ROBOT MODELING AND CONTROL

ECE470S

LAB3 : Motion Planning for the PUMA 560 Manipulator

1 Purpose

The purpose of this lab is to develop a Matlab function the performs a motion planning algorithm. The function will use artificial potential fields to find a path from an initial point and orientation in task space to a final point and orientation in task space while avoiding obstacles modeled as cylinders and spheres.

2 Preparation

1. Your work from Lab 1 is required for Lab 3.
2. Read the textbook Chapter 5 paying particular attention to Sections 5.2 and 5.5.
3. Write neatly on paper the expressions for $\|o_i(q) - b\|$ and $o_i(q) - b$ appearing in equations (5.6) and (5.7) of the textbook in the two cases:

   `case 1:` the obstacle is a sphere of radius R centred at $c = (c_x, c_y, c_z)$.

   `case 2:` the obstacle is a cylinder of infinite height with the following specs: centre at $c = (c_x, c_y)$; axis parallel to the z_0 axis; the radius is R.
4. Study this entire document ahead of the lab, so that you are prepared to complete all steps within the lab period.

3 Experiment

In this lab you will write a Matlab function that performs the motion planning algorithm developed in Chapter 5 of your textbook, and presented in class. This function will be used in Lab 4 of this course, so it is important that you test your function accurately. You will use the robotics toolbox to define and plot the robot configuration in three-space.

At the heart of the artificial potential method, there is the gradient descent algorithm

$$q^{k+1} = q^k + \alpha_k \sum_{i=1}^{6} J_{o_i}(q^k)^\top \left[F_{i,att}(o_i^0(q^k)) + F_{i,rep}(o_i^0(q^k))\right], \tag{1}$$

where

$$J_{o_i}(q^k) = \left[J_{v_1} \cdots J_{v_i} \mid 0_{3\times(6-i)}\right], \tag{2}$$

and

$$J_{v_j} = \begin{cases} z_{j-1}^0 & \text{if joint } j \text{ is } P \\ z_{j-1}^0 \times (o_i^0 - o_{j-1}^0) & \text{if joint } j \text{ is } R. \end{cases}$$

The vectors z_{j-1}^0, o_i^0, and so on are computed using forward kinematics with joint vector q^k.

3.1 The Attractive Field

The objective here is to develop a function providing the attractive component of the gradient descent algorithm in (1). Write a function `att.m` that has these arguments

```
tau = att(q,q2,myrobot)
```

where `q` is a column vector of the actual joint angles, `q2` is a column vector of the final joint angles, and `myrobot` is the robot structure from Lab 1. The function will return the vector

$$\texttt{tau} = \sum_{i=1}^{6} J_{o_i}(q)^\top F_{i,att}(o_i^0(q)).$$

For each link, you function will compute the forward kinematics for the current position of origin o_i and the forward kinematics for the final position of origin o_i. After the forward kinematics, your function will compute the artificial forces $F_{i,att}(q)$ for $i = 1, \ldots, 6$ as in (5.4) of the textbook. Take $\zeta_i = 1$ initially, but you may modify the values of ζ_i to achieve better results. Next the function will compute the Jacobians J_{o_i} in equation (2) above.

The output `tau` is a **row**[1] vector of "torques" which is the sum of each of the individual "torques" computed for each o_i. **Ensure that you normalize the output `tau` such that `norm(tau) = 1`**. Type `help norm` to see how the `norm` function of Matlab works.

Test your function as follows. You should get[2]

```
>>H1 = eul2tr([0 pi pi/2]); % eul2tr converts ZYZ Euler angles to a hom.  tsf.  mtx
>>H1(1:3,4)=100*[-1; 3; 3;]/4; % This assigns the desired displacement to the hom.tsf.mtx.
>>q1 = inverse(H1,myrobot);
% This is the starting joint variable vector.
>>H2 = eul2tr([0 pi -pi/2]);
>>H2(1:3,4)=100*[3; -1; 2;]/4;
>>q2 = inverse(H2,myrobot);
% This is the final joint variable vector
>>tau = att(q1,q2,myrobot)

tau =

-0.9050 -0.0229 -0.4003 -0.0977 0.1034 0.0000
```

Note that it may be necessary to add 2π to `q2(4)` depending on how your inverse function returns. For better debugging, you may want to test the values of the six attractive forces $F_{att,i}$. Using the data above, you should obtain

```
[ Fatt1 Fatt2 Fatt3 Fatt4 Fatt5 Fatt6] =
0 0.7003 0.7003 1.0000 1.0000 1.0000
0 -0.3662 -0.3662 -1.0000 -1.0000 -1.0000
0 -0.0996 -0.0996 -0.2500 -0.2500 -0.2500
```

Note: multiply these results by 100 to convert to units of cm.

3.2 Motion Planning without Obstacles

The objective of this task is to develop a function that implements the motion planning using the gradient descent algorithm. Write a function `motionplan.m` that has these arguments

[1] The reason for having τ be a row vector is that we will want to store the gradient descent iterations as rows in a matrix, see Section 3.2.

[2] Here we are assuming that the unit of length used in your DH table is centimetres.

```
qref = motionplan(q0,q2,t1,t2,myrobot,obs,tol)
```

where `qref` is a piecewise cubic polynomial (PWCP), `q0` is a column vector of the initial actual joint angles, `q2` is a column vector of the final joint angles, `t1` is the start time of the PWCP, `t2` is the finish time of the PWCP, `myrobot` is the robot structure from Lab 1, `obs` is an obstacle structure, and `tol` is the tolerance used to terminate the algorithm. The function `motionplan` will make function calls to `att` and `rep` and will implement the gradient descent algorithm of Section 5.2.4. By setting $\alpha^i = 0.01$ the output of the `att` function is scaled appropriately. If α^i is too large, the robot arm will oscillate around the final position.

The gradient descent algorithm produces waypoints q^i, $i = 1, \ldots, N$. Each waypoint is a vector with six components. You will collect the waypoints in a matrix `q`. The i-th row of `q` will contain the waypoint q^i (thus q^i is stored as a row vector, not a column vector). The matrix `q` will have dimension $N \times 6$.

Terminate your algorithm when `norm(q(end,1:5)-q2(1:5))<tol` (recall that `q2` is the desired joint variable vector, as discussed in the previous section). Note that the outputs of `att` and `rep` do not affect `q(6)`. Simply use the command `linspace` to make `q(6)` transition from its initial value to the desired final value.

Once the gradient algorithm terminates producing the matrix `q`, the cubic spline `qref` can be created as follows.

```
t = linspace(t1,t2,size(q,1));
qref =  spline(t,q'); % defines a spline object with interpolation
                      % times in t and interpolation values the columns of q
```

Issue the following commands to plot the trajectory of your robot arm using the command

```
qref = motionplan(q1,q2,0,10,myrobot,[],0.01);
t=linspace(0,10,300);
q = ppval(qref,t)';
plot(myrobot,q)
```

and verify that your Puma 560 converges to the desired final position.

3.3 Motion Planning with Obstacles

Download the `setupobstacle.m` and `plotobstacle.m` files from the course website. The `setupobstacle` file creates a 6-cell array of obstacle data. Your function will use this data to compute the repulsive fields. Each cell contains the data for one obstacle. There are two types of obstacles that we will be using: cylinders and spheres. The choice for these will become apparent when you will compute the distance from origin o_i to the closest point on the obstacle. The cylinder obstacle has the following fields (all units in cm):

`R`: the actual radius of the cylinder
`c`: the center (x,y) of the cylinder
`rho0`: the radius of influence of the cylinder
`h`: the height of the cylinder, used for plotting only
`type`: equal to `'cyl'` for cylinders

The sphere obstacle has these fields:

`R`: the actual radius of the sphere
`c`: the center (x,y,z) of the sphere
`rho0`: the radius of influence of the sphere
`type`: equal to `'sph'` for sphere

You will now create a function `rep` which computes the repulsive potential field for each obstacle. Write a function `rep.m` that has these arguments

```
tau = rep(q,myrobot,obs)
```

where `q` is a column vector of the actual joint angles, `myrobot` is the robot structure from Lab 1, and `obs` is the obstacle data generated by `setupobstacle.m`. The function will return the row vector

$$\texttt{tau} = \sum_{i=1}^{6} J_{o_i}(q)^\top F_{i,rep}(o_i^0(q)).$$

Using equation (5.5), (5.6), and (5.7) in the textbook you will compute the artificial repulsive force for each obstacle. As before, the forces must be mapped to joint torques using ${J_{o_i}}^\top$. For the output, you will need to sum these torques over each origin. Your function `rep` should output the repulsive torque only for one obstacle. **Ensure that you normalize the output `tau` such that `norm(tau) = 1`.**

Test your function as follows using `q1` and `q2` as before. You should get

```
>>setupobstacle
>>q3 = 0.9*q1+0.1*q2;
>>tau = rep(q3,myrobot,obs{1}) % This tests the torque for the cylinder obstacle

tau =

0.9950 0.0291 -0.0504 0.0790 0.0197 0.0000
```

In particular, the repulsive forces should be as follows:

```
[ Frep1 Frep2 Frep3 Frep4 Frep5 Frep6 ] =
0 0 0 -106.4269 -106.4269 -106.4269
0 0 0 -3.6940 -3.6940 -3.6940
0 0 0 0 0 0
```

Note: multiply these results by 10^{-6} to convert to units of cm.

Next, test your function using the sphere obstacle. You should get

```
>> q = [pi/2 pi 1.2*pi 0 0 0];
>> tau = rep(q,myrobot,obs{6})

tau =

-0.1138 -0.2140 -0.9702 0 -0.0037 0
```

Now you will modify your `motionplan` function to include the repulsive fields by simply adding the artificial repulsive torques to the artificial attractive torques. Ensure to scale the output of the `rep` function by 0.01 in the same manner as the `att` function.

Using the obstacles given in `setupobstacle`, `q1` as before, and `q2` as before, we will now plot the robot arm with the obstacles by issuing the following commands

```
>>setupobstacle
>>hold on
>>axis([-100 100 -100 100 0 200])
>>view(-32,50)
>>plotobstacle(obs);
>>qref = motionplan(q1,q2,0,10,myrobot,obs,0.01);
>>t=linspace(0,10,300);
```

```
>>q=ppval(qref,t)';
>>plot(myrobot,q);
>>hold off
```

Feel free to experiment with different obstacles and different artificial torque scaling factors to see how results change.

4 Submission

Each student will submit the lab preparation and the following on Quercus.

1. A zipped folder containing ALL of your Matlab functions including functions from Lab 1. Thoroughly comment your code. Your folder may contain intermediate Matlab functions, but calls to the main functions `motionplan`, `att`, and `rep` must work without any modification required.
2. A Matlab file `lab3.m` working out the various steps in Section 3.