

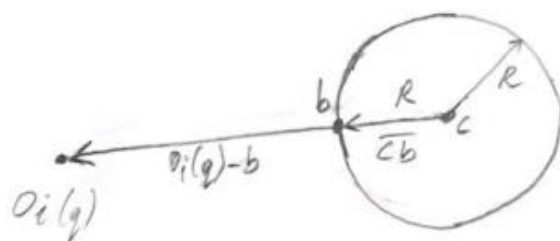
ECE470 LAB-3 REPORT AND PREPARATION

Deniz Akyildiz

1003102224

PREPARATION

Case 1: sphere with R , centered at $c = (c_x, c_y, c_z)$



* \vec{cb} is in the direction of $o_i(q) - b$ and $o_i(q) - c$

$$\text{Now, } b = c + R \cdot \frac{o_i(q) - c}{\|o_i(q) - c\|}$$

unit vector in the direction
of $o_i(q) - b$ and $o_i(q) - c$
(just to give direction to R)

Thus,

$$\Rightarrow o_i(q) - b = o_i(q) - c - R \cdot \frac{o_i(q) - c}{\|o_i(q) - c\|} = (o_i(q) - c) \left(1 - \frac{R}{\|o_i(q) - c\|}\right)$$

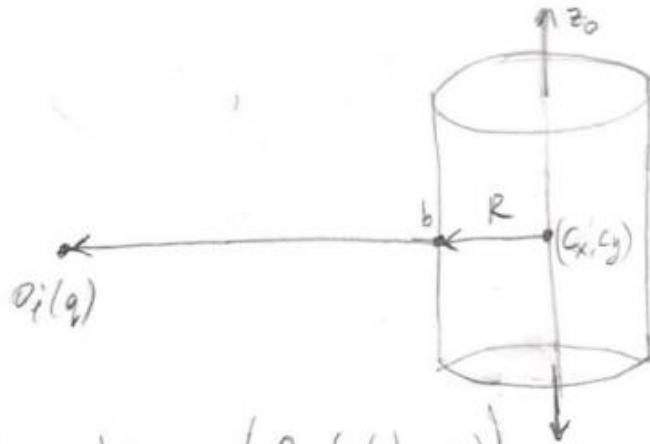
From the figure we see that

$$o_i(q) - c = \vec{cb} + o_i(q) - b$$

$$\text{And magnitude wise: } \|o_i(q) - c\| = \underbrace{\|\vec{cb}\|}_R + \|o_i(q) - b\|$$

$$\Rightarrow \|o_i(q) - b\| = \|o_i(q) - c\| - R$$

Case 2: cylinder of inf. height, centered at $C = (c_x, c_y)$
axis parallel to z_0 , radius R



$$\text{Now, } b_x = c_x + \left(\frac{R \cdot (p_i(q)_x - c_x)}{\|p_i(q) - c\|} \right)$$

$$b_y = c_y + \left(\frac{R \cdot (p_i(q)_y - c_y)}{\|p_i(q) - c\|} \right)$$

$$b_z = p_i(q)_z$$

$$\Rightarrow p_i(q) - b = \begin{bmatrix} (p_i(q)_x - c_x) \\ (p_i(q)_y - c_y) \\ 0 \end{bmatrix} \cdot \left(1 - \frac{R}{\|p_i(q) - c\|} \right)$$

$$\Rightarrow \|p_i(q) - b\| = \|p_i(q) - c\| - R \quad (\text{similar to part 1})$$

EXPERIMENT

3.1 The Attractive Field

In this part, the attractive part that is needed for gradient descent is computed. The computation of the attractive torque is done in att.m as seen in Figure 1.

This function returns the expression:

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

In order to find tau (attractive torques), att.m function computes the forward kinematics for each link for the given positions (elements of q and q2, which represent the initial and final joint angles). Then, the attractive forces are computed using the expression 5.4 from the textbook:

$$F_{att,i}(q) = \begin{cases} -\zeta_i(o_i(q) - o_i(q_f)) & : \|o_i(q) - o_i(q_f)\| \leq d \\ -d\zeta_i \frac{(o_i(q) - o_i(q_f))}{\|o_i(q) - o_i(q_f)\|} & : \|o_i(q) - o_i(q_f)\| > d \end{cases}$$

It is important to note that here the d value is taken as infinite, which means that only the first part of the equation is used. Finally, the corresponding Jacobian as stated in the handout is computed, and tau value is returned after normalization. The verification of this function could be seen in Figure 2, where the output is the expected result from the handout. ζ value is taken as 1, following the handout suggestion.

```
function tau = att(q,q2,myrobot)
    % Solve FK for qs and qf
    H_start = forward_jacobian(q, myrobot);
    H_final = forward_jacobian(q2, myrobot);
    % z value could be adjusted here
    z = 1;
    tau = zeros(1,6);
    for i = 1:6
        o_start(:,i) = H_start(1:3,4,i);
        o_final(:,i) = H_final(1:3,4,i);
        F_att(:,i) = -z * (o_start(:,i) - o_final(:,i));
        J = jacobian_i(q, myrobot,i);
        J_o_i = horzcat(J(1:3, 1:i), zeros(3,6-i));
        tau = tau + (J_o_i' * F_att(:,i))';
    end
    if norm(tau) ~= 0
        tau = tau / norm(tau);
    end
end
```

Figure 1: Attractive torques computation (att.m)

```
13 %% 3.1 The Attractive Field
14
15 H1 = eul2tr([0 pi pi/2]); % eul2tr converts ZYZ Euler angles to a hom. tsf. mtx
16 H1(1:3,4)=100*[-1; 3; 3;]/4; % This assigns the desired displacement to the hom.tsf.mtx.
17 q1 = inverse(H1,myrobot);
18 % This is the starting joint variable vector.
19 H2 = eul2tr([0 pi -pi/2]);
20 H2(1:3,4)=100*[3; -1; 2;]/4;
21 q2 = inverse(H2,myrobot);
22 % This is the final joint variable vector
23 disp('The resulting attractive component:')
24 tau = att(q1,q2,myrobot)
25
```

Command Window

New to MATLAB? See resources for [Getting Started](#).

The resulting attractive component:

tau =

-0.9049 -0.0228 -0.4005 -0.0977 0.1034 0.0000


 >> |

Figure 2: Attractive torques verification

3.2 Motion Planning without Obstacles

In this part, motion planning is performed for the robot in an obstacle free setting. Figure 3 shows the motion planning algorithm implementation with gradient descent and attractive torques only. There is no need for repulsive components since there are no obstacles.

```
function qref = motionplan_free(q0,q2,t1,t2,myrobot,obs,tol)
    % motion planning with no obstacles
    q = q0;
    alpha = 0.001;
    while norm(q(end,1:5)-q2(1:5)) >= tol
        % Gradient descent algorithm:
        tau = alpha*att(q(end,:), q2, myrobot);
        q(end+1,:) = q(end,:) + tau;
    end
    q(:,6) = linspace(q0(6),q2(6),size(q,1));
    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
end
```

Figure 3: Motion planning without obstacles (motionplan.m initial version, named motionplan_free to distinguish from the motionplan function for obstacles)

In order to verify the motion planning function in Figure 3, a trajectory is calculated for given start and end states. Figure 4 shows the resulting position of the robot, where it ended up in the desired coordinates. To observe the full motion, section 3.2 in the Matlab code (Lab3.m) could be run. The α value here is chosen smaller (compared to given 0.01 value) to stop the oscillation around final position.

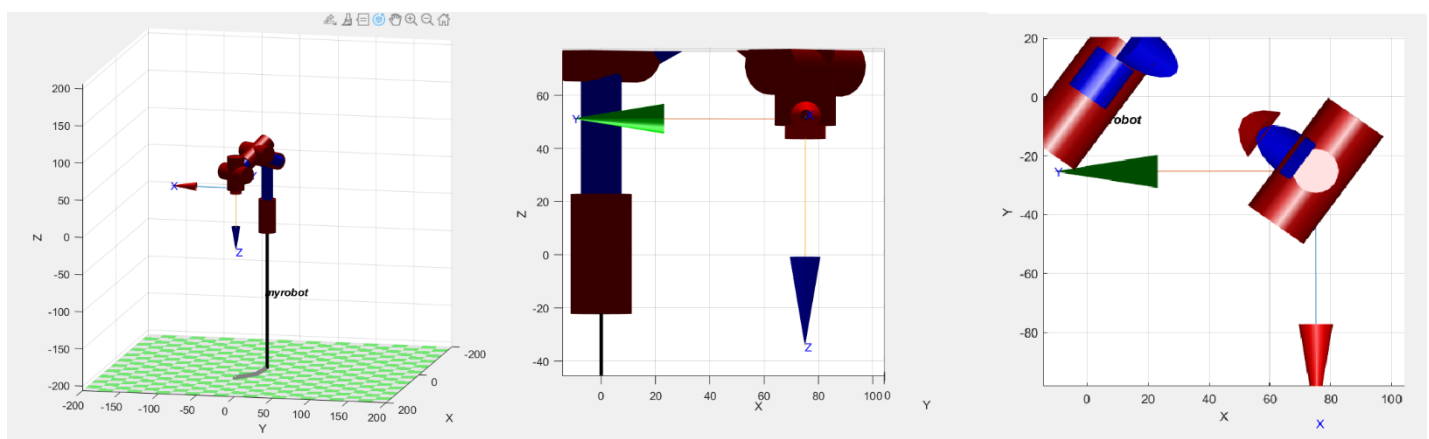


Figure 4: The end position of the robot after running the verification code from the handout

3.3 Motion Planning with Obstacles

In this part, motion planning with obstacles is performed for the robot. With the introduction of obstacles to the environment, there is a need to calculate the repulsive torques. The computation of the repulsive torque is done in rep.m as seen in Figure 5.

This function returns the expression:

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

In order to find τ (repulsive torques), rep.m function computes the forward kinematics for each link for the given positions (elements of q and q_2 , which represent the initial and final joint angles). Then, the repulsive forces are computed using the expression 5.6 and 5.7 from the textbook:

$$F_{rep,i}(q) = \eta_i \left(\frac{1}{\rho(o_i(q))} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2(o_i(q))} \nabla \rho(o_i(q)) \quad \nabla \rho(x) \Big|_{x=o_i(q)} = \frac{o_i(q) - b}{\|o_i(q) - b\|}$$

The detailed computation of what these equations correspond to with a spherical or cylindrical obstacle are shown in the preparation. After this step, similar to part 3.1, the corresponding Jacobian as stated in the handout is computed and τ value is returned after normalization.

```
function tau = rep(q,myrobot,obs)
% Repulsive component calculation
H = forward_jacobian(q, myrobot);
eta = 0.01;
tau = zeros(1,6);
for i = 1:6
    o_i(:,i) = H(1:3,4,i);
    if obs.type == 'cyl'
        oi_b_x = (o_i(1,i) - obs.c(1)) * (1 - (obs.R / norm(o_i(1:2,i) - obs.c)));
        oi_b_y = (o_i(2,i) - obs.c(2)) * (1 - (obs.R / norm(o_i(1:2,i) - obs.c)));
        oi_b = [oi_b_x; oi_b_y; 0];
        n = norm(o_i(1:2,i) - obs.c) - obs.R;
    end
    if obs.type == 'sph'
        oi_b = (o_i(:,i) - obs.c) * (1 - (obs.R / norm(o_i(:,i) - obs.c)));
        n = norm(o_i(:,i) - obs.c) - obs.R;
    end
    J = jacobian_i(q, myrobot,i);
    J_o_i = horzcat(J(1:3, 1:i), zeros(3,6-i));
    if n <= obs.rho0
        F_rep(:,i) = eta * ((1/n) - (1/obs.rho0)) * (1/(n*n)) * (oi_b/n);
    else
        F_rep(:,i) = zeros(3,1);
    end
    tau = tau + (J_o_i' * F_rep(:,i))';
end
if norm(tau) ~= 0
    tau = tau / norm(tau);
end
end
```

Figure 5: Repulsive torques computation (rep.m)

The verification of this function could be seen in Figure 6, where the output is the expected result from the handout. η value is taken as 0.01, after a trial-and-error process.

```

%% 3.3
- setupobstacle
- q3 = 0.9*q1+0.1*q2;
- disp('The resulting repulsive component for obstacle 1:')
- tau = rep(q3,myrobot,obs{1})

- q = [pi/2 pi 1.2*pi 0 0 0];
- disp('The resulting repulsive component for obstacle 6:')
- tau = rep(q,myrobot,obs{6})

```

Command Window

new to MATLAB? See resources for [Getting Started](#).

The resulting repulsive component for obstacle 1:

tau =

0.9950	0.0291	-0.0504	0.0790	0.0197	-0.0000
--------	--------	---------	--------	--------	---------

The resulting repulsive component for obstacle 6:

tau =

-0.1135	-0.2143	-0.9701	0.0000	-0.0037	0.0000
---------	---------	---------	--------	---------	--------

Figure 6: Repulsive torques verification

After making sure that the rep.m function is working properly, it is used in motion planning. Similar to obstacle free case, a gradient descent algorithm is implemented, with the addition of repulsive torques. Since it is possible to have multiple obstacles, this function expects an array of obstacles (obs) and loops to find the total repulsive torques on every joint, calling rep.m function every time for each obstacle. The implementation of motion planning for an environment with multiple obstacles could be seen in Figure 7.

Finally, the robot movement that is planned by this function could be observed in Figure 8, where all the obstacles are enabled. It could be observed that the robot avoids all the obstacles, and to see the full motion, part 3.3. of Lab3.m could be run. It is possible to remove or add any obstacle with the current set up of the motionplan.m function. After trial and error, the chosen values are: $\eta = 0.01$, $\zeta = 1$, and $\alpha = 0.01$. At some parts, the robot seems like it is getting stuck in between two obstacles, however eventually it reaches the desired position as requested.


```

function qref = motionplan(q0,q2,t1,t2,myrobot,obs,tol)
    q = q0;
    alpha = 0.01;
    while norm(q(end,1:5)-q2(1:5)) >= tol
        % Gradient descent algorithm:
        tau_rep = zeros(1,6);
        for k=1:size(obs,2)
            tau_rep = tau_rep + alpha*rep(q(end,:),myrobot,obs{k});
        end
        tau = alpha*att(q(end,:), q2, myrobot) + tau_rep;
        q(end+1,:) = q(end,:) + tau;
    end
    q(:,6) = linspace(q0(6),q2(6),size(q,1));
    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
end

```

Figure 7: Motion planning with obstacles

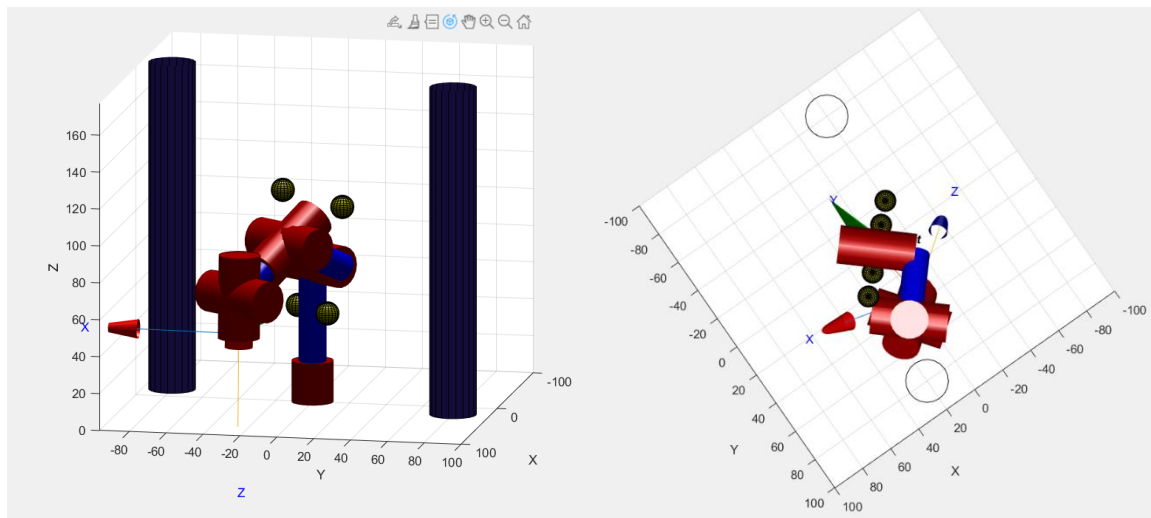


Figure 8: Robot motion around the obstacles