Introduction to Robotics

Lab 5b

Shaaf Farooque 08405, Mysha Zulfiqar 08443

Task 5.5 Desired end-effector velocity (10 points)

Formulate an expression to determine the desired end-effector velocity, ${}^{s}v_{e}$, for straight-line motion, given the current position of the origin of our end-effector frame in the fixed frame, ${}^{s}p_{fk}$, as determined by our forward kinematics mapping and the desired position, ${}^{s}p_{pg}$, as determined by ${}^{s}T_{pg}$.

Provide a MATLAB `function ve = makeVE(p_cur,p_des,speed)` that accepts the current position, the desired position, and the linear speed as arguments and returns the desired end-effector velocity.

```
function ve = makeVE(p_cur, p_des, speed)
    direction = p_des - p_cur;
    distance = norm(direction);
    if distance < 1e-6 %Avoid division by zero
    ve = [0; 0; 0];
    else
        ve = min(speed / distance, 1) * direction; %ensure scaling is not larger than 1
    end
end
```

```
p_cur = [1;2;3];
p_des = [4;5;6];
speed = 1;
ve = makeVE(p_cur,p_des,speed)
```

```
ve = 3×1
    0.5774
    0.5774
    0.5774
```

Task 5.6 Desired joint angles (20 points)

Provide a MATLAB `function q = makeQ(ve,q0,dt,T)` that applies the RRMC algorithm to determine the desired joint angles vector, q, when it is provided a desired end-effector velocity, v_e, the current joints position vector, q_0, the time step, Δt, and the final time, T. It's a good idea to first determine the symbolic expression for g and save it to be computed at any q.

```
function q = makeQ(ve, q0, dt, T)
    N = round(T / dt);
    q = zeros(N+1, length(q0));
    q(1, :) = q0;
    [~, ~, ~, ~, Tsb] = getTsb();
    Rsb = Tsb(1:3,1:3);
    Rbs = Rsb.';
    Jb = getBodyJacobian();
    syms theta_1 theta_2 theta_3 theta_4
    for k = 1:N
        theta1 = q(k, 1);
        theta2 = q(k, 2);
        theta3 = q(k, 3);
        theta4 = q(k,4);
        Jb_val = double(subs(Jb, [theta_1 theta_2 theta_3 theta_4], [theta1 theta2 theta3 theta4]));
        Jv = Jb_val(4:6, 1:3);
        J_pinv = pinv(Jv);
        q_dot = J_pinv * subs(Rbs,[theta_1 theta_2 theta_3 theta_4],[theta1 theta2 theta3 theta4]) * ve;
        q_dot = [q_dot; 0];
        q(k+1, :) = q(k, :) + (q_dot' * dt);
    end
end
```

```
q0 = [0 0 0 0];
dt = 0.1;
T = 1;
ve = makeVE([0; 0; 45.2], [0; -20; 25.2], 1);
q = makeQ(ve, q0, dt, T)
```

q = 11×4

	1	2	3	4
1	0	0	0	0
2	0	0.0016	0.0011	0
3	0	-6.2549	9.6084	0
4	0	-6.2827	9.6174	0
5	0	-6.3100	9.6262	0
6	0	-6.3368	9.6348	0
7	0	-6.3631	9.6433	0
8	0	-6.3891	9.6518	0

Task 5.7 Communicating with motors (10 points)

Provide a MATLAB `function errorCode = setPosition(jointAngles)` that accepts joint angles of Phantom X Pincher as argument, and sets them as goal positions for the respective motors in the arm. The function should be properly commented, especially the error codes should be explained in detail.

- The `jointAngles` vector contains servo joint angles, in order from the base to the wrist. The angles should either be in radians or angles.
- Remember that the library method `arb.setpos` expects angles in radians.
- The angle limits for the motors are $[-150°, 150°]$, and your function should output an error and stop execution, if a provided joint angle is outside this limit.
- You have freedom to choose complexity of the error reporting system. It could be as simple as `errorCode=0` if the instructions are being executed by the motors, and `errorCode=1`, if they're not.

```
function errorCode = setPosition(jointAngles)
    lower_limit = deg2rad(-150); % Lower limit
    upper_limit = deg2rad(150); % Upper limit
    % returns 1 if any of the joints exceed the limits
    for i = 1:length(jointAngles)
        if jointAngles(i) < lower_limit || jointAngles(i) > upper_limit
            errorCode = 1;
            return;
        end
    end
    errorCode = 0;
end
```

Task 5.8 Executing straight-line motion (20 points)

Write a MATLAB function `function errorCode = pickObject(pose_obj)` that moves the end-effector from its current configuration to the pre-grasp pose in a straight-line motion. Be mindful of the following:

- Choose a small value for the speed of motion first, e.g. 0.01 m/s, and then gradually increase it. This is because we have not yet derived the maximum possible end-effector speed from the maximum possible joint speeds.
- Choose a reasonably big timestep first, e.g. $\Delta t = 3s$, and then test the effects of decreasing it. If the timestep is too small, then you're sending frequent updates to the Arbotix controller and clogging bandwidth and compute; if the timestep is too big, then the joints slow down to reach q_k and then again accelerate towards q_{k+1}, resulting in non-smooth or jerky motion.
- The task requires commands to be sent to the Arbotix controller at a fixed real rate. The MATLAB object `rateControl` can be used for such loops.
- Verify that no joint angle, θ_i, in the generated joint angles trajectory, q, exceeds the angular bounds of the servo motors. Also, verify that the joint trajectory is not passing through singularities. You may have to plot q.
- We've not yet implemented a collision-checker and the generated joint trajectory could result in collisions with the objects in the workspace, e.g. the base board, or in self-collisions, i.e. a link colliding with another link. Be alert and immediately unplug the arm if it is heading towards a collision.

```
function errorCode = pickObject(pose_obj, arb)
   p_dest = pose_obj %+ [0 0 5]; %pre grasp pose 5cm above final pose
   IK = [-0.8948 1.3346 0.5471 1.2425 1.8868]; % Manually getting destination joint angles for now
   jointAngles = []; % Use a function here to calculate joint angles from pose_obj
   jointAngles = IK(1:4);
   errorCode = setPosition(jointAngles);
   if errorCode ~= 0
       disp("Error.");
       return;
   end
   angles_cur = arb.getpos(); % To get current joint angles
   [x,y,z,~] = pincherFK(angles_cur(1:4)); % Get X Y Z from current joint angles
   p_cur = [x,y,z];
   speed = 1; % linear speed, can be set as per requirements or as an arguement of the function
   ve = makeVE(p_cur, p_dest, speed);
   ve = ve.';
   dt = 0.1; % Time step, can be taken as function input
   T = round(norm(p_dest-p_cur)/speed,3); % Total time, can be taken as function input
   q = makeQ(ve, angles_cur(1:4), dt, T); % Compute joint angles over time
   for i = q(:,3)
       if i == 0
           disp('Passing through singularity')
           errorCode = 1;
           return;
       end
   end
   disp(q(end,:));
   rate = rateControl(10);
   reset(rate);
   for j = 1:length(q)
       arb.setpos([q(j,1) q(j,2) q(j,3) q(j,4) 0],[100 100 100 100 100])
       waitfor(rate);
   end
end
```