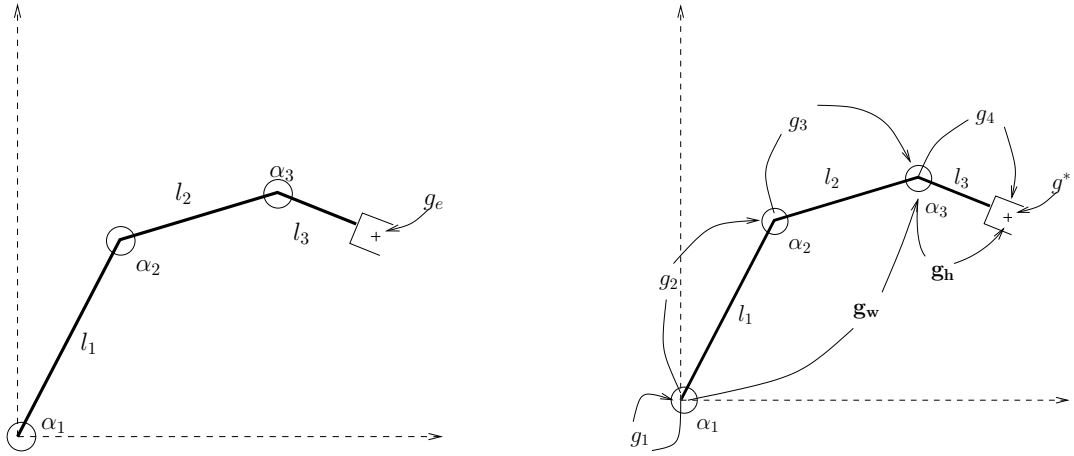
ECE4560 - Solution #9

Problem 1. [30 pts] Work out the inverse kinematics for the three-link rotational planar manipulator depicted in Figure 3. Since it is fully controllable, we can use a version of Pieper's solution for the plane, but use the geometric method (from the homework) for the position part.

The last joint/link combination could be considered the hand, with the first two joint/link combinations being the base to wrist transformation. If the forward kinematics are given by $g_e(\alpha) = g_1(\alpha_1)g_2(\alpha_2)g_3(\alpha_3)g_4$, then use the first two joints to solve for the position of the wrist, given by the position of the following wrist configuration $g_w = g_e^* \, g_4^{-1}$. Once this position has been solved, you can solve for α_3 in order to get the orientation to work out properly for the end-effector (don't forget that $g_3(\alpha_3)$ gets broken up into $\bar{g}_3\tilde{g}_3(\alpha_3)$).

Hopefully the drawing in Figure 1(b) helps a little bit, since it contains additional details to help you visualize what is going on. The whole point of Pieper's idea was to use the orientation control of the wrist-hand connection to define the hand joints using the desired orientation and for the base to wrist portion of the kinematic chain to be used to find appropriate position for the wrist so that the hand could actually achieve the orientation task.



(a) Manipulator

(b) Pieper's solution visualization.

Figure 1: Planar 3R Manipulator.

So, in short

(a) For link lengths $l_1 > l_2 > l_3$, what are the inverse kinematics for the manipulator depicted in Figure 3.

(b) Supposing that $l_1 = 1$, $l_2 = \frac{1}{2}$, and $l_3 = \frac{1}{4}$, what joint angles would be used to get to the configuration $g_e^* = (1.5560, 0.7288, 0.7854)$? (the angle is in radians, which would be 45.00 degrees).

Because of the way that angles work, I could also add or subtract 2π to the desired orientation and get a valid solution. You may see such behavior pop up as you work out the inverse kinematics, so don't be confused about that. This goes along with my mentioning that sometimes 2π should be added/subtracted to the angle as part of the inverse kinematics solution in order for things to work out. If you'd like to visualize and verify things, then there is a function called `planarR3_display` on the class wiki page (`pvela.gatech.edu/classes`). Follow the *Display/Plotting* link.

Solution 1. The solution to this requires first, for us to determine the forward kinematics, then break it down to get the base frame to wrist frame transformation. The forward kinematics are:

$$g_e(\alpha) = \left\{ \begin{array}{c} l_1\cos(\alpha_1) + l_2\cos(\alpha_1+\alpha_2) + l_3\cos(\alpha_1+\alpha_2+\alpha_3) \\ l_1\sin(\alpha_1) + l_2\sin(\alpha_1+\alpha_2) + l_3\sin(\alpha_1+\alpha_2+\alpha_3) \\ \alpha_1+\alpha_2+\alpha_3 \end{array} \right\}.$$

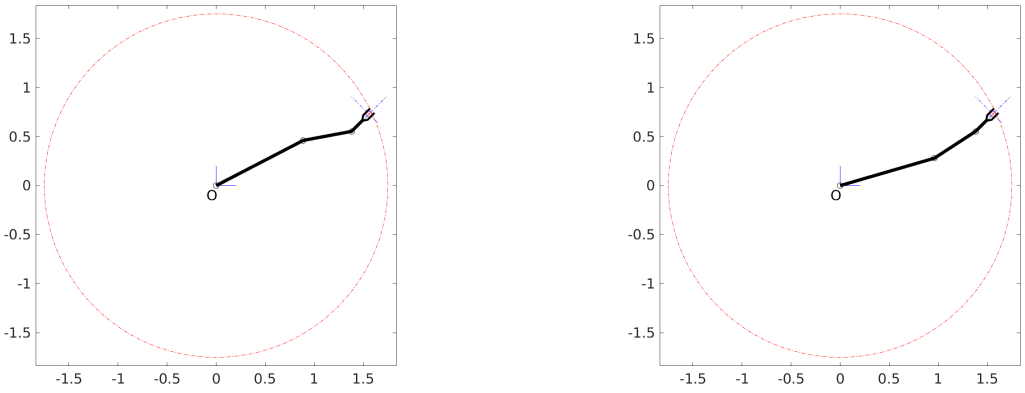


Figure 2: Visualization of manipulator for the two joint configuration solutions of Problem 1.

If we just want to make it to the wrist, then we transform up to, but not including, the third joint angle. This is

$$g_w(\alpha) = \begin{Bmatrix} l_1\cos(\alpha_1) + l_2\cos(\alpha_1+\alpha_2) \\ l_1\sin(\alpha_1) + l_2\sin(\alpha_1+\alpha_2) \\ \alpha_1+\alpha_2 \end{Bmatrix}.$$

Note that, for both cases, the end-effector configuration is given in vector form. The inverse kinematics for the wrist portion will utilize the *triangles method* described in class.

(a) The overall procedure is as follows:

1. Compute the desired wrist configuration, $g_w^* = g_e^* g_4^{-1}$.
2. The triangles method uses the following angles to generate the solution,

$$\begin{aligned} \gamma &= \arctan 2(y_w^*, x_w^*), \\ \delta &= \arccos\left(\frac{l_1^2 + l_2^2 - r^2}{2l_1l_2}\right) - \pi \quad \text{or} \quad \pi - \arccos\left(\frac{l_1^2 + l_2^2 - r^2}{2l_1l_2}\right), \\ \beta &= \arccos\left(\frac{l_1^2 + r^2 - l_2^2}{2l_1r}\right), \text{and} \end{aligned}$$

   where $r = \sqrt{(x_w^*)^2 + (y_w^*)^2}$.
3. Choose either the pairing $\alpha_1 = \gamma + \beta$ and $\alpha_2 = \alpha - \pi$, or the pairing $\alpha_1 = \gamma - \beta$ and $\alpha_2 = \pi - \alpha$, to get the solution to the first two joints.
4. The last joint is determined by setting $\alpha_3 = \theta_e^* - \alpha_1 - \alpha_2$. If α_3 is bigger than π then subtract 2π. If α_3 is less than $-\pi$, then add 2π. It seems more natural to have the last joint angle lie inside of the range $[-\pi, \pi]$.

The solutions may vary somewhat because I could always add or subtract 2π to the solution. The solutions above should result in angles that lie within the range $[-\pi, \pi]$ which seems more natural. Note that the solution to $\arccos$ is defined to lie within the range $[0, \pi]$, hence δ should be in the desired range.

(b) Applying the above algorithm, I get the joint configuration $\alpha = (0.4785, -0.2944, 0.6012)^T$ radians, which is equivalent to $\alpha = (27.4171, -16.8659, 34.4487)^T$ degrees. For the second possible solution my results give $\alpha = (0.2829, 0.2944, 0.2081)^T$ radians, or equivalently $\alpha = (16.2096, 16.8659, 11.9246)^T$ degrees. The two solutions are depicted in Figure 2.

Of course, if your joint angles differ by 2π radians (or 360 degrees) in some of the cases, that's OK, so long as you have verified that the final end-effector configuration under those joint angles is indeed correct. In practice this would not be feasible, and angles respecting given limits would be necessary.

Problem 2. [10 pts] Complete the adjoint code for $SE(3)$ for the case of computing the adjoint of a Lie group element g and a twist ξ (e.g., Lie algebra element). Verify against the following Matlab run:

```
>> g1 = SE3([-5;6;-1] , [1 0 0;0 0 -1;0 1 0]);
>> g2 = SE3([ 0;0; 0] , [ 1/sqrt(2) 0 1/sqrt(2);0 1 0;-1/sqrt(2) 0 1/sqrt(2)]);
>> adjoint(g1, g2)

ans =

    0.7071   -0.7071         0    2.7782
    0.7071    0.7071         0    5.2929
         0         0    1.0000         0
         0         0         0    1.0000


>>g = SE3([1;0;-1],[sqrt(2)/2 0 -sqrt(2)/2 ; 0 1 0; sqrt(2)/2 0 sqrt(2)/2]);
>>xi = [0; 1; 0; pi/10; 0; pi/12];

>>adjoint(g, xi)

ans =

         0
    0.5557
         0
    0.0370
         0
    0.4073
```

If $\xi = (1, 0, 1, 0, \pi/6, -\pi/12)^T$, then what results from the adjoint operation?

Solution 2. My code for the adjoint looks like:

```
function z = adjoint(g, x)

if (isa(x,'SE3'))                                % SE(3)/SE(3) adjoint.
  z = g*x*inv(g);
elseif ( (size(x,1) == 6) && (size(x,2) == 1) ) % SE(3)/se(3) adjoint.
  R = g.M(1:3,1:3);                              %  vector form.
  d = g.M(1:3,4);
  z = [R , hat(d)*R ; zeros(3) , R]*x;
elseif ( (size(x,1) == 4) && (size(x,2) == 4) ) % SE(3)/se(3) adjoint.
  z = g.M*x*inv(g.M);                            %  homogeneous form.
end

end


function omegahat = hat(omega)

omegahat = [0 , -omega(3) , omega(2); ...
            omega(3), 0, -omega(1); ...
            -omega(2), omega(1), 0];

end
```

I verified that it gives the above and then tried it for the new twist,

```
>> xi2 = [1, 0, 1, 0, pi/6, -pi/12]'
>> adjoint(g, xi2)

ans =

    0.5236
         0
    1.9378
    0.1851
    0.5236
   -0.1851
```

Problem 3. [40 pts] Let's consider the three-link rotational planar manipulator from the previous homeworks, c.f. Figure 3. The link lengths were specified to be $l_1 = 1$, $l_2 = \frac{1}{2}$, and $l_3 = \frac{1}{4}$. Let's work out how to get trajectories in the α space given a desired trajectory in the (x, y)-position space. Here we will be ignoring the orientation. That makes our robot a kinematically redundant manipulator since the input space is three dimensions and the output space is only two.

The task here is to perform resolved rate trajectory generation. As hinted in earlier homeworks, the Jacobian is a useful little beast. It lets us maps joint velocities to position velocities:

$$\dot{p} = J(\alpha)\dot{\alpha}$$

assuming that α is a function of time. Resolve rate trajectory generation takes the Jacobian $J(\alpha)$ and inverts it. However, here the Jacobian matrix is not square, it is 2×3. There actually is still an inverse to this particular problem. It is called the pseudo-inverse and is actually implemented by Matlab (the function is `pinv`). I believe that Matlab can also use the back-slash or forward-slash (I don't use either, so I don't know which one is correct) to also solve for $\dot{\alpha}$ given $\dot{p}$. Anyhow, performing the pseudo-inverse gives:

$$\dot{\alpha} = J^{\dagger}(\alpha)\dot{p}$$

where $\cdot^{\dagger}$ is the symbol for the pseudo-inverse. Now, given an initial α and a desired end-effector velocity, we can integrate the above using `ode45` to get a trajectory for α that takes use from one point to another. Put succintly, you will have to integrate the following differential equation

$$\dot{\alpha}(t) = J^{\dagger}(\alpha(t))\dot{p}_{des}(t), \quad \alpha(0) = \alpha_0,$$

with a known initial condition (the initial joint configuration) and known end-effector position velocity (here it is constant). The actual function that computes the derivative $\dot{\alpha}$ will need to invoke code associated to the manipulator Jacobian. It should also have access to the desired end-effector body velocity (again, constant so that's not a problem).

Suppose that your initial configuration is $\alpha(0) = (-\pi/4, 2\pi/3, -\pi/5)$ and that your desired velocity is $\dot{p} = (-0.5, 0.5)$ which will be applied for 2 seconds. Perform resolved rate trajectory generation to get the $\alpha(t)$ trajectory. Plot **(a)** the resulting joint angles as a function of time, **(b)** the end-effector position as a function of time (so, you will have to map the α values through the forward kinematics for position the plot), and **(c)** a parametric plot of the manipulator's position. In the parametric plot, you should also overlay the plot of the manipulator in its initial configuration, in it's final configuration, and one to two intermediate configurations.

You can always view it as a movie using the hints and code from prior homeworks. Sample code should be on the class wiki page for ECE4560 (located at `pvela.gatech.edu/classes` and following the Display/Plotting link). The plot code for the manipulator is here too.

Solution 3. The first thing to do is to write a function that returns the Jacobian associated to the manipulator. Let's suppose that it is called `pJac` and takes in an α. The Jacobian for this was already worked out in one of the early homeworks, so it won't be covered further here. Then, it is just a matter of constructing the `ode45` function for the velocity in α:

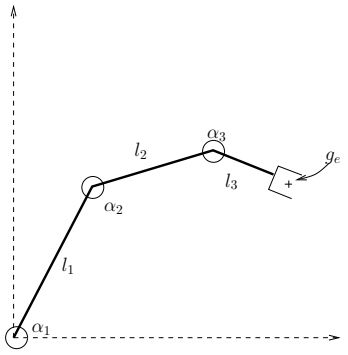


Figure 3: Planar 3R Manipulator.

```
function alphadot = rrDiffEq(t, alpha)

J = pJac(alpha);
alphadot = pinv(J)*vdes;

end
```

where it was assumed that `vdes` is available as a variable in the workspace of the function. Since we used a constant value, it could have just been coded into the function. After that, it is a matter of just running `ode45` with the function above, the desired timespane, and the proper initial condition. That goes something like this:

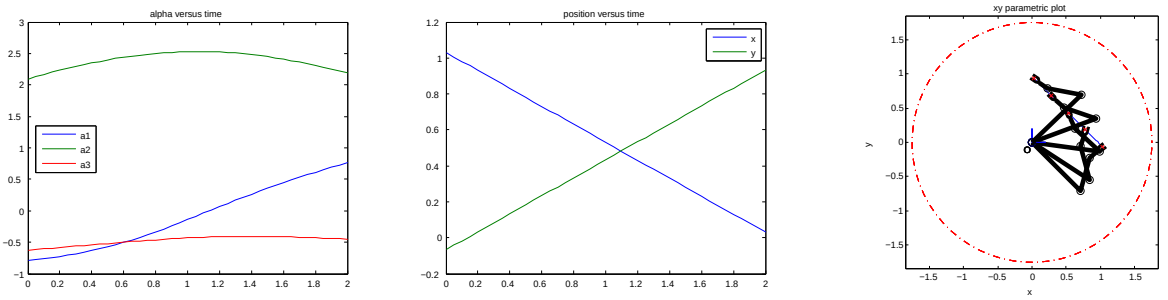
```
[tsol, asol] = ode45(@rrDiffEq, tspan, alphai);
```

Then, to visualize the sequence I did something like this:

```
tsteps = linspace(tspan(1),tspan(2),5);
figure(3);
plot(psol(1,:),psol(2,:));
  title('xy parametric plot');
  xlabel('x');
  ylabel('y');
  hold on;
  for ii=1:length(tsteps)
    planarR3_display(afun(tsteps(ii)));
  end
  hold off;
```

much like noted in the documentation for an earlier code stub provided to the class. A similar for loop could be used to visualize the movement as a movie. Just make sure to add a `drawnow` function call at the end of the loop. Note that the parametric plot requires having a variable called `psol`. It is generated by taking each `asol` and running through the forward kinematics for it. To do so, I also created a forward kinematics function based on the solutions from an earlier homework. I actually created a vectorized version of the forward kinematics, so that a matrix of multiple α vectors would return a matrix of multiple g_e vectors. This can only really be done for $SE(2)$ since $SE(3)$ does not have a realiable vector form. The alternative is to use a `for` loop to transform each `asol` joint vector.

The joint angles versus time plot is given in Figure 4(a). The parametric plot of the end-effector position is given in Figure 4(b). There are snapshots of the manipulator at for initial, intermediate, and final manipulator configurations along the trajectory in Figure 4(c). Notice that the manipulator follows the linear trajectory right on (the middle plots are straight lines). The joint angles coordinate through the (pseudo)inverse of the manipulator Jacobian to follow the desired body velocity. When it is possible to follow the desired velocity, then the manipulator will do so.



(a) Joint angles for linear trajectory. (b) Parametric plot of trajectory position. (c) Snapshots of manipulator.

Figure 4: Plots and graphs associated with the desired velocity.