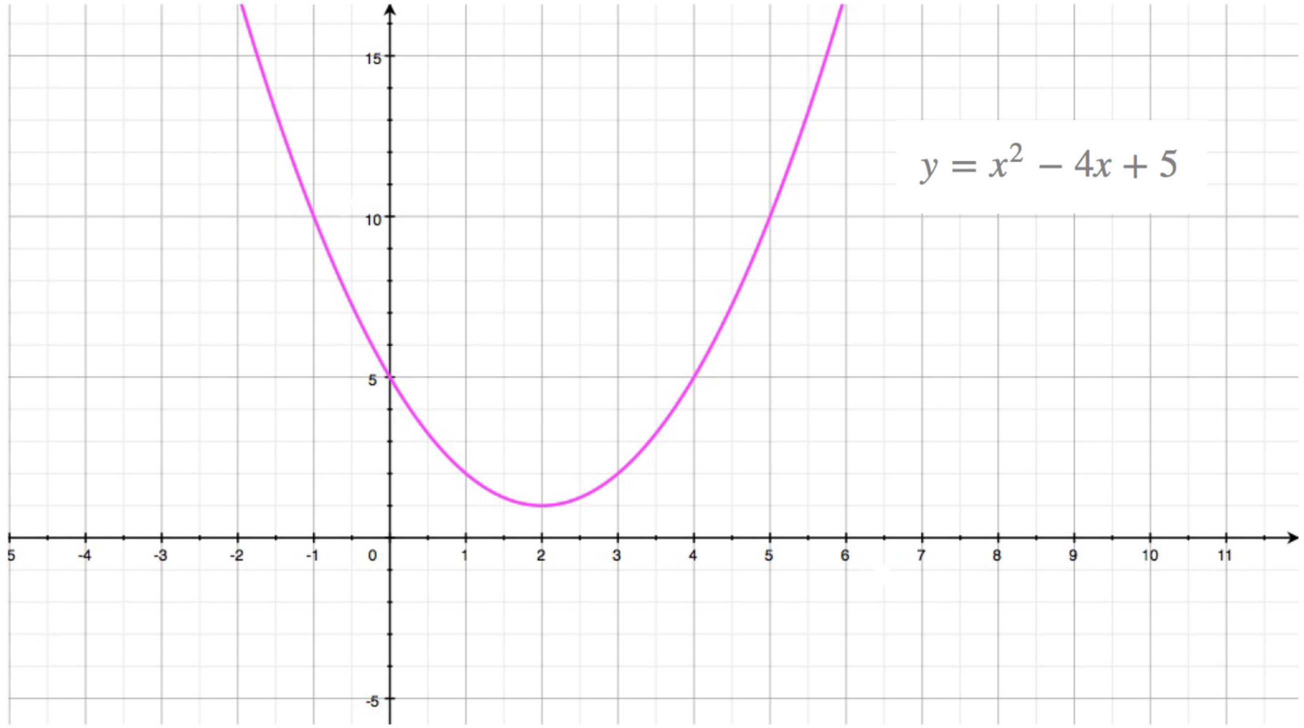


Cost function: sum of distances to line



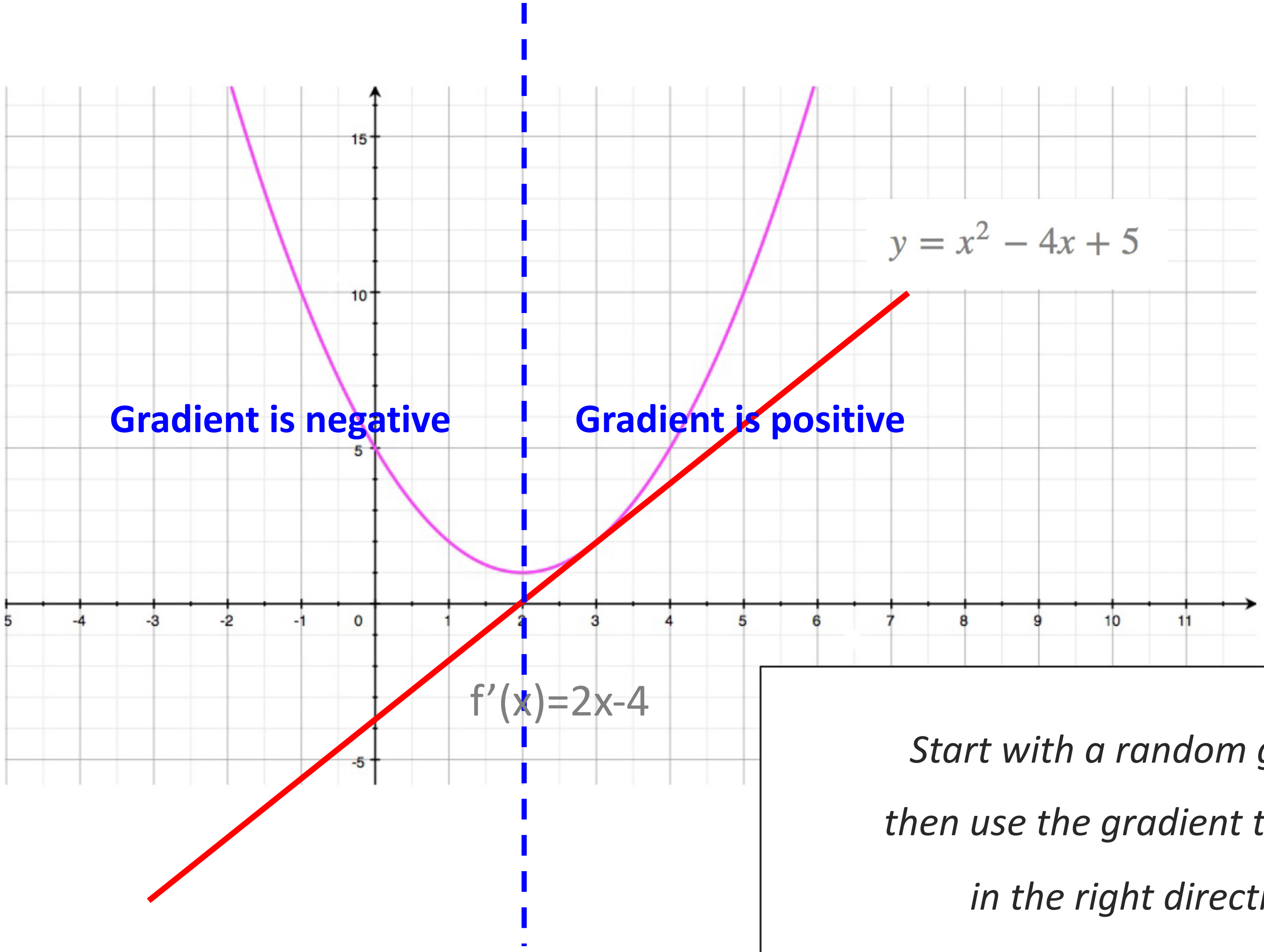
E = f(M,B,p_1, …p_N)

A primer on gradient descent



- Where is the minimum?

A primer on gradient descent



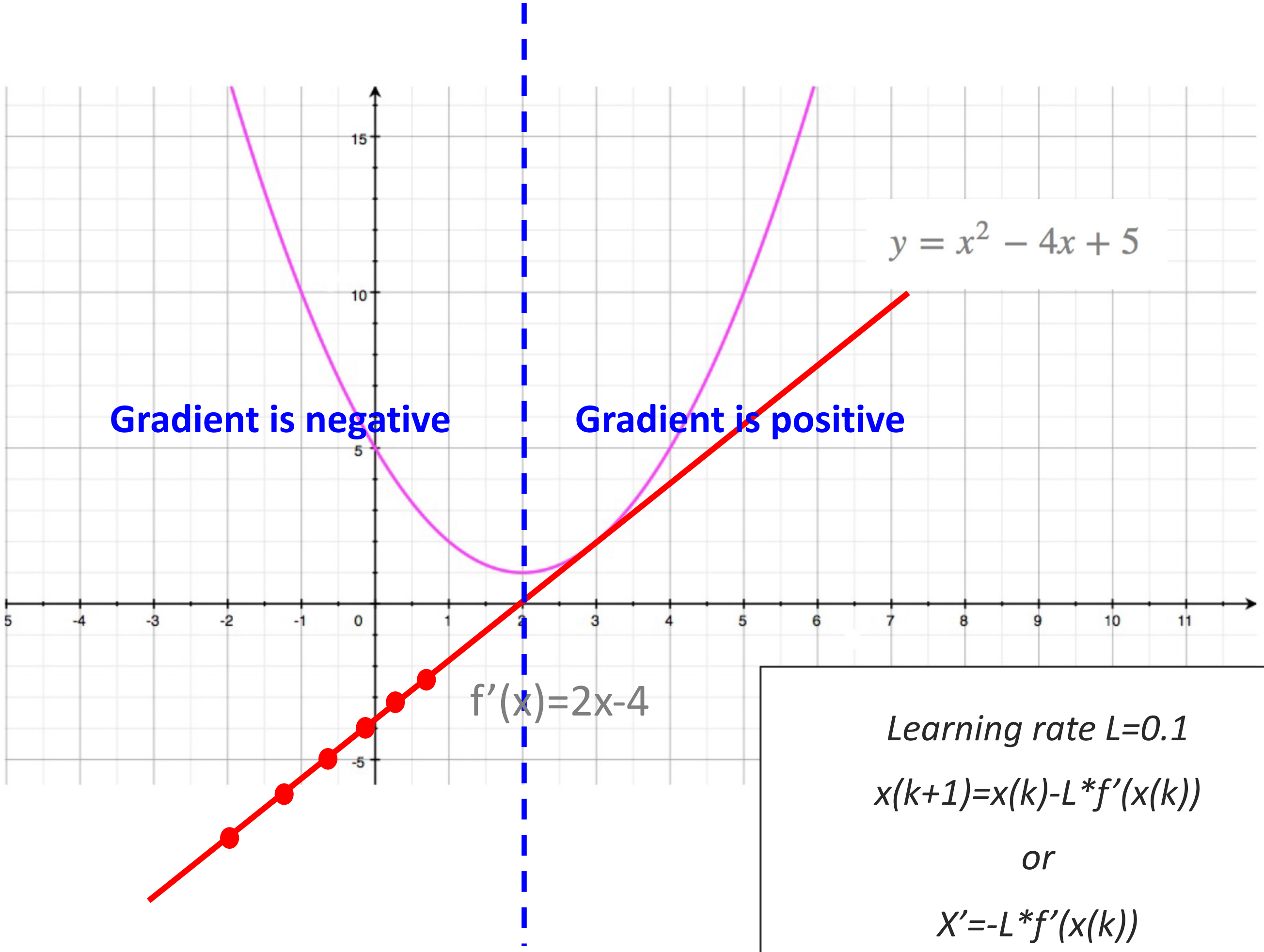
- Where is the minimum?
- Its where f'(x)=0!
- f'(x)=2x-4

Start with a random guess,

then use the gradient to move

in the right direction

A primer on gradient descent



Learning rate L=0.1

*x(k+1)=x(k)-L*f'(x(k))*

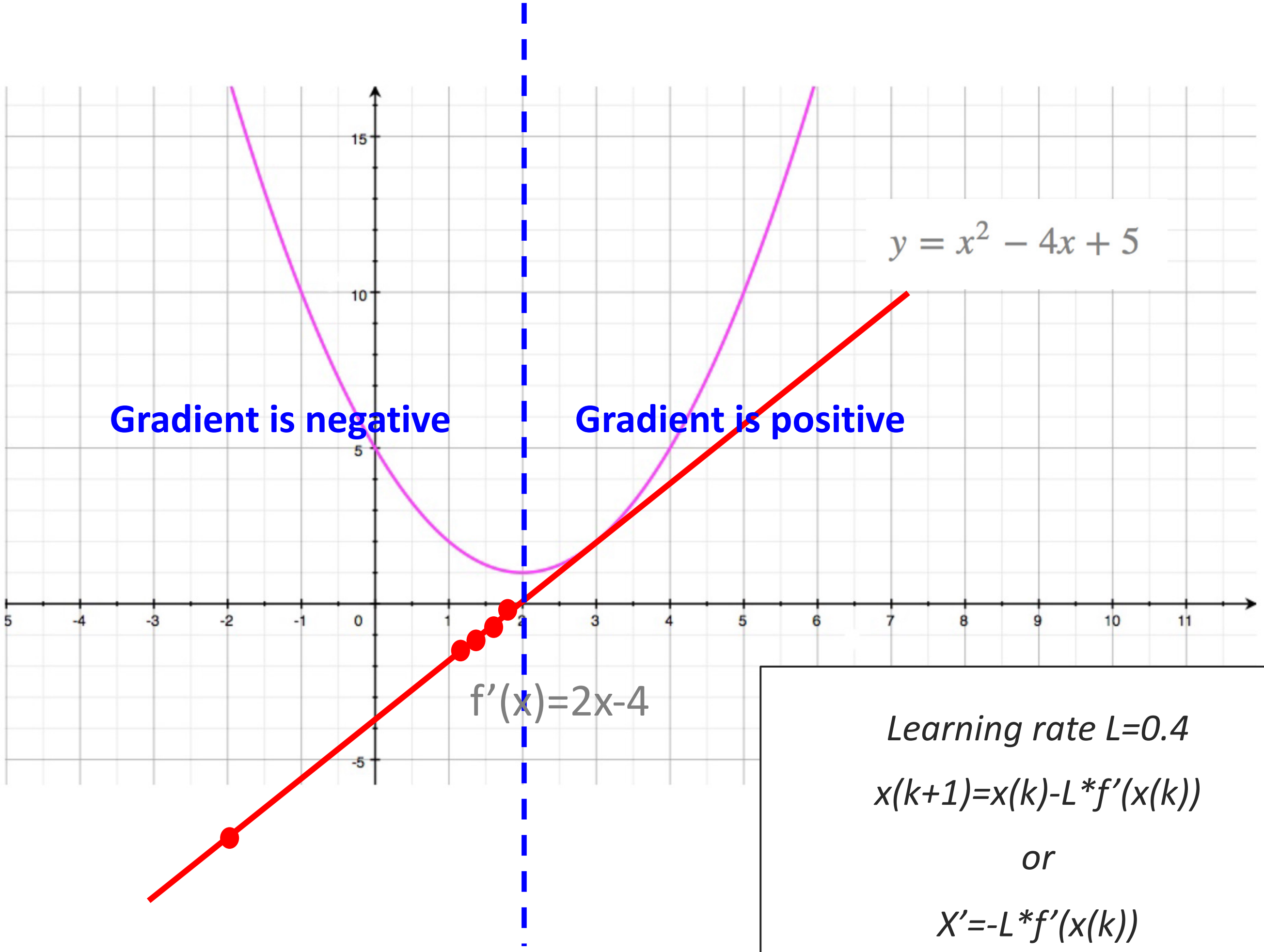
or

*X'=-L*f'(x(k))*

k	x	f'(x)
0	-2	-8
1	-1.2	-6.4
2	-0.56	-5.12
3	-0.048	-4.096
4	0.3616	-3.2768
5	0.69	-2.62
…		…

L=0.1

A primer on gradient descent



Learning rate L=0.4

*x(k+1)=x(k)-L*f'(x(k))*

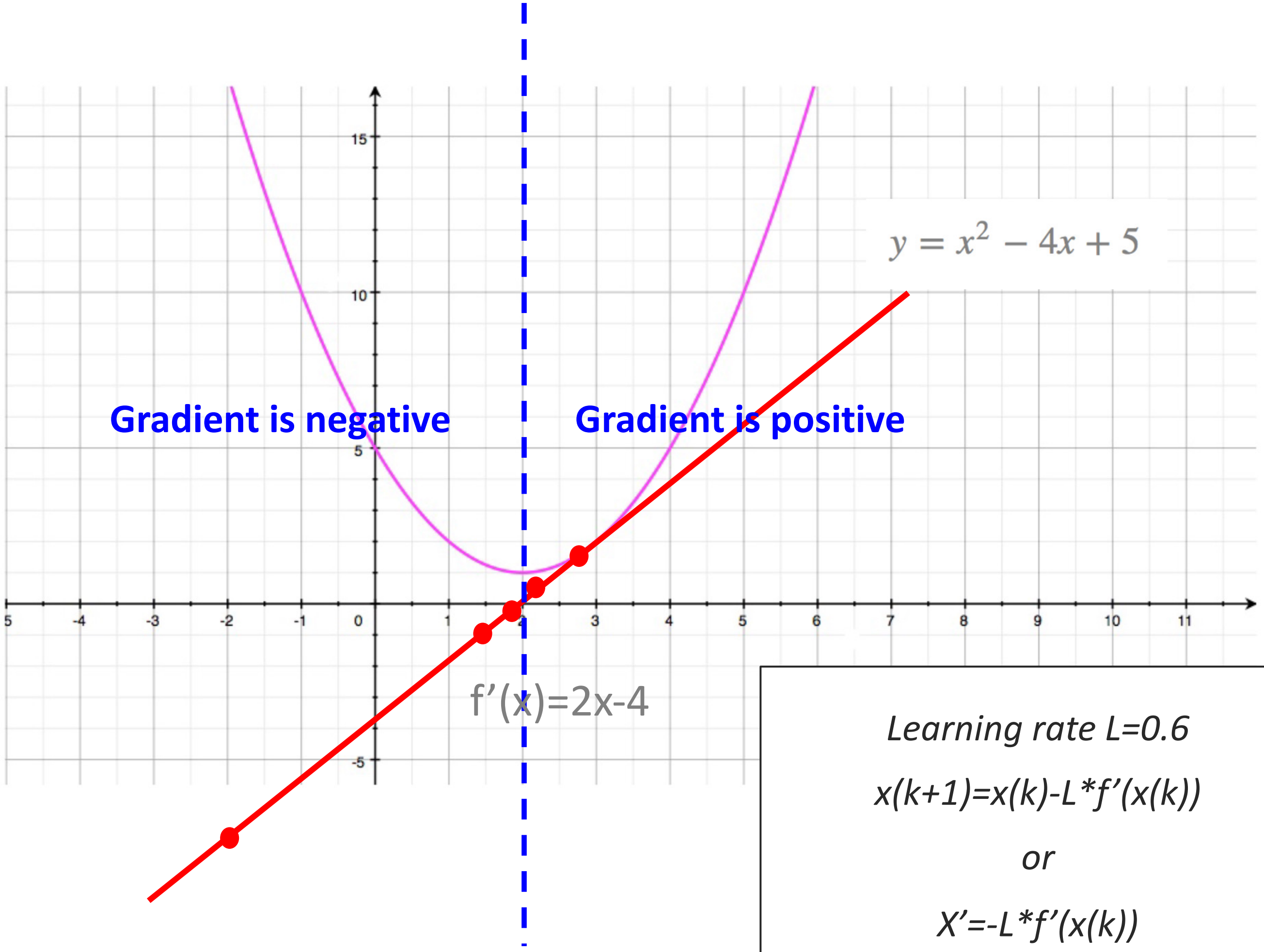
or

*X'=-L*f'(x(k))*

k	x	f'(x)
0	-2	-8
1	1.2	-1.6
2	1.84	-0.32
3	1.968	-0.064
4	1.9936	-0.0128
…	…	…

L=0.4

A primer on gradient descent



Learning rate L=0.6

*x(k+1)=x(k)-L*f’(x(k))*

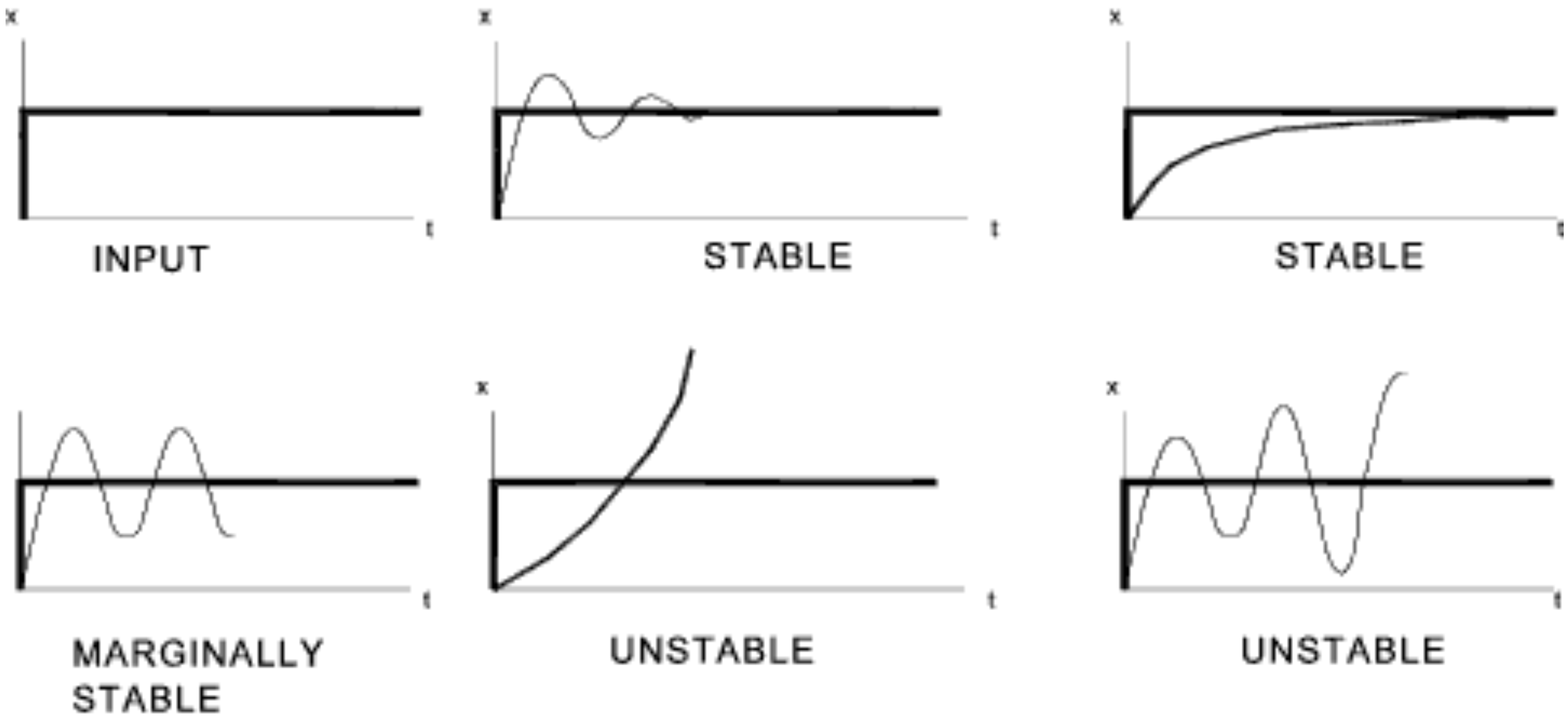
or

*X’=-L*f’(x(k))*

k	x	f’(x)
0	-2	-8
1	2.8	1.6
2	1.84	-0.32
3	2.032	0.064
4	1.9936	-0.0128
…	…	…

L=0.6

Typical controller behavior



Gradient Descent for Solving IK

Given a "distance-from-goal" function f and motors $\alpha_0, \alpha_1, \alpha_2$:

$$\nabla f(\alpha_0, \alpha_1, \alpha_2) = [\nabla f_{\alpha_0}(\alpha_0, \alpha_1, \alpha_2), \nabla f_{\alpha_1}(\alpha_0, \alpha_1, \alpha_2), \nabla f_{\alpha_2}(\alpha_0, \alpha_1, \alpha_2)]$$

Gradient

- $\nabla f_{\alpha_0} = (\alpha_0, \alpha_1, \alpha_2) \approx \frac{f(\alpha_0 + \Delta_x, \alpha_1, \alpha_2) - f(\alpha_0, \alpha_1, \alpha_2)}{\Delta x}$
- $\nabla f_{\alpha_1} = (\alpha_0, \alpha_1, \alpha_2) \approx \frac{f(\alpha_0, \alpha_1 + \Delta_y, \alpha_2) - f(\alpha_0, \alpha_1, \alpha_2)}{\Delta y}$
- $\nabla f_{\alpha_2} = (\alpha_0, \alpha_1, \alpha_2) \approx \frac{f(\alpha_0, \alpha_1, \alpha_2 + \Delta_z) - f(\alpha_0, \alpha_1, \alpha_2)}{\Delta z}$

Controller

$$\alpha_0 \leftarrow \alpha_0 - L\nabla f_{\alpha_0}(\alpha_0, \alpha_1, \alpha_2)$$
$$\alpha_1 \leftarrow \alpha_1 - L\nabla f_{\alpha_1}(\alpha_0, \alpha_1, \alpha_2)$$
$$\alpha_2 \leftarrow \alpha_2 - L\nabla f_{\alpha_2}(\alpha_0, \alpha_1, \alpha_2)$$

Relating gradients in joint space to gradients in operational space

- Linear equations dictate end-effector position:

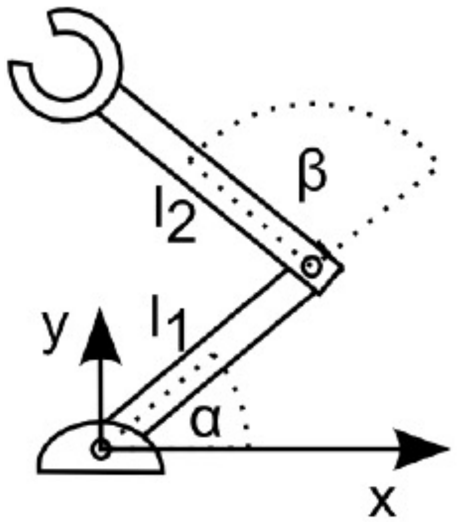
$$\left.\begin{aligned} x_e(\alpha,\beta) &= l_1\cos(\alpha) + l_2\cos(\alpha+\beta) \\ y_e(\alpha,\beta) &= l_1\, sin(\alpha) + l_2 \sin(\alpha+\beta) \end{aligned}\right\} \text{Forward Kinematics Equations}$$

- Relationship between position change and angle change:

$$\Delta x_e = \frac{\partial x_e(\alpha,\beta)}{\partial\alpha}\Delta\alpha + \frac{\partial x_e(\alpha,\beta)}{\partial\beta}\Delta\beta$$

$$\Delta y_e = \frac{\partial y_e(\alpha,\beta)}{\partial\alpha}\Delta\alpha + \frac{\partial y_e(\alpha,\beta)}{\partial\beta}\Delta\beta$$

- $J = \begin{bmatrix} \frac{\partial x_e}{\partial\alpha} & \frac{\partial x_e}{\partial\beta} \\ \frac{\partial y_e}{\partial\alpha} & \frac{\partial y_e}{\partial\beta} \end{bmatrix} \quad \begin{bmatrix}\Delta x_e \\ \Delta y_e\end{bmatrix} = J\cdot\begin{bmatrix}\Delta\alpha \\ \Delta\beta\end{bmatrix}$



x_e: x position of end effector

y_e: y position of end effector

Using the Jacobian to Move the Robot

- $\frac{dp_e}{dt} = \boldsymbol{J}\frac{d\boldsymbol{q}}{dt}$, or in other words , $v_e = \boldsymbol{J} \cdot \dot{\boldsymbol{q}}$

- $\dot{q} = \boldsymbol{J}^{-\mathbf{1}} \cdot \left[v_{e,d} + K\left(p_{e,d} - p\right)\right]$

$\dot{q}$: Change in C-space

K: gain

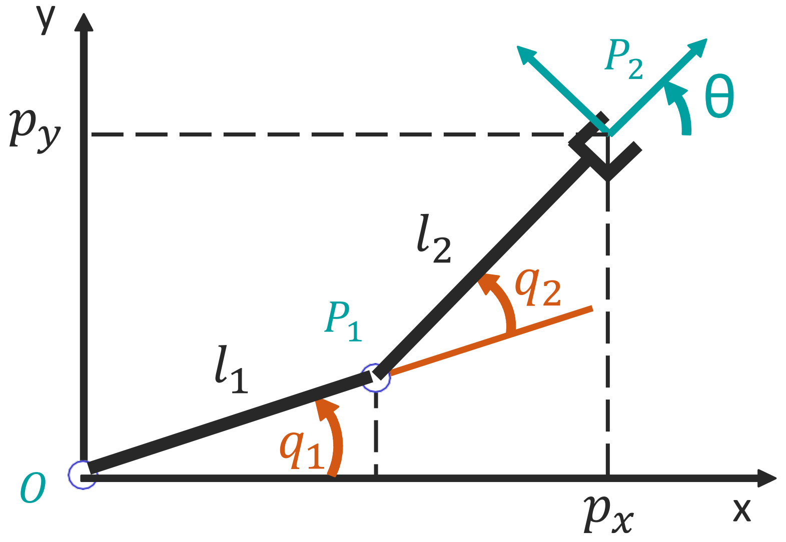
$v_{e,d}$: Desired velocity

$p_{e,d}$: Desired position

Section 3.3.3 in the book

Inverse Kinematics

IK for manipulator

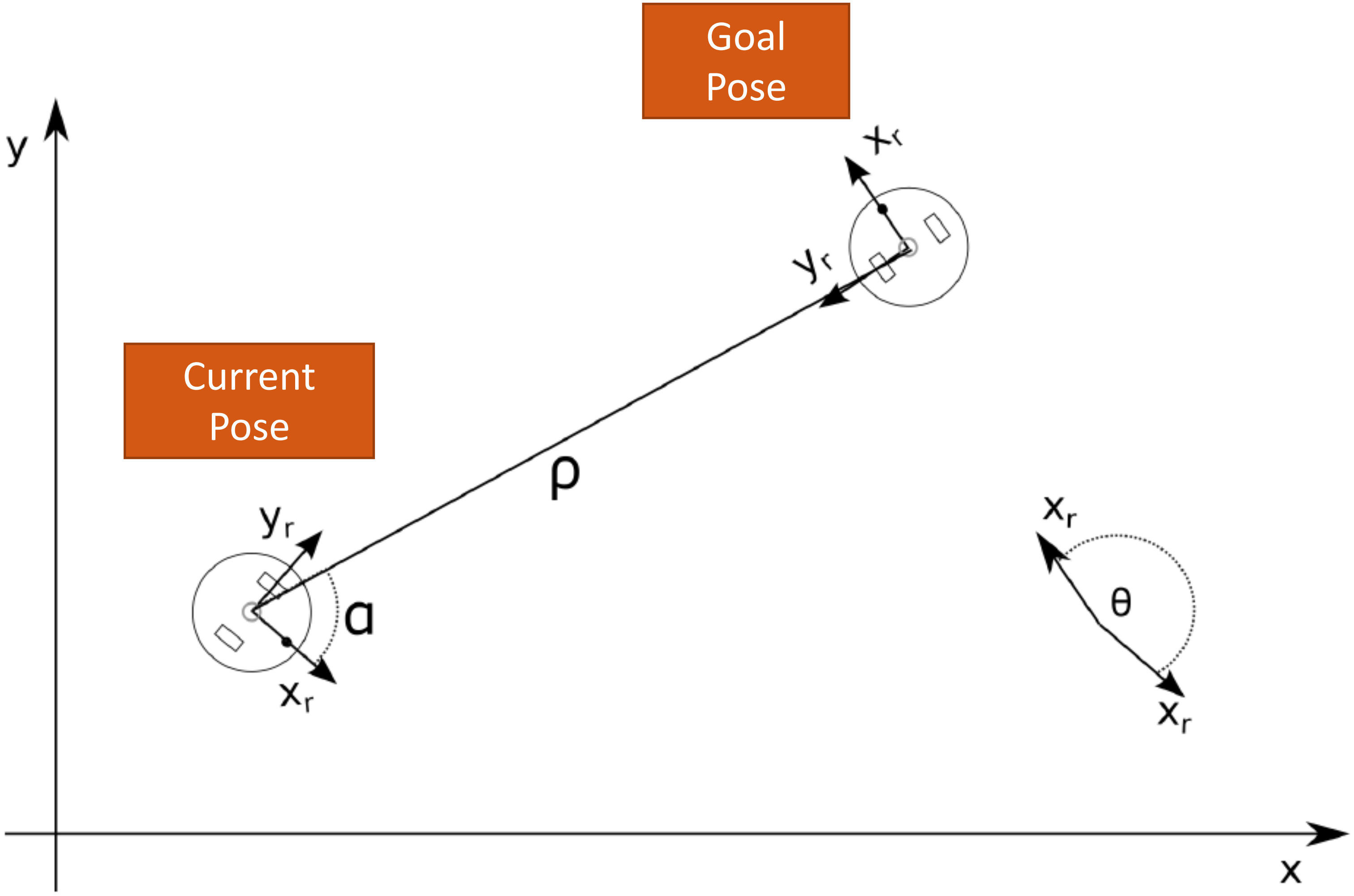


Given point (x, y, θ) Find angles (q_1, q_2)

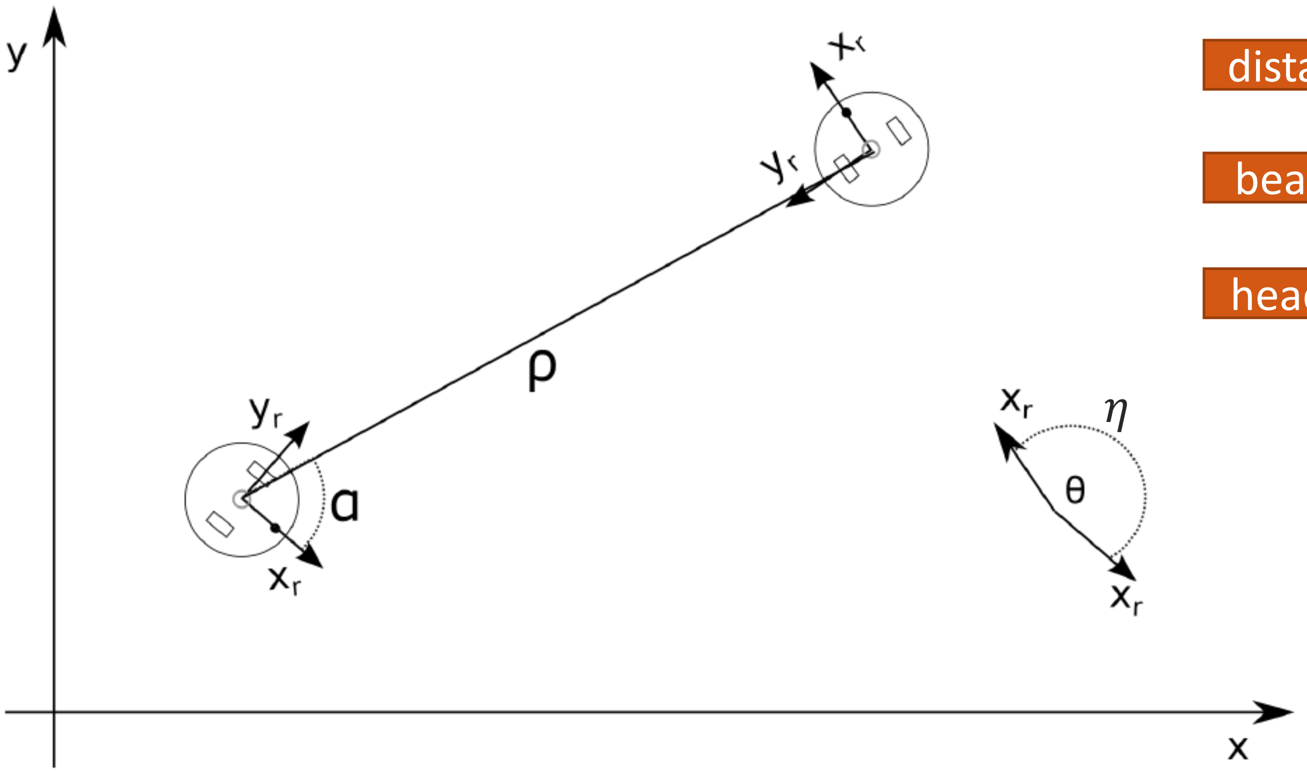
IK for e-puck

Given: (x, y, θ) Find: (ϕ_l, ϕ_r)

Position Change Using Feedback Control



Position Change Using Feedback Control



Error Terms

distance $\rho = \sqrt{(x_r - x_g)^2 + (y_r - y_g)^2}$

bearing $\alpha = \tan^{-1}\left(\frac{y_g - y_r}{x_g - x_r}\right) - \theta_r$

heading $\eta = \theta_g - \theta_r$

Update Rules

translation $\dot{x} = p_1\rho$

rotation $\dot{\theta} = p_2\alpha + p_3\eta$

p_1, p_2, p_3 are controller gains

Position Change Using Feedback Control

Error Terms

$$\rho = \sqrt{(x_r - x_g)^2 + (y_r - y_g)^2}$$

$$\alpha = \tan^{-1}\left(\frac{y_g - y_r}{x_g - x_r}\right) - \theta_r$$

$$\eta = \theta_g - \theta_r$$

Update Rules

translation $\dot{x} = p_1\rho$

rotation $\dot{\theta} = p_2\alpha + p_3\eta$

p_1, p_2, p_3 are controller gains

Jacobian: relate joint to operational velocities

$$\begin{bmatrix} \dot{x}_R \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} \frac{r\dot{\phi}_l}{2} + \frac{r\dot{\phi}_r}{2} \\ \frac{\dot{\phi}_r r}{d} - \frac{\dot{\phi}_l r}{d} \end{bmatrix} =$$

$$\frac{dp_e}{dt} = \boldsymbol{J}\frac{d\boldsymbol{q}}{dt}$$

Inverse kinematics

$$\begin{bmatrix} \dot{x}_R \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ -\frac{r}{d} & \frac{r}{d} \end{bmatrix} \begin{bmatrix} \dot{\phi}_l \\ \dot{\phi}_r \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$\begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ -\frac{r}{d} & \frac{r}{d} \end{bmatrix}^{-1} \begin{bmatrix} \dot{x}_R \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} \dot{\phi}_l \\ \dot{\phi}_r \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -d/2 \\ 1 & d/2 \end{bmatrix} \begin{bmatrix} \dot{x}_R \\ \dot{\theta}_R \end{bmatrix}$$

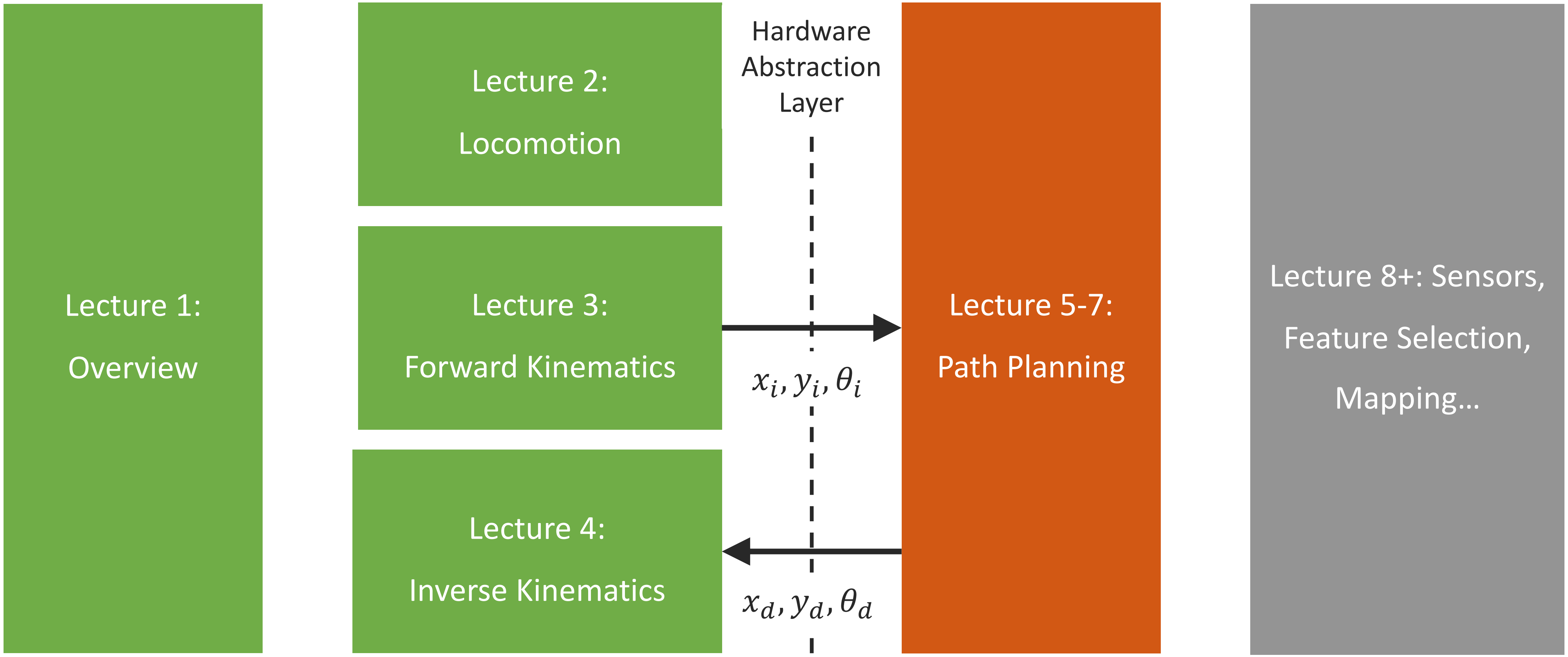
Where do we go from here?

Chapter 4

Module 2 – COMPUTATION

Part I – Motion and Trajectory Planning

Roadmap


Lecture 1:
Overview
Lecture 2:
Locomotion
Lecture 3:
Forward Kinematics
Lecture 4:
Inverse Kinematics
Hardware
Abstraction
Layer
x_i, y_i, θ_i
x_d, y_d, θ_d
Lecture 5-7:
Path Planning
Lecture 8+: Sensors,
Feature Selection,
Mapping…

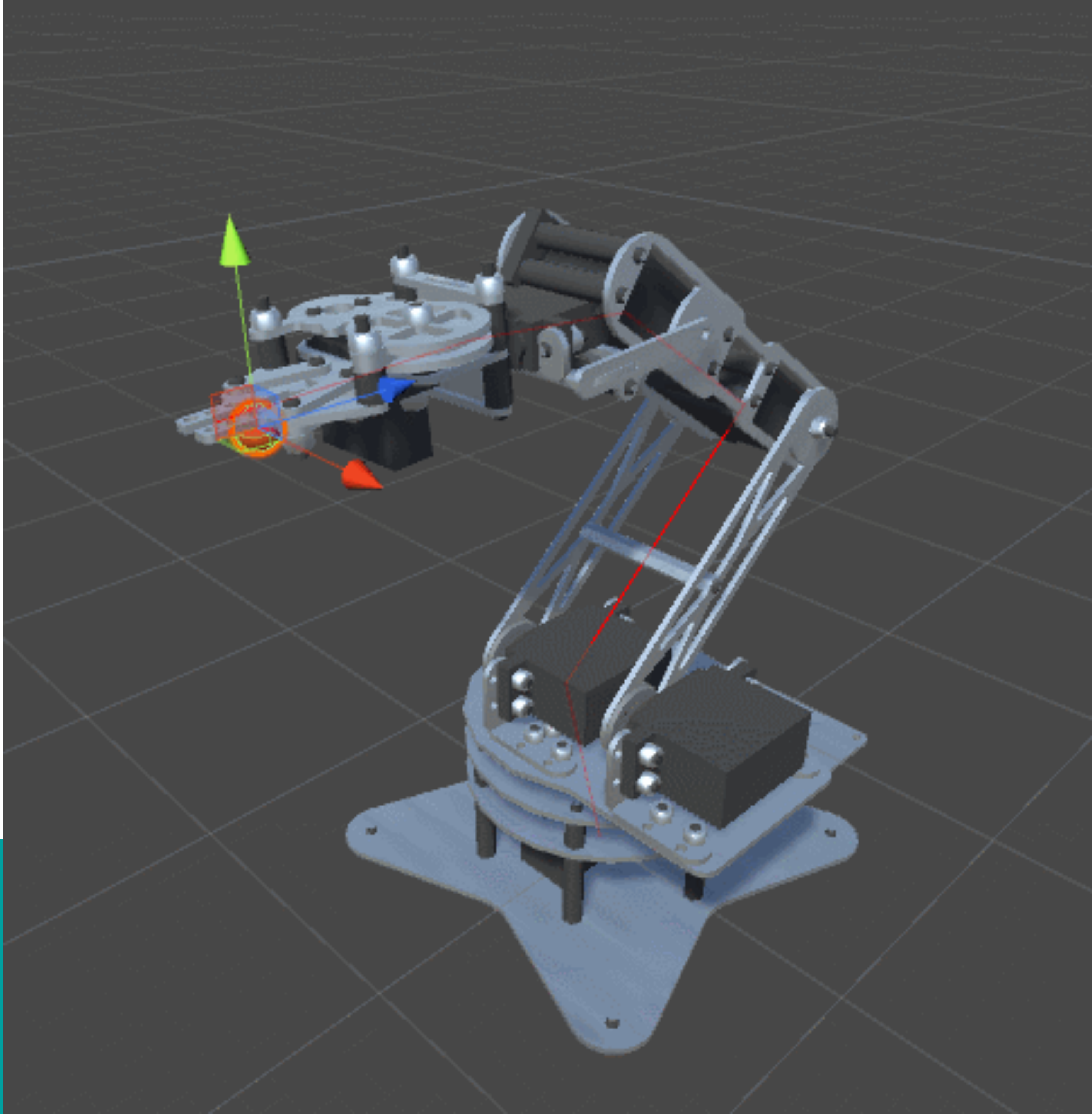
Motion Planning

Some taxonomy

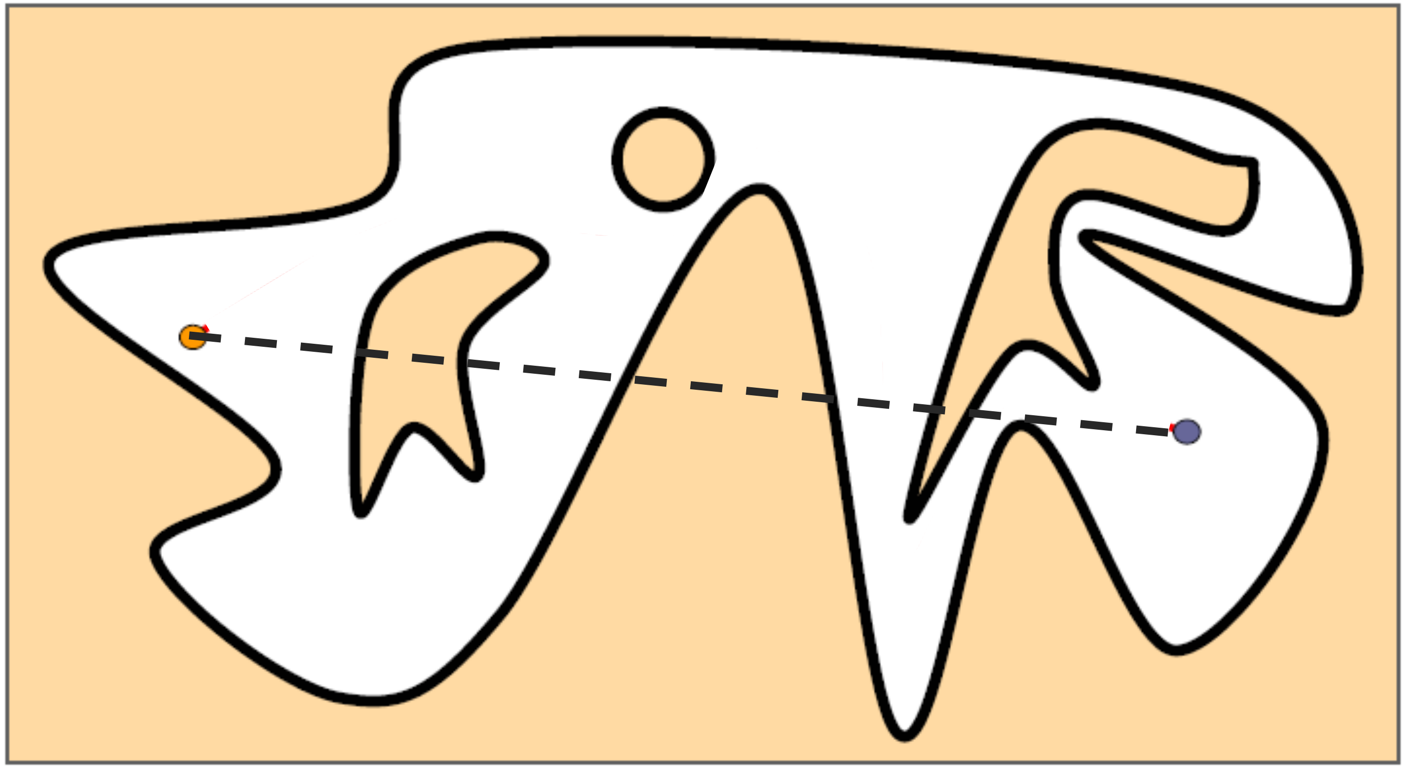
- **Kinematics**: geometrical relationships in terms of position/velocity between joint-space and task-space
- **Dynamics**: relationships between joint torques and dynamical properties of the plant with links/end-effector motions
- **Control**: computation of the control actions (i.e. joint torques) to achieve a desired motion.
- **Planning**: planning of the *desired* movements of the manipulator

Motivating Problem 1: Joint Angle Limitation Problems!

How do we fix this?



Motivating Problem 2: Moving e-puck from start to goal state



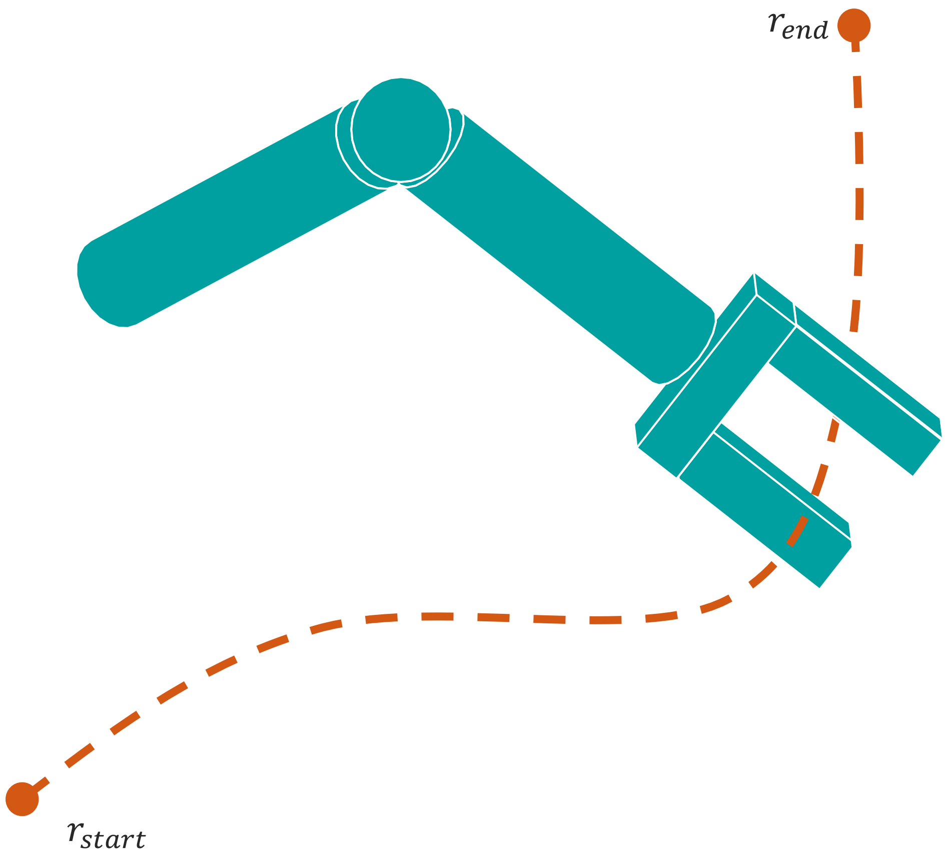
Using distance and angle as ”error” is not sufficient anymore.

Moving E-puck from Start to Goal state

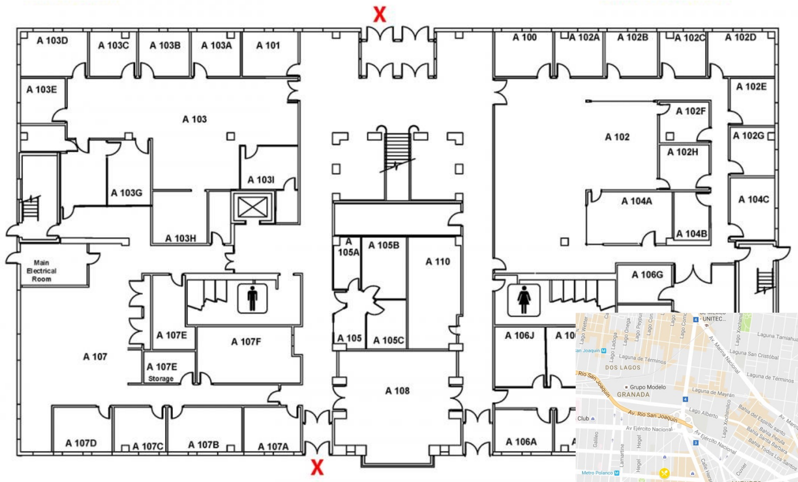
- We need to know where we are ✓
- We need a controller to drive the robot from point A to point B ✓
- We need to have a *map* of the environment that contains obstacles
- We need to compute a *trajectory* of safe intermediate points

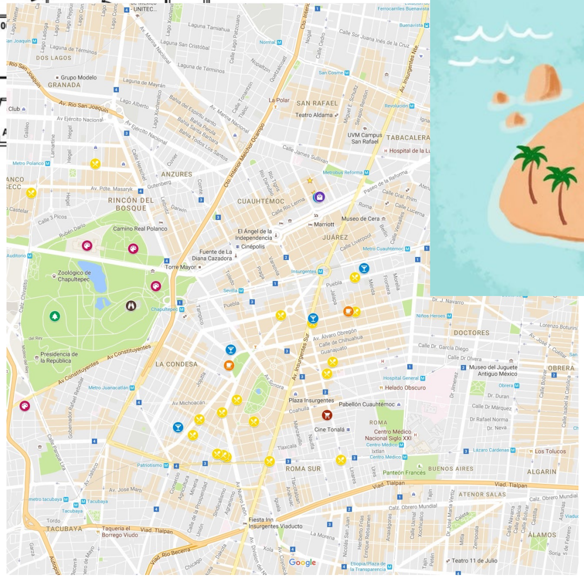
What is a trajectory?

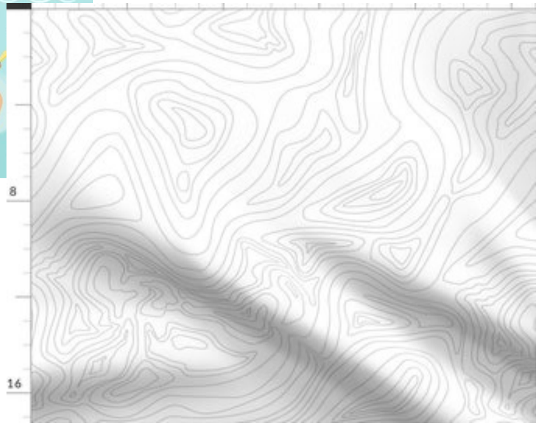
- **Path**: geometric description of a series of poses from r_{start} to r_{end} that respect specifications (e.g. avoiding obstacles or following a specific force profile)
- **Trajectory**: execution of path over time $r(t)$



Maps

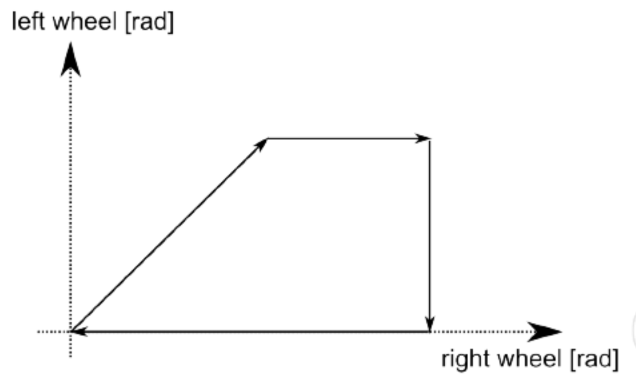


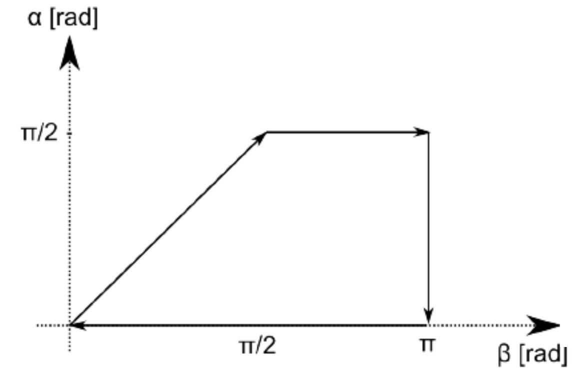




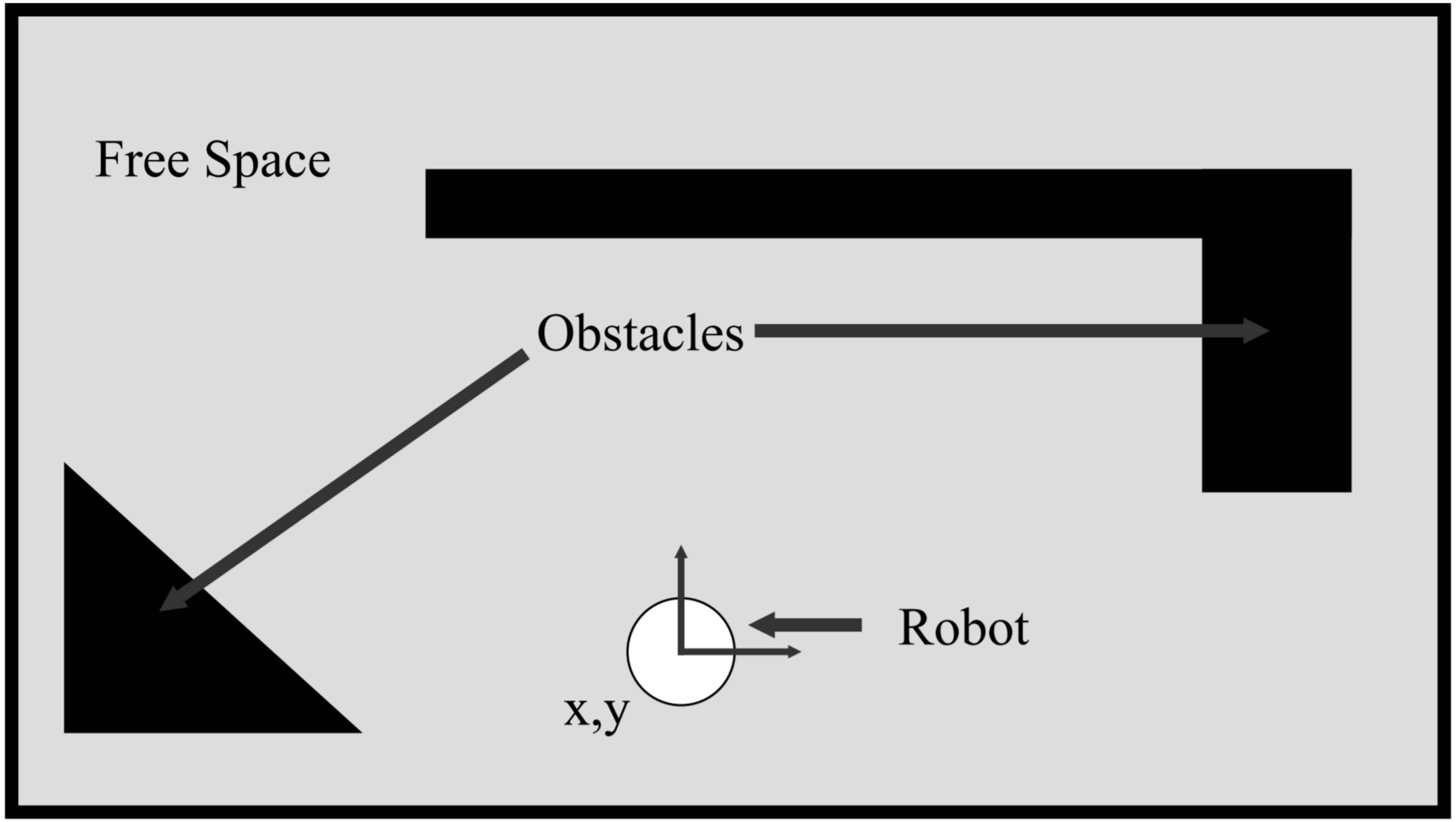
Configuration Space (joint space)

- Allows us to reduce robot positions to a single point
  - Very convenient for planning!
- One axis of configuration space for each degree of freedom of the robot
  - Convenient for planning as the space is the same as what the robot controls.
  - Removes the need to figure out how to get to a point in (X, Y)-space
- We still need to figure out how to put obstacles into configuration space!

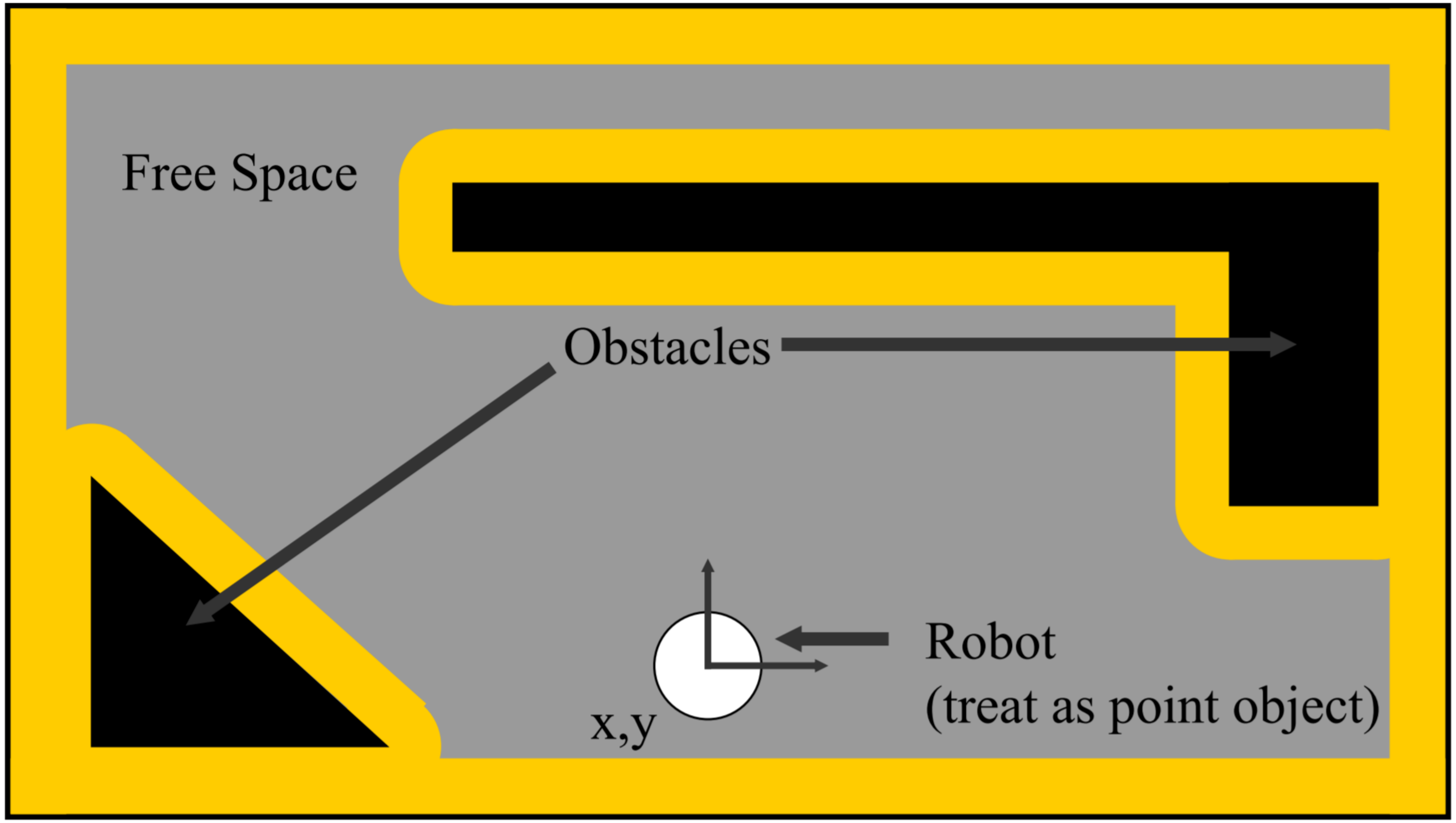




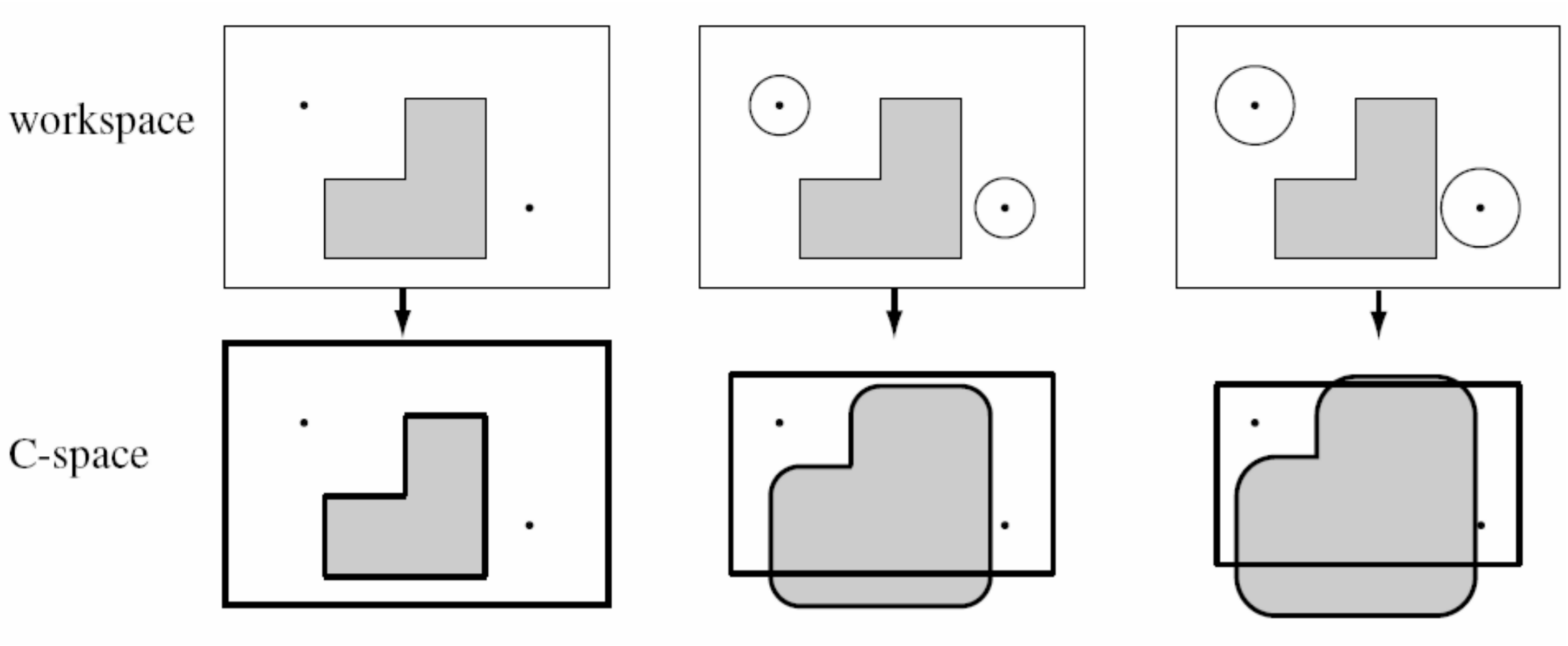
Configuration Space (operational space)



Configuration Space

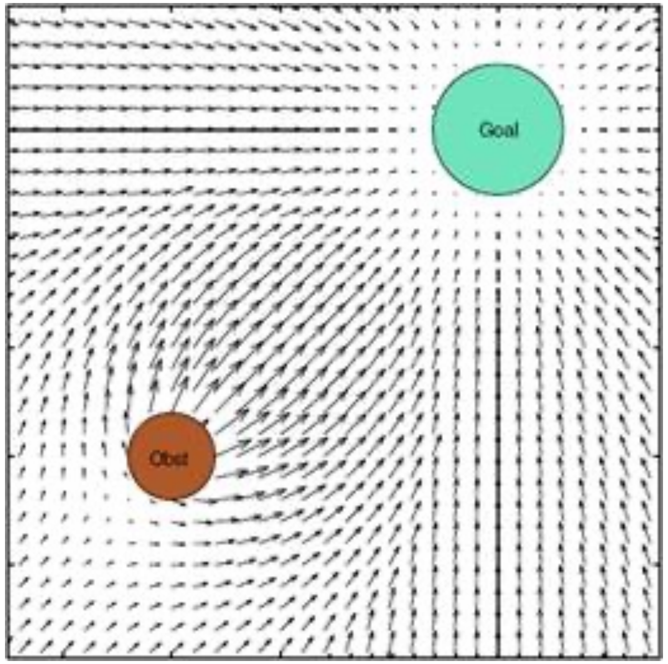


Configuration Space

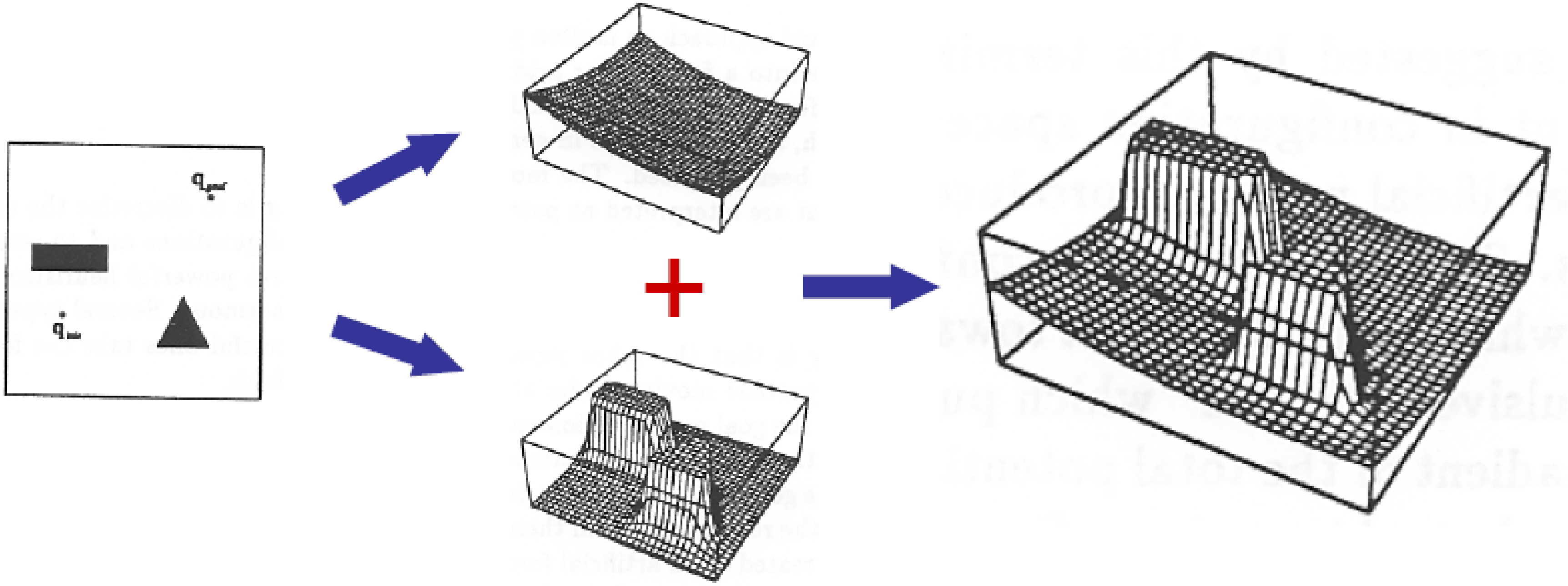


Reactive planning: Potential Fields

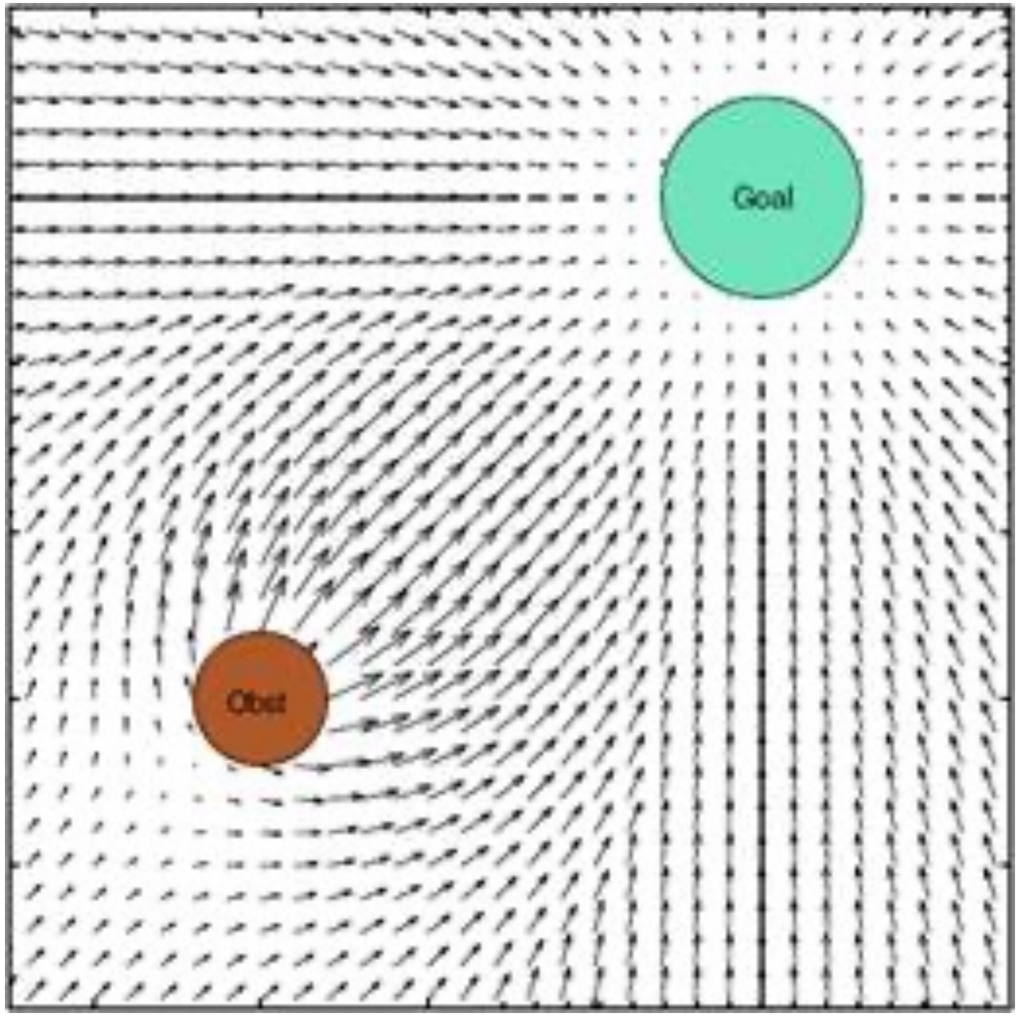
- Initially proposed for real-time collision avoidance [Khatib 1986].
- A potential field is a scalar function over the free space.
- To navigate, the robot applies a force proportional to the negated gradient of the potential field.
- A navigation function is an ideal potential field that
  - has global minimum at the goal
  - has no local minima
  - grows to infinity near obstacles
  - is smooth



Including obstacles



Potential Fields: Strengths/Weaknesses



Moving E-puck from Start to Goal state

- We need to know where we are
- We need a controller to drive the robot from point A to point B
- We need to have a *map* of the environment
- We need to compute a *trajectory* of safe intermediate points
- We need to perform planning to get *complete* solutions

Motion Planning

$\xi \rightarrow$ "xi"
pronounced "ksee"

- **Goal: Find a trajectory in configuration space from one point to another**
  - A trajectory is usually denoted as $\xi : t \in [0, T] \rightarrow C$
  - *(A function mapping time to robot configurations)*

- The Bad News:
  - All complete search algorithms scale exponentially!!
  - Motion planning problems are high dimensional, e.g., big exponent.