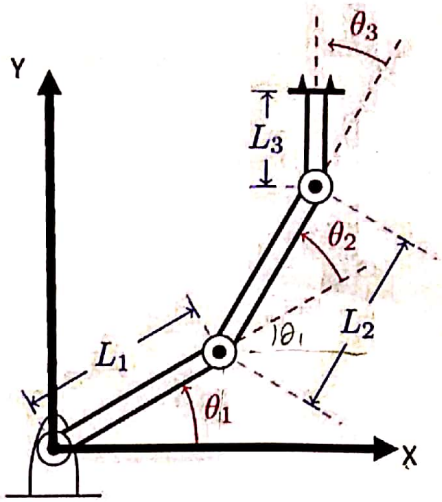
ICP 8: Robot Trajectory Generation

1. The robot:
   a. The forward kinematics problem
   (suppose the robot has N joints)

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} L_1 \cos\theta_1 \\ L_1 \sin\theta_1 \end{bmatrix} + \begin{bmatrix} L_2 \cos(\theta_1+\theta_2) \\ L_2 \sin(\theta_1+\theta_2) \end{bmatrix} + \begin{bmatrix} L_3 \cos(\theta_1+\theta_2+\theta_3) \\ L_3 \sin(\theta_1+\theta_2+\theta_3) \end{bmatrix}$$

$$= \begin{bmatrix} L_1 C_1 + L_2 C_{12} + L_3 C_{123} \\ L_1 S_1 + L_2 S_{12} + L_3 S_{123} \end{bmatrix}$$



   b. Jacobian matrix

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix}_{2\times 1} = J_{2\times N} \times \begin{bmatrix} \dot{\theta}1 \\ \vdots \\ \dot{\theta}N \end{bmatrix}_{N\times 1}$$

$$\frac{\partial X}{\partial t} = J \cdot \frac{\partial \theta}{\partial t}$$

Matrix J is of dimension (rows: 2 x columns: N)

$$J = \frac{\partial X}{\partial t}\frac{\partial t}{\partial \theta} = \frac{\partial X}{\partial \theta}$$

$$J = \begin{bmatrix} -L_1 S_1 - L_2 S_{12} - L_3 S_{123} & -L_2 S_{12} - L_3 S_{123} & -L_3 S_{123} \\ L_1 C_1 + L_2 C_{12} + L_3 C_{123} & L_2 C_{12} + L_3 C_{123} & L_3 C_{123} \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \cdots & \frac{\partial x}{\partial \theta_N} \\ \frac{\partial y}{\partial \theta_1} & \cdots & \frac{\partial y}{\partial \theta_N} \end{bmatrix}$$

$\theta_1, \theta_2, \theta_3$ should be known

   c. Inverse kinematics
   Explain the relationship between Jacobian matrices and IK:

$$\Delta X = J \times \Delta\theta$$

$$\Delta\theta = J^{-1} \Delta X$$

need to do pinv(J)
pseudo inverse

efbtfc0024z

2. Trajectory generation:

What is your plan for generating the trajectories?

Given:

(1) joint_angles: the current robot joint angles

(2) joint_goal_angles: the final robot joint angles when the goal is reached.

joint-angles = joint-angles + K_p · angle-diff) × dt (step size)

decreasing

domain · 2D

domain 3D

domain

Our plan to generate trajectoy is curve fitting a sinusoidal function

$\dot{\theta}$ → $\dot{\theta}_{max}$

$\dot{\theta} = \sin(wt)$ $\quad$ $\ddot{\theta} = w\cos(wt)$

slope = $\ddot{\theta}_{max}$ $\quad$ t_f

$\dot{\theta}_{max} = \sin\left(w\frac{t_f}{2}\right)$ $\quad$ $\ddot{\theta}_{max} = w\cos(wt)\big|_{t=0} = w$

m > n

n × n

$(A'A)^{-1}A$

In your design, is the EE speed, acceleration and the amount of joint angle changes regulated or capped?

I collaborated with Alex, in our design, we set up the maximum speed and acceleration to get a sin function to follow.

List two benefits of your design:

① I can control the $\ddot{\theta}_{max}$ which means I can control my torque. because $\tau = I\ddot{\theta}$, thus make my motor can perform in their capability.

② I can control the $\dot{\theta}_{max}$ which makes the robot be safer.

3. Python Hands-On Task:
   a. Download the code from here to your virtual machine
      i. Unzip it and put it under a directory of your choice
      ii. Go to the directory: **$ cd ~/director_of_your_choice**
   b. Make files executable

      $ chmod +x NLinkArm.py

      $ chmod +x RobotTrajGen.py
   c. Make edits to the python file: RobotTrajGen.py
      i. Find all the places with keyword **"TODO"**
      ii. Implment your own trajectory generation algorithm

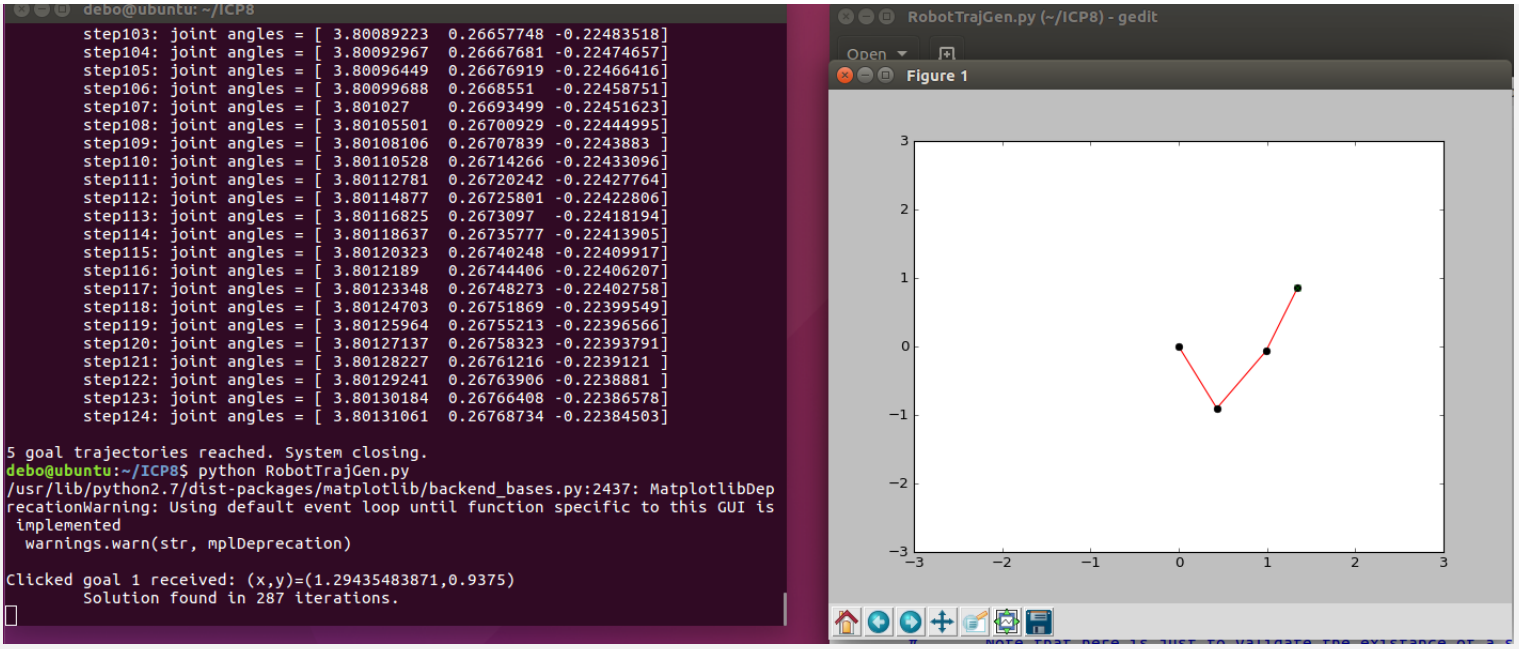
d. Run the code:
   i. Command:
      $ cd ~/director_of_your_choice
      $ python RobotTrajGen.py
   ii. show_animation flag: This is a flag on top of the "RobotTrajGen.py " file. Toggle the Boolean variable to change between:
      1. Randomly generated goals & numeric outputs on the terminal
      2. User defined goals (by right mouse click) & visualization

4. What to submit:

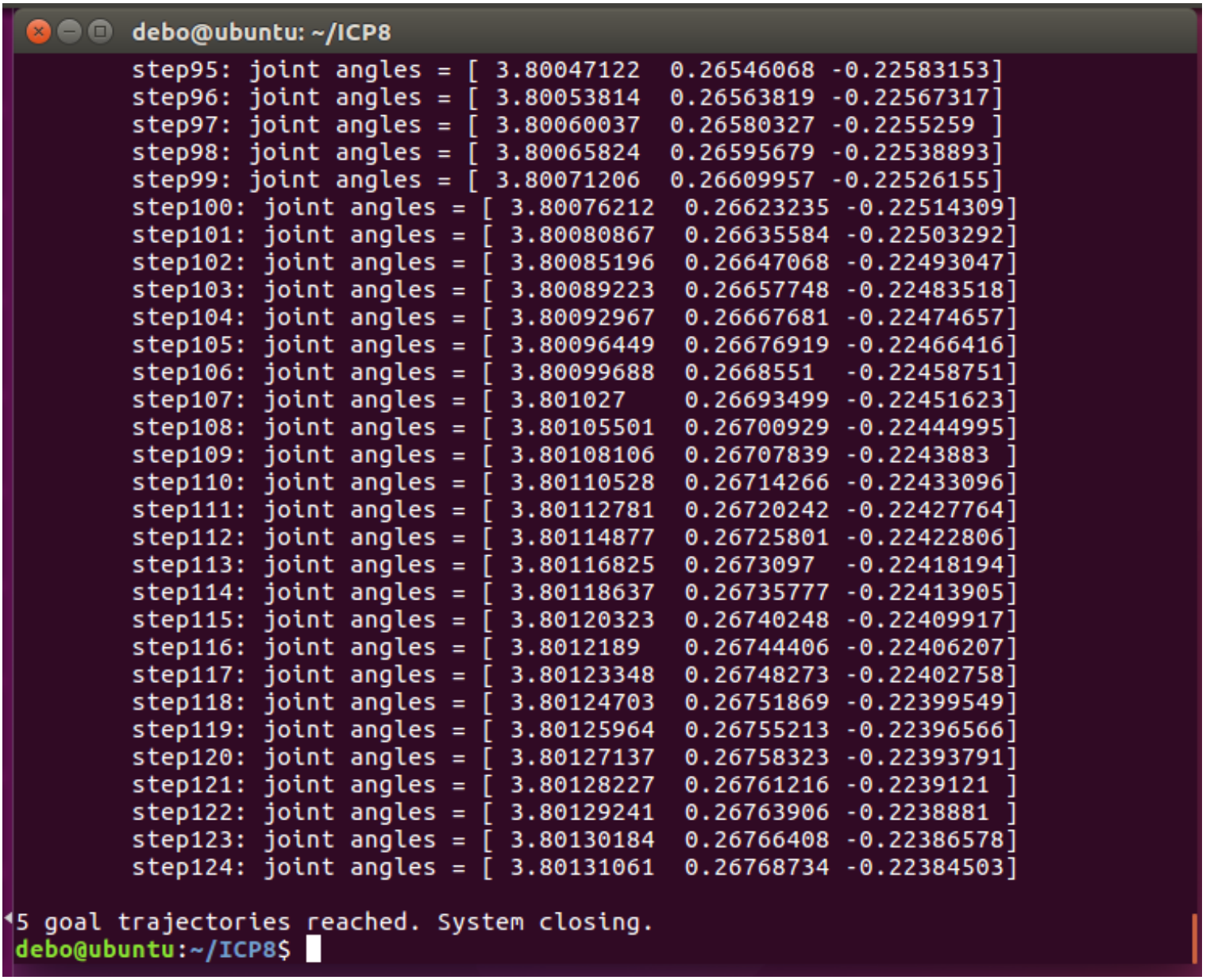
a. Questions:
   i. Answer the questions in 1. And 2.
   ii. Answer the three questions below:

When designing the trajectory, we consider the goal and current joint angles instead of the cartesian position difference. List the downsides or risks if you do it in the cartesian space.	Ans: In cartesian space, maybe some joints other than the EE will collide with obstacle when EE is safe.
In previous lectures we were using geometric methods and **algebraic methods to solve for** IK, but here using Jacobians seems like a very systematic way. Does IK using inverse jacobian always work for every robotic application? Explain your reasons.	Ans: Actually, doing inverse jacobian is a really time consuming procedure, while using geometric or algebraic method to get the closed form solution is less complicated to get the joint value. Thus in real-time manipulation, inverse jacobian cannot handle.
In our application, the trajectory is purely a smooth transition from initial to target joint angles. What if there are obstacles in the workspace? Explain an idea to perform **trajectory generation while** taking into consideration of the obstacles in the environment.	Ans: ICP8 → HW4 If there are obstacles in workspace, I'll sample some points in joint space to check the range of collision part and make my trajectory generation don't pass through the range / region, thus make the trajectory be safe. And also I will consider the distance between joints and obstacle as a cost, to make it safe. And also the speed when approaching obstacles, I'll also slow it down.

Default link # and link length:
Show_animation = True:



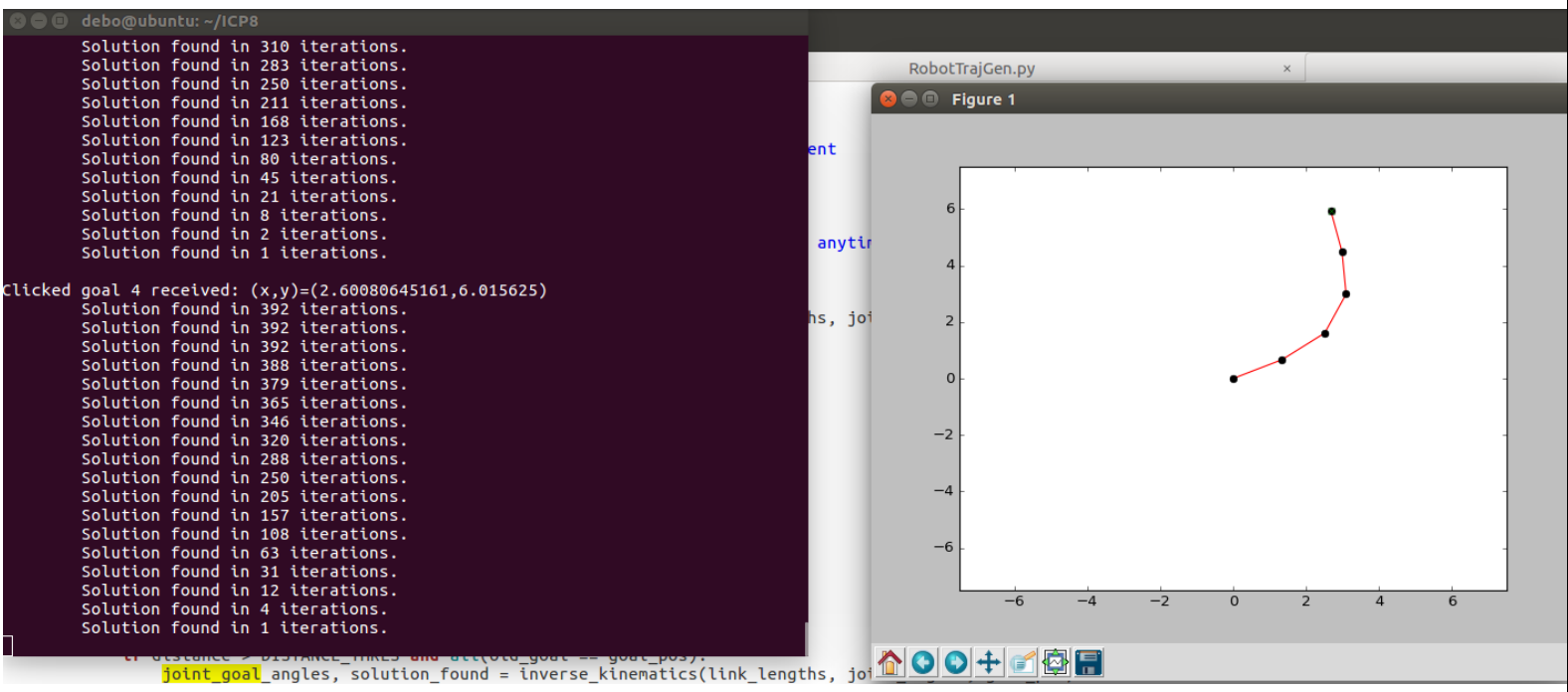
Default link # and link length:
Show_animation =False:



Modifiedlink # and link length:

Show_animation = True:

What's your choice of link # and link length? 5 links and link length 1.5



Modifiedlink # and link length:

Show_animation = False:

What's your choice of link # and link length? 5 links and link length 2.5

```
debo@ubuntu: ~/ICP8
       Solution found in 1 iterations.
       step326: joint angles = [-54.52112207  -1.6054765    2.13918889 -38.3605535   41.44651381
  10.54553217]
       Solution found in 1 iterations.
       step327: joint angles = [-54.5405866   -1.54117123   2.11569053 -38.33047219  41.44610675
  10.55729008]

Random goal 5 generated: (x,y)=(3.81820154696,3.00713901424)
       Solution found in 39 iterations.
       Solution found in 39 iterations.
       step1: joint angles = [-55.22565235  -1.69029447   2.06709019 -38.06673715  42.13158419
  11.10318794]
       Solution found in 17 iterations.
       step2: joint angles = [-55.44759872  -1.70874335   2.05628443 -37.97952196  42.30803237
  11.27036521]
       Solution found in 31 iterations.
       step3: joint angles = [-55.45786658  -1.68916279   2.15324107 -37.96217493  42.23663624
  11.25757777]
       Solution found in 29 iterations.
       step4: joint angles = [-55.41721812  -1.65587281   2.28186081 -37.95105086  42.11656782
  11.20274699]
       Solution found in 24 iterations.
       step5: joint angles = [-55.39305635  -1.62229571   2.40350725 -37.92802884  42.02468864
  11.16385882]
       Solution found in 14 iterations.
       step6: joint angles = [-55.4015844   -1.59362974   2.5066647  -37.89354109  41.97469863
  11.15424874]
       Solution found in 1 iterations.
       step7: joint angles = [-55.43320286  -1.57058876   2.5969315  -37.85504713  41.94699326
  11.16328115]

5 goal trajectories reached. System closing.
debo@ubuntu:~/ICP8$
```