CS 4/6545 Autonomous Robotics:
Differential kinematics and inverse kinematics

Read all instructions carefully.

Submission instructions: Your submission will consist of all code and images (described below) proving that you solved the assignment properly. The easiest way to submit the assignment to me is to "tar" or "zip" up your assignment:

```
IK $ cd ..
AuRo $ tar czf IK.tgz IK
```

This will yield the file `IK.tgz` in my `AuRo` directory. The `tar czf` command stores the entire `IK` directory and compresses it.

Assignment overview: In this assignment, you will practice differential kinematics and implement resolved rate motion control (RMRC) using the planar arm and—for advanced students—the industrial robot arm. The code you need to modify will be in `IKPlanar.cpp` (and, for advanced students, `IKSpatial.cpp`).

1. **Implement `FKin(.)` (the forward kinematics function)** Implementing this function mainly requires you to transfer your code from the `coord_frame_planar2.cpp` (in the *coord frames* assignment) into a new function.

2. **Implement `CalcOSDiff(.)` on the planar robot** `CalcOSDiff(.)` should take two 3D poses and return a three dimensional vector that represents the *differential* between the two, which you will recall is roughly defined as the linear and angular change needed to get from one to the other. If we call the two angles of rotation around $\hat{\boldsymbol{z}}$ θ_1 and θ_2 and the translations along $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ as x_1, x_2 and y_1, y_2, then we want:

$$\begin{bmatrix} c_{\theta_2} & -s_{\theta_2} & x_2 \\ s_{\theta_2} & c_{\theta_2} & y_2 \\ 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} c_{\theta_1} & -s_{\theta_1} & x_1 \\ s_{\theta_1} & c_{\theta_1} & y_1 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ \theta_2 - \theta_1 \end{bmatrix} = \Delta \boldsymbol{x} \tag{1}$$

   Do not forget to correct for the angular component of $\Delta \boldsymbol{x}$ so that the change in rotation is within the interval $[-\pi, \pi]$ by implementing `WrapAngle(.)`.

3.* **Implement `CalcOSDiff(.)` on the spatial robot** (*For advanced undergraduates (bonus points) and graduate students*) This means that you have to calculate the difference between two rotation matrices, as discussed in class, expressed as an angular velocity vector:

$$\mathbf{R}_{\text{des}} - \mathbf{R}(\boldsymbol{q}) \rightarrow \Delta \boldsymbol{x}_{\text{rot}} \tag{2}$$

   After you compute the differences in translation ($\Delta \boldsymbol{x}_{\text{trans}}$):

$$\Delta \boldsymbol{x} \equiv \begin{bmatrix} \Delta \boldsymbol{x}_{\text{trans}} \\ \Delta \boldsymbol{x}_{\text{rot}} \end{bmatrix} \tag{3}$$

4. **Copy CalcJacobian(.) for the planar arm from the *Jacobians* assignment** 'Nuff said.

5.* **Implement `CalcJacobianNumerical(.)` on the spatial robot** (*For advanced undergraduates (bonus points) and graduate students*)

   This task requires you to use the following algorithm (introduced in class):

function `CalcJacobianNumerical`($\boldsymbol{q}$) computes the Jacobian for a robot *around configuration* $\boldsymbol{q}$
$\boldsymbol{x} \leftarrow$`Fkin`($\boldsymbol{q}$)
for $i = 1, \ldots, n$ **do**
  $q_i \leftarrow q_i + \Delta$
  $\boldsymbol{j}_i \leftarrow$ `CalcOSDiff`($\boldsymbol{x}$, `FKin`($\boldsymbol{q}^*$))
  $q_i \leftarrow q_i - \Delta$
end for
$\mathbf{J} \leftarrow \mathbf{J}/\Delta$
return $\boldsymbol{q}$

6. **Implement `DoIK(.)` in `IKPlanar.cpp`** This is the Resolved Motion Rate Control Algorithm. I've done the heavy lifting for you (the backtracking ray search, for example). Implementing the pieces that are left should give you insight into the workings of the algorithm. Note that you need to be able to be prepared to use either the Jacobian Transpose (`TransposeJ(.)`) or the Jacobian Pseudo-Inverse (`SolveJ(.)`) approaches.

7. **Test inverse kinematics** The plugin takes the following steps:

   (a) Selects a random configuration vector $\boldsymbol{q}_{\text{rand}}$ vector

   (b) Computes the desired pose using `FKin(`$\boldsymbol{q}_{\text{rand}}$`)`

   (c) Solves the inverse kinematics problem from $\boldsymbol{q}_{\text{init}} \equiv \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$

   (d) Uses PD control to move the arm to the IK solution (should be $\boldsymbol{q}_{\text{rand}}$)

   Test both IK approaches (Jacobian Transpose and Jacobian Pseudo-Inverse), and count how many iterations are necessary for convergence. *Do not count consider iterations when the algorithm must restart.*