ECE470 LAB-2 PREPARATION

Deniz Akyildiz

1003102224

PREPARATION

3.1 Geometric Jacobian Pseudocode

3.1) J_g(myrobot , q) // geometric Jacobian

for i from 1 to 6: → this could be outside the loop too

H_{i-1}^0 = forward(q, myrobot) // compute H_i^0 using forward.m from lab1 (might need to slightly modify)

$z_{i-1}^0 = H_{i-1}^0(1{:}3, 3)$ // extract z_i^0 and store

$o_{i-1}^0 = H_{i-1}^0(1{:}3, 4)$ // extract o_i^0 and store

for i from 1 to 6:

$J_{v_i} = z_{i-1}^0 \times (o_6^0 - o_{i-1}^0)$ // since all joints are revolute

$J_{w_i} = z_{i-1}^0$

$J = [J_v ; J_w]$

return J

3.2 Analytic Jacobian Pseudocode

3.2) J_a (myrobot , q)

H_6^0 = forward (q, myrobot)

q_inv = inverse (H_6^0, myrobot) // this also need to be modified, since euler angles are different

J_geom = J_g (myrobot, q)

ϕ, θ, ψ = q_inv[4,:], q_inv[5,:], q_inv[6,:] // extract euler angles

$B = [0, -s\phi, c\phi s\theta;\ 0, c\phi, s\phi s\theta;\ 1, 0, c\theta]$

$J\_a = \begin{bmatrix} I_3 & 0 \\ 0 & B^{-1} \end{bmatrix} \cdot J\_geom$

return J_a

3.3 Geometric Jacobian Code

Using the pseudocode from section 3.1, the algorithm to calculate the geometric Jacobian was implemented in Matlab. The forward kinematics function from Lab1 was used with slight modification here, in order to store all the homogenous transformations instead of only returning the last one. This new forward kinematics function is called *forward_jacobian*. The function was verified by testing with the given values in the handout and the output is shown in Figure 3. Note that the handout assumes the units are in meters, however throughout the lab centimeters were used, thus there is a scaling difference between the output and the expected value.

```
function H_all = forward_jacobian(joint, myrobot)
    % Calculates forward kinematics given joint angles and robot model.
    H = eye(4);
    H_all = zeros(4,4,6);
    % Loop to create individiual matrices from ith to (i-1)th reference
    % frames, and multiply them to get the forward kinematics matrix.
    for i = 1:6
        theta_i = joint(i);
        alpha_i = myrobot.alpha(i);
        a_i = myrobot.a(i);
        d_i = myrobot.d(i);
        H_i = [cos(theta_i) -sin(theta_i)*cos(alpha_i) ...
               sin(theta_i)*sin(alpha_i) a_i*cos(theta_i); ...
               sin(theta_i) cos(theta_i)*cos(alpha_i) ...
               -cos(theta_i)*sin(alpha_i) a_i*sin(theta_i); ...
               0 sin(alpha_i) cos(alpha_i) d_i; ...
               0 0 0 1];
        H = H * H_i;
        H_all(:,:,i) = H;
end
```

Figure 1: Modified forward kinematics function

```
function J = jacobian(joint, myrobot)
    % Function to compute the geometric jacobian
    % given the robot structure and joint variables
    H_all = forward_jacobian(joint, myrobot);
    o_0_6 = H_all(1:3,4,6);
    J_v = zeros(3,6);
    J_w = zeros(3,6);
    for i = 1:6
        if i == 1
            o = zeros(3,1);
            z = [0;0;1];
        else
            o = H_all(1:3,4,i-1);
            z = H_all(1:3,3,i-1);
        end
        Jv_i = cross(z, (o_0_6 - o));
        Jw_i = z;
        J_v(:,i) = Jv_i;
        J_w(:,i) = Jw_i;
    end
    J = [J_v ; J_w];
end
```

Figure 2: Geometric Jacobian calculation (jacobian.m)

```
Verify geometric jacobian:

ans =

   -9.9507   13.0699   39.5488    0.6699   19.6958         0
  -13.4955   13.0699   39.5488   -9.3301    2.3753   -0.0000
         0   -2.5065  -24.1265    3.5355    2.5365         0
         0    0.7071    0.7071   -0.3536    0.0670    0.1603
         0   -0.7071   -0.7071   -0.3536   -0.9330   -0.3397
    1.0000    0.0000    0.0000   -0.8660    0.3536   -0.9268
```

Figure 3: Verify geometric Jacobian (jacobian.m)

3.4 Analytic Jacobian Code

Using the pseudocode from section 3.2, the algorithm to calculate the analytic Jacobian was implemented in Matlab. The function was verified by testing with the given values in the handout and the output is shown in Figure 5. Note that the handout assumes the units are in meters, however throughout the lab centimeters were used, thus there is a scaling difference between the output and the expected value.

```
function Ja = ajacobian(joint, myrobot)
    % Function to compute the analytic jacobian
    % given the robot structure and joint variables
    H = forward(joint, myrobot);
    R = H(1:3, 1:3); % could not use inv. kin. here since euler angles are different
    fi = atan2(R(2,3), R(1,3));
    theta = atan2(sqrt(1- R(3,3)^2), R(3,3));
    Jg = jacobian(joint, myrobot);
    B = [0, -sin(fi), cos(fi) * sin(theta); ...
         0, cos(fi), sin(fi) * sin(theta); ...
         1, 0, cos(theta)];
    Ja = ([eye(3) zeros(3,3); zeros(3,3) inv(B)] * Jg);
end
```

Figure 4: Analytic Jacobian calculation (ajacobian.m)

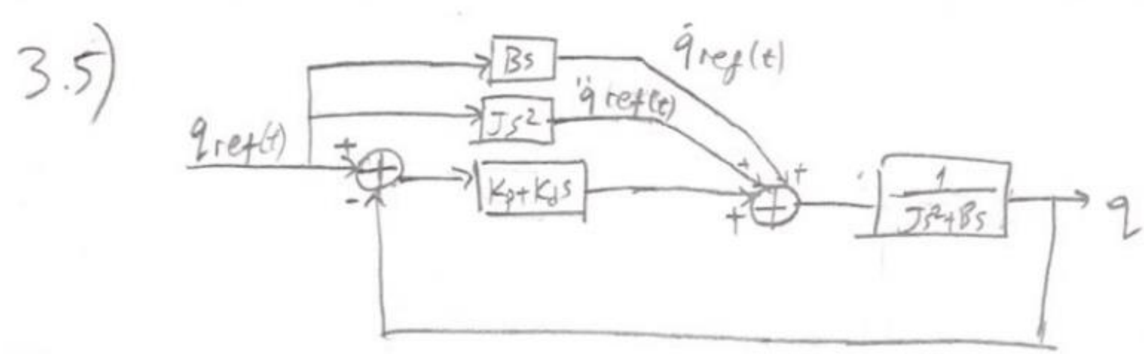
```
Verify analytic jacobian:

ans =

   -9.9507   13.0699   39.5488    0.6699   19.6958         0
  -13.4955   13.0699   39.5488   -9.3301    2.3753   -0.0000
         0   -2.5065  -24.1265    3.5355    2.5365         0
    1.0000    2.3225    2.3225   -0.4495    2.5060    0.0000
         0    0.3377    0.3377   -0.4706   -0.3377    0.0000
         0    2.5060    2.5060    0.4495    2.3225    1.0000
```

Figure 5: Verify analytic Jacobian (ajacobian.m)

3.5 Controller Block Diagram



3.6 Gain Values

3.6) We need the transfer function which is

$$T(s) = \frac{(Js^2+Bs)\left((K_p+K_d s)+(Js^2+Bs)\right)}{Js^2+Bs+K_p+K_d s}$$

where characteristic polynomial is $Js^2+Bs+K_p+K_d s$

we know the poles are at $s=-2$

$$\Rightarrow (s+2)^2 = s^2+\left(\frac{B+K_d}{J}\right)s+\frac{K_p}{J}$$

$$s^2+4s+4 = s^2+\left(\frac{B+K_d}{J}\right)s+\frac{K_p}{J}$$

$$\Rightarrow \boxed{\begin{array}{l} K_d = 4J-B \\ K_p = 4J \end{array}}$$

And we get 12 values by:

$$\left.\begin{array}{l} K_{p1} = 8\times10^{-4} \\ K_{p2} = 8\times10^{-4} \\ K_{p3} = 8\times10^{-4} \\ K_{p4} = K_{p5} = K_{p6} = 1.32\times10^{-4} \end{array}\right\}$$

$K_{d1} = 8\times10^{-4} - 14.8\times10^{-4} = -6.8\times10^{-4}$

$K_{d2} = -0.17\times10^{-4} = -1.7\times10^{-5}$

$K_{d3} = (8-13.8)\times10^{-4} = -5.8\times10^{-4}$

$K_{d4} = 6.08\times10^{-5}$

$K_{d5} = 4.94\times10^{-5}$

$K_{d6} = 9.53\times10^{-5}$