Assignment 1

Name:

Session (ROB 599 or ROB 498):

Reading Assignment: Chapter 5 & 6 of Modern Robotics (MR)

YouTube Playlist: https://www.youtube.com/playlist?list=PLEYKx4BGriISagN3Ihc-9L6Cw44vtzTD9

Submission: Export this file as a PDF and submit to Canvas together with the writing part.

Deadline: 9/17/2025

- If you have any questions about the assignment or software, please use the Discussion function on Canvas to seek help
- If your question was not addressed on Canvas, then reach out to GSI: Yulun Zhuang
- The live script for homework is still evolving, please check Canvas periodically to make sure you are working on the latest version

Written Assignment

Use LaTex for homework

Use LaTex syntax to anwer the written part of the assignment

Latex expressions can be brought up by pressing ctrl + shift + L in Windows or cmd + shift + L in Mac

Table of Contents

Problem 1: Rotation Matrix [5 pts]

1a) [1 pt] Properties of a rotation matrix

If we view the rotation matrix $R = [r_1, r_2] \in SO(2)$ as a matrix with 2 column vectors, write down the 3 properties of the rotation matrix

Properties of a rotation matrix:

1. $R^\top R = I$
2. $\det(R) = 1$
3. $r_1^\top r_2 = 0, \quad \|r_1\| = \|r_2\| = 1$

1b) [1 pt] Proof: Prove that $R^{-1} = R^\top$

Proof:

$R^\top R = I$

Left multiply both sides by R^{-1}

$R^{-1} R^\top R = R^{-1} I \Rightarrow R^{-1} = R^\top$

1c) [3 pts] Prove the equation below:

$\widehat{R\omega} = R\hat{\omega}R^\top$

where ω is the angular velocity

Hint*: the following distribution law holds:

$R(a \times b) = (Ra) \times (Rb)$

Proof:

$\widehat{R\omega}x = (R\omega) \times x = (R\omega) \times (RR^\top x) = R\big(\omega \times (R^\top x)\big) = R\hat{\omega}(R^\top x) = (R\hat{\omega}R^\top)x$

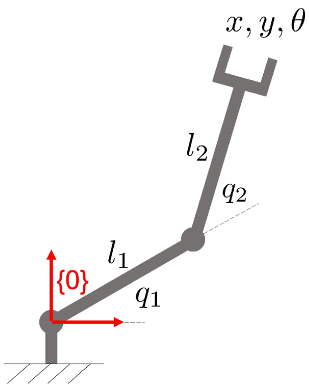
For any x, we conclude:

$\widehat{R\omega} = R\hat{\omega}R^\top$

Problem 2: Analytic Inverse Kinematics [3 pts]

2a) [2 pts] For the 2-link robot, derive the inverse kinematics

express q_1, q_2 using l_1, l_2, x, y



$x = l_1 cosq_1 + l_2 cos(q_1 + q_2)$

$y = l_1 sinq_1 + l_2 sin(q_1 + q_2)$

$r^2 = x^2 + y^2$

$c_2 = \dfrac{r^2 - l_1^2 - l_2^2}{2 l_1 l_2}$

$s_2 = \pm\sqrt{1 - c_2^2}$

$q_2 = arctan(s_2, c_2)$

$q_1 = arctan(y, x) \; - arctan\left(l_2 s_2, \; l_1 + l_2 c_2\right)$

2b) [1 pt] What are the set of q that correspond to singularities?

$$\mathcal{S} = \{ \; q \in \mathbb{R}^2 \mid q_2 = 0 \;\; \text{or} \;\; q_2 = \pi \, \}$$

Coding Assignment

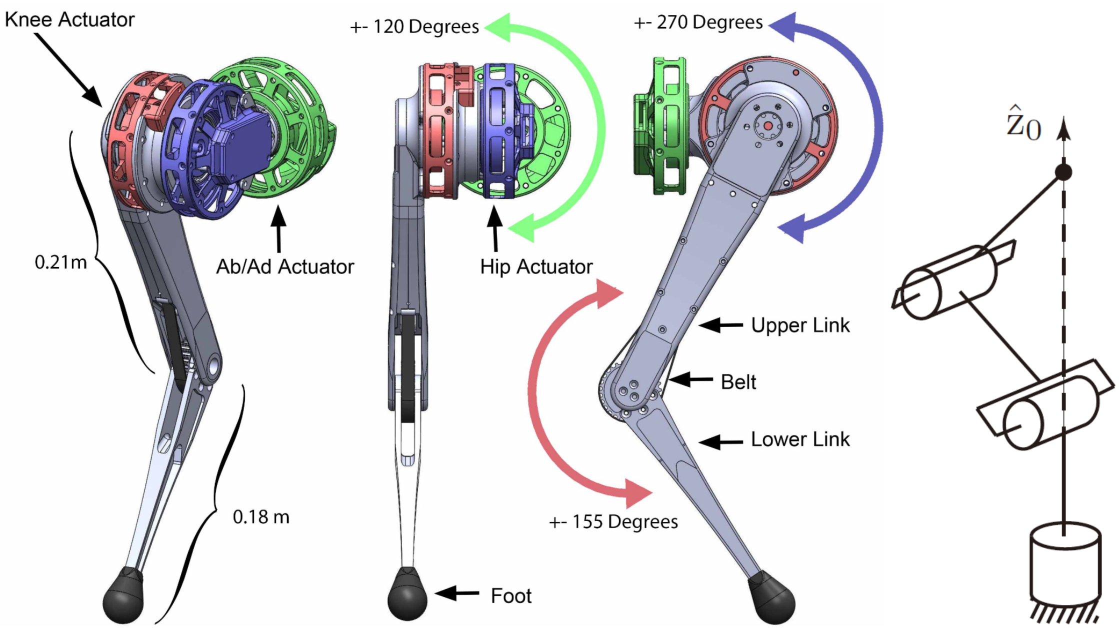
Codebase Structure & Setup

Execute the `setup.m` script to add paths of `spatical_v2` and check installation status of CasADi.

```
clc; clear; close all;
syms q dq p v tau f l [3,1] real;
robot = model_inverted_leg();
```

Problem Context

In this assignment, we will work on a 3 DOF leg from MIT minicheetah (left), which can be abstracted as a 3R robot arm (right)



The variable names and symbol are listed:

* EE: End-Effector

Notations	Symbol	Size
Joint Angles	q	3
Joint Velocities	$\dot{q}$	3
Joint Torques	τ	3
EE Position	p	3
EE Velocity	v	3
EE Force	f	3
EE Jacobian	J	3x3

The constant and value are listed:

Notations	Symbol	Value
Ab/Ad Offset	l_1	0.08
Upper Link	l_2	0.3
Lower Link	l_3	0.25

Problem 3 - Forward Kinematics (FK) [7 pts]

3a) [2 pts] Derive the analytic FK of the inverted leg via Homogeneous Transformation Matrices.

Assume the configuration with all zero joint position (home configuration) is when both links are pointing vertically up

```
pos_ee_gt = [
    cos(q1) * (l3 * sin(q2 + q3) + l2 * sin(q2));
    sin(q1) * (l3 * sin(q2 + q3) + l2 * sin(q2));
    l1 + l3 * cos(q2 + q3) + l2 * cos(q2)
]
```

pos_ee_gt =

$$\begin{pmatrix} \cos(q_1)\ (l_3 \sin(q_2+q_3) + l_2 \sin(q_2)) \\ \sin(q_1)\ (l_3 \sin(q_2+q_3) + l_2 \sin(q_2)) \\ l_1 + l_3 \cos(q_2+q_3) + l_2 \cos(q_2) \end{pmatrix}$$

```
pos_ee_htm = []; % (3, 1) analyitical expression of end-effector position derived
from HTM
%%% YOUR CODE START %%%
Rz_q1 = [cos(q1) -sin(q1) 0;
         sin(q1)  cos(q1) 0;
         0        0       1];

Ry_q2 = [cos(q2)  0  sin(q2);
         0        1  0;
        -sin(q2)  0  cos(q2)];

Ry_q3 = [cos(q3)  0  sin(q3);
         0        1  0;
        -sin(q3)  0  cos(q3)];

T01 = [Rz_q1 [0;0;l1]; 0 0 0 1];
T12 = [Ry_q2 [0;0;0];   0 0 0 1];
T23 = [Ry_q3 [0;0;l2];  0 0 0 1];
T3E = [eye(3) [0;0;l3]; 0 0 0 1];

T0E = simplify(T01 * T12 * T23 * T3E);

pos_ee_htm = simplify(T0E(1:3,4));

%%% YOUR CODE END %%%

assert(isequaln(pos_ee_gt, simplify(expand(pos_ee_htm))), "FK didn't match!")
```

3b) [1 pt] Compute the EE postiion at configuration $q = [1, 0.8, 0.5]^T$

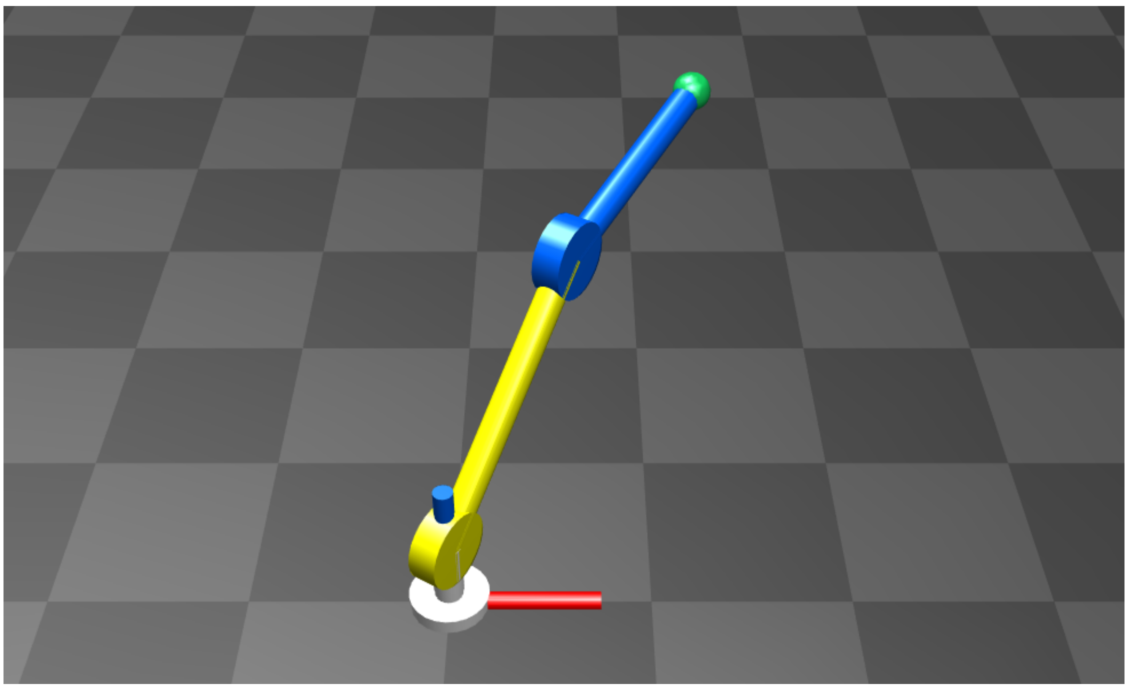
Note: use the $subs$ function: https://www.mathworks.com/help/symbolic/sym.subs.html

```
pos_ee_val = zeros(3, 1); % (3, 1) numerical values of end-effector position
q_val = [1, 0.8, 0.5]';
l_val = [0.08, 0.3, 0.25]';
%%% YOUR CODE START %%%

pos_ee_val = double(subs(pos_ee_htm, ...
    {q1, q2, q3, l1, l2, l3}, ...
    {q_val(1), q_val(2), q_val(3), l_val(1), l_val(2), l_val(3)}));

%%% YOUR CODE END %%%
assert(norm(pos_ee_val - [0.2464    0.3838    0.3559]')<1e-3, 'EE position error!')

% visualize the robot
showmotion(robot, [1, 2], [q_val, zeros(3, 1)])
```



3c) [2 pts] Compute the EE velocity at configuration $q = [1, 0.8, 0.5]^T$ **and** $\dot{q} = [0.2, 0.5, 1]$

Note: compute Jacobian and use $v = J\dot{q}$

```
vel_ee_val = zeros(3, 1); % (3, 1) numerical value of end-effector velocity
q_val = [1, 0.8, 0.5]';
dq_val = [0.2, 0.5, 1]';
%%% YOUR CODE START %%%

J = jacobian(pos_ee_htm, [q1 q2 q3]);

J_val = double(subs(J, {q1, q2, q3, l1, l2, l3}, ...
    {q_val(1), q_val(2), q_val(3), 0.08, 0.3, 0.25}));

vel_ee_val = J_val * dq_val;

```

```
%%% YOUR CODE END %%%
assert(norm(vel_ee_val - [0.0339    0.2216    -0.4689]')<1e-3, 'EE velocity error!')
```

3d) [1 pt] Outout Matlab functions

Output EE position, velocity, Jacobian as separte matlab functions

Note: use the *matlabFunction* function: https://www.mathworks.com/help/symbolic/sym.matlabfunction.html

As an example, the way to generate matlab function for EE position is:

```
matlabFunction(pos_ee_htm, 'File', 'fcn_pos_ee.m', 'Vars', {q, l})
```

```
ans = function_handle with value:
    @fcn_pos_ee
```

```
%%% YOUR CODE START %%%

matlabFunction(pos_ee_htm, ...
    'File', 'fcn_pos_ee.m', ...
    'Vars', {q, l});

J = jacobian(pos_ee_htm, q);
matlabFunction(J, ...
    'File', 'fcn_jacobian_ee.m', ...
    'Vars', {q, l});

vel_ee_sym = J * dq;
matlabFunction(vel_ee_sym, ...
    'File', 'fcn_vel_ee.m', ...
    'Vars', {q, dq, l});

%%% YOUR CODE END %%%
```

3e) [1 pt] evaluate matlab functions for EE position, velocity and Jacobian

As an example, the way to compute EE position is:

```
pos_ee_val_m = fcn_pos_ee(q_val, l_val)
```

```
pos_ee_val_m = 3×1
    0.2464
    0.3838
    0.3559
```

```
%%% YOUR CODE START %%%

vel_ee_val_m = fcn_vel_ee(q_val, dq_val, l_val);
J_val_m = fcn_jacobian_ee(q_val, l_val);

%%% YOUR CODE END %%%
```

Problem 4 - Numerical Inverse Kinematics (IK) [3 pts]

4a) [2 pts] Use nonlienar least square to solve the IK numerically.

Note: use the function *lsqnonlin*: https://www.mathworks.com/help/optim/ug/lsqnonlin.html#buuhcjf-2

```
pos_ee_des = [0.3; 0.3; 0.3];
q_guess = [0; 0; 0];
q_sol = []; % (3, 1) solved joint positions
%%% YOUR CODE START %%%

ik_error_fun = @(q) fcn_pos_ee(q, l_val) - pos_ee_des;
opts = optimoptions('lsqnonlin', 'Display', 'off');
q_sol = lsqnonlin(ik_error_fun, q_guess, [], [], opts);

%%% YOUR CODE END %%%

pos_ee_calc = fcn_pos_ee(q_sol, l_val)
```

```
pos_ee_calc = 3×1
    0.3000
    0.3000
    0.3000
```

```
error = norm(pos_ee_calc - pos_ee_des);
assert(error < 1e-3, 'IK solution error!')
```

4b) [1 pt] Solve for joint space trajectory from task space trajectory

Given a task space trajectory, solve for the corresponding joint trajectories.

The test case is generated with all zero initial guesses, but a good practice is to use the previous solution as the next initial guess. Both solutions are acceptable for this assignment.

Hint: you can visualize the solved trajectory by running the following in **terminal**.

`close all force`

`showmotion(robot, linspace(0, 1, Nt), q_traj)`

```
Nt = 11;
t = linspace(0,2*pi,Nt);
pos_ee_traj = [0.3 + 0.1 * cos(t);
               0.3 + 0.1 * sin(t);
               0.3 * ones(1,Nt)];

q_traj = zeros(3, Nt); % (3, Nt) Solved joint trajectory
ops = optimoptions(@lsqnonlin,'Display','off');
%%% YOUR CODE START %%%

q_guess = [0; 0; 0];

for i = 1:Nt
    pos_des = pos_ee_traj(:, i);
    ik_error_fun = @(q) fcn_pos_ee(q, l_val) - pos_des;
```

```
    if i > 1
        q_guess = q_traj(:, i-1);
    end

    q_sol = lsqnonlin(ik_error_fun, q_guess, [], [], ops);

    q_traj(:, i) = q_sol;
    q_guess = q_sol;
end

%%% YOUR CODE END %%%

q_traj_gt = ...
    [0.6435, 0.7555, 0.8736, 0.9729, 1.0226, 0.9828, 0.8334, 0.6508, 0.5544,
0.5645, 0.6435;
    1.0497, 1.1728, 1.1672, 0.8709, 0.6095, 0.4005, 0.2880, 0.3277, 0.4970, 0.7336,
1.0497;
    0.2346, 0.0001, 0.0002, 0.5926, 1.0654, 1.3973, 1.5569, 1.5021, 1.2494, 0.8484,
0.2346];
error = norm(q_traj - q_traj_gt);
assert(error < 1.5e-4, 'joint trajectory solution error!')
```

Problem 5 - Euler–Lagrange Dynamics [7 pts]

5a) [2 pts] Compute the linear velocities of the center of mass (CoM) at each link

Assume the center of mass for each link is in the middle.

```
%%% YOUR CODE START %%%

Rz1 = [cos(q1) -sin(q1) 0;
       sin(q1)  cos(q1) 0;
       0        0       1];
Ry2 = [cos(q2) 0 sin(q2);
       0       1 0;
      -sin(q2) 0 cos(q2)];
Ry3 = [cos(q3) 0 sin(q3);
       0       1 0;
      -sin(q3) 0 cos(q3)];


T01 = [Rz1 [0;0;l1]; 0 0 0 1];
T12 = [Ry2 [0;0;0];  0 0 0 1];
T23 = [Ry3 [0;0;l2]; 0 0 0 1];
T3C = [eye(3) [0;0;l3/2]; 0 0 0 1];
T2C = [eye(3) [0;0;l2/2]; 0 0 0 1];
T1C = [eye(3) [0;0;l1/2]; 0 0 0 1];
```

```
p1c = (T01*T1C)*( [0;0;0;1] );
p2c = (T01*T12*T2C)*( [0;0;0;1] );
p3c = (T01*T12*T23*T3C)*( [0;0;0;1] );

Jv1 = jacobian(p1c(1:3), [q1 q2 q3]);
Jv2 = jacobian(p2c(1:3), [q1 q2 q3]);
Jv3 = jacobian(p3c(1:3), [q1 q2 q3]);

v1_c = simplify( Jv1 * [dq1; dq2; dq3] );
v2_c = simplify( Jv2 * [dq1; dq2; dq3] );
v3_c = simplify( Jv3 * [dq1; dq2; dq3] );


v_com_world = [v1_c, v2_c, v3_c];

%% YOUR CODE END %%%
```

5b) [1 pt] Compute the angular velocities of each link expressed in the corresponding body frame

Hint: compute angular velocities via stacking and angular velocities of all preceding axes

```
%%% YOUR CODE START %%%

z1_world = [0;0;1];
y2_world = Rz1*[0;1;0];
y3_world = (Rz1*Ry2)*[0;1;0];

w1_world = dq1*z1_world;
w2_world = w1_world + dq2*y2_world;
w3_world = w2_world + dq3*y3_world;

R01 = Rz1;
R02 = Rz1*Ry2;
R03 = Rz1*Ry2*Ry3;

w1_body = simplify( R01.' * w1_world );
w2_body = simplify( R02.' * w2_world );
w3_body = simplify( R03.' * w3_world );


w_body = [w1_body, w2_body, w3_body];

%% YOUR CODE END %%%
```

5c) [2 pts] Compute the kinetic energy and potential energy

```
N = 3;
M = {};  % link mass
I = {};  % link inertia matrix in body frame
% Retrive mass and inertia for each link
for idx = 1:N
```

```
    [mass, ~, inertia] = mcI(robot.I{idx});
    M{idx} = mass;
    I{idx} = inertia;
end

%%% YOUR CODE START %%%

syms g real
T = 0;   % kinetic energy
V = 0;   % potential energy

for i = 1:N
    m = M{i};
    Ic = I{i};

    vi = v_com_world(:,i);
    T = T + 0.5*m*(vi.'*vi);

    wi = w_body(:,i);
    T = T + 0.5*(wi.'*Ic*wi);

    pi = eval(['p' num2str(i) 'c']);
    V = V + m*g*pi(3);
end

T = simplify(T);
V = simplify(V);

%%% YOUR CODE END %%%
```

5d) [1 pt] Compute the mass matrix

```
H = []; % (3, 3) mass matrix
%%% YOUR CODE START %%%

L = T - V;
q = [q1; q2; q3];
H = sym(zeros(3,3));

for i = 1:3
    for j = 1:3
        H(i,j) = diff(diff(T, dq(i)), dq(j));
    end
end

H = simplify(H);

%% YOUR CODE END %%%

q_val = [0; 0; 0];
dq_val = [0; 0; 0];
```

```
[H_gt, bias_gt] = HandC(robot, q_val, dq_val);

H_val = double(subs(H, [q, l], [q_val, l_val]));

error_H = norm(H_val - H_gt)
```

```
error_H =
1.3878e-17
```

```
assert(error_H < 1e-7, "Mass matrix error!")
```

5e) [1 pt] Compute the gravitational vector

```
G = []; % (3, 1) generized gravity
%%% YOUR CODE START %%%

q = [q1; q2; q3];
for i = 1:3
    G(i) = diff(V, q(i));
end
G = simplify(G);

%%% YOUR CODE END %%%
```

Helper Functions

```
function R = rot(th, a)
    % Create coordinate rotation matrix
    c = cos(th);
    s = sin(th);
    if a == 'x'
        R = [ 1  0  0;
              0  c  -s;
              0  s  c];
    elseif a == 'y'
        R = [ c  0  s;
              0  1  0;
             -s  0  c];
    elseif a == 'z'
        R = [ c -s  0;
              s  c  0;
              0  0  1];
    else
        disp('specify rotation axis\n')
    end
end
```