```
function [contact_pts_O, contact_pts_P,msg] = getFingerContactPt(joint_angles,joint_axis,q_joint,joint_types,body_config,body_consts,base_pts,elli
addpath('/Users/guoyongxin/Desktop/Assignment_Academics/Senior_Second semester/MEC529/Myfunctions');

% Contact_pts_O is w.r.t Object frame {O}, and contact_pts_P is w.r.t
% Palm frame {P}.
% Body_configs is the transformation matrix of the ellipsoid.
% body_consts is the constant length of the links for three fingers.
% joint_angles stores 2 angles for thumb, and 3 angles for 2 3-link
% fingers respectively.
% ellipsoid_para = [a;b;c].

% assign config.
T_OToP = body_config;

% assign lengths
l1_1 = body_consts(1);
l2_1 = body_consts(2);
l1_2 = body_consts(3);
l2_2 = body_consts(4);
l3_2 = body_consts(5);
l1_3 = body_consts(6);
l2_3 = body_consts(7);
l3_3 = body_consts(8);
% assign base coordinate for three fingers.
P_b1 = base_pts(:,1); % base point for thumb
P_b2 = base_pts(:,2); % base point for F1
P_b3 = base_pts(:,3); % base point for F2
% assign joint types. They are all revolute joints R.
thumbType = joint_types(1:2);
F1Type = joint_types(3:5);
F2Type = joint_types(6:8);
% assign q axis
q_1 = q_joint(:,1:2); % q for thumb.
q_2 = q_joint(:,3:5); % q for F1.
q_3 = q_joint(:,6:8); % q for F2.
% assign joint axis
w_1 = joint_axis(:,1:2); % w for thumb
w_2 = joint_axis(:,3:5); % w for F1
w_3 = joint_axis(:,6:8); % w for F2
% assign joint angles
thumbAngle = joint_angles(1:2);
F1Angle = joint_angles(3:5);
F2Angle = joint_angles(6:8);

% initial config for three fingers, gst0_i, for i=1...3
I = eye(3);
z = zeros(1,3);
P_1 = P_b1 + [0;0;l1_1+l2_1];
P_2 = P_b2 + [0;0;l1_2+l2_2+l3_2];
P_3 = P_b3 + [0;0;l1_3+l2_3+l3_3];
gst0_1 = [I,P_1;z,1];
gst0_2 = [I,P_2;z,1];
gst0_3 = [I,P_3;z,1];

% compute forward kinematics
gst_theta_1 = manipdkin(gst0_1,w_1,q_1,thumbType,thumbAngle);
gst_theta_2 = manipdkin(gst0_2,w_2,q_2,F1Type,F1Angle);
gst_theta_3 = manipdkin(gst0_3,w_3,q_3,F2Type,F2Angle);

% compute contact_pts expressed in frame {P}
origin_FingerTip = [0;0;0;1];
% get the position vector by multiplying by orgin at finger tip. It is like extracting P from gst_theta.
contact_pts_1_P = gst_theta_1 * origin_FingerTip;
contact_pts_2_P = gst_theta_2 * origin_FingerTip;
contact_pts_3_P = gst_theta_3 * origin_FingerTip;
contact_pts_P_homo = [contact_pts_1_P,contact_pts_2_P,contact_pts_3_P]; % homogenous representation
contact_pts_P = contact_pts_P_homo(1:3,:); % without homogenous representation.

% compute contact_pts expressed in frame {O}
T_OToP_Inv = getTransInv(T_OToP); % get inverse T matrix.
contact_pts_O_homo = zeros(4,3);
contact_pts_O = zeros(3,3);
msg = ""; % initialize the message string.

for i = 1:3
    % expressed in {O} frame
    contact_pts_O_homo(:,i) = T_OToP_Inv * contact_pts_P_homo(:,i); % homogenous representation.
    contact_pts_O(:,i) = contact_pts_O_homo(1:3,i); % without homogenous representation.
    % check if the finger is touching the object or not.
    RHS = 0; % initialize right-hand side of the ellipsoid surface equation.
    for j = 1:3 % index for x, y and z component of the contact point in frame {O}
        RHS = RHS + (contact_pts_O(j,i)/ellipsoid_para(j))^2;
    end

    if RHS ~= 1 % doesn't satisfy the ellipsoid surface equation.
        switch i
            case 1 % thumb
                msg = msg + "Thumb (finger 1): No Contact!" + newline;
            case 2 % F1 finger
                msg = msg + "F1 (finger 2): No Contact!" + newline;
            otherwise % F2 finger
                msg = msg + "F2 (finger 3): No Contact!" + newline;
        end
    else % satisfy the ellipsoid surface equation.
        switch i
            case 1 % thumb
```

```
                msg = msg + "Thumb (finger 1): Contact!" + newline;
            case 2 % F1 finger
                msg = msg + "F1 (finger 2): Contact!" + newline;
            otherwise % F2 finger
                msg = msg + "F2 (finger 3): Contact!" + newline;
        end
    end

end


end
```

```
Not enough input arguments.

Error in getFingerContactPt (line 13)
T_OToP = body_config;
```

Published with MATLAB® R2018b