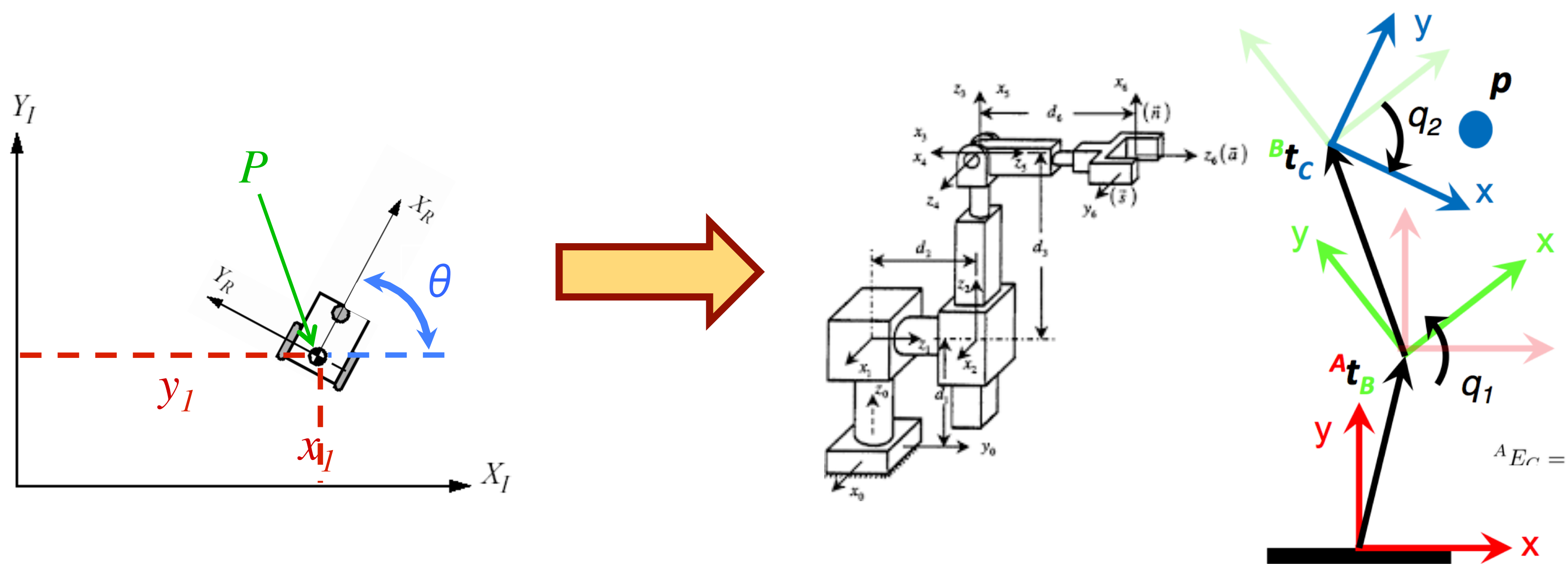
Kinematics

Manipulator Kinematics



Many slides adapted from:
Siegwart, Nourbakhsh and Scaramuzza, Autonomous Mobile Robots
Renata Melamud, An Introduction to Robot Kinematics, CMU
Rick Parent, Computer Animation, Ohio State
Steve Rotenberg, Computer Animation, UCSD

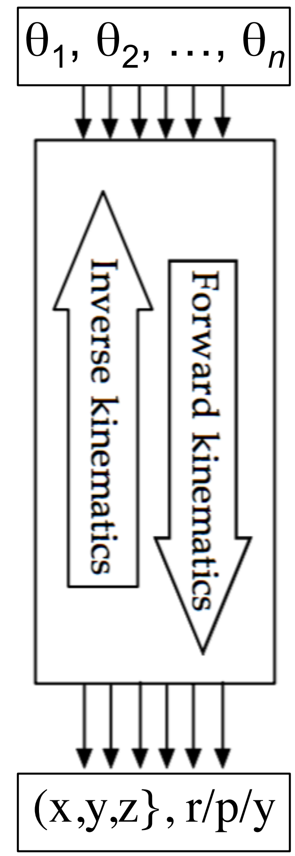
Bookkeeping

- Upcoming:
  - Projects:
    - Wiki permissions – http://tiny.cc/robotics-team-schedules
  - Posted tonight:
    - Quiz 3: Manipulation, Grasping, Kinematics
    - **Concepts!**
  - Homework 2
    - Resolution, Kinematics & IK, Course Progress
- Today: Inverse kinematics

Forward & Inverse

- Forward:
  - Inputs: joint angles
  - Outputs: coordinates of end-effector
- Inverse:
  - Inputs: desired coordinates of end-effector
  - Outputs: joint angles
- Inverse kinematics are tricky
  - Multiple solutions
  - No solutions
  - Dead spots

Joint space (robot space – previously *R*)



Cartesian space (global space – previously *I*)

Forward Kinematics

- We will sometimes use the vector **Φ** to represent the array of **M** joint values:

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1 & \phi_2 & \dots & \phi_M \end{bmatrix}$$

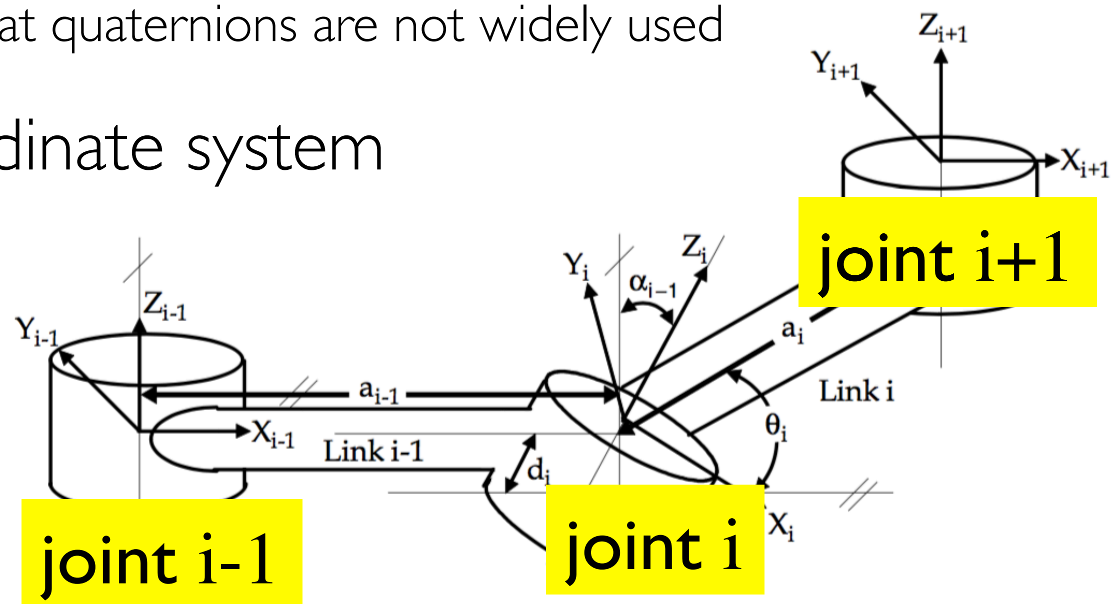
- We will sometimes use the vector **e** to represent an array of **N** joint values that describe the end effector in world space:

$$\mathbf{e} = \begin{bmatrix} e_1 & e_2 & \dots & e_N \end{bmatrix}$$

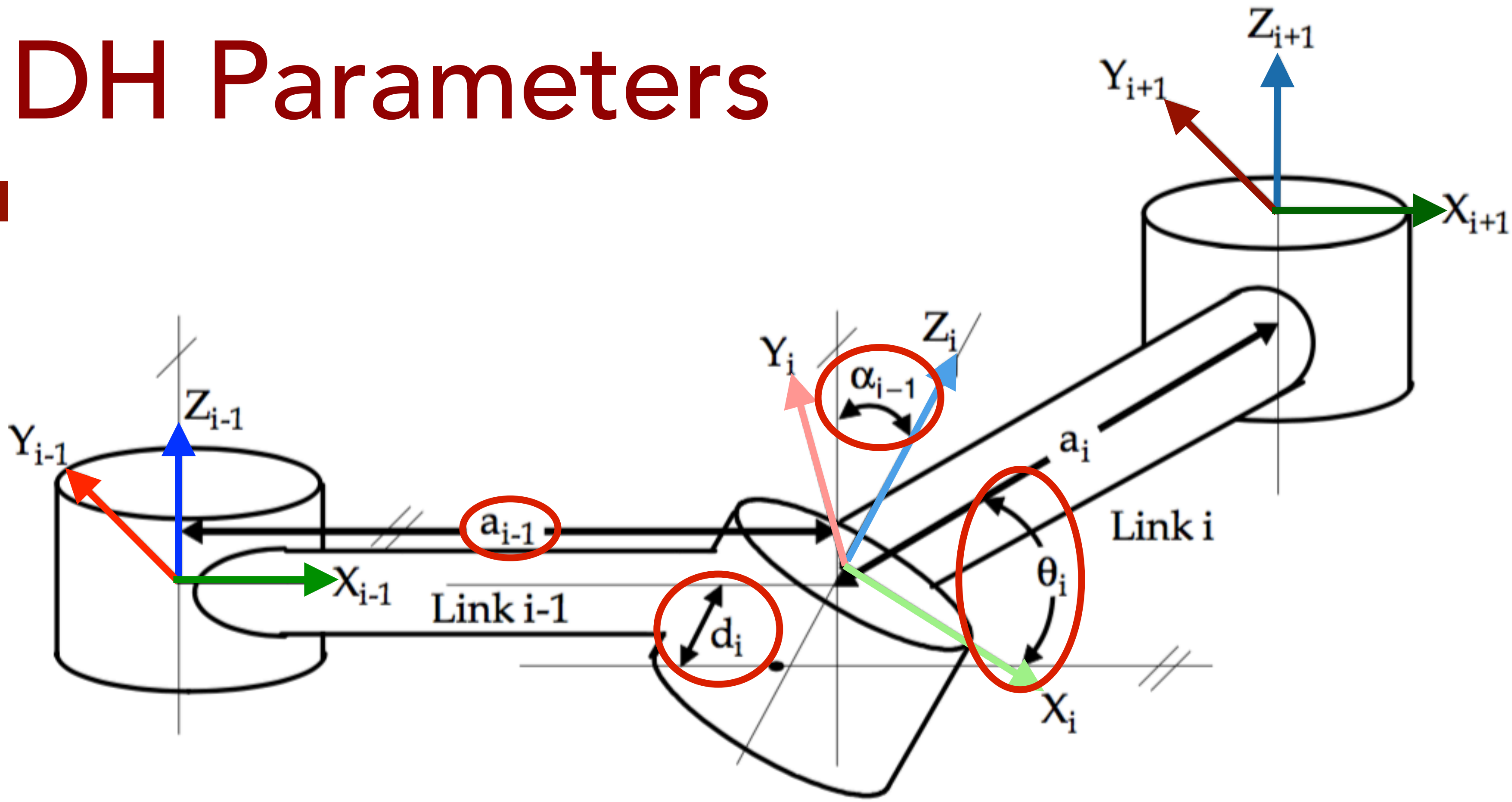
- Example:
  - If our end effector is a full joint with orientation, **e** would contain 6 DOFs: 3 translations and 3 rotations. If we were only concerned with the end effector position, **e** would just contain the 3 translations.

Describing A Manipulator

- Arm made up of links in a chain
  - How to describe each link?
  - Many choices exist
  - DH parameters widely used
    - Although it's not true that quaternions are not widely used
- Joints each have coordinate system
  - {x,y,z}, r/p/y — *OR!!*
- *DH parameters*
  - Denavit-Hartenberg
  - $a_{i-1}, \alpha_{i-1}, d_i, \theta_2$



DH Parameters



a_{i-1} : link length – distance Z_{i-1} and Z_i along X_i

α_{i-1} : link twist – angle Z_{i-1} and Z_i around X_i

d_i : link offset – distance X_{i-1} to X_i along Z_i

θ_2 : joint angle – angle X_{i-1} and X_i around Z_i

Forward: i → i-1

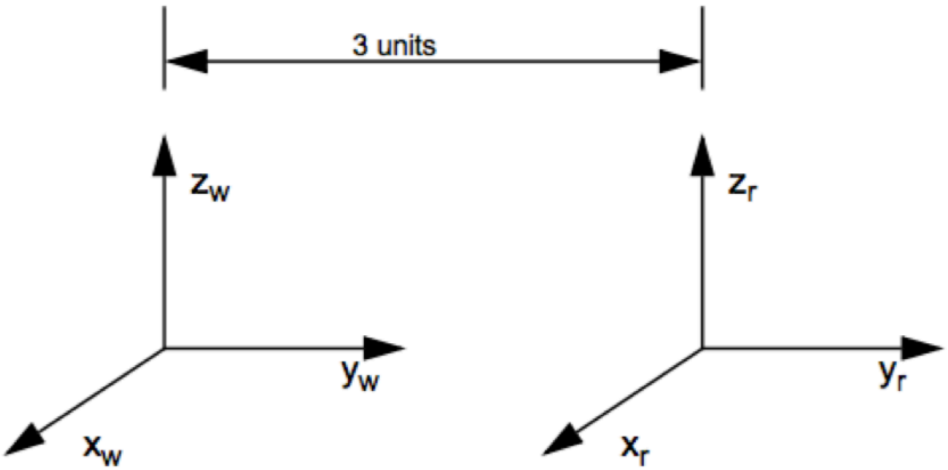
- We are we looking for:

  Transformation matrix **T** going from **i** to **i-1**:

$$ {}^{i-1}T_i \qquad \left(\text{or } {}^{i-1}_{\;\;i}T \right) $$

- Determine position and orientation of end-effector as function of displacements in joints
- Why?
  - We can multiply out along all joints

Translation



$$\xi_I = \begin{bmatrix} x_I \\ y_I \\ z_I \\ \theta \end{bmatrix} \qquad \xi_R = \begin{bmatrix} x_R \\ y_R \\ z_R \\ \theta \end{bmatrix}$$

Origin of R in I:

$$\begin{bmatrix} 0 \\ 3 \\ 0 \\ 1 \end{bmatrix}$$

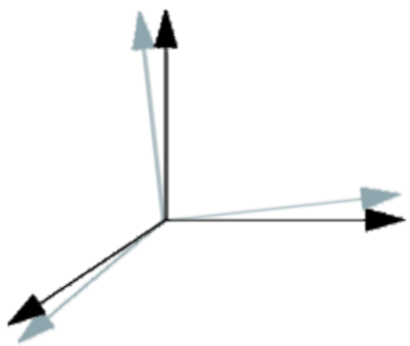
In 3D:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Generally:

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation



$$\xi_I = \begin{bmatrix} x \\ y \\ z \\ \theta_I \end{bmatrix} \qquad \xi_R = \begin{bmatrix} x \\ y \\ z \\ \theta_R \end{bmatrix}$$

Generally:

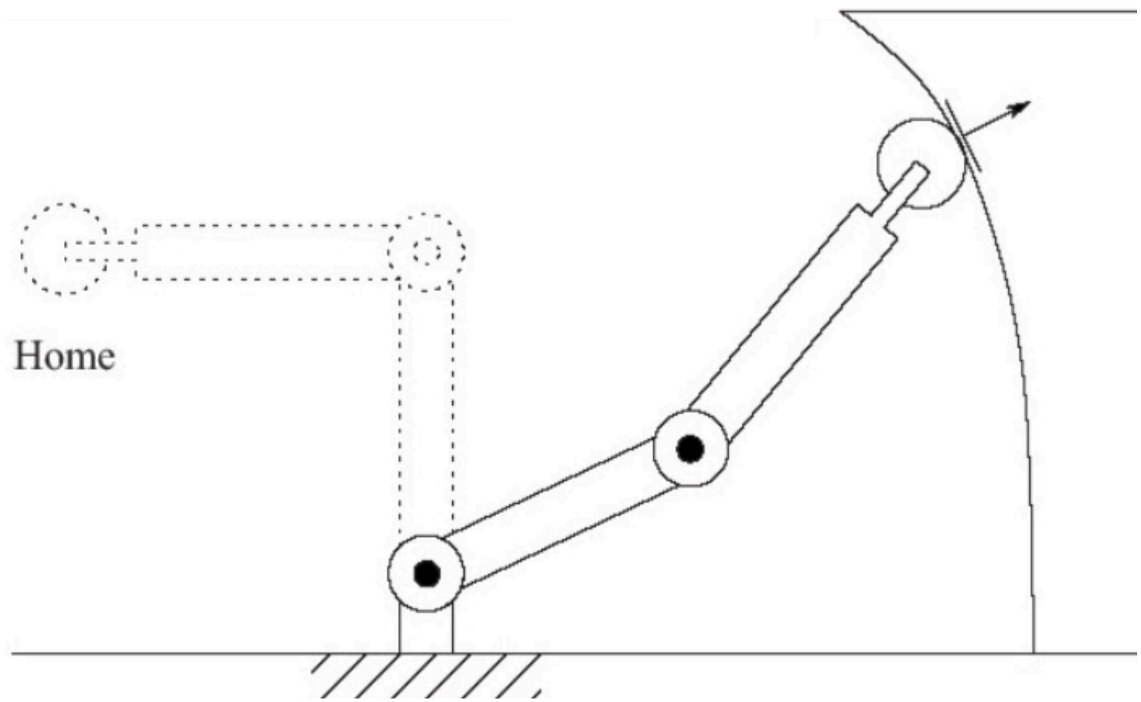
$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Review?

Introduction to Homogeneous Transformations & Robot Kinematics

Jennifer Kay 2005

Example: Rotation in Plane



$$x = a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2)$$
$$y = a_1 \sin\theta_1 + a_2 \sin(\theta_1 + \theta_2)$$
$$a_i = \text{ the length of } i\text{th link}$$

Transformation i to i-1 (2)

a_{i-1} : distance Z_{i-1} and Z_i along X_i
α_{i-1} : angle Z_{i-1} and Z_i around X_i

} screw displacement:

$$[X_i] = \text{Trans}_{X_i}(a_{i,i+1})\,\text{Rot}_{X_i}(\alpha_{i,i+1})$$

d_i : distance X_{i-1} to X_i along Z_i
θ_2 : angle X_{i-1} and X_i around Z_i

} screw displacement:

$$[Z_i] = \text{Trans}_{Z_i}(d_i)\,\text{Rot}_{Z_i}(\theta_i)$$

- Coordinate transformation:

$$^{i-1}T_i = [Z_i][X_i] = \text{Trans}_{Z_i}(d_i)\,\text{Rot}_{Z_i}(\theta_i)\,\text{Trans}_{X_i}(a_{i,i+1})\,\text{Rot}_{X_i}(\alpha_{i,i+1}),$$

Transformation i to i-1 (3)

$$\mathrm{Trans}_{Z_i}(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{Rot}_{Z_i}(\theta_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{Trans}_{X_i}(a_{i,i+1}) = \begin{bmatrix} 1 & 0 & 0 & a_{i,i+1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{Rot}_{X_i}(\alpha_{i,i+1}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_{i,i+1} & -\sin\alpha_{i,i+1} & 0 \\ 0 & \sin\alpha_{i,i+1} & \cos\alpha_{i,i+1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation in DH:

$${}^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_{i,i+1} & \sin\theta_i\sin\alpha_{i,i+1} & a_{i,i+1}\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_{i,i+1} & -\cos\theta_i\sin\alpha_{i,i+1} & a_{i,i+1}\sin\theta_i \\ 0 & \sin\alpha_{i,i+1} & \cos\alpha_{i,i+1} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

Inverse Kinematics

- Goal:
  - Compute the vector of joint DOFs that will cause the end effector to reach some desired goal state
  - In other words, it is the inverse previous problem
- Instead of function from world space to robot space.

$$\mathbf{e} = f(\mathbf{\Phi}) \leftrightarrow \mathbf{\Phi} = f^{-1}(\mathbf{e})$$

Inverse Kinematics Issues

- IK is challenging!
  - $f()$ is (usually) relatively easy to evaluate
  - $f^{-1}()$ usually isn't
- Issues:
  - There may be several possible solutions for Φ
  - There may be no solutions
  - If there is a solution, it may be expensive to find it
  - There are some local-minimum "stuck" configurations
- Many different approaches to solving IK problems

Analytical vs. Numerical

- One major way to classify IK-solving approaches: **analytical** vs **numerical** methods
- Analytical
  - Find an exact solution by directly inverting the forward kinematics equations.
  - Works on relatively simple chains.
- Numerical
  - Use approximation and iteration to converge on a solution.
  - More expensive, more general purpose.
- We will look at one technique: Jacobians

Calculus Review

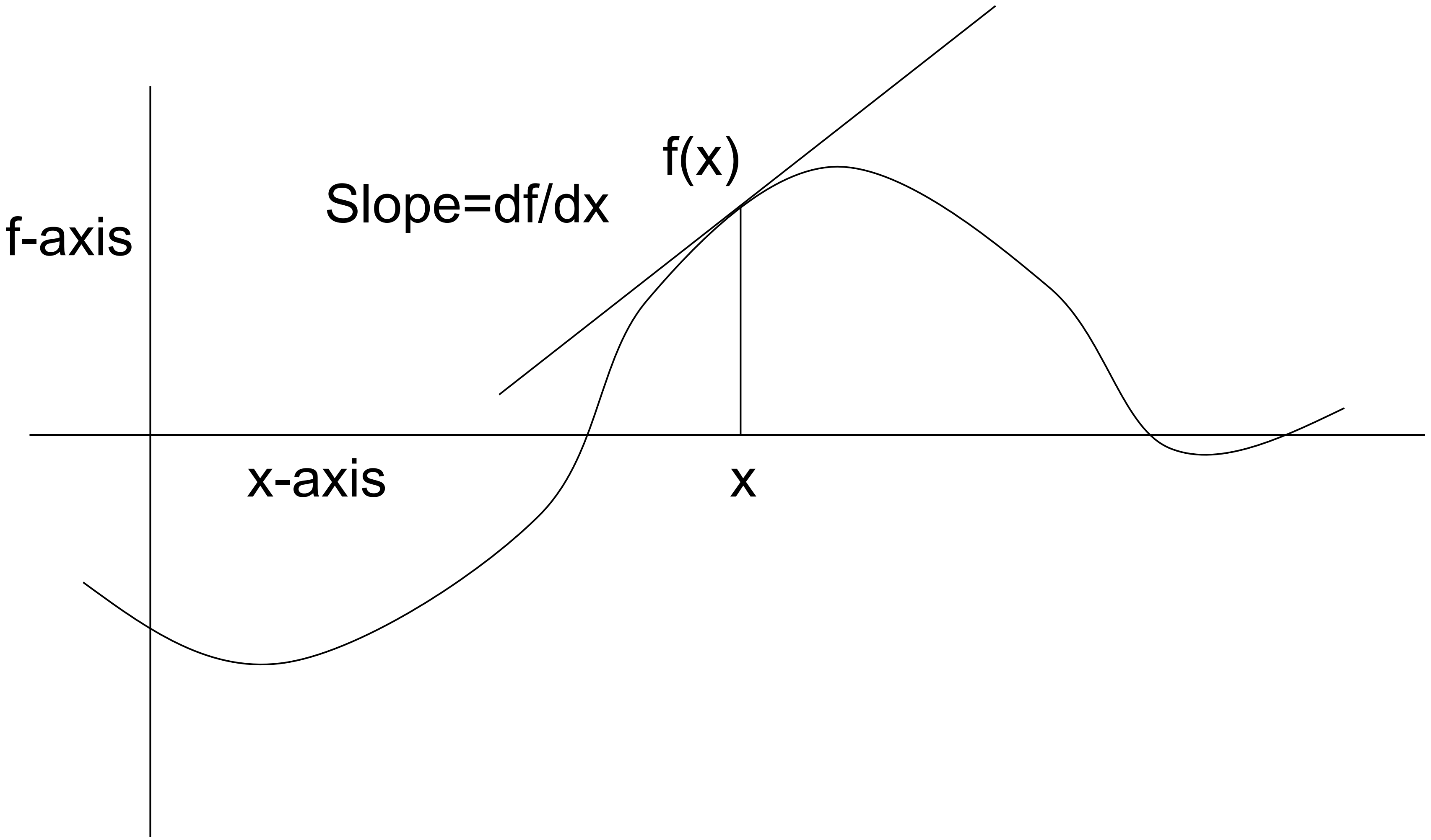
Review adapted from:
Steve Rotenberg, Computer Animation, UCSD
http://graphics.ucsd.edu/courses/cse169_w05

Derivative of a Scalar Function

- If we have a scalar function f of a single variable x, we can write it as f(x)
- Derivative of function with respect to x is df/dx
- The derivative is defined as:

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{\Delta f}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Derivative of a Scalar Function



Derivative of f(x)=x^2

For example: $f(x) = x^2$

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{(x+\Delta x)^2 - (x)^2}{\Delta x} \qquad = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

$$= \lim_{\Delta x \to 0} \frac{x^2 + 2x\Delta x + \Delta x^2 - x^2}{\Delta x}$$

$$= \lim_{\Delta x \to 0} \frac{2x\Delta x + \Delta x^2}{\Delta x}$$

$$= \lim_{\Delta x \to 0} (2x + \Delta x) = \boxed{2x}$$

Exact vs. Approximate

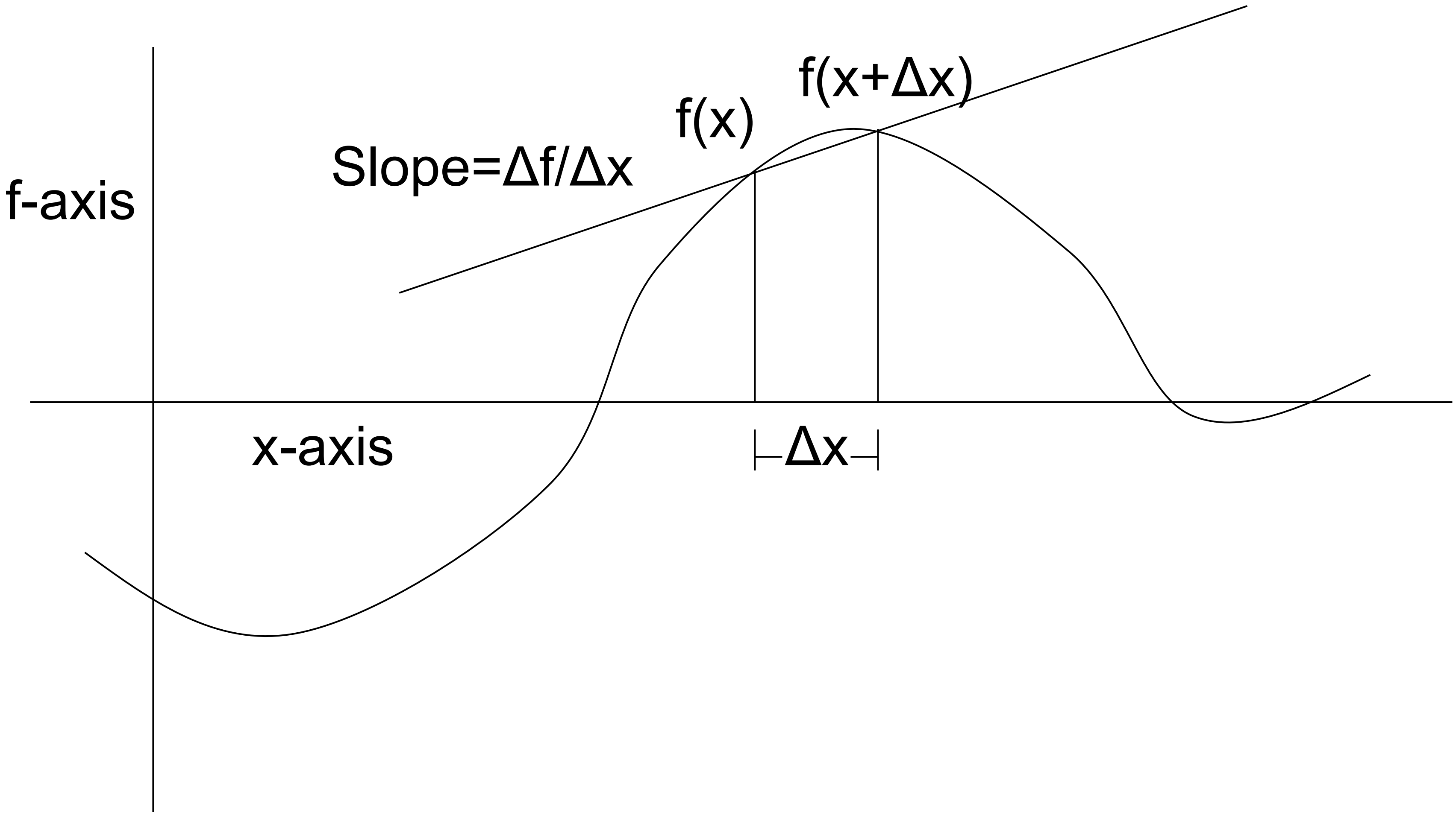
- Many algorithms require the computation of derivatives
- Sometimes, we can compute them. For example:

$$f(x) = x^2 \qquad \frac{df}{dx} = 2x$$

- Sometimes function is complex, can't compute an exact derivative
- As long as we can evaluate the function, we can always approximate a derivative

$$\frac{df}{dx} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x} \quad \text{for small } \Delta x$$

Approximate Derivative



Nearby Function Values

- If we know the value of a function and its derivative at some x, we can estimate what the value of the function is at other points near x

$$\frac{\Delta f}{\Delta x} \approx \frac{df}{dx}$$

$$\Delta f \approx \Delta x \frac{df}{dx}$$

$$f(x+\Delta x) \approx f(x) + \Delta x \frac{df}{dx}$$

Finding Solutions to f(x)=0

- There are many mathematical and computational approaches to finding values of x for which f(x)=0
- One such way is the *gradient descent* method
- If we can evaluate f(x) and df/dx for any value of x, we can always follow the gradient (slope) in the direction (currently) headed towards 0

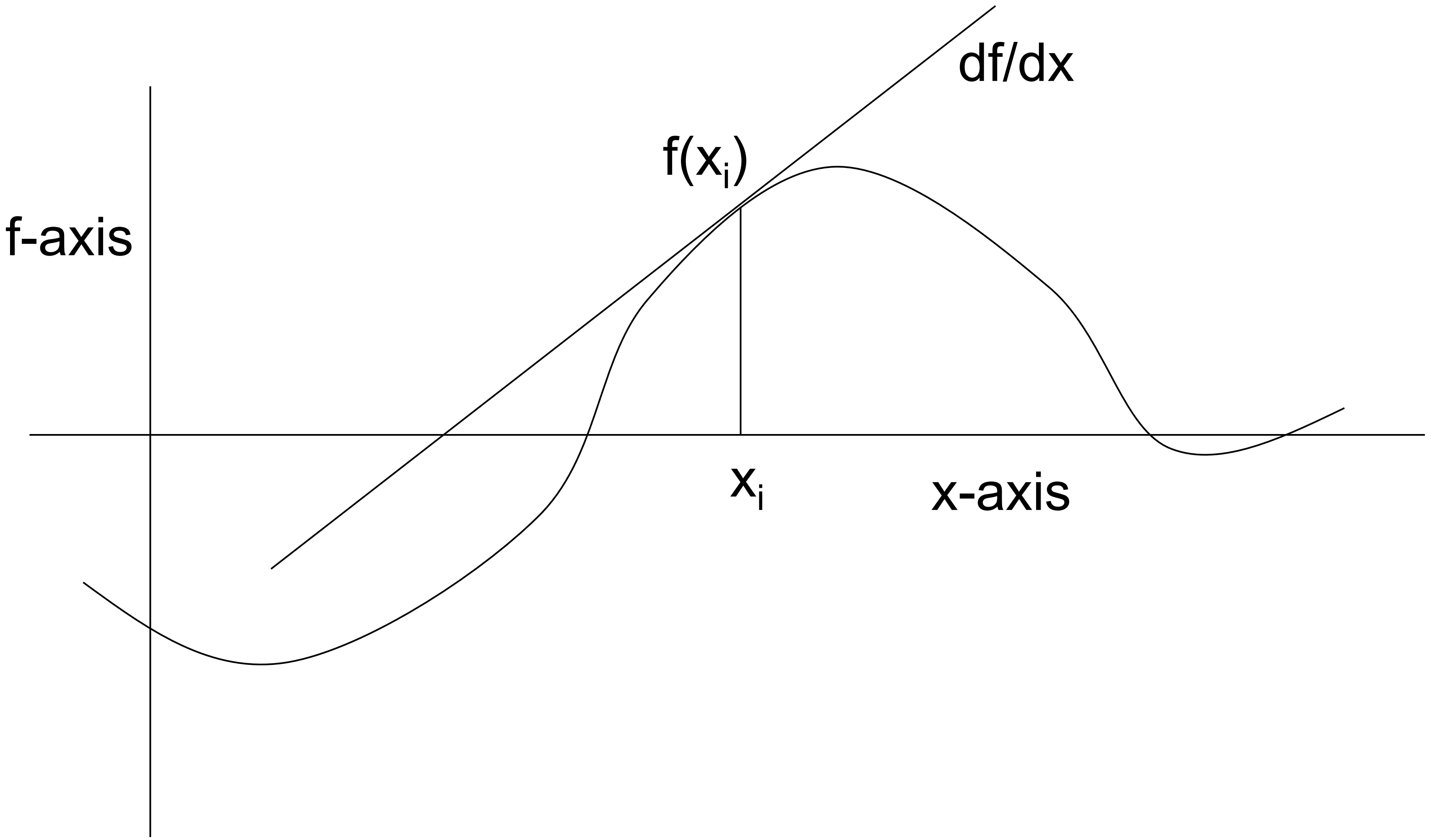
Gradient Descent

- We want to find the value of x that causes f(x) to equal 0
- We will start at some value x_0 and keep taking small steps:

  $$x_{i+1} = x_i + \Delta x$$

  until we find a value x_N that satisfies $f(x_N)=0$
- For each step, (try to) choose a value of Δx that gets closer to our goal
- Use the derivative as approximation of slope of function
- Use this to move 'downhill' towards zero

Gradient Descent



Minimization

- If $f(x_i)$ is not 0, the value of $f(x_i)$ can be thought of as an error
- Goal of gradient descent: minimize this error
  - Making it a member of the class **minimization algorithms**
- Each step Δx results in function changing its value
  - Call this Δf
- Ideally, $\Delta f = -f(x_i)$ – in other words, want to take a step Δx that causes Δf to cancel out the error
- Realistically, hope each step brings us closer, and we can eventually stop when we get close enough
- This iterative process is consistent with *numerical* algorithms

Choosing Δx Step

- Safety vs. efficiency
  - If step size is too small, converges very slowly
  - If step size is too large, algorithm not reduce f.
    - Because the first order approximation is valid only locally.
- If function varies widely, what is safest?
- If we have a relatively smooth function?
- If we feel very confident?
  - We could try stepping directly to where linear approximation passes through 0