

# Game Architecture

3/29/16: Inverse Kinematics

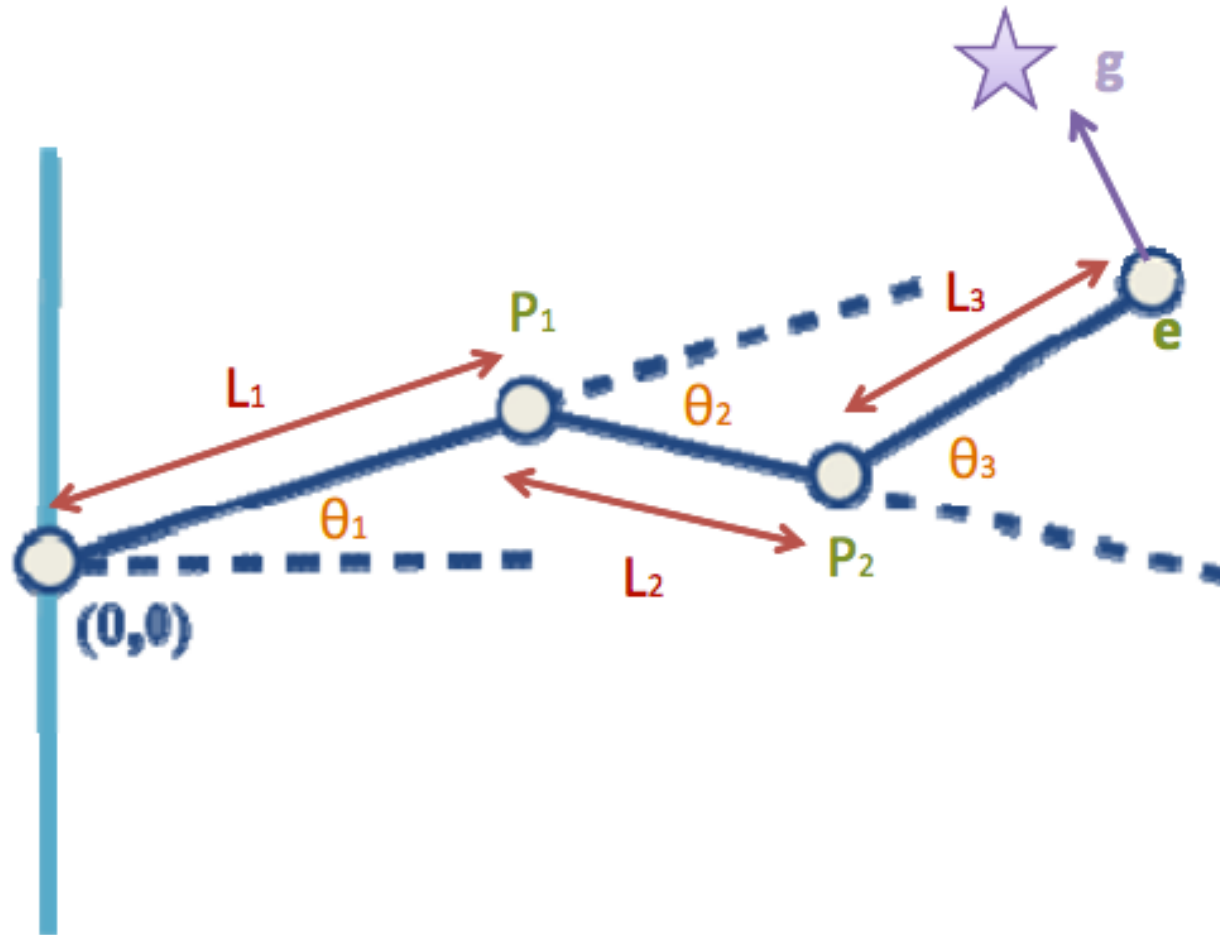
# Missile Guidance

$$e = f(\theta)$$

Forward Kinematics (FK)

$$\theta = f^{-1}(e)$$

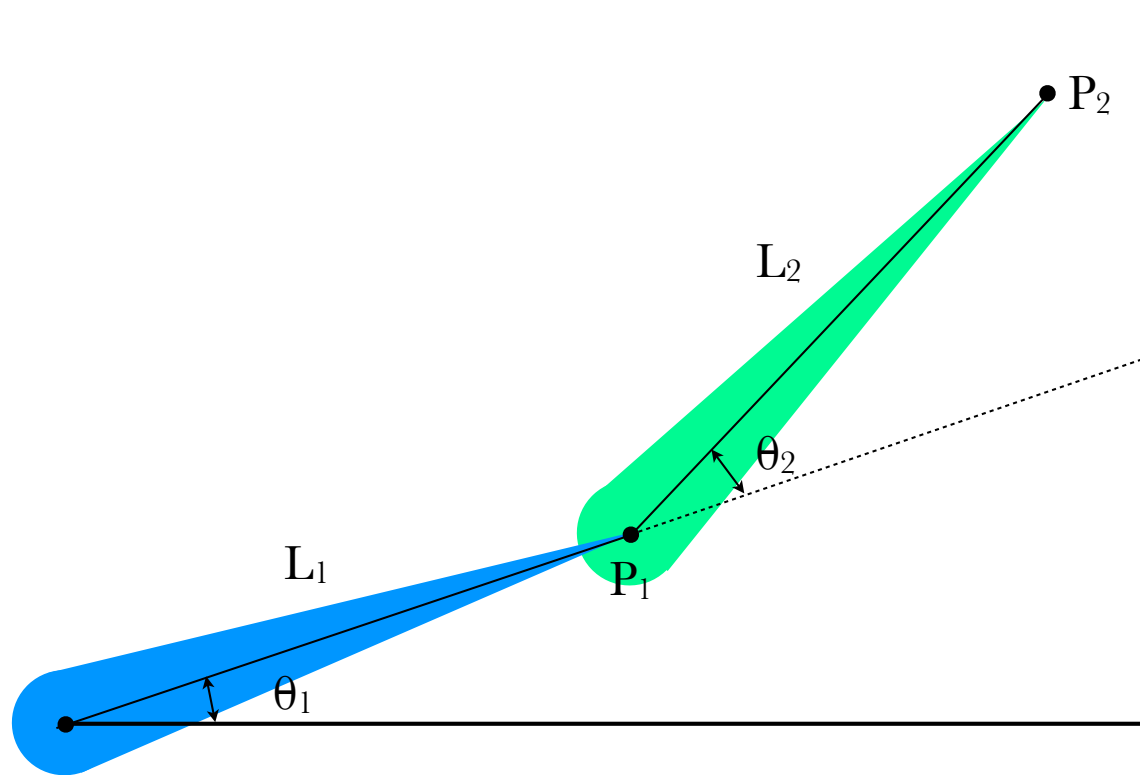
Inverse Kinematics (IK)



# Types of Solutions

- Analytical
  - Equation that can be directly solved
  - Preferred, practically impossible as a general solution
- Numerical
  - Expensive
  - Inaccurate
  - Unfortunately only practical option for general solver.

# Closed Form / Analytical Solution



$$Px_1 = L_1 * \cos(\theta_1)$$

$$Py_1 = L_1 * \sin(\theta_1)$$

$$Px_2 = Px_1 + (L_2 * \cos(\theta_1 + \theta_2))$$

$$Py_2 = Py_1 + (L_2 * \sin(\theta_1 + \theta_2))$$

$$Px_2 = (L_1 * \cos(\theta_1)) + (L_2 * \cos(\theta_1 + \theta_2))$$

$$Py_2 = (L_1 * \sin(\theta_1)) + (L_2 * \sin(\theta_1 + \theta_2))$$

Using:

$$\cos(a + b) = \cos(a) \cos(b) - \sin(a) \sin(b)$$

$$\sin(a + b) = \cos(a) \sin(b) + \sin(a) \cos(b)$$

we get:

$$Px_2 = L_1 \cos \theta_1 + L_2 \cos \theta_1 \cos \theta_2 - L_2 \sin \theta_1 \sin \theta_2$$

$$Py_2 = L_1 \sin \theta_1 + L_2 \cos \theta_1 \sin \theta_2 + L_2 \sin \theta_1 \cos \theta_2$$

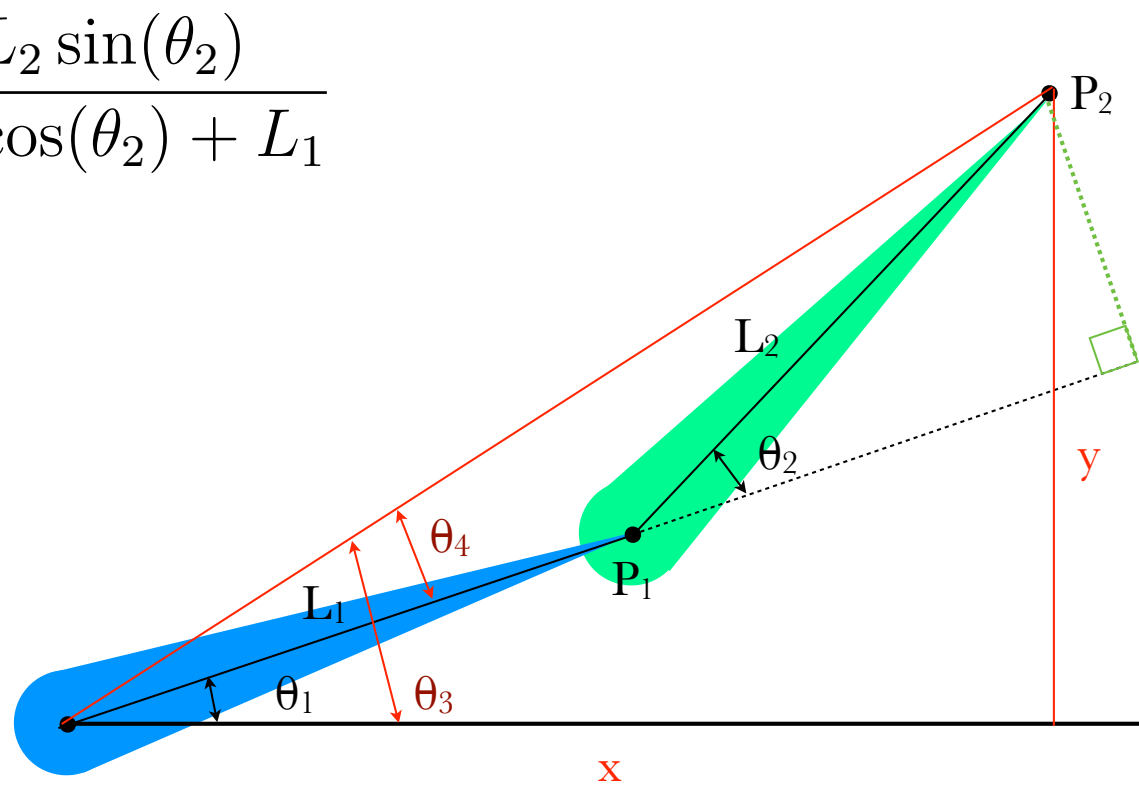
after rearranging this a whole bunch to get rid of the  $\theta_1$  term:

$$\theta_2 = \arccos \left( \frac{Px_2^2 + Py_2^2 - L_1^2 - L_2^2}{2L_1L_2} \right)$$

$$\tan(\theta_3) = y/x$$

$$\tan(\theta_4) = \frac{L_2 \sin(\theta_2)}{L_2 \cos(\theta_2) + L_1}$$

$$\theta_1 = \theta_3 - \theta_4$$





Using:

$$\tan(a - b) = \frac{\tan(a) - \tan(b)}{1 + \tan(a) \tan(b)}$$

and doing some substitution:

$$\tan(\theta_1) = \frac{\frac{y}{x} - \frac{L_2 \sin(\theta_2)}{L_2 \cos(\theta_2) + L_1}}{1 + \left( \frac{y}{x} * \frac{L_2 \sin(\theta)}{L_2 \cos(\theta_2) + L_1} \right)}$$

simplifying:

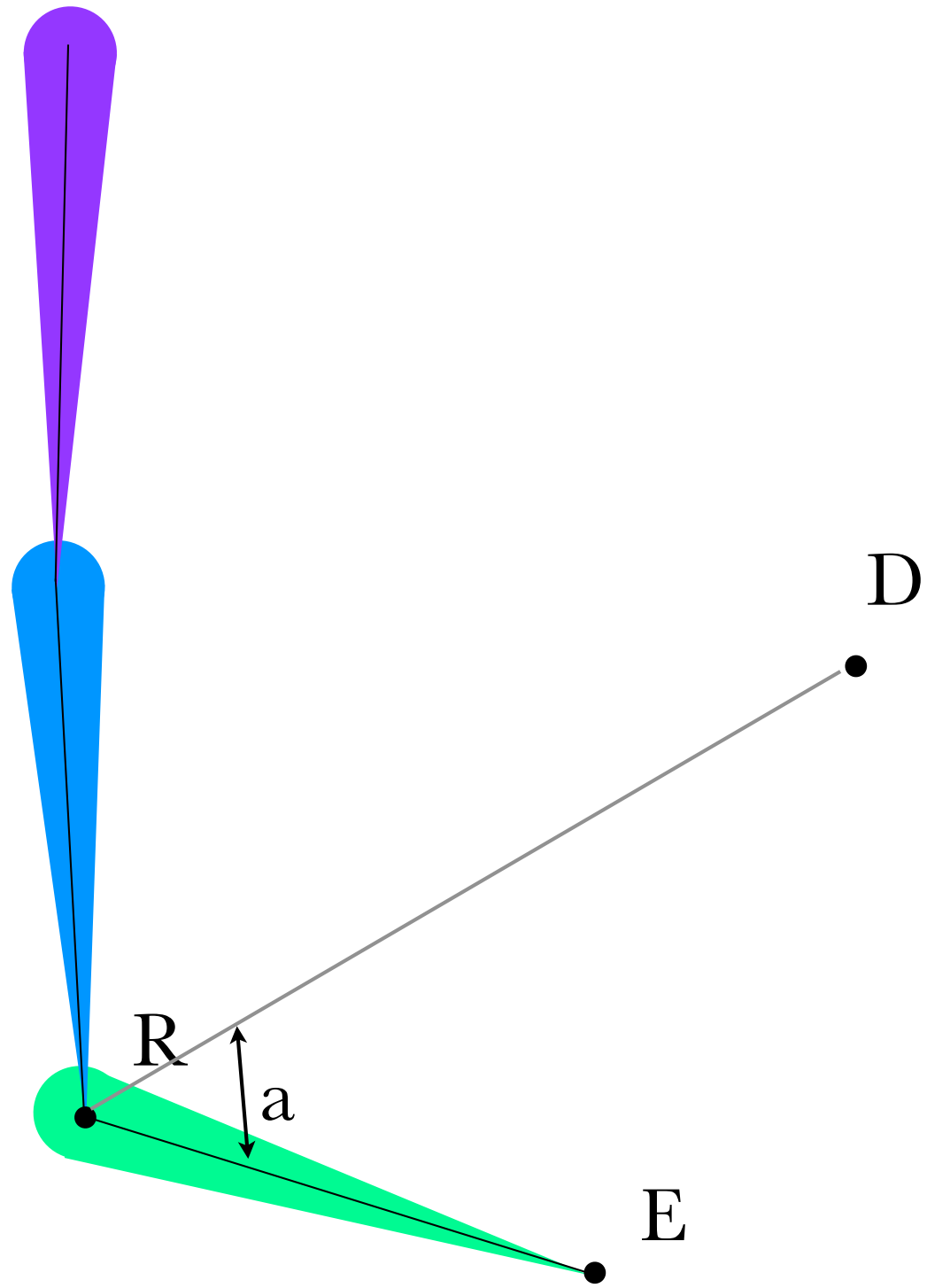
$$\theta_1 = \arctan \left( \frac{y(L_2 \cos(\theta_2) + L_1) - x(L_2 \sin(\theta_2))}{x(L_2 \cos(\theta_2) + L_1) + y(L_2 \sin(\theta_2))} \right)$$

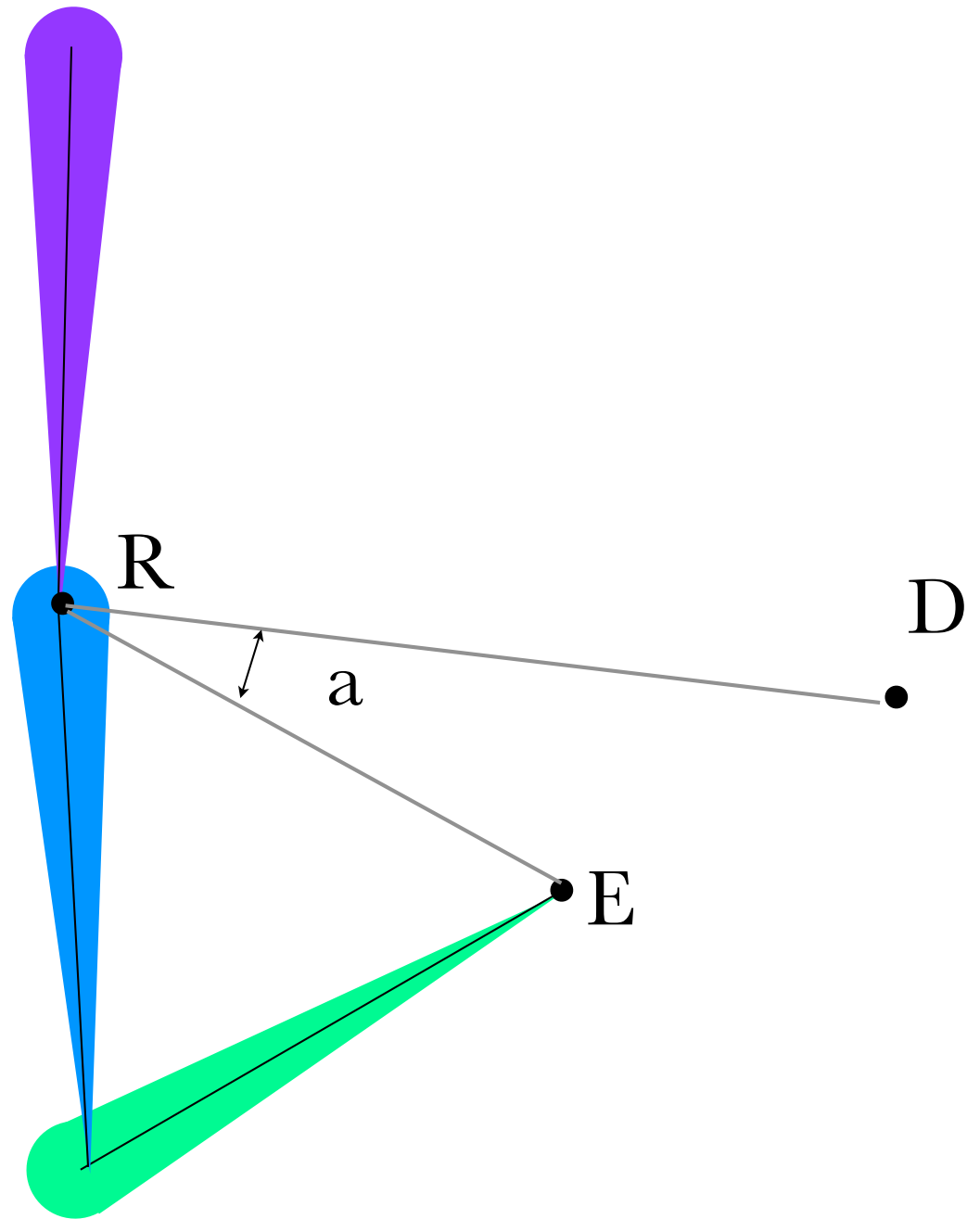
# Problems

- Gets complicated as  $n$  increases
- Humans have more DOFs than needed to reach goal
- Sets of nonlinear equations often have no closed form
- Iterate numerically

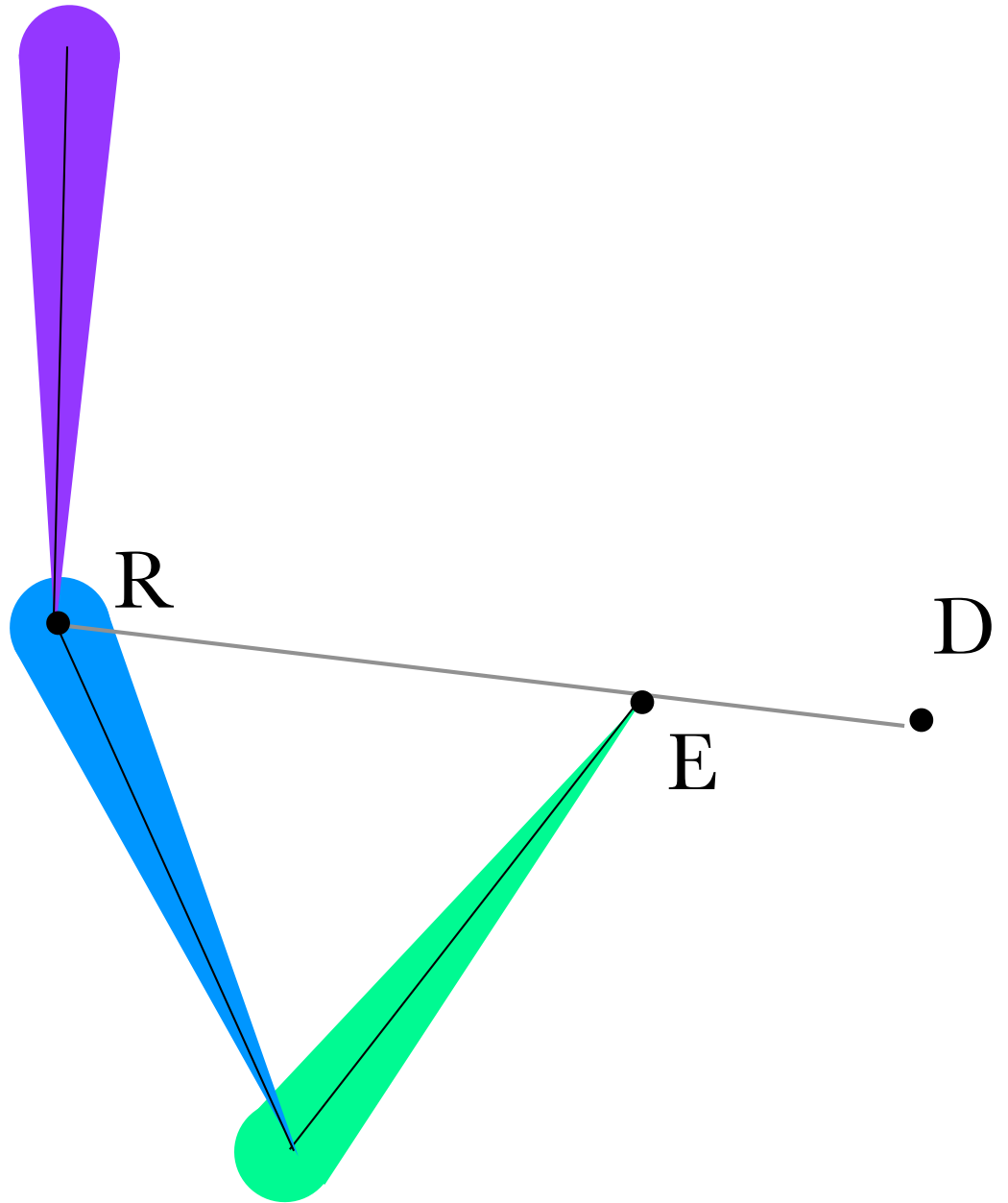
# Cyclic-Coordinate Descent

$$\cos(a) = \overrightarrow{RD} \cdot \overrightarrow{RE}$$





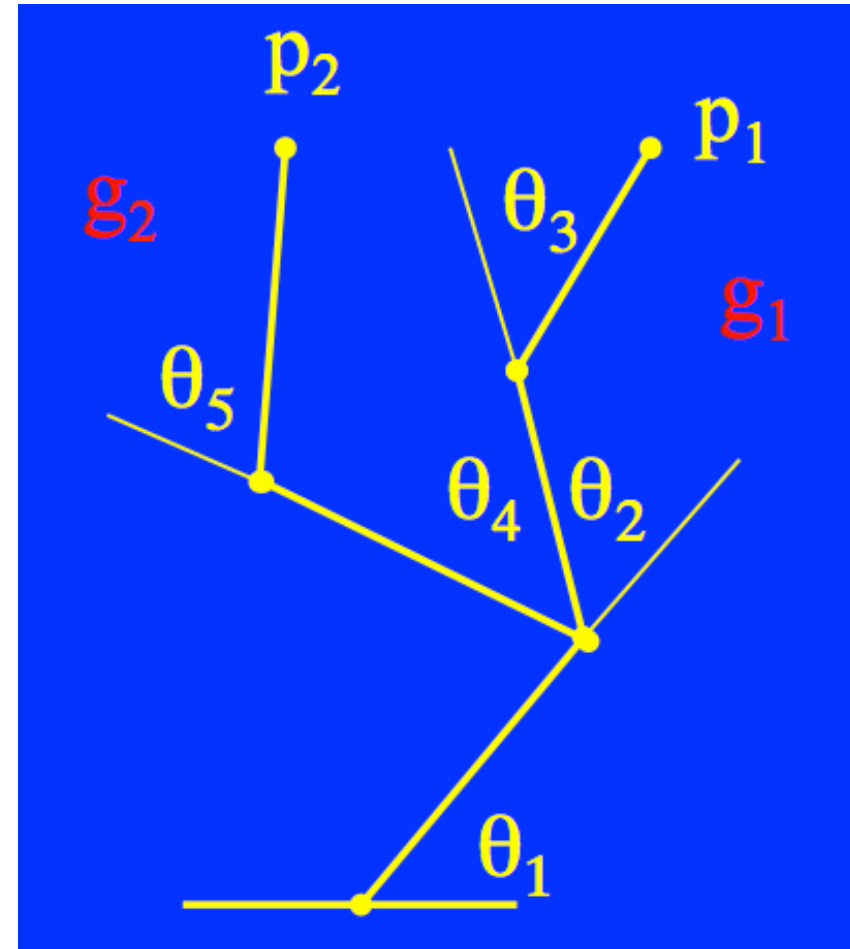
$$\cos(a) = \overrightarrow{RD} \cdot \overrightarrow{RE}$$



$$\cos(a) = \overrightarrow{RD} \cdot \overrightarrow{RE}$$

# Problems

- Only handles serial chains
- Multiple goals needed for humans
- How to distribute desired angle behavior?
- Can just average, but won't satisfy/prioritize all goals



# Jacobian Inverse



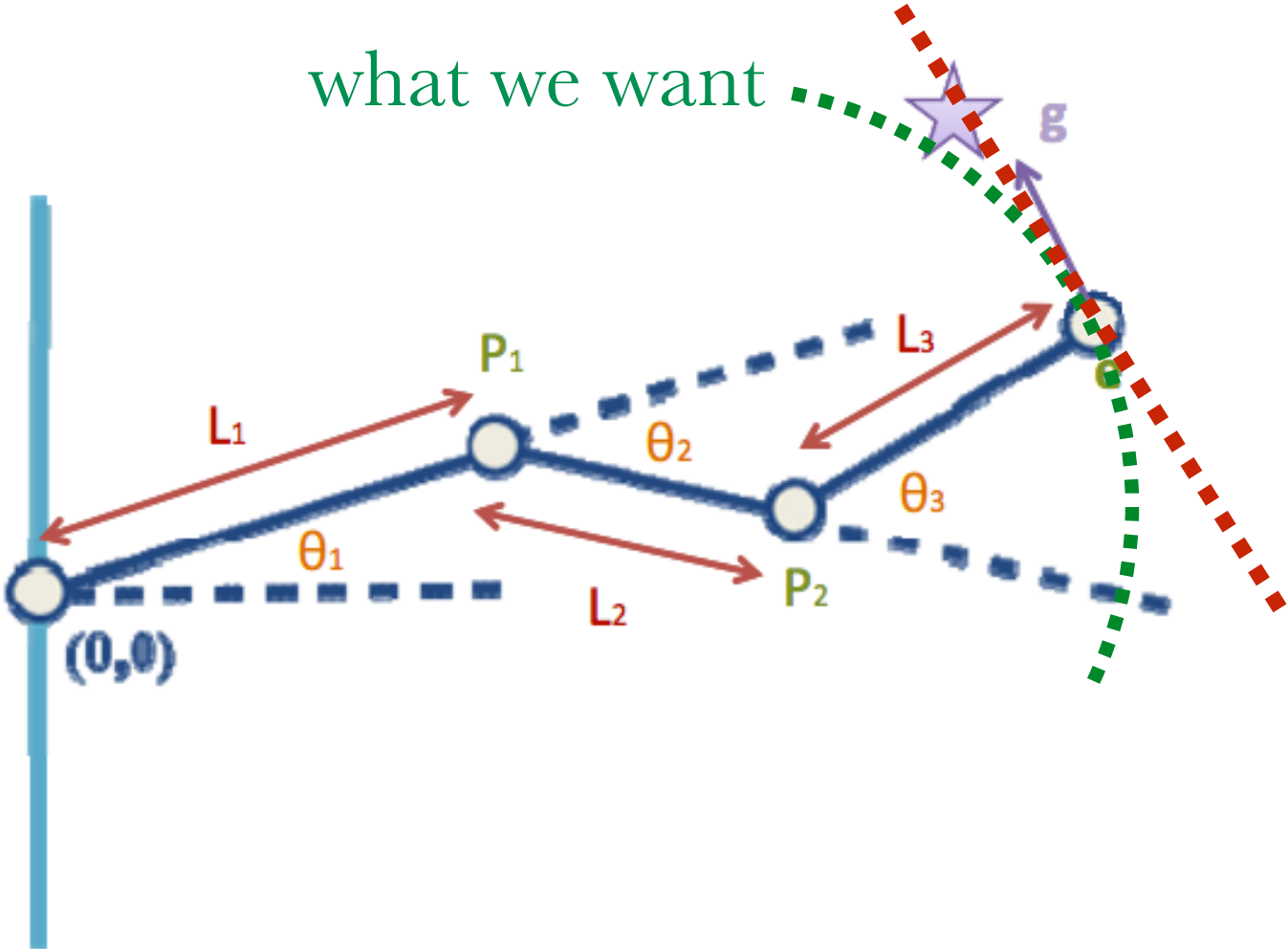
# The Jacobian

- What is it? A linear approximation to  $f$
- Matrix of partial derivatives of entire system.
- In this case, defined relative to  $e$
- Defines how  $e$  changes relative to instantaneous changes in the system.

$$\partial e = J \partial \theta$$

linear approximation

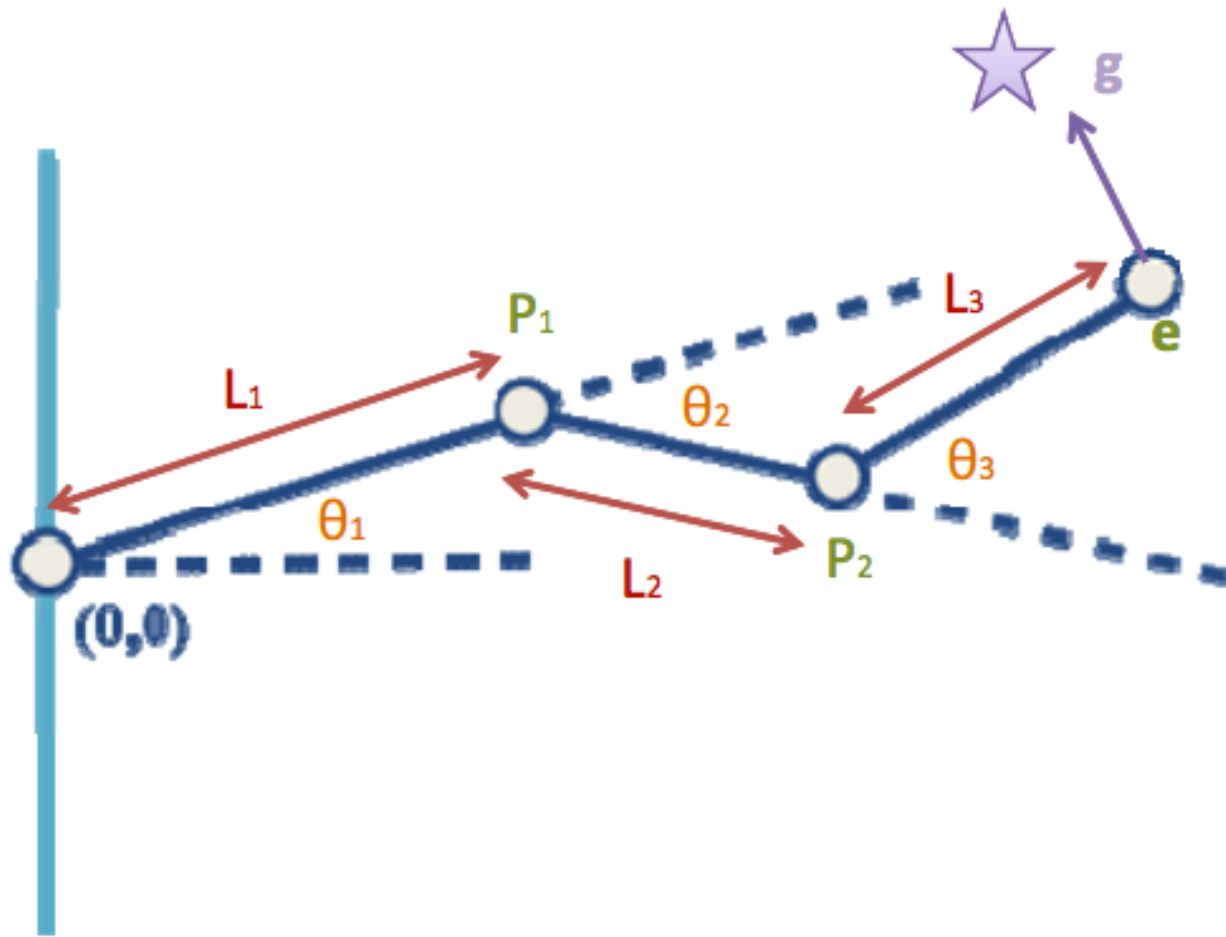
what we want



# Jacobian Entries

- The entries in the Jacobian matrix are usually very easy to calculate.
- If the  $j^{\text{th}}$  joint is a rotational joint with a single degree of freedom, the joint angle is a single scalar  $\theta_j$ . Let  $\mathbf{p}_j$  be the position of the joint and let  $\mathbf{v}_j$  be a unit vector pointing along the current axis of rotation for the joint
- In this case, if angles are measured in radians with the direction of rotation given by the right-hand rule and if the  $i^{\text{th}}$  end effector is affected by the joint, then the corresponding entry in the Jacobian is:

$$\frac{\partial \mathbf{s}_i}{\partial \theta_j} = \mathbf{v}_j \times (\mathbf{s}_i - \mathbf{p}_j)$$



$$J = \begin{bmatrix} ((0, 0, 1) \times (\mathbf{e}))_x & ((0, 0, 1) \times (\mathbf{e} - \mathbf{P}_1))_x & ((0, 0, 1) \times (\mathbf{e} - \mathbf{P}_2))_x \\ ((0, 0, 1) \times (\mathbf{e}))_y & ((0, 0, 1) \times (\mathbf{e} - \mathbf{P}_1))_y & ((0, 0, 1) \times (\mathbf{e} - \mathbf{P}_2))_y \\ ((0, 0, 1) \times (\mathbf{e}))_z & ((0, 0, 1) \times (\mathbf{e} - \mathbf{P}_1))_z & ((0, 0, 1) \times (\mathbf{e} - \mathbf{P}_2))_z \end{bmatrix}$$

# Except

- Jacobian defines changes in  $e$  relative to changes in  $\theta$
- Want to know how a desired change in  $e$  maps to changes in  $\theta$
- Recall that the equation of inverse kinematics is defined as  $\theta = f^{-1}(e)$
- Therefore we need  $\partial\theta = J^{-1}\partial e$

# What if We Can't Invert?

- Cheat!
- Pseudo-inverse

$$J^+ = (J^T J)^{-1} J^T$$

# Algorithm

- Calculate Jacobian
- Calculate pseudo inverse
- Calculate  $\partial e$  (goal-current)
- Calculate error  $|(I - J^+ J)\partial e|$
- If error  $>$  tolerance, repeat previous step with halved  $\partial e$
- Calculate  $\partial\theta = J^+ \partial e$
- Apply to whole system
- Repeat either max steps or until  $e$  is close to goal

we try to solve:

$$y_1 = f(x_0 + \Delta x)$$

where  $y_1$  is a desired end effector position,  $x_0$  is a current parameter vector, and  $\Delta x$  is the unknown we're trying to find that modifies  $x_0$  to reach  $y_1$

Taylor expansion:

$$y_1 = f(x_0) + \frac{\partial f}{\partial x}(x_0)\Delta x + O\|\Delta x\|^2$$

since  $f(x)$  is a vector function, we now have a matrix of first-order partial derivatives, often denoted  $J$

$$y_1 = f(x_0) + J(x_0)\Delta x$$

$$\Delta x = J(x_0)^{-1}(y_1 - y_0)$$



Broyden–Fletcher–Goldfarb–Shanno  
(BFGS) method

search direction is found by solving:

$$p_k^N = -\nabla^2 f_k^{-1} \nabla f_k$$

where  $\nabla^2 f_k$  is the Hessian matrix and  $\nabla f_k$  is the gradient of the objective function, given by

$$\nabla f_k = J(e - g)$$

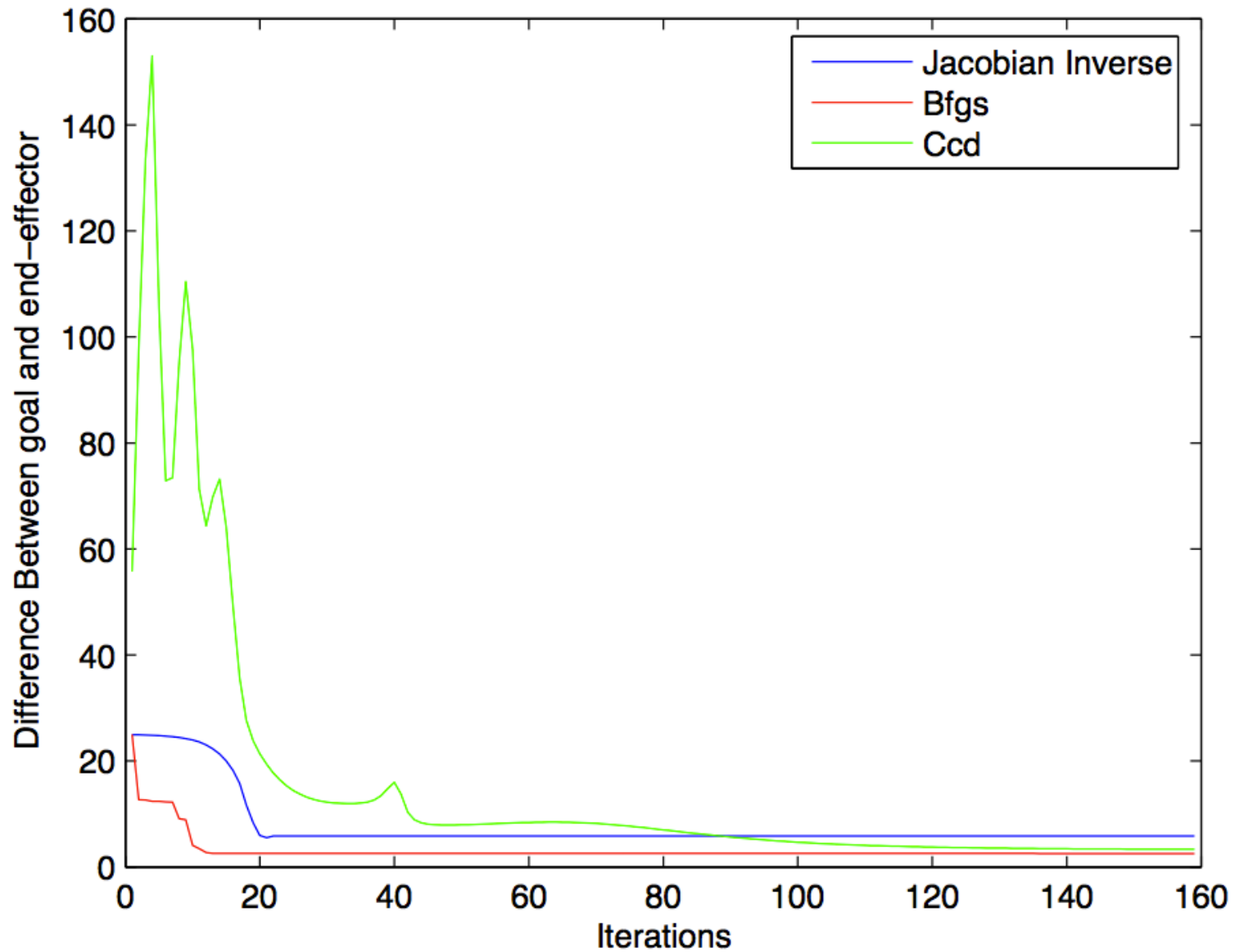
where  $e$  is the end-effector position and  $g$  is the goal position.

$$B_{i+1} = B_i - \frac{B_i s_i s_i^T B_i}{s_i^T B_i s_i} + \frac{g_i g_i^T}{g_i^T s_i}$$

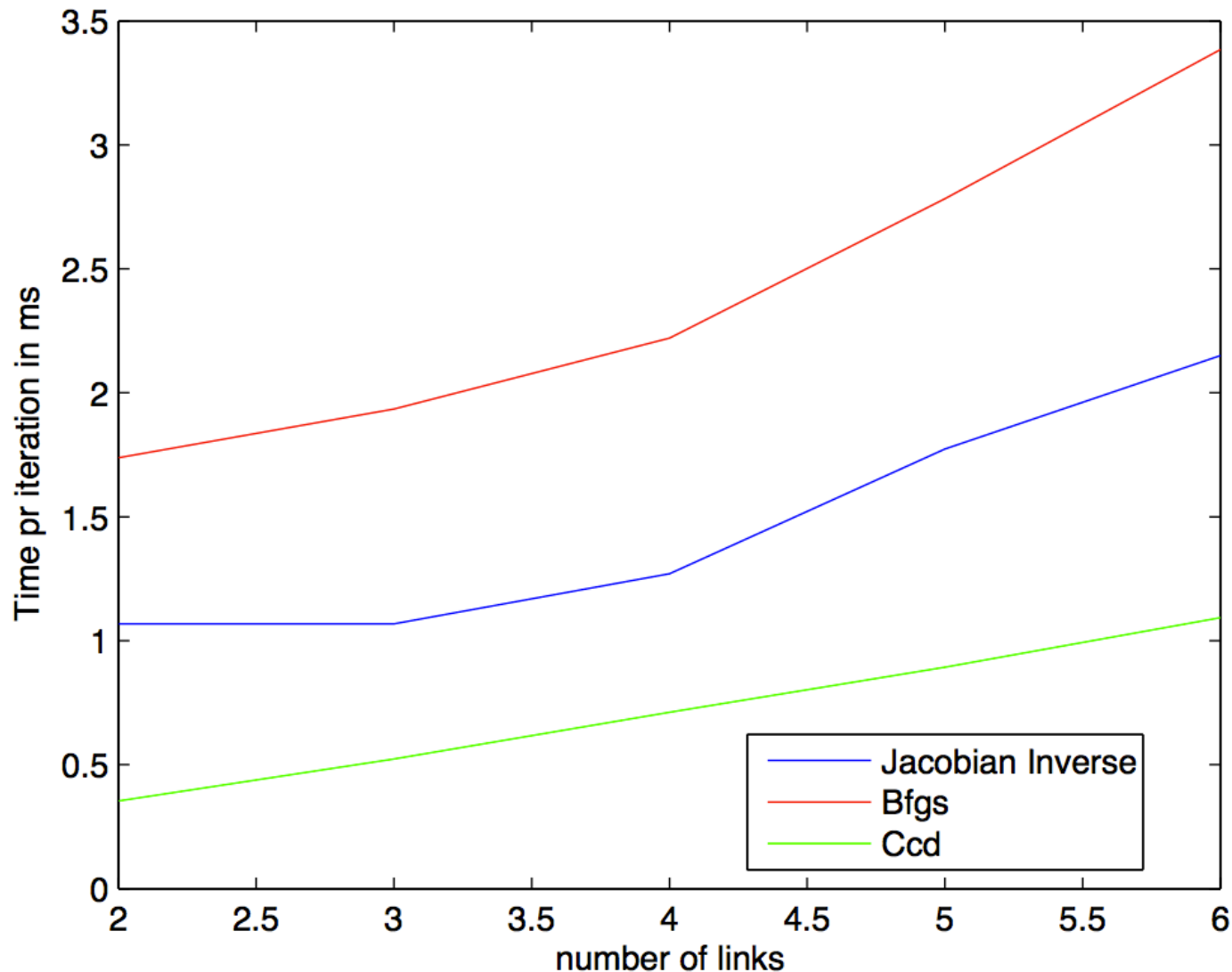
$$s_i = x_{i+1} - x_i$$

$$g = \nabla f(x_{i+1}) - \nabla f(x_i)$$

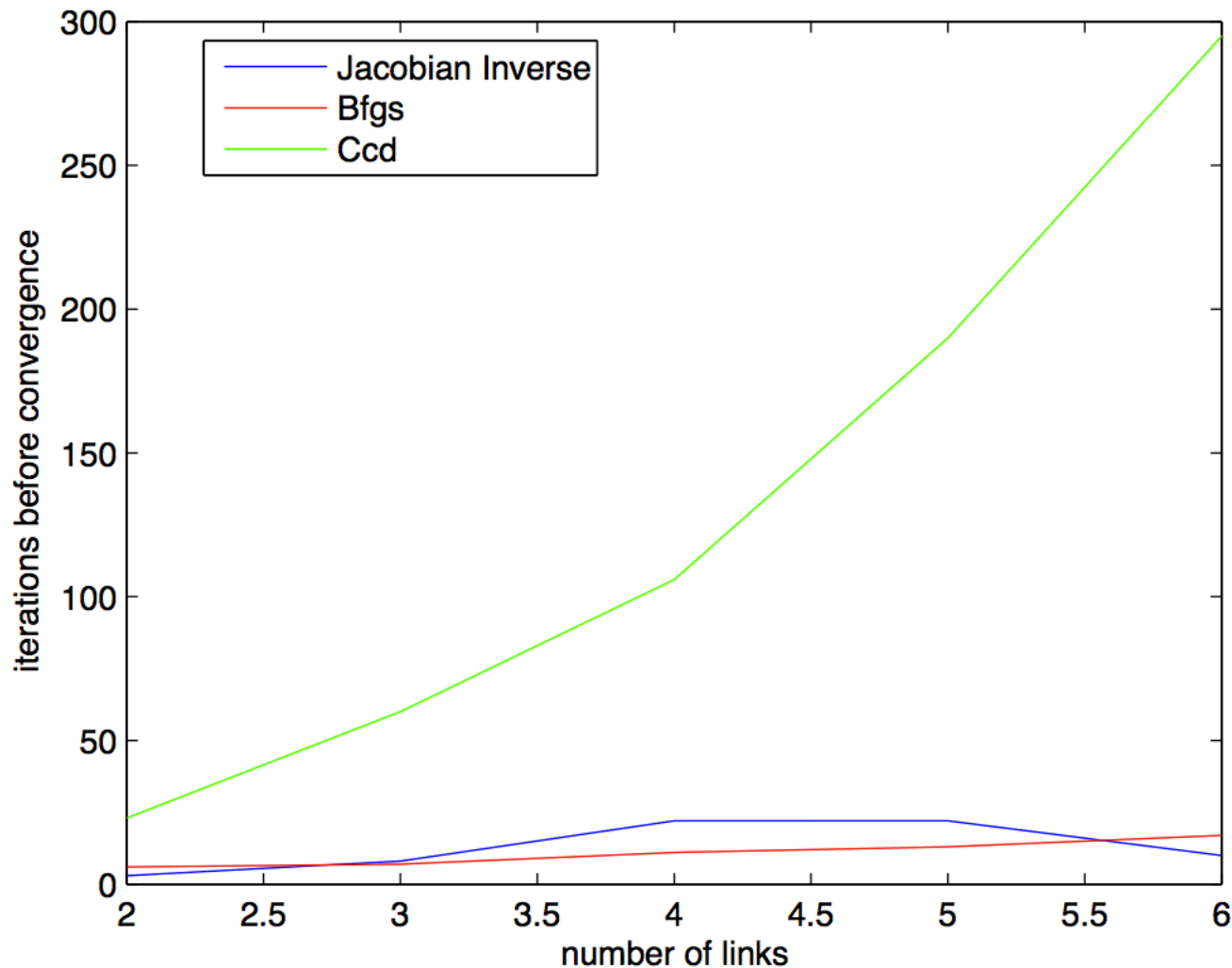
# Convergence



# Milliseconds per Iteration for Increasing Number of Joints



# Number of Iterations as a Function of Chain Length



# Precision

Method	Distance to Goal
JI	5.853667
CCD	3.368891
BFGS	2.534875
Optimal	2

# Qualitatively...

Method	Convergence	Speed per Iteration	Number of Iterations	Precision
CCD	Meh	Pretty good	Awful	Not Bad
JI	Good	Right in the middle	Good	Pretty Bad
BFGS	Awesome	Not great	Good	Good