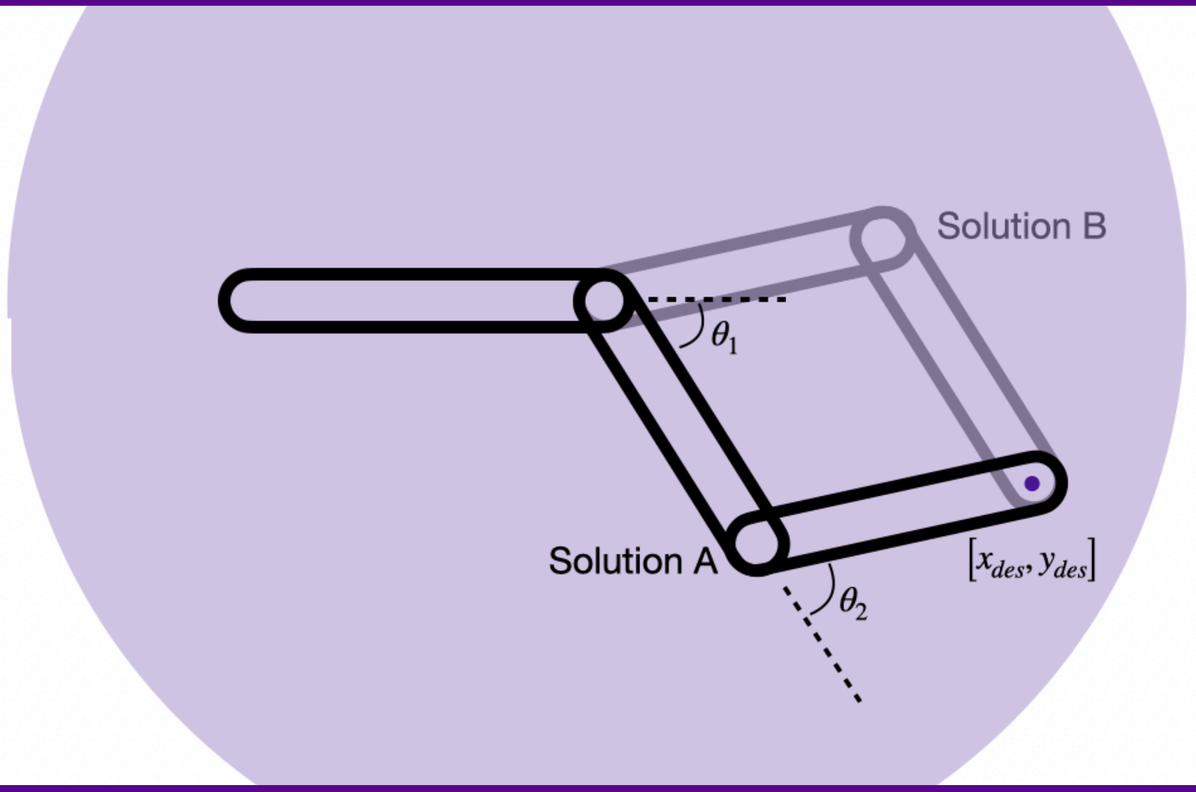
ROB-UY 2004

Robotic Manipulation & Locomotion

Lecture 06A - Inverse Kinematics



Agenda

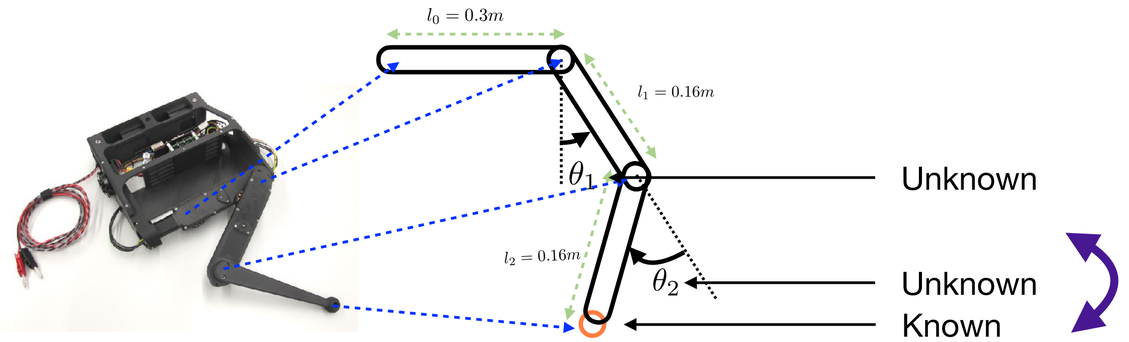
1. Inverse Kinematics Introduction
2. How many solutions?
3. Optimization
4. Python Code Examples

Agenda

1. **Inverse Kinematics Introduction**
2. How many solutions?
3. Optimization
4. Python Code Examples

Inverse Kinematics

- Given the pose of the end-effector (or some specific part of the robot) with respect to a stationary base frame, determine the joint angles of the robot's revolute joints, (and/or lengths of the robot's prismatic joints).



Forward Kinematics

$$T_{SF} = \begin{bmatrix} cos(\theta_1+\theta_2) & -sin(\theta_1+\theta_2) & 0 & (l_3 cos\theta_2 + l_2)cos\theta_1 - l_3 sin\theta_1 sin\theta_2 + l_1 \\ sin(\theta_1+\theta_2) & cos(\theta_1+\theta_2) & 0 & (l_3 cos\theta_2 + l_2)sin\theta_1 + l_3 cos\theta_1 sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

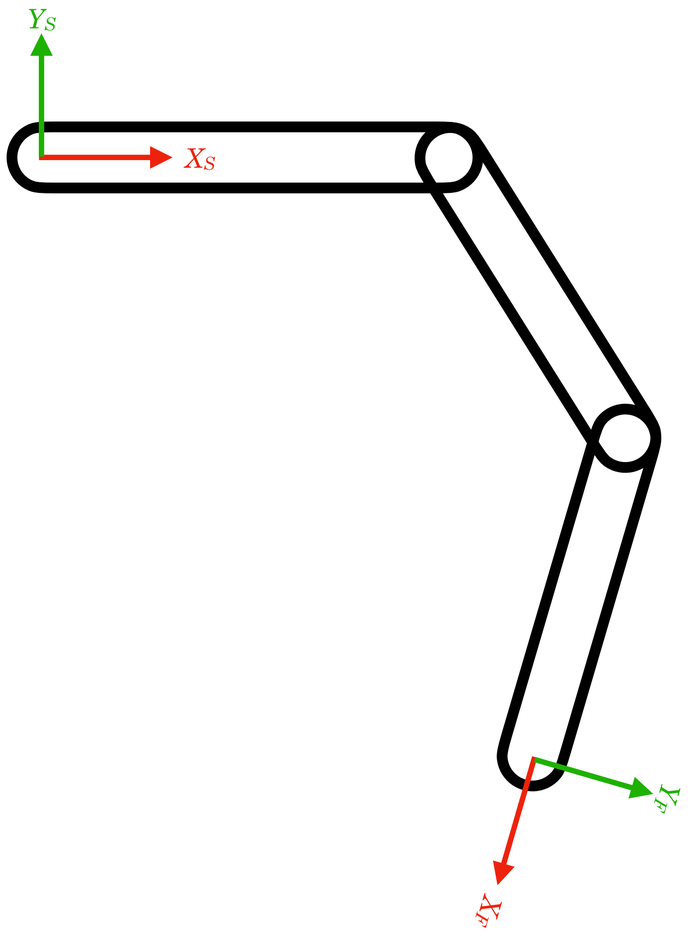
The position of the foot in the frame $\{S\}$!

Forward Kinematics

- Solve for x_f and y_f
- They are a function of known variables

$$x_{SF} = (l_3 cos\theta_2 + l_2)cos\theta_1 - l_3 sin\theta_1 sin\theta_2 + l_1$$

$$y_{SF} = (l_3 cos\theta_2 + l_2)sin\theta_1 + l_3 cos\theta_1 sin\theta_2$$

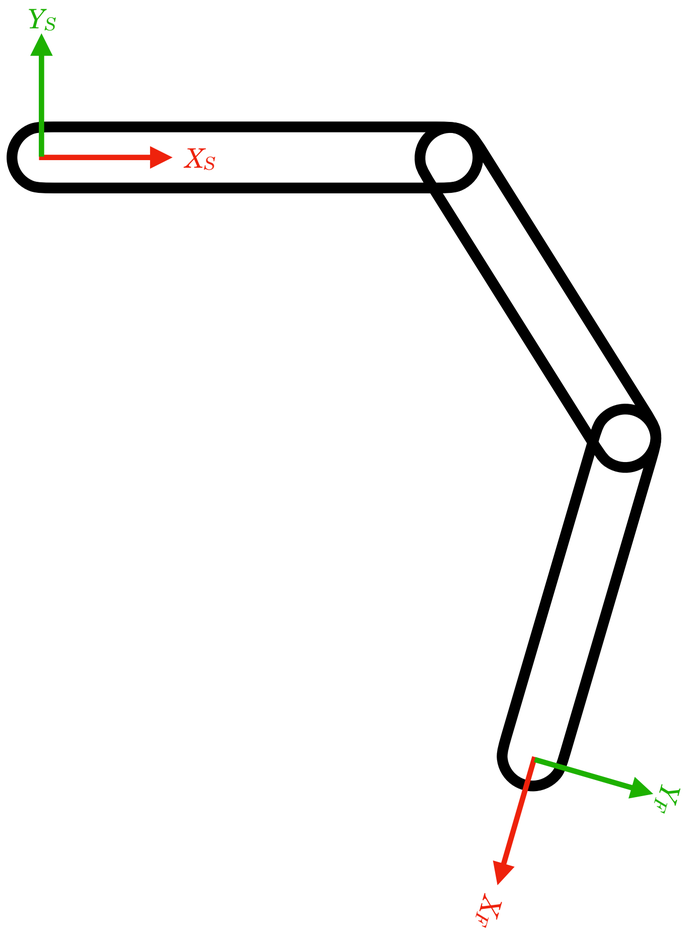


Inverse Kinematics

- Solve for θ_1 and θ_2
- Two equations, two unknowns
- Nonlinear

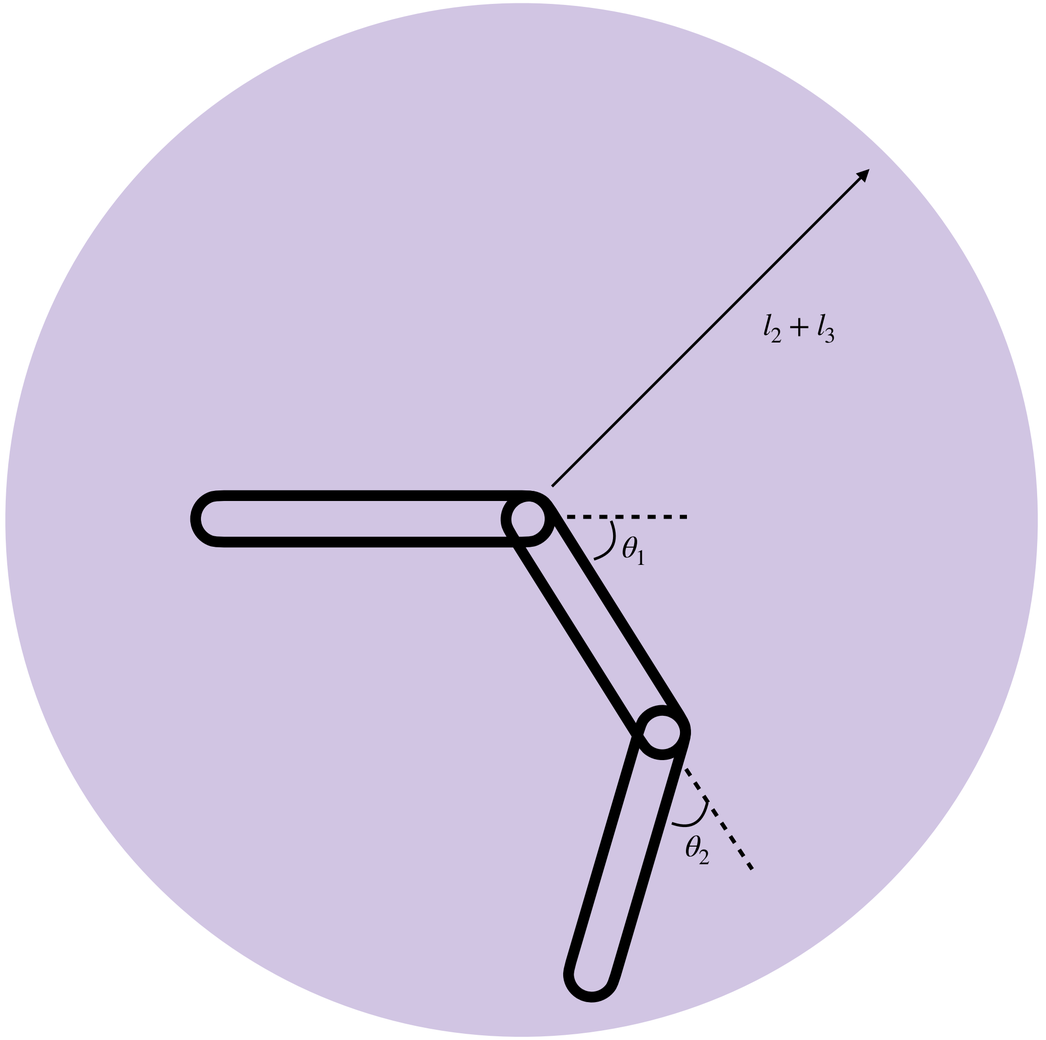
$$(l_3 cos\theta_2 + l_2)cos\theta_1 - l_3 sin\theta_1 sin\theta_2 + l_1 = x_{SF}$$

$$(l_3 cos\theta_2 + l_2)sin\theta_1 + l_3 cos\theta_1 sin\theta_2 = y_{SF}$$



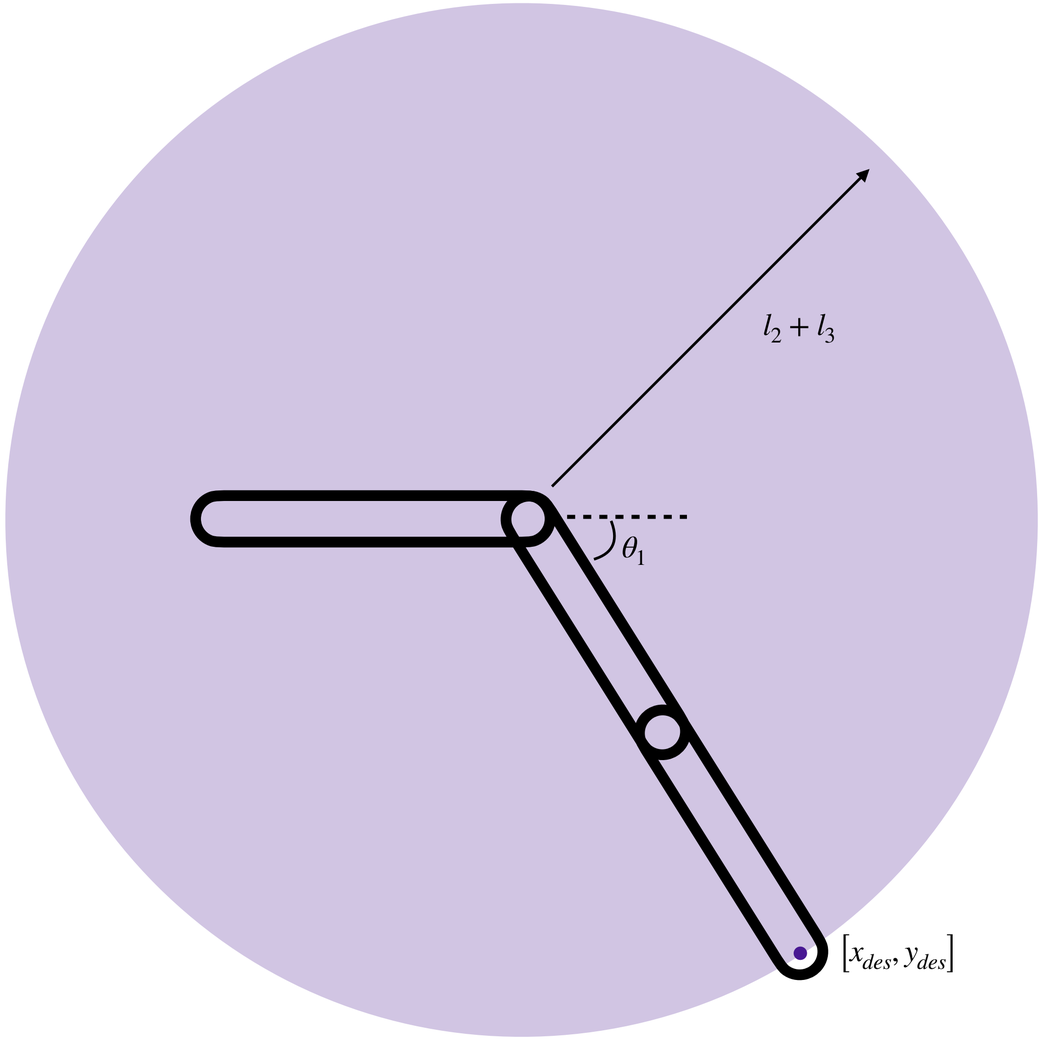
Agenda

1. Inverse Kinematics Introduction
2. **How many solutions?**
3. Optimization
4. Python Code Examples


$l_2 + l_3$
θ_1
θ_2

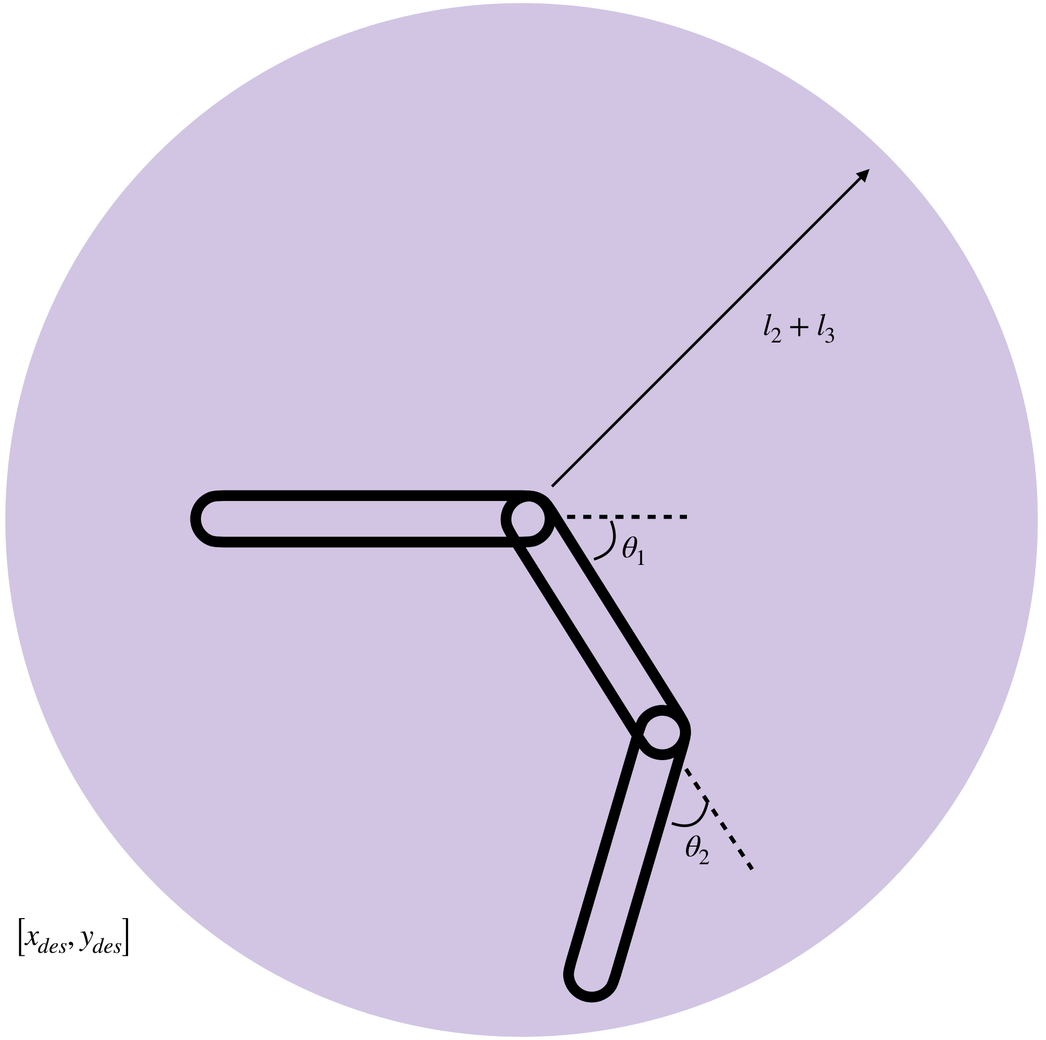
Case 1 $\sqrt{(x_{desired} - l_0)^2 + (y_{desired})^2} = l_1 + l_2$

$\theta_2 = 0$ and $\theta_1 = \frac{\pi}{2} + \arctan 2(y_{des}, x_{des} - l_0)$



Case 2 $\sqrt{(x_{desired} - l_0)^2 + (y_{desired})^2} > l_1 + l_2$

No solutions!



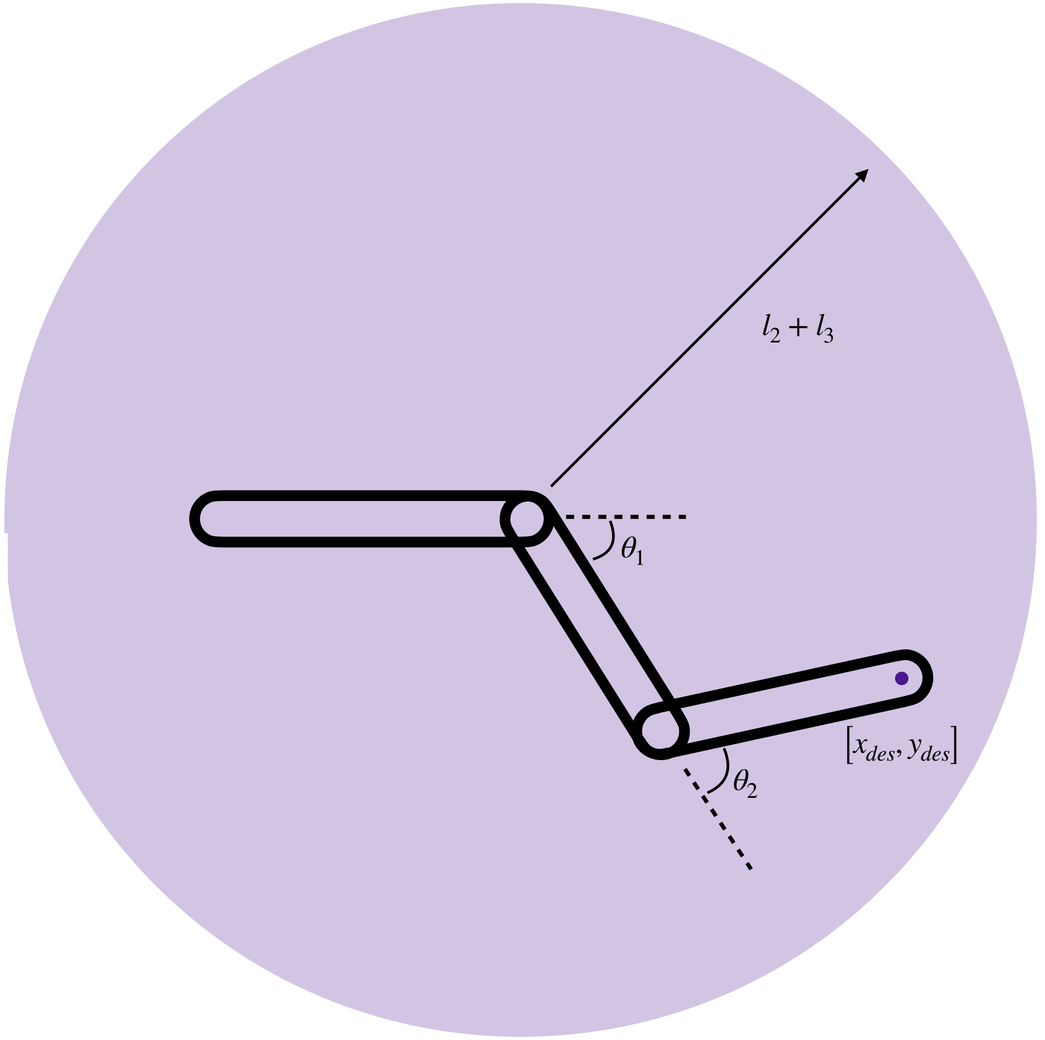
- $[x_{des}, y_{des}]$

Case 3 $\sqrt{(x_{desired} - l_0)^2 + (y_{desired})^2} < l_1 + l_2$

2 solutions!

$$\theta_1 = \frac{\pi}{2} + \arctan 2(y_{des}, x_{des} - l_0) \mp \arccos(\frac{-l_2^2 + l_1^2 + l_{des}^2}{2 l_1 l_{des}})$$

$$\theta_2 = \pm \arccos(\frac{l_{des}^2 - l_1^2 - l_2^2}{l_1 l_2})$$

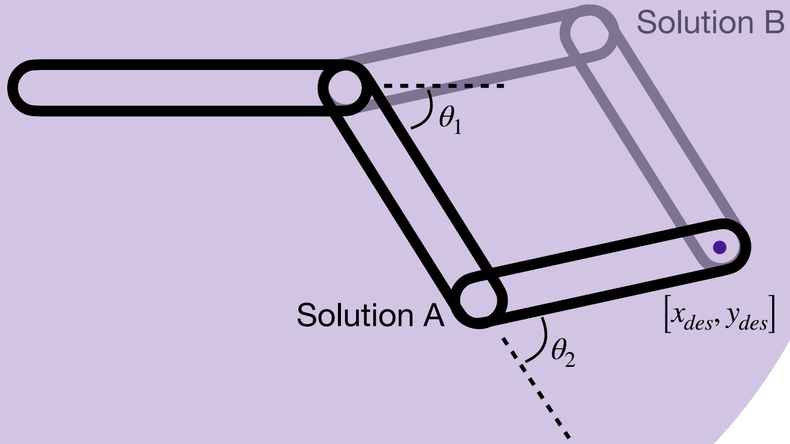


Case 3 $\sqrt{(x_{desired} - l_0)^2 + (y_{desired})^2} < l_1 + l_2$

2 solutions!

$$\theta_1 = \frac{\pi}{2} + \arctan 2(y_{des}, x_{des} - l_0) \mp \arccos(\frac{-l_2^2 + l_1^2 + l_{des}^2}{2 l_1 l_{des}})$$

$$\theta_2 = \pm \arccos(\frac{l_{des}^2 - l_1^2 - l_2^2}{l_1 l_2})$$

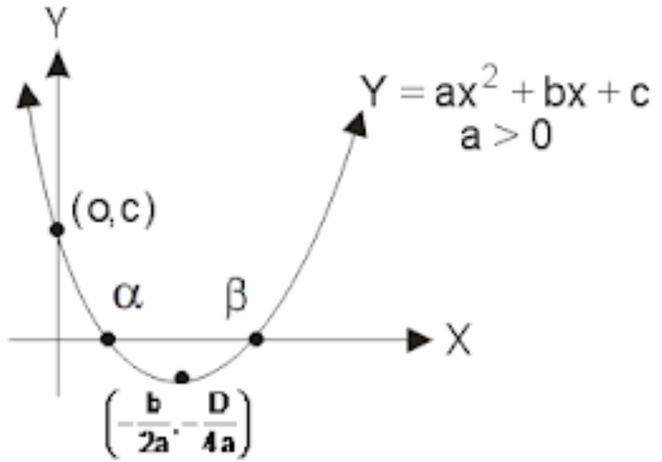


Agenda

1. Inverse Kinematics Introduction
2. How many solutions?
3. **Optimization**
4. Python Code Examples

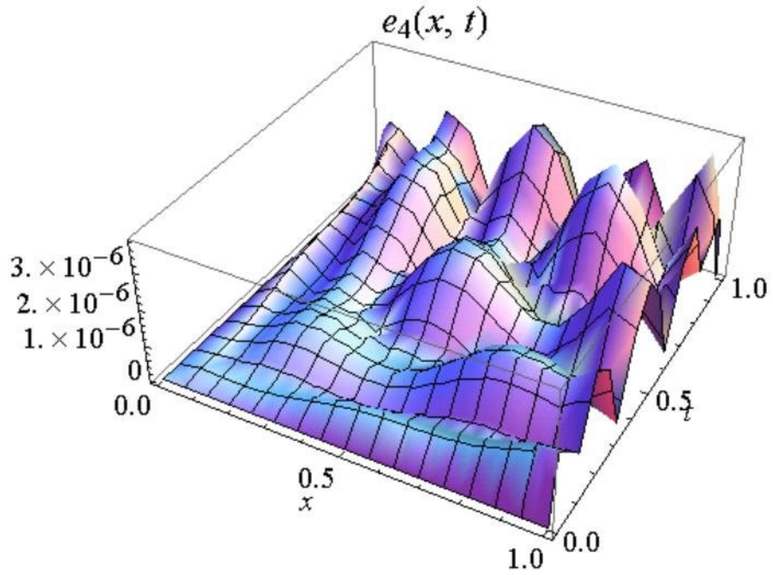
Optimization

- Let's review
  - Find the minimum value of a function
  - Easy for quadratics!
    - E.g. $y = ax^2 + bx + c$



Optimization

- Often, there is no closed form solution of the equation, so we use an “unconstrained” optimization algorithm



Yadollah Ordokhani

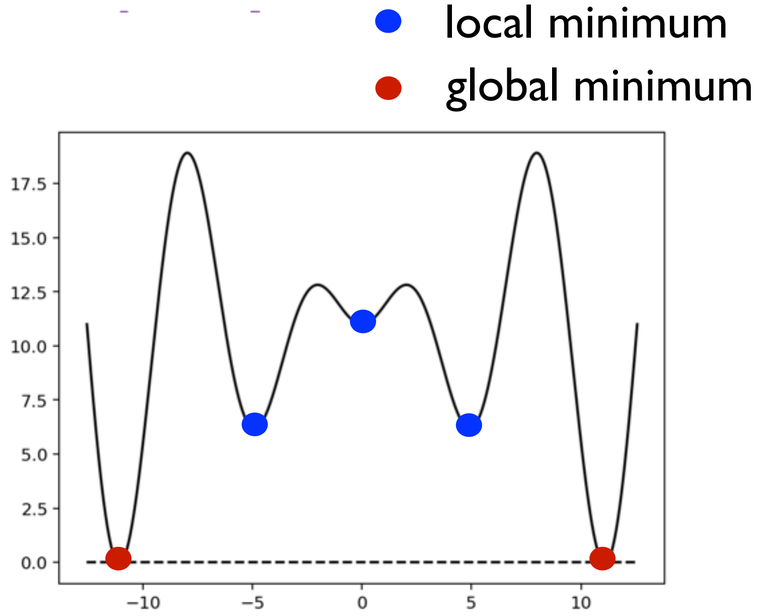
Optimization

- Often, there is no closed form solution of the equation, so we use an “unconstrained” optimization algorithm
- In this lecture, we use SciPy


```
scipy.optimize.minimize(fun, x0, args=(), method=None,
```

Non Linear Optimization

- Only local minimum found
- Results depend on local guess
- Constraints can be added, but make problem harder



Agenda

1. Inverse Kinematics Introduction
2. How many solutions?
3. Optimization
4. **Python Code Examples**

```python
def forward_kinematics_leg(theta):
    theta_1 = theta[0]
    theta_2 = theta[1]
    x_fk = (l3*math.cos(theta_2) + l2)*math.cos(theta_1) - l3 * math.sin(theta_1)*math.sin(theta_2)+l1
    y_fk = (l3*math.cos(theta_2) + l2)*math.sin(theta_1) + l3 * math.cos(theta_1)*math.sin(theta_2)

    return np.array([x_fk, y_fk])
```

FK Solver

```python
def forward_kinematics_leg(theta):
    theta_1 = theta[0]
    theta_2 = theta[1]
    x_fk = (l3*math.cos(theta_2) + l2)*math.cos(theta_1) - l3 * math.sin(theta_1)*math.sin(theta_2)+l1
    y_fk = (l3*math.cos(theta_2) + l2)*math.sin(theta_1) + l3 * math.cos(theta_1)*math.sin(theta_2)

    return np.array([x_fk, y_fk])


def get_error_leg(theta, x_des):
    x = forward_kinematics_leg(theta)
    error = x_des - x

    return error.dot(error)
```

FK Solver

Error Def’n

```python
def forward_kinematics_leg(theta):
    theta_1 = theta[0]
    theta_2 = theta[1]
    x_fk = (l3*math.cos(theta_2) + l2)*math.cos(theta_1) - l3 * math.sin(theta_1)*math.sin(theta_2)+l1
    y_fk = (l3*math.cos(theta_2) + l2)*math.sin(theta_1) + l3 * math.cos(theta_1)*math.sin(theta_2)

    return np.array([x_fk, y_fk])


def get_error_leg(theta, x_des):
    x = forward_kinematics_leg(theta)
    error = x_des - x

    return error.dot(error)


def inverse_kinematics_leg(x_des):
    x_0 = np.array([0,0])
    res = scipy.optimize.minimize(get_error_leg, x_0, args = (x_des) )

    return res.x
```

FK Solver

Error Def’n

IK Solver

```python
def forward_kinematics_leg(theta):
    theta_1 = theta[0]
    theta_2 = theta[1]
    x_fk = (l3*math.cos(theta_2) + l2)*math.cos(theta_1) - l3 * math.sin(theta_1)*math.sin(theta_2)+l1
    y_fk = (l3*math.cos(theta_2) + l2)*math.sin(theta_1) + l3 * math.cos(theta_1)*math.sin(theta_2)

    return np.array([x_fk, y_fk])


def get_error_leg(theta, x_des):
    x = forward_kinematics_leg(theta)
    error = x_des - x

    return error.dot(error)


def inverse_kinematics_leg(x_des):
    x_0 = np.array([0,0])
    res = scipy.optimize.minimize(get_error_leg, x_0, args = (x_des) )

    return res.x


############ MAIN ################
x_des = np.array([l1+0.2, 0.2])
theta = inverse_kinematics_leg(x_des)
print("IK solution for leg angles: ", theta, 'with error: ', get_error_leg(theta, x_des))
plot_leg(theta, x_des)
```

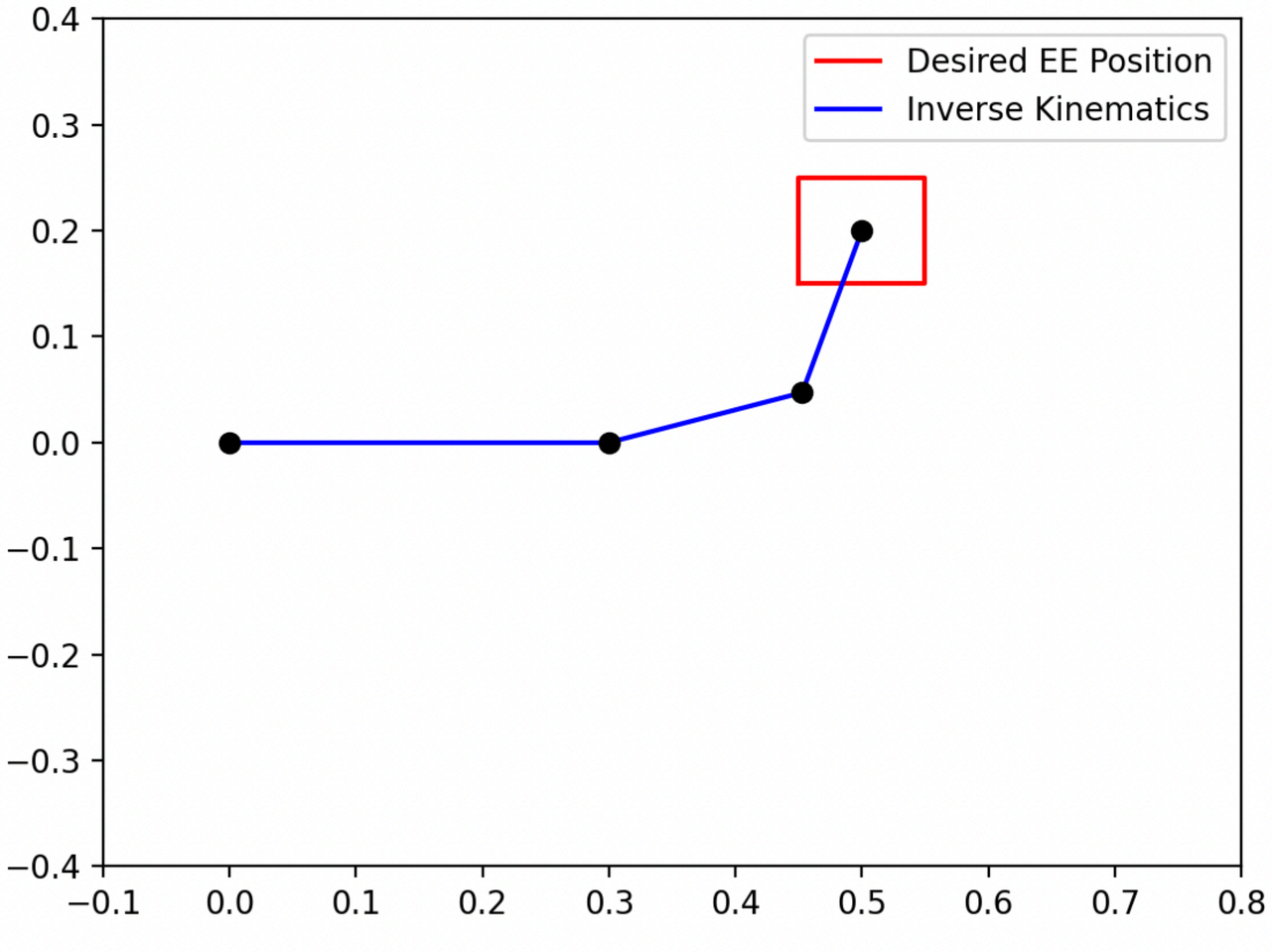
FK Solver

Error Def’n

IK Solver

ROB-UY 2004

Robotic Manipulation & Locomotion



Examples

1. One solution
   - $[x_{des}, y_{des}] = [l_1 + l_2 + l_3, 0.0]$
   - $[x_{des}, y_{des}] = [0.0, -l_1 - l_2 - l_3]$
2. Two solutions
   - $[x_{des}, y_{des}] = [l_1 + l_2, l_3]$
3. No solution
   - $[x_{des}, y_{des}] = [0.0, 2l_1 + l_2 + l_3]$

Inverse Kinematics of Kuka Robot

Inverse Kinematics of Kuka Robot

```python
## first we need to write a function that implements the cost function we want to minimize, i.e. ||p(theta) - p_desired||^2
def get_error(theta, desired_position):
    """
    theta is the current joint configuration
    desired_position is the desired end-effector position (a 3D vector)
    """
    # we compute the end-effector position from the current guess theta
    current_guess_position = forward_kinematics_KUKA(theta)[0:3,3]

    # we compute and return error (as Euclidian distance)
    error = current_guess_position - desired_position
    return error.dot(error)

# our initial guess
x0 = np.zeros(7)

# the desired position
des_pos = np.array([1.5, 0.5, 0.5])

# we minimize
res = scipy.optimize.minimize(get_error, x0, args = (des_pos))

# the result is
print(res.x)
```