Supplementary material

Matlab code for AMF file coding starting from the pressure distribution obtained with a baropodometric board

Initial part of preliminary cleaning

```
clear all
close all
```

Register Image and Numerical Values from Colormap

```
img=imread("imm.png"); % Load image containing pressure map

r=size(img,1);
c=size(img,2);

img_bordo=zeros(r,c); % Initialize matrix for borders
% Represent borders with ones and leave the rest as zeros

for i=1:r

    for j=1:c

        if((img(i,j,1)==255)&&(img(i,j,2)==255)&&(img(i,j,3)==255))
            img_bordo(i,j)=1; % If it's white, it's a border

        end
    end

end

figure;
imshow(img_bordo(:,:)) % Display the found border
```

Divide the images into left and right parts

```
img_sx=img_bordo(:,1:round(c/2));
img_dx=img_bordo(:,round(c/2):end);


figure;
subplot(1,2,1);
imshow(img_sx(:,:)) % Display the left border


subplot(1,2,2);
imshow(img_dx(:,:)) % Display the right border
```

Start registration protocol

```
% Create a point cloud of the image containing borders


r1 = 1; % Essential for the for loop


for i=1:r
    for j=1:round(c/2)


        if(img_sx(i,j)==1)
            Punti_img_sx(r1,:)=[i,j,0];
            r1=r1+1;
        end



    end



end


figure;
plot3(Punti_img_sx(:,1),Punti_img_sx(:,2),Punti_img_sx(:,3),"*")
axis equal
```

```
%% End point creation %%
```

Load the STL file corresponding to the insole

```
[TR]=stlread("patient_name.stl");  %% Import STL


% Save point and connectivity list


Punti_stl=TR.Points;
Triangoli_stl=TR.ConnectivityList;




Punti_stl(:,3)=0; % Add a third coordinate equal to 0
% Essential for good matrix operations on point clouds


plot3(Punti_stl(:,1),Punti_stl(:,2),Punti_stl(:,3),"*") % Display the generated
points


axis equal
```

Move the STL in space to prepare it for the next part

```
baricentro_stl=(sum(Punti_stl)/size(Punti_stl,1));
baricentro_img_sx=(sum(Punti_img_sx)/size(Punti_img_sx,1));



b_stl=Punti_stl-baricentro_stl;
b_img_sx=Punti_img_sx-baricentro_img_sx;




figure;
subplot(1,2,1);

plot3(b_stl(:,1),b_stl(:,2),b_stl(:,3),"*")

axis equal
```

```
subplot(1,2,2);
plot3(b_img_sx(:,1),b_img_sx(:,2),b_img_sx(:,3),"*")
axis equal
```

Start actual registration

```
% Calculate registration using the inertia matrix method and SVD decomposition
% The method is in Professor Ferrari's registration notes

I_stl = zeros(3,3); % Initialize the matrix

for i=1:size(b_stl,1) % Fill it with its values
    x=b_stl(i,1);
    y=b_stl(i,2);
    z=b_stl(i,3);

    I_stl=[((y^2)+(z^2)) -x*y -x*z; -x*y (x^2+z^2) -y*z; -x*z -y*z (x^2+y^2)];

end

% Do the same thing for the other set of points

I_imgsx=zeros(3,3);

for i=1:size(img_sx,1)
    x=img_sx(i,1);
    y=img_sx(i,2);
    z=img_sx(i,3);

    I_imgsx=[((y^2)+(z^2)) -x*y -x*z; -x*y (x^2+z^2) -y*z; -x*z -y*z (x^2+y^2)];

end

[U,S,V] = svd(b_stl'*b_stl);
```

```
[U1,S1,V1] = svd(b_img_sx'*b_img_sx);


if(det(U)<0)
    U(3,3)=-1*U(3,3);
end


if(det(U1)<0)
    U1(3,3)=-1*U1(3,3);
end


R=U1*U';
save('matrice_registrazione.mat','R');
P2=b_img_sx*R;




% Plot the result


figure;
hold on
plot3(b_stl(:,1),b_stl(:,2),b_stl(:,3),"*")
xlabel('X')
ylabel('Y')
axis equal


plot3(P2(:,1),P2(:,2),P2(:,3),"*")
hold off


% So I have a rotation of R and translation of -b_img_sx
```

From colormap to numeric value

(This section is used to extract numeric values from color intensity)

```
Mappa=imread("Mappa.png");
Mappa=Mappa(1,:,:);
```

```
Mappa(1,end+1,1)=255;% Correction due to the low resolution of the image
massimo=10;
minimo=1;


img_sx_mappato=zeros(size(img,1),size(img,2));


soglia=0.1;


for i=1:size(img,1)
    for j=1:size(img,2)


        if((img(i,j,1)~=img(i,j,2))&&(img(i,j,2)~=img(i,j,3)))


            for k=1:size(Mappa,2)
                r=abs(Mappa(1,k,1)-img(i,j,1));
                g=abs(Mappa(1,k,2)-img(i,j,2));
                b=abs(Mappa(1,k,3)-img(i,j,3));


                if((r<soglia)&&(g<soglia)&&(b<soglia))
                    img_sx_mappato(i,j)=k;


                end


            end




        end



    end
```

```
end

figure;

imshow((img_sx_mappato(:,:)))

caxis([0 max(max(img_sx_mappato)')]);

colorbar;

colormap jet


save('Mappa_pressioni.mat','img_sx_mappato');



% Show on screen the colormap used to estimate values
Mappacolore=[Mappa;Mappa;Mappa;Mappa;Mappa;Mappa;Mappa;Mappa;Mappa;Mappa;Mappa];

Mappacolore=[Mappacolore;Mappa;Mappa;Mappa;Mappa;Mappa;Mappa;Mappa;Mappa;Mappa;Mapp
a];


% Plot components of color in the colormap
imshow(Mappacolore(:,:,:))

figure;

hold on


plot(1:size(Mappa,2),Mappa(1,:,1),'Color','RED');
plot(1:size(Mappa,2),Mappa(1,:,2),'Color','green')

plot(1:size(Mappa,2),Mappa(1,:,3),'Color','blue')

hold off
```

Shapetostl

Convert from the description of a surface in space to a description of a volume in space. This part is necessary since we were given an STL that is not useful for printing, and we need to repair it.

```
[TR]=stlread("patient_name.stl");  % Import the STL


% Save point and connectivity variables
Punti=TR.Points;

Triangoli=TR.ConnectivityList;
```

```
% Plot the surface with its normals
Grafica(Triangoli,Punti)
```

Algorithm for smoothing

Reference : https://it.mathworks.com/matlabcentral/fileexchange/77004-mesh-processing-toolbox?focused=51c47026-b470-4984-ae22-cf52d1e3444d&tab=example

```
boundaries = detect_mesh_holes_and_boundary(Triangoli);



nb_iterations = 10;
ngb_degre = 6;

V_out = smooth_mesh_boundaries(Punti,boundaries,nb_iterations,ngb_degre);



Grafica(Triangoli,V_out) % Function present at the end of the script
```

Project

Create my volume

```


Proiezione=Punti;
Proiezione_M=Proiezione;

Proiezione(:,3)=0;

Proiezione_M(:,3)=Proiezione_M(:,3)/2;

Punti2=[Punti;Proiezione];
```

Save the new STL

```
shp = alphaShape(Punti2);

a = criticalAlpha(shp,'one-region');

shp = alphaShape(Punti2,a);

plot(shp)

[T, P] =  alphaTriangulation(shp);

TR=triangulation(T,P);

[F,P] = freeBoundary(TR);

TR=triangulation(F,P);
```

```
stlwrite(TR,"create_mesh_from_shape.stl")
```

Modify STL

```
[TR]=stlread("create_mesh_from_shape.stl");  % Import the STL


% Save point and connectivity variables
Punti=TR.Points;

Triangoli=TR.ConnectivityList;


Grafica(Triangoli,Punti)


nb_iterations = 2;
ngb_degre = 2;

V = mesh_smooth(Punti,Triangoli,nb_iterations,ngb_degre);

Grafica(Triangoli,V)

TR=triangulation(Triangoli,V);

stlwrite(TR,"Modeladapted.stl")
```

From colormap to point cloud

Convert colormap data to numeric values

```
load('Mappa_pressioni.mat');


imshow((img_sx_mappato(:,:)))


caxis([0 max(max(img_sx_mappato)')]);
colorbar;

colormap jet


% Take the identical left foot for the right foot


ass_sx=img_sx_mappato(:,1:round(size(img_sx_mappato,2)/2));
imshow((ass_sx(:,:)))

caxis([0 max(max(ass_sx)')]);
```

```
colorbar;
colormap jet
```

Data densification and Gaussian filter

```
ass_sx_fill=zeros(size(ass_sx,1),size(ass_sx,2));
for i=2:(size(ass_sx,1)-1)

    for j=2:(size(ass_sx,2)-1)

        Voxel=[ass_sx(i-1,j-1), ass_sx(i-1,j),ass_sx(i,j-1), ass_sx(i,j),
ass_sx(i+1,j), ass_sx(i,j+1), ass_sx(i+1,j-1), ass_sx(i-1,j+1), ass_sx(i+1,j+1)];



        if(max(Voxel)>0)
            ass_sx_fill(i,j)=round(median(nonzeros(Voxel)));



        end



    end



end



imshow((ass_sx_fill(:,:)))
caxis([0 max(max(ass_sx_fill)')]);
colorbar;
colormap jet


Num_iteraz=1;
```

```
if(Num_iteraz>0)
    ass_sx=ass_sx_fill;


    for it=1:Num_iteraz
        for i=2:(size(ass_sx_fill,1)-1)


            for j=2:(size(ass_sx_fill,2)-1)


                Voxel=[ass_sx(i-1,j-1), ass_sx(i-1,j),ass_sx(i,j-1), ass_sx(i,j),
ass_sx(i+1,j), ass_sx(i,j+1), ass_sx(i+1,j-1), ass_sx(i-1,j+1), ass_sx(i+1,j+1)];



                if(max(Voxel)>0)
                    ass_sx_fill(i,j)=round(median(nonzeros(Voxel)));



                end




            end



        end


    end


end
imshow((ass_sx_fill(:,:)))
caxis([0 max(max(ass_sx_fill)')]);
colorbar;
colormap jet
```

```
%S is the standard deviation of the Gaussian, which determines the size of the
kernel


S=2;


ass_sx = imgaussfilt(ass_sx,S);
imshow((ass_sx(:,:)))

caxis([0 max(max(ass_sx)')]);

colorbar;

colormap jet
```

Convert from color values to pressure values

```
valore_ass=[min(min(ass_sx)'),mean(mean(ass_sx)'),max(max(ass_sx)')]


% Minimum, mean, max


valore_press=[0,1971,3468]; %[g/cm^2]


press_sx = interp1(valore_ass,valore_press,ass_sx);


load('Matrice_riempimento_moduloelsatico.mat');
% First row: apparent elastic modulus

% Second row: associated filling
```

Definition of the number of thresholds

```




ass_linea=nonzeros(press_sx);
ass_linea=ass_linea(:);


% num => Number of thresholds


num=5




h=histogram(ass_linea,num);
```

```
numBins=h.NumBins;
[counts,edges]=histcounts(ass_linea,numBins);



for i = 1:numBins
    binValues = ass_linea(ass_linea >= edges(i) & ass_linea <= edges(i+1));

    binMatrix(i, 1:numel(binValues)) = binValues;

end

binMatrix=binMatrix';



% Matrix where the number of columns is equal to the number of bins
% and the elements on the rows are the elements within that bin



valori_medi=mean(binMatrix);
% Calculate the values at the center of each edge

valpress=[(edges(1,1)+edges(1,2))/2,(edges(1,2)+edges(1,3))/2,(edges(1,3)+edges(1,4
))/2,(edges(1,4)+edges(1,5))/2,(edges(1,5)+edges(1,6))/2]

% Calculate the mean

m_valpress=mean(valpress);
```

Selection of the basic elastic modulus

```
Emedio=40; % Average elastic modulus



Eperparte=Emedio*(valpress/m_valpress); % Elastic modulus associated with each zone
```

Load Corrected STL

```
[TR]=stlread("Modeladapted.stl");



Punti=TR.Points;
Triangoli=TR.ConnectivityList;
```

Create point cloud from left image

```
% Radius size around each point


r1=1



for i=1:size(press_sx,1)
    for j=1:size(press_sx,2)


        if(press_sx(i,j)>0)
            cloud_point_from_press(r1,:)=[i,j,press_sx(i,j)];
            r1=r1+1;
        end



    end



end



plot(cloud_point_from_press(:,1),cloud_point_from_press(:,2),"*");
axis equal


% Find centroid and perform translation


baricentro=mean(cloud_point_from_press);


cloud_point_from_press(:,1)=cloud_point_from_press(:,1)-baricentro(1);


cloud_point_from_press(:,2)=cloud_point_from_press(:,2)-baricentro(2);


plot(cloud_point_from_press(:,1),cloud_point_from_press(:,2),"*");
axis equal
```

```
%% Now perform rotation


load('matrice_registrazione.mat')


cloud_point_from_press_reg=cloud_point_from_press*R;
cloud_point_from_press_reg(:,3)=cloud_point_from_press(:,3);

plot(cloud_point_from_press_reg(:,1),cloud_point_from_press_reg(:,2),"*");

axis equal
```

Create divisions for various Infill

```
for i=1:size(cloud_point_from_press_reg,1)

    for j=1:size(binMatrix,1)

        for k=1:size(binMatrix,2)

            if(cloud_point_from_press_reg(i,3)==binMatrix(j,k))
                cloud_point_from_press_reg(i,4)=k;

            end


        end


    end


end


% Added a column to see which filling class my points belong to
```

Create the 5 filling groups

```
r1=1;
r2=1;
r3=1;
r4=1;
r5=1;


for i=1:size(cloud_point_from_press_reg,1)

    if(cloud_point_from_press_reg(i,4)==1 && 1<=num)
        infill_1(r1,1:3)=cloud_point_from_press_reg(i,1:3);
        r1=r1+1;
    end

    if(cloud_point_from_press_reg(i,4)==2 && 2<=num)
        infill_2(r2,1:3)=cloud_point_from_press_reg(i,1:3);
        r2=r2+1;
    end

    if(cloud_point_from_press_reg(i,4)==3 && 3<=num)
        infill_3(r3,1:3)=cloud_point_from_press_reg(i,1:3);
        r3=r3+1;
    end

    if(cloud_point_from_press_reg(i,4)==4 && 4<=num)
        infill_4(r4,1:3)=cloud_point_from_press_reg(i,1:3);
        r4=r4+1;
    end

    if(cloud_point_from_press_reg(i,4)==5 && 5<=num)
        infill_5(r5,1:3)=cloud_point_from_press_reg(i,1:3);
        r5=r5+1;
    end
```

```
end


%% General plot of registered points and points from the STL


figure;
plot(Punti(:,1),Punti(:,2));

axis equal

hold on


plot(cloud_point_from_press_reg(:,1),cloud_point_from_press_reg(:,2),"*");
plot(infill_1(:,1),infill_1(:,2),"*");


if( exist ("infill_2") == true)
    plot(infill_2(:,1),infill_2(:,2),"*");

end


if(exist ("infill_3") == true)
    plot(infill_3(:,1),infill_3(:,2),"*");

end


if(exist ("infill_4") == true)
    plot(infill_4(:,1),infill_4(:,2),"*");

end


if(exist ("infill_5") == true)
    plot(infill_5(:,1),infill_5(:,2),"*");

end


hold off
```

Check the extent of the area, if too few points, merge them

```
subplot(1,5,1)


plot(infill_1(:,1),infill_1(:,2),"*");
```

```
title("Infill 1")




if( exist ("infill_2") == true)
    subplot(1,5,2)
    plot(infill_2(:,1),infill_2(:,2),"*");
    title("Infill 2")
end


if( exist ("infill_3") == true)
    subplot(1,5,3)
    plot(infill_3(:,1),infill_3(:,2),"*");
    title("Infill 3")
end


if( exist ("infill_4") == true)
    subplot(1,5,4)
    plot(infill_4(:,1),infill_4(:,2),"*");
    title("Infill 4")
end


if( exist ("infill_5") == true)
    subplot(1,5,5)
    plot(infill_5(:,1),infill_5(:,2),"*");
    title("Infill 5")
end
```

Decide if Infill 1 is sufficient; if yes, good, otherwise promote to the next infill

```
infill_2=[infill_2;infill_1];




title("Infill 2")
```

```
clear infill_1
Eperparte=[Eperparte(2:end)];


for i=1:size(cloud_point_from_press_reg,1)
    if(cloud_point_from_press_reg(i,4)==1)
        cloud_point_from_press_reg(i,4)=2;
    end



end
```

Decide if Infill 2 is sufficient; if yes, good, otherwise promote to the next infill

```
infill_3=[infill_3;infill_2];


title("Infill 3")


clear infill_2
Eperparte=[Eperparte(1),Eperparte(3:end)];


for i=1:size(cloud_point_from_press_reg,1)
    if(cloud_point_from_press_reg(i,4)==2)
        cloud_point_from_press_reg(i,4)=3;
    end



end
```

Decide if Infill 3 is sufficient; if yes, good, otherwise promote to the next infill

```
infill_4=[infill_4;infill_3];


clear infill_3
```

```
Eperparte=[Eperparte(1:2),Eperparte(4:end)];


for i=1:size(cloud_point_from_press_reg,1)
    if(cloud_point_from_press_reg(i,4)==3)
        cloud_point_from_press_reg(i,4)=4;
    end



end
```

Decide if Infill 4 is sufficient; if yes, good, otherwise promote to the next infill

```
infill_5=[infill_5;infill_4];


clear infill_4
Eperparte=[Eperparte(1:3),Eperparte(5)];


for i=1:size(cloud_point_from_press_reg,1)
    if(cloud_point_from_press_reg(i,4)==4)
        cloud_point_from_press_reg(i,4)=5;
    end



end
```

Replot everything

```
figure;
plot(Punti(:,1),Punti(:,2));
axis equal
hold on


plot(cloud_point_from_press_reg(:,1),cloud_point_from_press_reg(:,2),"*");
```

```
if( exist ("infill_1") == true)
    plot(infill_1(:,1),infill_1(:,2),"*");
end


if( exist ("infill_2") == true)
    plot(infill_2(:,1),infill_2(:,2),"*");
end


if(exist ("infill_3") == true)
    plot(infill_3(:,1),infill_3(:,2),"*");
end


if(exist ("infill_4") == true)
    plot(infill_4(:,1),infill_4(:,2),"*");
end


if(exist ("infill_5") == true)
    plot(infill_5(:,1),infill_5(:,2),"*");
end


hold off
```

Classify points of the STL using a brute force method

```
r=2; % radius of comparison


for i=1:size(cloud_point_from_press_reg,1)


    for j=1:size(Punti,1)


        radice=(Punti(j,1)-cloud_point_from_press_reg(i,1))^2+(Punti(j,2)-
cloud_point_from_press_reg(i,2))^2;
        distanza=sqrt(radice);


        if(distanza<=r)
```

```
            Punti(j,4)=cloud_point_from_press_reg(i,4);
        end




    end









end


colore = Punti(:,4);
figure;
scatter3(Punti(:,1), Punti(:,2), Punti(:,4)*10, 50, colore, 'filled');
axis equal
```

Create clusters of points that will form the various infills

```
indi_1=1;
indi_2=1;
indi_3=1;
indi_4=1;
indi_5=1;
for i=1:size(Punti,1)

    if(Punti(i,4)==1)
        Punti_1(indi_1,:)=Punti(i,1:3);
        indi_1=indi_1+1;
    end
```

```
    if(Punti(i,4)==2)
        Punti_2(indi_2,:)=Punti(i,1:3);
        indi_2=indi_2+1;
    end

    if(Punti(i,4)==3)
        Punti_3(indi_3,:)=Punti(i,1:3);
        indi_3=indi_3+1;
    end

    if(Punti(i,4)==4)
        Punti_4(indi_4,:)=Punti(i,1:3);
        indi_4=indi_4+1;
    end

    if(Punti(i,4)==5)
        Punti_5(indi_5,:)=Punti(i,1:3);
        indi_5=indi_5+1;
    end
end
```

Create Triangulation for infill 1

```
if( exist ("Punti_1") == true)



    alfa1=10.6;

    shp1 = alphaShape(Punti_1,alfa1);
    suggerito=criticalAlpha(shp1,'all-points')
    %plot(shp1)
    [T1, P1] =  alphaTriangulation(shp1);
```

```
    TR1=triangulation(T1,P1);
    [F1,P1] = freeBoundary(TR1);
    TR1=triangulation(F1,P1);
    Grafica(F1,P1)



    % salvo la triangolazione sotto forma di file stl
    stlwrite(TR1,"infill1.stl")

end
```

Create Triangulation for infill 2

```
if( exist ("Punti_2") == true)

    alfa2=11;

    shp2= alphaShape(Punti_2,alfa2);
    suggerito=criticalAlpha(shp2,'all-points')
    %plot(shp1)
    [T2, P2] =  alphaTriangulation(shp2);
    TR2=triangulation(T2,P2);
    [F2,P2] = freeBoundary(TR2);
    TR2=triangulation(F2,P2);
    Grafica(F2,P2)

    % salvo la triangolazione sotto forma di file stl
    stlwrite(TR2,"infill2.stl")
end
```

Create Triangulation for infill 3

```
if( exist ("Punti_3") == true)

    alfa3=7.7;
```

```
    shp3= alphaShape(Punti_3,alfa3);
    suggerito=criticalAlpha(shp3,'all-points')
    %plot(shp1)
    [T3, P3] =  alphaTriangulation(shp3);
    TR3=triangulation(T3,P3);
    [F3,P3] = freeBoundary(TR3);
    TR3=triangulation(F3,P3);
    Grafica(F3,P3)

    % salvo la triangolazione sotto forma di file stl
    stlwrite(TR3,"infill3.stl")
end
```

Create Triangulation for infill 4

```
if( exist ("Punti_4") == true)
    alfa4=7,8;

    shp4= alphaShape(Punti_4,alfa4);
    suggerito=criticalAlpha(shp4,'all-points')
    %plot(shp1)
    [T4, P4] =  alphaTriangulation(shp4);
    TR4=triangulation(T4,P4);
    [F4,P4] = freeBoundary(TR4);
    TR4=triangulation(F4,P4);
    Grafica(F4,P4)



    % salvo la triangolazione sotto forma di file stl
    stlwrite(TR4,"infill4.stl")

end
```

Create Triangulation for infill 5

```
if( exist ("Punti_5") == true)


    alfa5=7;


    shp5= alphaShape(Punti_5,alfa5);
    suggerito=criticalAlpha(shp5,'all-points')
    %plot(shp1)
    [T5, P5] =  alphaTriangulation(shp5);
    TR5=triangulation(T5,P5);
    [F5,P5] = freeBoundary(TR5);
    TR5=triangulation(F5,P5);
    Grafica(F5,P5)



    % save the triangulation as an stl file
    stlwrite(TR5,"infill5.stl")


end


% save the elastic modulus for each infill
save("Eperparte");
```

Load final Mesh

```
clear all


load("Eperparte.mat");
load("Matrice_riempimento_moduloelsatico.mat");


[TR]=stlread("Modeladapted.stl");  %% import STL


% save the variables of points and connectivities
Punti=TR.Points;
Triangoli=TR.ConnectivityList;
```

```
% View STL
figure;


Grafica(Triangoli,Punti)
title('Insole')
```

Load my infills

```
if isfile("infill1.stl")

    [TR1]=stlread("infill1.stl");   %% import the stl

    % save the variables of points and connectivities
    Punti1=TR1.Points;
    Triangoli1=TR1.ConnectivityList;

end



if isfile("infill2.stl")

    [TR2]=stlread("infill2.stl");   %% import the stl

    % save the variables of points and connectivities
    Punti2=TR2.Points;
    Triangoli2=TR2.ConnectivityList;

end

if isfile("infill3.stl")

    [TR3]=stlread("infill3.stl");   %% import the stl

    % save the variables of points and connectivities
```

```
    Punti3=TR3.Points;
    Triangoli3=TR3.ConnectivityList;

end



if isfile("infill4.stl")

    [TR4]=stlread("inifll4.stl");   %% import the stl

    % save the variables of points and connectivities
    Punti4=TR4.Points;
    Triangoli4=TR4.ConnectivityList;

end



if isfile("infill5.stl")

    [TR5]=stlread("inifll5.stl");   %% import the stl

    % save the variables of points and connectivities
    Punti5=TR5.Points;
    Triangoli5=TR5.ConnectivityList;

end
```

My new list of connectivities

```
Punti_Modellone=[Punti]; % Base Points
Triangoli_Modellone=[Triangoli]; % Base Triangles


if( exist ("Punti1") == true)% verify that the infill exists
```

```
    appoggio=size(Punti_Modellone,1);

    Punti_Modellone=[Punti_Modellone;Punti1]; %%add the points to the overall


    Triangoli_Appoggio1=Triangoli1+appoggio; %scale the triangle list

    Triangoli_Modellone=[Triangoli_Modellone;Triangoli_Appoggio1]; %update the
triangle list


end


if( exist ("Punti2") == true)
    appoggio=size(Punti_Modellone,1);

    Punti_Modellone=[Punti_Modellone;Punti2];


    Triangoli_Appoggio2=Triangoli2+appoggio;

    Triangoli_Modellone=[Triangoli_Modellone;Triangoli_Appoggio2];


end


if( exist ("Punti3") == true)
    appoggio=size(Punti_Modellone,1);

    Punti_Modellone=[Punti_Modellone;Punti3];


    Triangoli_Appoggio3=Triangoli3+appoggio;

    Triangoli_Modellone=[Triangoli_Modellone;Triangoli_Appoggio3];

end


if( exist ("Punti4") == true)
    appoggio=size(Punti_Modellone,1);

    Punti_Modellone=[Punti_Modellone;Punti4];


    Triangoli_Appoggio4=Triangoli4+appoggio;

    Triangoli_Modellone=[Triangoli_Modellone;Triangoli_Appoggio4];

end


if( exist ("Punti5") == true)
```

```
    appoggio=size(Punti_Modellone,1);
    Punti_Modellone=[Punti_Modellone;Punti5];


    Triangoli_Appoggio5=Triangoli5+appoggio;
    Triangoli_Modellone=[Triangoli_Modellone;Triangoli_Appoggio5];
end
```

calculate the infill for each part

```
infill=interp1(Materiale(1,:),Materiale(2,:),Eperparte);


infill=round(infill);
```

Infill

```


inifillbasesug=round(interp1(Materiale(1,:),Materiale(2,:),Emedio));


disp("valore consigliato infillbase "+inifillbasesug);


inifillbase="48";
inifillbase=inifillbase+"%";


if( exist ("Punti1") == true)
    disp("valore consigliato infill1 "+infill(1));


    inifill1="21";
    inifillbase=inifill1+"%";
end


if( exist ("Punti2") == true)
    disp("valore consigliato infill2 "+infill(2));


    inifill2="41";
```

```
    inifillbase=inifill2+"%";
end


if( exist ("Punti3") == true)
    disp("valore consigliato infill3 "+infill(3));


    inifill3="48";
    inifillbase=inifill3+"%";
end




if( exist ("Punti4") == true)
    disp("valore consigliato infill4 "+infill(4));


    inifill4="61";
    inifillbase=inifill4+"%";
end




if( exist ("Punti5") == true)
    disp("valore consigliato infill5 "+infill(5));


    inifill5="66";
    inifillbase=inifill5+"%";
end
```

Export file part

```
% Save a file from mesh to .amf
% Reference
% https://en.wikipedia.org/wiki/Additive_manufacturing_file_format


tic


fid = fopen("ModelOne.amf", 'w'); %create the file
```

```
filename = fopen(fid);            %acquire the file name



% first line is necessary for everything to work
fprintf(fid, '<?xml version="1.0" encoding="utf-8"?>\n');


% unit of measure
fprintf(fid, '<amf unit="millimeter">\n');


% model name
fprintf(fid, '<metadata type="name">%s</metadata>\n',filename);


% Start writing volume


fprintf(fid, '<object id="%s">\n',"0");


fprintf(fid, '<mesh>\n');


fprintf(fid, '    <vertices>\n');


% list of total triangle vertices
for i=1:size(Punti_Modellone,1)


    px1=Punti_Modellone(i,1);

    py1=Punti_Modellone(i,2);

    pz1=Punti_Modellone(i,3);

    fprintf(fid,'
<vertex><coordinates><x>%f</x><y>%f</y><z>%f</z></coordinates></vertex>\n',px1,py1,
pz1);



end



% end the list of vertices
fprintf(fid, '        </vertices>\n');
```

```
% list of volumes

%total volume

fprintf(fid, '        <volume>\n');



%add metadata



fprintf(fid, '            <metadata
type="slic3r.fill_density">%s</metadata>\n',inifillbase);
fprintf(fid, '            <metadata
type="slic3r.fill_pattern">cubic</metadata>\n');







%list of total connectivities including modifiers;

for i=1:size(Triangoli_Modellone,1)



    v1=Triangoli_Modellone(i,1)-1;

    v2=Triangoli_Modellone(i,2)-1;

    v3=Triangoli_Modellone(i,3)-1;

    fprintf(fid,'
<triangle><v1>%f</v1><v2>%f</v2><v3>%f</v3></triangle>\n',v1,v2,v3);


end



% end of volume list
fprintf(fid, '            </volume>\n');
```

```
%%%
```

```
end

%%Volume2%%%%%%%%%%%%%

if( exist ("Punti2") == true)

    fprintf(fid, '    <volume>\n');



    % add metadata



    fprintf(fid, '    <metadata
type="slic3r.fill_density">%s</metadata>\n',inifill2);
    fprintf(fid, '    <metadata type="slic3r.fill_pattern">cubic</metadata>\n');




    % List of connectivities for the selected model;

    for i=1:size(Triangoli_Appoggio2,1)



        v1=Triangoli_Appoggio2(i,1)-1;
        v2=Triangoli_Appoggio2(i,2)-1;
        v3=Triangoli_Appoggio2(i,3)-1;
        fprintf(fid,'
<triangle><v1>%f</v1><v2>%f</v2><v3>%f</v3></triangle>\n',v1,v2,v3);

    end



    % end of volume list
```

```
    fprintf(fid, '    </volume>\n');



end


%%%Volume3%%%%%%

if( exist ("Punti3") == true)

    fprintf(fid, '    <volume>\n');



    % add metadata

    fprintf(fid, '    <metadata
type="slic3r.fill_density">%s</metadata>\n',inifill3);
    fprintf(fid, '    <metadata type="slic3r.fill_pattern">cubic</metadata>\n');




    % List of connectivities for the selected model;

    for i=1:size(Triangoli_Appoggio3,1)



        v1=Triangoli_Appoggio3(i,1)-1;
        v2=Triangoli_Appoggio3(i,2)-1;
        v3=Triangoli_Appoggio3(i,3)-1;
        fprintf(fid,'
<triangle><v1>%f</v1><v2>%f</v2><v3>%f</v3></triangle>\n',v1,v2,v3);

    end



    % end of volume list
```

```
    fprintf(fid, '    </volume>\n');


end


%%%Volume4%%%%%%%%%%%%%

if( exist ("Punti4") == true)

    fprintf(fid, '    <volume>\n');


    % add metadata


    fprintf(fid, '    <metadata
type="slic3r.fill_density">%s</metadata>\n',inifill4);
    fprintf(fid, '    <metadata type="slic3r.fill_pattern">cubic</metadata>\n');



    % List of connectivities for the selected model;

    for i=1:size(Triangoli_Appoggio4,1)


        v1=Triangoli_Appoggio4(i,1)-1;
        v2=Triangoli_Appoggio4(i,2)-1;
        v3=Triangoli_Appoggio4(i,3)-1;
        fprintf(fid,'
<triangle><v1>%f</v1><v2>%f</v2><v3>%f</v3></triangle>\n',v1,v2,v3);

    end
```

```
    % end of volume list
    fprintf(fid, '    </volume>\n');


end

%%%%Volume5%%%%%%%%%

if( exist ("Punti5") == true)

    fprintf(fid, '    <volume>\n');


    % add metadata


    fprintf(fid, '    <metadata
type="slic3r.fill_density">%s</metadata>\n',inifill5);
    fprintf(fid, '    <metadata type="slic3r.fill_pattern">cubic</metadata>\n');




    % List of connectivities for the selected model;

    for i=1:size(Triangoli_Appoggio5,1)


        v1=Triangoli_Appoggio5(i,1)-1;
        v2=Triangoli_Appoggio5(i,2)-1;
        v3=Triangoli_Appoggio5(i,3)-1;
        fprintf(fid,'
<triangle><v1>%f</v1><v2>%f</v2><v3>%f</v3></triangle>\n',v1,v2,v3);


    end
```

```
    % end of volume list
    fprintf(fid, '    </volume>\n');


end




% close the mesh
fprintf(fid, '</mesh>\n');


% end of object section
fprintf(fid, '</object>\n');
% file closure
fprintf(fid, '</amf>\n');


fclose(fid); % close the file
toc
```

Useful functions

```
function [ R ] = tro( d )
% Function that creates elementary homogeneous translation matrices
%   d is the displacement vector used for description


R=[1 0 0 d(1); 0 1 0 d(2);0 0 1 d(3); 0 0 0 1];
```

```
end


function Grafica(Triangoli,Punti)
figure;

patch('Faces',Triangoli,'Vertices',Punti,'EdgeColor','none','FaceColor',[0.5 0.5
1],'LineWidth',2);

camlight(-37.5,-30);

view([37.5 30])

axis equal



end
```