Table of Contents

Function: src/rotateTriangulationX.m

```
function TRG = rotateTriangulationX(TRG, theta)

    % Create the rotation matrix
    R = [1 0 0; 0 round(cosd(theta), 15) -round(sind(theta), 15); 0
 round(sind(theta), 15) round(cosd(theta), 15)];
    % Rotate the points in the triangulation object
    Points = TRG.Points * R;

    % Create a new triangulation object with the rotated points and the same
 connectivity list
    TRG = triangulation(TRG.ConnectivityList, Points);

end
```

Function: src/moveTriangulationZ.m

```
function TRG = moveTriangulationZ(TRG, depth)

    minZ = min(TRG.Points(:, 3));
    Points = TRG.Points;
    Points(:, 3) = Points(:, 3) - minZ - depth*1000;

    % Create a new triangulation object with the shifted points and the same
 connectivity list
    TRG = triangulation(TRG.ConnectivityList, Points);

end
```

Function: src/getStlData.m

```
function [points, normals, areas, depth_list] = getStlData(TRG, Points, List)

    points = (incenter(TRG)').';
    normals = (faceNormal(TRG)').';
    a = vecnorm(Points(:, List(1,:)) - Points(:, List(2,:)));
    b = vecnorm(Points(:, List(2,:)) - Points(:, List(3,:)));
    c = vecnorm(Points(:, List(1,:)) - Points(:, List(3,:)));
    s = (a + b + c) / 2;
    areaarray = sqrt(s .* (s - a) .* (s - b) .* (s - c));

    areas = areaarray.' / 1000000;                          % mm2 to m2
    depth_list = points(:,3) / 1000;                        % mm to m

end
```

Function: src/calcVelocity.m

```
function [movement, movement_normalized] = calcVelocity(points,
 direction_vector, linear_velocity, rotation, angular_velocity, threshold)

    n_elements = size(points, 1);

    vcor = linear_velocity .* direction_vector .* 1000;
    movement = ones(n_elements,1) .* vcor ;

    if rotation
        r_list = [points(:,1) points(:,2) points(:,3) + ones(n_elements,1) .*
 100];
        v_sum = cross(ones(n_elements,1) .* angular_velocity, r_list) +
 movement;
        movement= round(v_sum, 15);
    end

    movement_normalized = movement ./ vecnorm(movement, 2, 2);

    movement(abs(movement) < threshold) = 0;
    movement_normalized(abs(movement_normalized) < threshold) = 0;

end
```

Function: src/checkConditions.m

```
function [include, normals_include, movement_normalized_include,
 movement_include, areas_include, points_include, depth_list_include] =
 checkConditions(points, normals, areas, movement, movement_normalized,
 depth_list, unit_test, threshold)

    is_leading_edge = dot(normals, movement_normalized, 2) >= -threshold;
    is_intruding = points(:,3) < 0;
    include = is_leading_edge & is_intruding;

    normals_include = normals(include,:);
    movement_normalized_include = movement_normalized(include,:);
    movement_include = movement(include,:);
    areas_include = areas(include,:);
    points_include = points(include,:);
    depth_list_include = depth_list(include,:);

    if unit_test
    pointsTest = [90 180];
    normals_include = normals_include(pointsTest,:);
    movement_normalized_include = movement_normalized_include(pointsTest,:);
    areas_include = areas_include(pointsTest,:);
    points_include = points_include(pointsTest,:);
    depth_list_include = depth_list_include(pointsTest,:);
    end

end
```

Function: src/findLocalFrame.m

```
function [z_local, r_local, theta_local] = findLocalFrame(normals_include,
 movement_include, movement_nn_include, include, unit_test)

    if unit_test
    z_local = repmat([0,0,1], numel(depth_list_include), 1);
    else
    z_local = repmat([0,0,1], sum(include), 1);
    end

    r_local = zeros(size(normals_include,1),3);
    for i = 1:size(normals_include,1)
    if (vecnorm(movement_include(i,:) - dot(movement_include(i,:),
 z_local(i,:), 3) .* z_local(i,:), 2, 2) == 0 && vecnorm(normals_include(i,:)
 - dot(normals_include(i,:), z_local(i,:), 3) .* z_local(i,:), 2, 2) == 0)
    r_local(i,:) = [1; 0; 0];
    elseif (vecnorm(movement_nn_include(i,:) - dot(movement_nn_include(i,:),
 z_local(i,:), 3) .* z_local(i,:), 2, 2) == 0 && vecnorm(normals_include(i,:)
 - dot(normals_include(i,:), z_local(i,:), 3) .* z_local(i,:), 2, 2) ~= 0)
    r_local(i,:) = (normals_include(i,:) - dot(normals_include(i,:),
 z_local(i,:), 3) .* z_local(i,:)) ./ vecnorm(normals_include(i,:) -
 dot(normals_include(i,:), z_local(i,:), 3) .* z_local(i,:), 2, 2);
    else
    r_local(i,:) = (movement_nn_include(i,:) - dot(movement_nn_include(i,:),
 z_local(i,:), 3) .* z_local(i,:)) ./ vecnorm(movement_nn_include(i,:) -
 dot(movement_nn_include(i,:), z_local(i,:), 3) .* z_local(i,:), 2, 2);
    end
    end

    theta_local = cross(z_local, r_local, 2);

end
```

Function: src/findAngles.m

```
function [beta, gamma, psi] = findAngles(normals_include,
 movement_normalized_include, r_local, z_local, theta_local)

    % beta - surface characteristic angle
    beta = zeros(size(normals_include,1),1);
    for i = 1:size(normals_include,1)
    if (dot(normals_include(i,:),r_local(i,:), 2) >= 0) &&
 (dot(normals_include(i,:),z_local(i,:), 2) >= 0)
        beta(i) = - round(acos(dot(normals_include(i,:),z_local(i,:), 2)),
 15);
    elseif  (dot(normals_include(i,:),r_local(i,:), 2) >= 0) &&
 (dot(normals_include(i,:),z_local(i,:), 2) < 0)
        beta(i) = +pi - round(acos(dot(normals_include(i,:),z_local(i,:), 2)),
 15);
    elseif  (dot(normals_include(i,:),r_local(i,:), 2) < 0) &&
 (dot(normals_include(i,:),z_local(i,:), 2) >= 0)
        beta(i) =     + round(acos(dot(normals_include(i,:),z_local(i,:), 2)),
 15);
    else
        beta(i) = -pi + round(acos(dot(normals_include(i,:),z_local(i,:), 2)),
 15);
    end
    end

    % gamma - velocity characteristic angle
    gamma = zeros(size(movement_normalized_include,1),1);
    for i = 1:size(movement_normalized_include,1)
    if dot(movement_normalized_include(i,:), z_local(i,:), 2) <= 0
    gamma(i) = round(acos(dot(movement_normalized_include(i,:), r_local(i,:),
 2)), 15);
    else
    gamma(i) = -round(acos(dot(movement_normalized_include(i,:), r_local(i,:),
 2)), 15);
    end
    end

    % psi - surface characteristic angle
    psi = zeros(size(normals_include,1),1);
    nr0_inc = zeros(size(normals_include,1),3);
    for i = 1:size(normals_include,1)
    nr0_inc(i,:) = (normals_include(i,:) -
 (dot(normals_include(i,:),z_local(i,:), 2) .* z_local(i,:))) ./
 vecnorm(normals_include(i,:) - (dot(normals_include(i,:),z_local(i,:), 2) .*
 z_local(i,:)),2,2);
    if vecnorm(normals_include(i,:) - (dot(normals_include(i,:),z_local(i,:),
 2) .* z_local(i,:)),2,2) == 0 || dot(nr0_inc(i,:),r_local(i,:),2) == 0
    psi(i) = 0;
    else
    psi(i) = round(atan( dot(nr0_inc(i,:),theta_local(i,:), 2) ./
 dot(nr0_inc(i,:),r_local(i,:), 2) ), 15);
    end
    end

end
```

Function: src/findFit.m

```
function [f1, f2, f3] = findFit(gamma, beta, psi, z_local,
 movement_normalized_include, normals_include, threshold, depth_list_include,
 include, unit_test)

    x1 = round(sin(gamma), 15);
    x2 = round(cos(beta), 15);
    x3 = round(cos(psi), 15) .* round(cos(gamma), 15) .* round(sin(beta), 15)
 + round(sin(gamma), 15) .* round(cos(beta), 15);

    y1 = dot(-z_local, movement_normalized_include,2);
    y2 = dot(-z_local, normals_include,2);
    y3 = dot(normals_include, movement_normalized_include,2);

    x1(abs(x1) < abs(threshold)) = 0;
    x2(abs(x2) < abs(threshold)) = 0;
    x3(abs(x3) < abs(threshold)) = 0;

    if unit_test
    unitx = ones(numel(depth_list_include), 1);
    else
    unitx = ones(sum(include), 1);
    end

    Tk = [unitx x1  x2  x3  x1.^2    x2.^2    x3.^2    x1.*x2  x2.*x3  x3.*x1
 x1.^3    x2.^3    x3.^3    x1.*x2.^2    x2.*x1.^2    x2.*x3.^2    x3.*x2.^2
  x3.*x1.^2  x1.*x3.^2  x1.*x2.*x3];

    c1k = [0.00212; -0.02320; -0.20890; -0.43083; -0.00259; 0.48872; -0.00415;
 0.07204; -0.02750; -0.08772; 0.01992; -0.45961; 0.40799; -0.10107; -0.06576;
 0.05664; -0.09269; 0.01892; 0.01033; 0.15120];
    c2k = [-0.06796; -0.10941; 0.04725; -0.06914; -0.05835; -0.65880;
 -0.11985; -0.25739; -0.26834; 0.02692; -0.00736; 0.63758; 0.08997; 0.21069;
 0.04748; 0.20406; 0.18589; 0.04934; 0.13527; -0.33207];
    c3k = [-0.02634; -0.03436; 0.45256; 0.00835; 0.02553; -1.31290; -0.05532;
 0.06790; -0.16404; 0.02287; 0.02927; 0.95406; -0.00131; -0.11028; 0.01487;
 -0.02730; 0.10911; -0.04097; 0.07881; -0.27519];

    f1 = Tk * c1k;
    f2 = Tk * c2k;
    f3 = Tk * c3k;

end
```

Function: src/findAlpha.m

```
function [alpha_gen, alpha_gen_n, alpha_gen_t, alpha] =
 findAlpha(normals_include, movement_normalized_include, beta, gamma, psi,
 r_local, theta_local, z_local, f1, f2, f3, mu_surf, xi_n)
```

```
    alpha_r_gen = f1 .* round(sin(beta), 15) .* round(cos(psi), 15) + f2 .*
 round(cos(gamma), 15);
    alpha_theta_gen = f1 .* round(sin(beta), 15) .* round(sin(psi), 15);
    alpha_z_gen = -f1 .* round(cos(beta), 15) - f2 .* round(sin(gamma), 15) -
 f3;

    alpha_gen = alpha_r_gen.*r_local + alpha_theta_gen.*theta_local +
 alpha_z_gen.*z_local;

    % Calculate the system specific alpha_n and alpha_t in the local
 coordinate frame
    % Correcting minor sign problems
    alpha_gen_n = zeros(size(normals_include,1),3);
    alpha_gen_t = zeros(size(normals_include,1),3);
    for i = 1:size(normals_include,1)
        if dot(alpha_gen(i,:),-normals_include(i,:),2) < 0
        alpha_gen_n(i,:) = -dot(alpha_gen(i,:),-normals_include(i,:),2) .* (-
normals_include(i,:));
        alpha_gen_t(i,:) = (alpha_gen(i,:) + alpha_gen_n(i,:));
        else
        alpha_gen_n(i,:) = dot(alpha_gen(i,:),-normals_include(i,:),2) .* (-
normals_include(i,:));
        alpha_gen_t(i,:) = (alpha_gen(i,:) - alpha_gen_n(i,:));
        end
    end

    for i = 1:size(normals_include,1)
        if dot(alpha_gen_t(i,:),-movement_normalized_include(i,:),2) < 0
        alpha_gen_t(i,:) = -(alpha_gen_t(i,:));
        end
    end

    alpha = xi_n .* (alpha_gen_n + min(mu_surf .* vecnorm(alpha_gen_n,2,2) ./
 vecnorm(alpha_gen_t,2,2),1) .* alpha_gen_t);

end
```

Function: src/getForces.m

```
function [forces, pressures, force_x, force_y, force_z, resultant] =
 getForces(depth_list_include, areas_include, alpha)

    forces = alpha .* abs(depth_list_include) .* areas_include;     % N
    pressures = forces ./ areas_include ./ 1000000;                 % N/mm²

    [force_x] = sum(forces(:,1),1);
    [force_y] = sum(forces(:,2),1);
    [force_z] = sum(forces(:,3),1);
    [resultant] = sqrt(force_x^2 + force_y^2 + force_z^2);

end
```

Function: src/getTorques.m

```
function [T, torque_x, torque_y, torque_z] = getTorques(points_include,
 depth_list_include, forces, unit_test, include)

    if unit_test
    c_null = zeros(numel(depth_list_include), 1);
    else
    c_null = zeros(sum(include),1);
    end

    c_x = [c_null points_include(:,2) points_include(:,3)];
    c_y = [points_include(:,1) c_null points_include(:,3)];
    c_z = [points_include(:,1) points_include(:,2) c_null];

    F_x = [c_null forces(:,2) forces(:,3)];
    F_y = [forces(:,1) c_null forces(:,3)];
    F_z = [forces(:,1) forces(:,2) c_null];

    T_x = cross(c_x,F_x) ./ 1000;
    T_y = cross(c_y,F_y) ./ 1000;
    T_z = cross(c_z,F_z) ./ 1000;

    T = [T_x(:,1) T_y(:,2) T_z(:,3)];

    [torque_x] = sum(T_x(:,1), 1);
    [torque_y] = sum(T_y(:,2), 1);
    [torque_z] = sum(T_z(:,3), 1);

end
```

Published with MATLAB® R2023a