

Lecture 13: inverse kinematics II

Prof. Katie Driggs-Campbell

March 24, 2026

Modern Robotics Ch 6

Administrivia

- Exam 2 is next week!
 - Topics will be on forward, inverse, and velocity kinematics

Who is Joseph Raphson?

- Joseph Raphson (1648 – 1715) was an English mathematician known best for the **Newton–Raphson method**
- Raphson's most notable work is *Analysis Aequationum Universalis* (1690), which approximates the roots of an equation
- Isaac Newton had developed a very similar formula in his Method of Fluxions, written in 1671, but published in 1736
- Raphson's method is simpler than Newton's, so Raphson's version is generally used in textbooks today

I Joseph Raphson of London Gent.

do grant and agree to and with the President, Council, and Fellows of the Royal Society of London for improving Natural knowledge, That so long as I shall continue a Fellow of the said Society, I will pay to the Treasurer of the said Society, for the time being, or to his Deputy, the sum of Fifty two shillings *per annum*; by four equal Quarterly payments, at the four usual days of payment, that is to say, the Feast of the Nativity of our Lord; the Feast of the Annuntiation of the Blessed Virgin Mary; the Feast of St. John Baptist; and the Feast of St. Michael the Archangel; the first payment to be made upon the *twenty fifth of December next being y^e feast of y^e nativity of our Lord* next ensuing the Date of these Presents; and I will pay in proportion, *viz.* One shilling per week, for any lesser time, after any the said days of payment, that I shall continue Fellow of the said Society. For the true payment whereof I bind my Self and my Heirs in the penal sum of twenty pounds. In witness whereof I have hereunto set my Hand and Seal this *fourth* day of *December* One thousand six hundred *eighty nine*

Joseph Raphson

*Sealed and Delivered
in the Presence of*

Edm. Hall
Hen. Hunt



Inverse Kinematics

- **Forward Kinematics:** computes the end-effector position from joint angles:

$$\begin{array}{ccc} \text{joint space} & \longrightarrow & SE(3) \\ \Theta & \longmapsto & T(\Theta) \end{array}$$

- **Inverse Kinematics:** computes the possible joint angles from the pose of the end-effector

$$\begin{array}{ccc} SE(3) & \longrightarrow & \text{joint space} \\ X & \longmapsto & \Theta \end{array}$$

Numerical Inverse Kinematics

- Given $T(\theta)$, find solutions θ that satisfy $T(\theta) = X$

$$f(\theta) = x$$

$$f(\theta) - x = 0$$

$$g(\theta) = 0$$

$$g(\theta)$$

- When the analytic solution is hard or impossible to come by, we **numerically** solve $T(\theta) - X = 0$

goal: find roots of g

Newton-Raphson Methods (1)

$$g(\theta) := f(\theta) - x_d$$

find θ_d s.t. $g(\theta_d) = 0$

to solve $g(\theta)=0$, g is differentiable

① start w/ initial guess θ^0 for θ_d

② write Taylor expansion of $g(\theta)$ at θ^0
up to first order

$$g(\theta) \approx g(\theta^0) + \underbrace{\frac{\partial g}{\partial \theta}(\theta^0)}_{\frac{\partial g}{\partial \theta} \Big|_{\theta^0}} (\theta - \theta^0) + \text{h.o.t.} = 0$$

Newton-Raphson Method (1.5)

$$\textcircled{3} \quad g(\theta^0) + \frac{\partial g}{\partial \theta} \bigg|_{\theta^0} (\theta - \theta^0) = 0$$

$$\theta^1 = \theta^0 - \left(\frac{\partial g}{\partial \theta} \bigg|_{\theta^0} \right)^{-1} g(\theta^0)$$

④ iterate w/ new estimate of θ :

$$\theta^{k+1} = \theta^k - \underbrace{\left(\frac{\partial g}{\partial \theta} \bigg|_{\theta^k} \right)^{-1}}_{\text{Hessian inverse}} g(\theta^k)$$

Newton-Raphson Method (2) $\theta^{k+1} = \theta^k - \left(\frac{\partial g}{\partial \theta} \Big|_{\theta^k} \right)^{-1} g(\theta^k)$

$$\frac{\partial g}{\partial \theta} \Big|_{\theta} = \begin{bmatrix} \frac{\partial g_1}{\partial \theta_1} \Big|_{\theta} & \dots & \frac{\partial g_1}{\partial \theta_n} \Big|_{\theta} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial \theta_1} \Big|_{\theta} & \dots & \frac{\partial g_n}{\partial \theta_n} \Big|_{\theta} \end{bmatrix} \in \mathbb{R}^{n \times n}$$
$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

we execute this iteration until some stopping condition is met:

$$\frac{|g(\theta^k) - g(\theta^{k+1})|}{|g(\theta^k)|} \leq \epsilon$$

Numerical IK

Given x_d : end-effector pose

$f(\theta)$: FK map

θ^0 : initial guess

$$x_d - f(\theta) = 0$$

find θ_d

$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

$$x_d = f(\theta) = \underbrace{f(\theta^0)}_{\text{arrow}} + \underbrace{\frac{df}{d\theta}}_{J(\theta^0)} \underbrace{(\theta - \theta^0)}_{\Delta\theta} \neq \text{not}$$

$$x_d - f(\theta^0) = J(\theta^0) \underline{\Delta\theta}$$

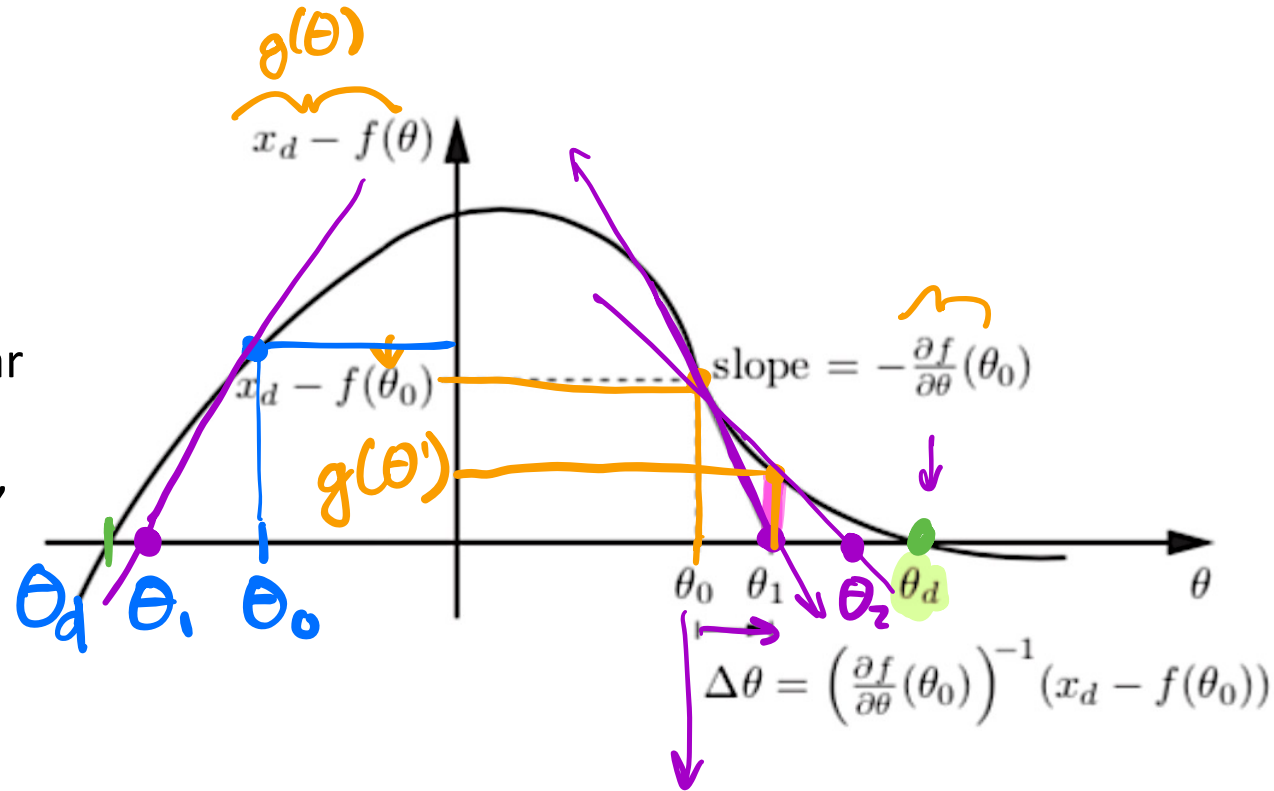
if J is invertible: $\Delta\theta = J^{-1}(\theta^0)(x_d - f(\theta^0))$

$$\rightarrow \theta' = \theta^0 + \Delta\theta$$

then reset initial guess, $\theta^0 \rightarrow \theta'$ & keep iterating
 $\{\theta^0, \theta^1, \theta^2, \dots, \theta^m\}$ converge to θ_d

Numerical Inverse Kinematics

- If the forward kinematics is linear in θ (i.e., h.o.t. are zero), then θ^1 exactly satisfies $x_d = f(\theta^1)$
- If the forward kinematics is not linear in θ (as is usually the case), then θ^1 should be closer to the root than θ^0 , but will require more iterations



$$\underline{J}^T J = I$$

Recall: Non-invertible Jacobians and the Pseudo-Inverse

- $J(\theta^0)$ may be singular ($\det(J(\theta^0)) = 0$) or may not be square
 - Non-invertible! Instead use Pseudo-Inverse!
- The Moore-Penrose pseudo-inverse for $J \in \mathbb{R}^{m \times n}$ is denoted $\underline{J}^\dagger \in \mathbb{R}^{n \times m}$
- Consider equation: $Jy = z$. We want to find the solution: $y^* = J^\dagger z$, such that:
 - One solution if $n = m$ and J is full rank
 - $y^* = J^{-1}z$
 - Many solutions if $n > m$
 - y^* exactly satisfies $Jy^* = z$, and gives the minimal norm solution ($\|y^*\| \leq \|y\|$)
 - No solutions if $n < m$ and z is not in the span of J
 - If no solution, y^* minimizes the two-norm of the error: $\|Jy^* - z\| \leq \|J\tilde{y} - z\|$ for all $\tilde{y}$

$$J^\dagger J = I$$

The Pseudo Inverse

- When J is full column rank ($n < m$, tall):

$$J^+ = (J^T J)^{-1} J^T \rightarrow \text{left inverse: } J^+ J = I$$

- When J is full row rank ($n > m$, wide):

$$J^+ = J^T (J J^T)^{-1} \rightarrow \text{right inverse: } J J^+ = I$$

- When $n = m$ and full rank:

$$J^+ = J^{-1}$$

$$\Delta \theta = J^+(\theta^0) (x_d - f(\theta^0))$$

Using the Newton-Raphson Method for IK

$$\begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$$

- The Newton-Raphson algorithm needs to be modified given that $X \in SE(3)$, which is not a general matrix in $\mathbb{R}^{4 \times 4}$ or a coordinate vector

Using the Newton-Raphson Method for IK

- The Newton-Raphson algorithm needs to be modified given that $X \in SE(3)$, which is not a general matrix in $\mathbb{R}^{4 \times 4}$ or a coordinate vector
- The error vector $e = x_d - f(\theta^i)$, represents the update needed to go from the current guess to the desired end-effector configuration (after being multiplied by the inverse Jacobian)
- Said otherwise, following the direction e for one second, starting from $f(\theta^i)$, should send us (approximately) to x_d

Using the Newton-Raphson Method for IK

- The Newton-Raphson algorithm needs to be modified given that $X \in SE(3)$, which is not a general matrix in $\mathbb{R}^{4 \times 4}$ or a coordinate vector
- The error vector $e = x_d - f(\theta^i)$, represents the update needed to go from the current guess to the desired end-effector configuration (after being multiplied by the inverse Jacobian)
- Said otherwise, following the direction e for one second, starting from $f(\theta^i)$, should send us (approximately) to x_d
- In our case, we are given $X \in SE(3)$, and instead of computing $X - T(\theta^i)$, we should compute the **twist** $\mathcal{V}_b$ which, if followed for one second, sends us from $T(\theta^i)$ to X

Numerical Inverse Kinematics with a Twist

- We want to find body twist $\mathcal{V}_b$ to move from $T_{sb}(\theta^i)$ to desired $T_{sd} = X$
current 'pose' guess *desired pose*
- The twist that sends us from $T_{sb}(\theta^i)$ to X satisfies:

$$T_{sd} := X = T_{sb} \underbrace{\ell^{[\mathcal{V}_b]}}_{T_{bd}}$$
$$\ell^{[\mathcal{V}_b]} = T_{sb}^{-1}(\theta^i) T_{sd}$$

recall matrix log:

$$[\mathcal{V}_b] = \log(T_{sb}^{-1}(\theta^i) T_{sd})$$

Numerical Inverse Kinematics: Algorithm

0. Given $X = T_{sd}$ and the forward kinematics map $T_{sb}(\theta)$

Given tolerances ϵ_ω and ϵ_v

1. Initialize θ^0 and set $i = 0$

2. While $\|\omega_b\| > \epsilon_\omega$ or $\|\omega_v\| > \epsilon_v$

1. Set $[\mathcal{V}_b] = \log T_{sb}^{-1}(\theta^i) T_{sd}$

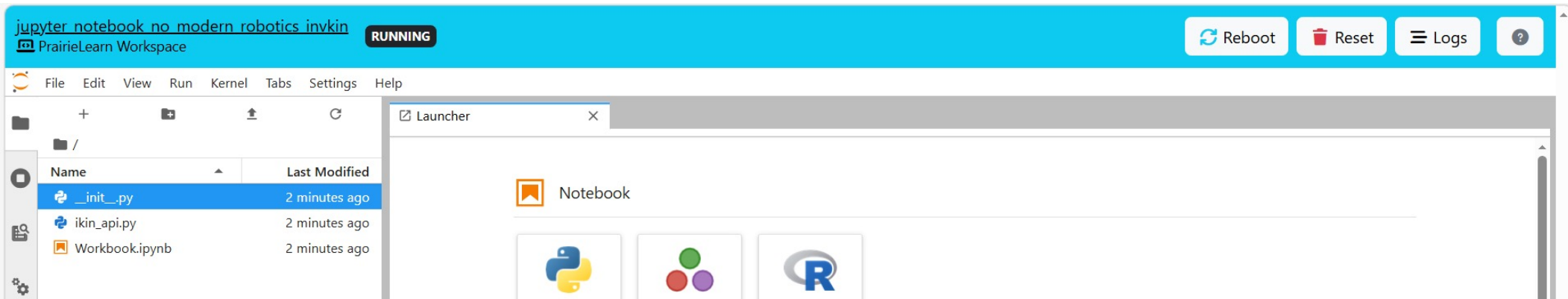
2. Set $\theta^{i+1} = \theta^i + J_b^\dagger(\theta^i) \mathcal{V}_b$

3. Increment i

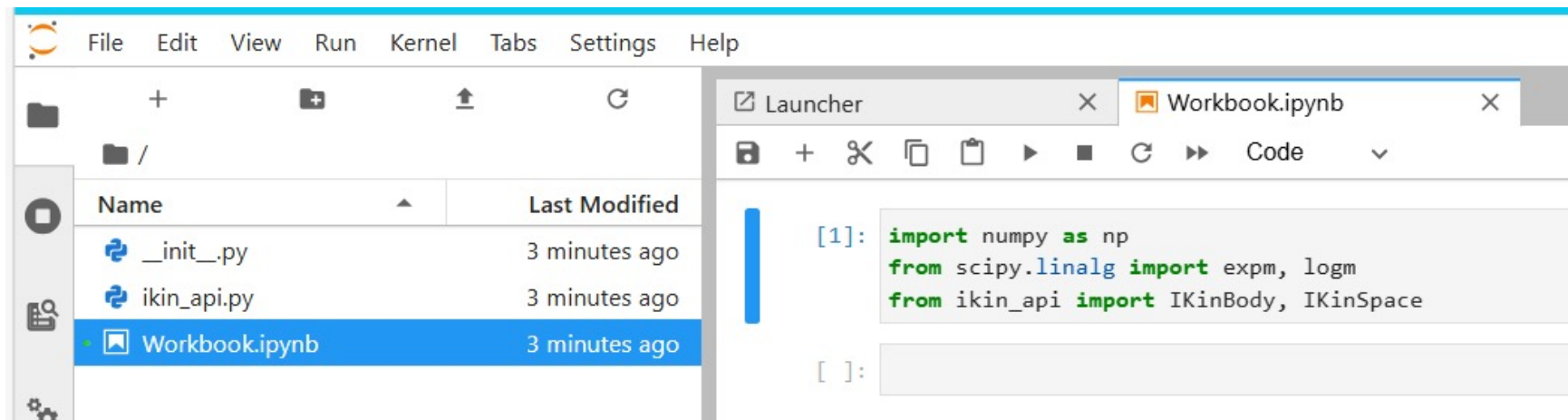
A few comments

- Re: algorithm on last slide:
 - An equivalent form can be derived in the space frame, using the spatial Jacobian and the spatial twist
- You can also extend this to find sequence of joint angles that will allow the end-effector to follow a trajectory $T_{sb}(t)$
 - Discretize the trajectory $T_{sb}(\theta_k)$ and compute θ_k such that $T_{sb}(\theta_k) = T_{sb}(t_k)$
 - Initialization is *really* important here!
- For your homework, you will not have to implement Newton-Raphson
 - Must have an understanding of the theory for conceptual questions on the upcoming exam

The Jupyter workspace now provides access to functions for inverse kinematics!
This is what you'll see when you startup the server:



In addition to numpy, expm, and logm, we now import IKinBody and IKinSpace by default:



File Edit View Run Kernel Tabs Settings Help

Launcher Workbook.ipynb ikin_api.py

```
1 from _internal import IKinBody, IKinSpace
2
3 """
4 def IKinBody(Blist, M, T, thetalist0, eomg, ev):
5     Computes inverse kinematics in the body frame for an open chain robot
6     :param Blist: The joint screw axes in the end-effector frame when the
7                   manipulator is at the home position, in the format of a
8                   matrix with axes as the columns
9     :param M: The home configuration of the end-effector
10    :param T: The desired end-effector configuration Tsd
11    :param thetalist0: An initial guess of joint angles that are close to
12                      satisfying Tsd
13    :param eomg: A small positive tolerance on the end-effector orientation
14                 error. The returned joint angles must give an end-effector
15                 orientation error less than eomg
16    :param ev: A small positive tolerance on the end-effector linear position
17              error. The returned joint angles must give an end-effector
18              position error less than ev
19    :return thetalist: Joint angles that achieve T within the specified
20                      tolerances,
21    :return success: A logical value where TRUE means that the function found
22                    a solution and FALSE means that it ran through the set
23                    number of maximum iterations without finding a solution
24                    within the tolerances eomg and ev.
25    Uses an iterative Newton-Raphson root-finding method.
26    The maximum number of iterations before the algorithm is terminated has
27    been hardcoded in as a variable called maxiterations. It is set to 20 at
28    the start of the function, but can be changed if needed.
29    Example Input:
30        Blist = np.array([[0, 0, -1, 2, 0, 0],
31                          [0, 0, 0, 0, 1, 0],
32                          [0, 0, 1, 0, 0, 0.1]]).T
33        M = np.array([[-1, 0, 0, 0],
34                      [0, 1, 0, 6],
```

ikin_api.py explains how to call
both functions and has
examples

You can ignore __init__.py

```
[1]: import numpy as np
from scipy.linalg import expm, logm
from ikin_api import IKinBody, IKinSpace

[2]: # Desired end effector pose - 4x4 homogeneous transformation matrix
T_lin0 = np.array([[0.64469519, 0.00045421, -0.76443961, -1.72006131], [-0.68757886, 0.43735862, -0.57961430, -3.20838971], [0.33407099, 0.00000000, 0.00000000, 0.00000000], [0.00000000, 0.00000000, 0.00000000, 0.00000000]])

# Initial pose of end effector - 4x4 homogeneous transformation matrix
M = np.array([[1.00000000, 0.00000000, 0.00000000, 2.00000000], [0.00000000, 1.00000000, 0.00000000, -4.00000000], [0.00000000, 0.00000000, 1.00000000, 0.00000000], [0.00000000, 0.00000000, 0.00000000, 1.00000000]])

# Screw axes - 6xNUM_JOINTS where each column is the screw of each joint (in this example, they are in the fixed frame)
S = np.array([[0.00000000, 0.00000000, 1.00000000, 0.00000000, 1.00000000, 0.00000000], [1.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000], [0.00000000, 1.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000], [0.00000000, 0.00000000, 0.00000000, 1.00000000, 0.00000000, 0.00000000], [0.00000000, 0.00000000, 0.00000000, 0.00000000, 1.00000000, 0.00000000], [0.00000000, 0.00000000, 0.00000000, 0.00000000, 0.00000000, 1.00000000]])

# Initial guess for what each theta should be - list of length NUM_JOINTS
thetalist0 = np.array([0., 0., 0., 0., 0., 0.])

# Tolerance on error of end effector orientation
eomg = 0.01

# Tolerance on error of end effector position
ev = 0.01

[5]: thetalist, success = IKinSpace(S, M, T_lin0, thetalist0, eomg, ev)

# success is True if the error of the solution is less than eomg and ev
# thetalist is of shape (NUM_JOINTS,)
# PrairieLearn expects thetalist is a column vector
thetalist = thetalist.reshape((1,-1)).T
print(thetalist.tolist())
print(thetalist.shape)

[[-0.35111565922256915], [-0.6162067021942831], [0.34969128094474744], [0.9405509369400207], [0.6555028624350827], [-0.17241756757718962]]
(6, 1)
```

Summary

- **Inverse Kinematics** gives us a method for finding the joint angles given an end-effector configuration
- This can sometimes be done analytically (geometrically), but this is often difficult so **numerical techniques** are more common in practice
- Introduced the **Newton-Raphson method**, which can be modified to solve inverse kinematics numerically