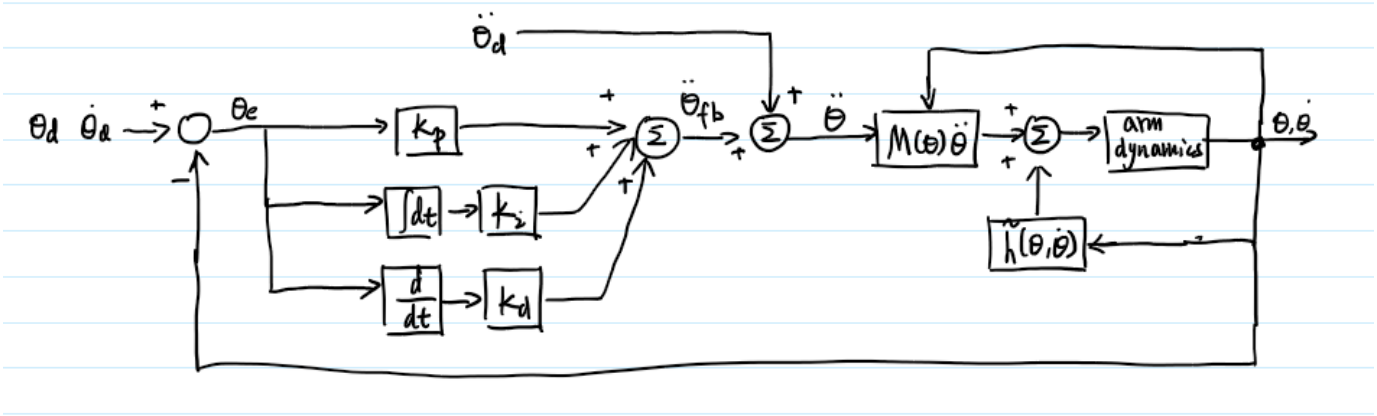
Name: Yusu (Cosmo) Zhao

Student Number: 20761282

PartA:

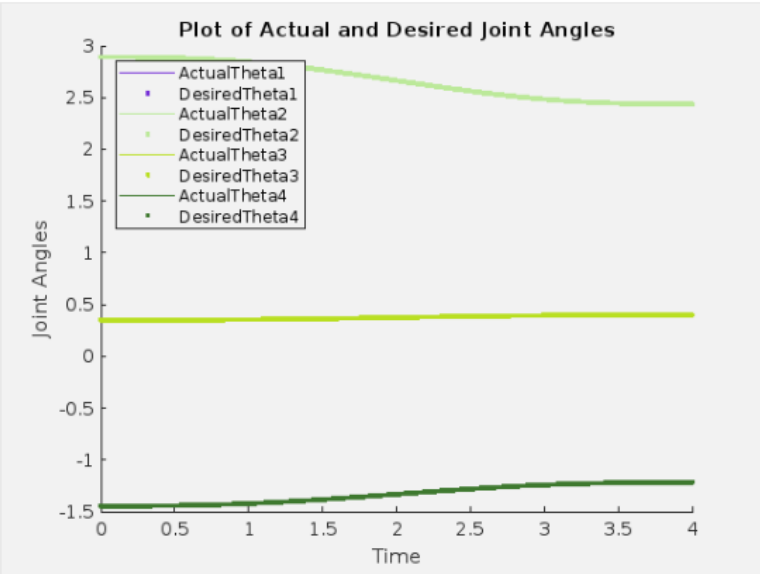
1.

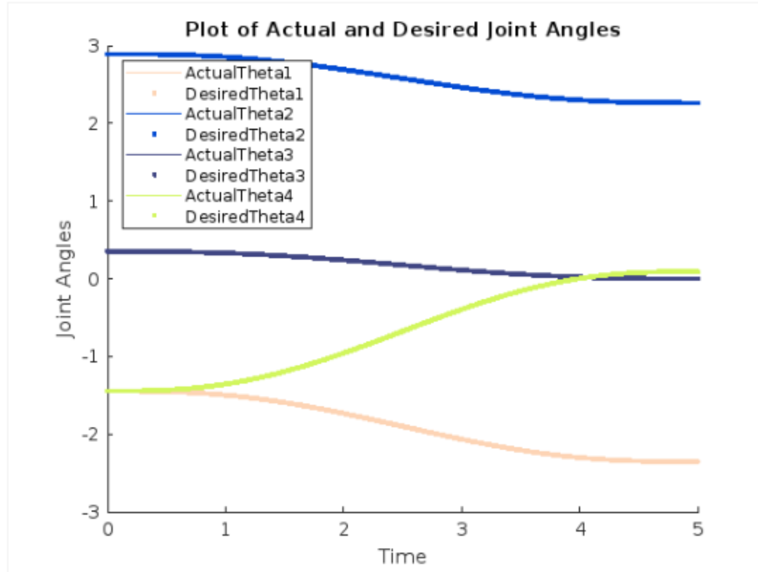


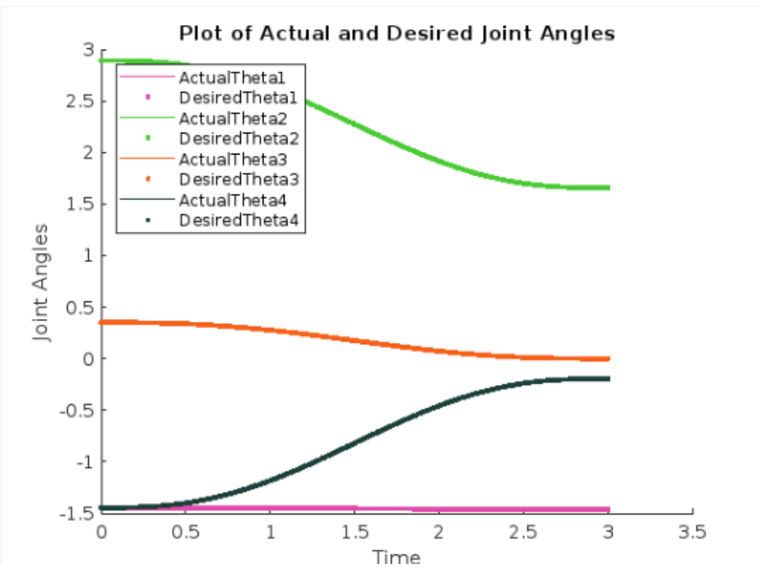
2.

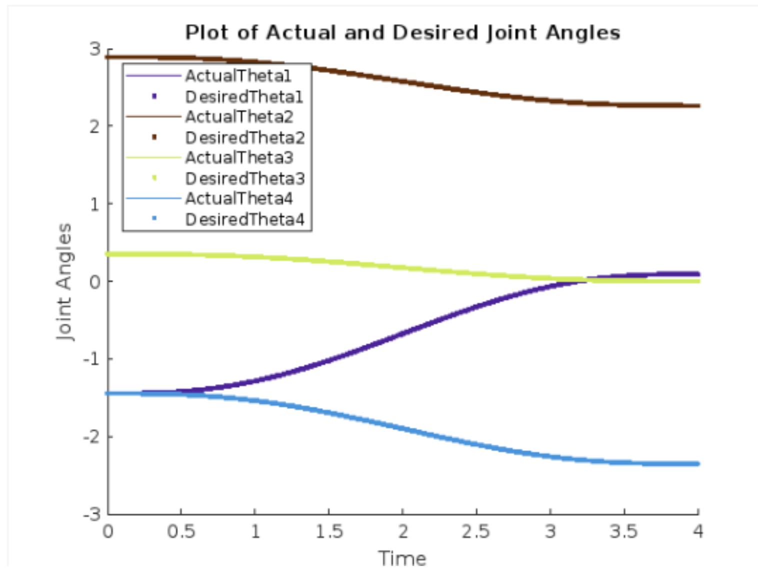
The parameter I chosed is K_p = 1600, K_d = 80, K_i = 0. I chose to have the 2% settling time to be 0.1 second, so the time constant becomes -1/40, and the roots are located at -40. So we get the equation (s+40)^2 and expand it, try to equal with the standard form s^2 + K_d s + K_p, and we get the values above. Then for the imperfect model case, those values will cause error by reaching the limitation, so I choose to take a longer settling time 0.2, and then added intergrator to remember the error. However, there are still some steady state error, so then I boost the K_p term to eliminate it. The final values are K_p = 800, K_d = 40, K_i = 2.

3.









PartB:

1. The code is simple. I copy-pasted most of the parts (robot designs and functions) from my previous part, and set the gains as the value I get from part A. These part can be used to actually design a controller in practice, or just as a test model to give some initial hints and then we can start to tuning the real robot based on the simulated value.

```
% Add the MATLAB library provided by http://modernrobotics.org
unzip('mr.zip');
ikFun = @inverse_kinematics;
genTraj = @generate_trajectory;

% Constants:
% T_sh - home pose of the paddle (relative to the fixed frame)
% g - acceleration due to gravity (relative to the fixed frame)
```

```
R_sh = [0 1 0; 0 0 1; 1 0 0]';
p_sh = [0.25 0 0.25]';
T_sh = RpToTrans(R_sh, p_sh);
g = [0; 0; -9.8];

% Robot Design (copied from Section 3: Design or your revised design)
% Define joint_types, joint_limits, B_list, S_list, and M
joint_types = ['R','R','P','R']; % the types of joints you have using a row vector of 'R' and
joint_limits = [[-pi;pi],[0;pi],[0;1],[-pi;pi]]; % must have the dimensions [2, n_joints]
S_list = [[0;0;1;0;0;0],[0;0;1;0;-1;0],[0;0;0;0;0;1],[0;0;1;0;-2;0]]; % must have the dimensior
B_list = [[0;1;0;2;0;0],[0;1;0;1;0;0],[0;0;0;0;1;0],[0;1;0;0;0;0]]; % must have the dimensions
M = [[0,0,1,2];[1,0,0,0];[0,1,0,-0.1];[0,0,0,1]]; % must be in SE(3)
speed_limits(joint_types == 'P') = 0.5;
speed_limits(joint_types == 'R') = 2;
n_joints = length(joint_types);

% Robot Dynamic Parameters
% Define M_list and G_list
M01 = vertcat(horzcat(eye(3), [0.5;0;-0.2]), [0 0 0 1]); % must be a 4x4 homogeneous transforma
M12 = vertcat(horzcat(eye(3), [1;0;0]), [0 0 0 1]); % must be a 4x4 homogeneous transformation
M23 = vertcat(horzcat(eye(3), [0.5;0;-0.5]), [0 0 0 1]); % must be a 4x4 homogeneous transforma
M34 = vertcat(horzcat(eye(3), [0;0;0.55]), [0 0 0 1]);
M45 = vertcat([[0;1;0], [0;0;1], [1;0;0], [0;0;0.05]], [0 0 0 1]);

M_list = cat(3, M01, M12, M23, M34, M45); % concatenates the matrices into a list of n+1 matric

G1 = blkdiag(diag([0.0005, 10.003/12, 10.003/12]) , 11 * eye(3)); % must be a 6x6 matrix with a
G2 = blkdiag(diag([0.0005, 10.003/12, 10.003/12]) , 11 * eye(3));
G3 = blkdiag(diag([10.003/12, 10.003/12, 0.0005]) , 11 * eye(3));
G4 = blkdiag(diag([0.0103/12, 0.0103/12, 0.00005]) , 1 * eye(3));

G_list = cat(3, G1, G2, G3, G4); % concatenates the matrices into a list of n matrices

% Place your gain here
Kp = 400
Kd = 40
Ki = 0
% Inverse Kinematics function (copied from the IK problem in Section 4: Validation)
% NOTE: This function has been modified so you can pass it p_sb *or* T_sc
% Inputs
% p_sb - The position of the ball (in the fixed frame)
% OR
% T_sc - The desired contact pose of the paddle (in the fixed frame)
% Outputs
% T_sc - The desired contact pose of the paddle (in the fixed frame)
% solution_list - A matrix of all valid IK solutions where each row is a list of joint position
function [T_sc, solution_list] = inverse_kinematics(var)
if size(var)==[4 4] & TestIfSE3(var)
    % The input variable is in SE(3), so set T_sc to the input
    T_sc = var;
```

```
else
    % The input variable is not in SE(3), so set p_sb to the input
    p_sb = reshape(var,[3,1]);

    % Generate a desired contact pose T_sc given the ball position p_sb
    % Note: Copy this section from the IK problem in Section 4: Validation
    contact_angle = 0;
    [T_sc] = make_contact(p_sb, contact_angle);
end

% Find all valid solutions of the IK problem for your robot to place the paddle in pose T_sc
% Note: Copy this section from the IK problem in Section 4: Validation
% Note: If T_sc is provided, it may be the home pose outside the cuboid
if size(var)==[4 4] & TestIfSE3(var)
    contact_angle = 0;
    p_sb = var(1:3,4);
elseif any(p_sb > [1.1;0.8;0.7] | p_sb < [0.3;-0.8;-0.1])
    solution_list = [];
    return;
end

    x = T_sc(1,4);
    y = T_sc(2,4);
    z = T_sc(3,4);
    gamma = atan2(y, x);
    alpha = acos((x.^2 + y.^2) / (2 * sqrt(x.^2 + y.^2)));
    theta1 = gamma - alpha;
    beta = acos(1 - (x.^2 + y.^2) / 2);
    theta2 = pi - beta;
    theta3 = z + 0.1;
    theta4 = - (theta1 + theta2);

    solution_list = [[theta1 theta2 theta3 theta4]]; % Each row is a valid set of joint positio
end

% Inputs
% p_sb - The position of the ball at time tc (in the fixed frame)
% t_c - The time at which the ball will be at p_sb
% freq - The sampling frequency (in Hz) used to generate your trajectory
% Outputs
% joint_traj - A matrix defining a joint trajectory were each row is a set of joint positions.
function joint_traj = generate_trajectory(p_sb, t_c, freq)
    % Find all valid configurations which contact the ball at p_sb
    [~, goal_configs] = inverse_kinematics(p_sb)
    if isempty(goal_configs)
        joint_traj = [];
        return;
    end

    % Find all valid home configurations
```

```
    T_sh = RpToTrans([0 1 0; 0 0 1; 1 0 0]', [0.25 0 0.25]');
    [~, home_configs] = inverse_kinematics(T_sh);

% Find a valid trajectory for your robot to follow to end up with the paddle at rest in pose T_
    N = t_c * freq;
    home_configs
    joint_traj = JointTrajectory(home_configs',goal_configs',t_c,N + 1,5);
end
function [T_sc] = make_contact(p_sb, theta)
% Inputs:
% p_sb - fixed ball position (relative to the fixed frame)
% theta - desired paddle contact angle about a vertical axis

% Constants:
% T_sh - home pose of the paddle (relative to the fixed frame)
R_sh = [0 0 1; 1 0 0; 0 1 0];
p_sh = [0.25; 0; 0.25];
T_sh = RpToTrans(R_sh, p_sh);

% Generate:
% p_hb - the position of the ball (relative to the home pose),
% T_sc - the contact pose (relative to the fixed frame), and
% T_hc - the motion required to move from the home pose to the contact pose.

p_hb = TransInv(T_sh) * [p_sb; 1];
p_hb = p_hb(1:3);
omg = [0; 1; 0] * theta; % Rotation about the z-axis by theta
so3mat = VecToso3(omg);  % Calculate the skew
R_hc = MatrixExp3(so3mat); % Calculate the exponential
offset = (R_hc * [0;0;1]) * 0.01;
p_hc = p_hb - offset; % 0.01m away from the ball
T_sc = T_sh * RpToTrans(R_hc, p_hc);
end
```

Question2:

```
% Add the MATLAB library provided by http://modernrobotics.org
unzip('mr.zip');
ikFun = @inverse_kinematics;
genTraj = @generate_trajectory;

% Constants:
% T_sh - home pose of the paddle (relative to the fixed frame)
% g - acceleration due to gravity (relative to the fixed frame)
R_sh = [0 1 0; 0 0 1; 1 0 0]';
p_sh = [0.25 0 0.25]';
T_sh = RpToTrans(R_sh, p_sh);
g = [0; 0; -9.8];

% Robot Design (copied from Section 3: Design or your revised design)
```

```
% Define joint_types, joint_limits, B_list, S_list, and M
joint_types = ['R','R','P','R']; % the types of joints you have using a row vector of 'R' and
joint_limits = [[-pi;pi],[0;pi],[0;1],[-pi;pi]]; % must have the dimensions [2, n_joints]
S_list = [[0;0;1;0;0;0],[0;0;1;0;-1;0],[0;0;0;0;0;1],[0;0;1;0;-2;0]]; % must have the dimensior
B_list = [[0;1;0;2;0;0],[0;1;0;1;0;0],[0;0;0;0;1;0],[0;1;0;0;0;0]]; % must have the dimensions
M = [[0,0,1,2];[1,0,0,0];[0,1,0,-0.1];[0,0,0,1]]; % must be in SE(3)
speed_limits(joint_types == 'P') = 0.5;
speed_limits(joint_types == 'R') = 2;
n_joints = length(joint_types);

% Robot Dynamic Parameters
% Define M_list and G_list
M01 = vertcat(horzcat(eye(3), [0.5;0;-0.2]), [0 0 0 1]); % must be a 4x4 homogeneous transforma
M12 = vertcat(horzcat(eye(3), [1;0;0]), [0 0 0 1]); % must be a 4x4 homogeneous transformation
M23 = vertcat(horzcat(eye(3), [0.5;0;-0.5]), [0 0 0 1]); % must be a 4x4 homogeneous transforma
M34 = vertcat(horzcat(eye(3), [0;0;0.55]), [0 0 0 1]);
M45 = vertcat([[0;1;0], [0;0;1], [1;0;0], [0;0;0.05]], [0 0 0 1]);

M_list = cat(3, M01, M12, M23, M34, M45); % concatenates the matrices into a list of n+1 matric

G1 = blkdiag(diag([0.0005, 10.003/12, 10.003/12]) , 11 * eye(3)); % must be a 6x6 matrix with a
G2 = blkdiag(diag([0.0005, 10.003/12, 10.003/12]) , 11 * eye(3));
G3 = blkdiag(diag([10.003/12, 10.003/12, 0.0005]) , 11 * eye(3));
G4 = blkdiag(diag([0.0103/12, 0.0103/12, 0.00005]) , 1 * eye(3));

G_list = cat(3, G1, G2, G3, G4); % concatenates the matrices into a list of n matrices

% Place your gain here
Kp = 800
Kd = 40
Ki = 2

% Inverse Kinematics function (copied from the IK problem in Section 4: Validation)
% NOTE: This function has been modified so you can pass it p_sb *or* T_sc
% Inputs
% p_sb - The position of the ball (in the fixed frame)
% OR
% T_sc - The desired contact pose of the paddle (in the fixed frame)
% Outputs
% T_sc - The desired contact pose of the paddle (in the fixed frame)
% solution_list - A matrix of all valid IK solutions where each row is a list of joint position
```

```
function [T_sc, solution_list] = inverse_kinematics(var)
if size(var)==[4 4] & TestIfSE3(var)
    % The input variable is in SE(3), so set T_sc to the input
    T_sc = var;
else
    % The input variable is not in SE(3), so set p_sb to the input
    p_sb = reshape(var,[3,1]);
```

```
    % Generate a desired contact pose T_sc given the ball position p_sb
    % Note: Copy this section from the IK problem in Section 4: Validation
    contact_angle = 0;
    [T_sc] = make_contact(p_sb, contact_angle);
end

% Find all valid solutions of the IK problem for your robot to place the paddle in pose T_sc
% Note: Copy this section from the IK problem in Section 4: Validation
% Note: If T_sc is provided, it may be the home pose outside the cuboid
if size(var)==[4 4] & TestIfSE3(var)
    contact_angle = 0;
    p_sb = var(1:3,4);
elseif any(p_sb > [1.1;0.8;0.7] | p_sb < [0.3;-0.8;-0.1])
    solution_list = [];
    return;
end

    x = T_sc(1,4);
    y = T_sc(2,4);
    z = T_sc(3,4);
    gamma = atan2(y, x);
    alpha = acos((x.^2 + y.^2) / (2 * sqrt(x.^2 + y.^2)));
    theta1 = gamma - alpha;
    beta = acos(1 - (x.^2 + y.^2) / 2);
    theta2 = pi - beta;
    theta3 = z + 0.1;
    theta4 = - (theta1 + theta2);

    solution_list = [[theta1 theta2 theta3 theta4]]; % Each row is a valid set of joint positio
end

% Inputs
% p_sb - The position of the ball at time tc (in the fixed frame)
% t_c - The time at which the ball will be at p_sb
% freq - The sampling frequency (in Hz) used to generate your trajectory
% Outputs
% joint_traj - A matrix defining a joint trajectory were each row is a set of joint positions.
function joint_traj = generate_trajectory(p_sb, t_c, freq)
    % Find all valid configurations which contact the ball at p_sb
    [~, goal_configs] = inverse_kinematics(p_sb)
    if isempty(goal_configs)
        joint_traj = [];
        return;
    end

    % Find all valid home configurations
    T_sh = RpToTrans([0 1 0; 0 0 1; 1 0 0]', [0.25 0 0.25]');
    [~, home_configs] = inverse_kinematics(T_sh);
```

```
% Find a valid trajectory for your robot to follow to end up with the paddle at rest in pose T_
    N = t_c * freq;
    joint_traj = JointTrajectory(home_configs',goal_configs',t_c,N + 1,5);
end
function [T_sc] = make_contact(p_sb, theta)
% Inputs:
% p_sb - fixed ball position (relative to the fixed frame)
% theta - desired paddle contact angle about a vertical axis

% Constants:
% T_sh - home pose of the paddle (relative to the fixed frame)
R_sh = [0 0 1; 1 0 0; 0 1 0];
p_sh = [0.25; 0; 0.25];
T_sh = RpToTrans(R_sh, p_sh);

% Generate:
% p_hb - the position of the ball (relative to the home pose),
% T_sc - the contact pose (relative to the fixed frame), and
% T_hc - the motion required to move from the home pose to the contact pose.

p_hb = TransInv(T_sh) * [p_sb; 1];
p_hb = p_hb(1:3);
omg = [0; 1; 0] * theta; % Rotation about the z-axis by theta
so3mat = VecToso3(omg);  % Calculate the skew
R_hc = MatrixExp3(so3mat); % Calculate the exponential
offset = (R_hc * [0;0;1]) * 0.01;
p_hc = p_hb - offset; % 0.01m away from the ball
T_sc = T_sh * RpToTrans(R_hc, p_hc);
end
```