3D Face Reconstruction using 2D Images

Brolin Fernandes, b.o.fernandes@student.utwente.nl, s2107112, M-ITECH
Gunish Alag, g.alag@student.utwente.nl, s2148439, M-MECH

ABSTRACT

3D face reconstruction using two stereo image pairs has been demonstrated (Appendix A) in this article. The overall flow of this procedure begins with camera calibration, followed by creating the required face masks. Stereo rectification aligns the images along the baseline, which further assists in the creation of the disparity maps. Once the depth maps are generated, we regenerate the scene, thereby creating a 3D point cloud which is further fine-tuned to create a mesh.

1. INTRODUCTION

3D reconstruction of images involves creation of 3 dimensional models using multiple 2D images. It is the reverse process of obtaining 2D images from 3D scenes using a single camera. 3D reconstruction of images has already been established as a field of varied applications, ranging from security & surveillance, virtual reality simulations, plastic surgery simulations, face recognition, face morphing, 3D games, human computer interaction and animations and even social media applications like Snapchat[1]. The human face reconstruction in a 3D model is an intricate process, as there is a lot that goes into generating 3D points from 2D scenes, such as the estimation of depth and other complexities. A stereo view with 2 cameras is required to generate an estimate of 3D scenes. When humans see with one eye closed, the depth estimation is not as optimal as when humans see with both eyes[2]. The human way to develop good 3D images involves having 2 eyes, in order to get a stereo view of the surrounding, thereby getting an estimate of the depth in view to the brain. A similar approach is generally used in image processing & computer vision, where a pair of cameras with identical camera calibration can be used for getting a stereo view from the set angles, in order to get an estimate of the depth in the image, which helps to create a 3D model. This is the approach used in this model, which will be explained in detail along with the methods and algorithms involved in the upcoming sections.

2. METHOD

MATLAB 2018b and its Image Processing and Computer Vision toolbox was used for implementation of 3D face reconstruction.

2.1 Background

Resources provided:

For the purpose of camera calibration, multiple images from different angles and orientations, from each camera (L- Left, M- Middle, R- Right) of a black & white checkerboard were provided. An example of these images is shown in Figure 1.

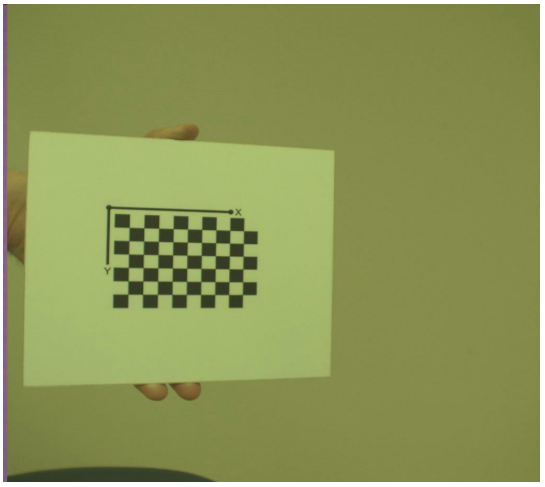


Figure 1: An image for camera calibration taken from the left camera

The given images are taken from aligned cameras (baseline is oriented in the x direction) the baseline horizontal direction of both the images are parallel and both the optical axes are parallel (along z-direction), the depth can be estimated using similar triangle principle as shown in Figure 2.

$$\frac{z}{f} = \frac{x}{x_l} = \frac{x-b}{x_r}$$

$$d = x_l - x_r = \frac{fb}{z}$$

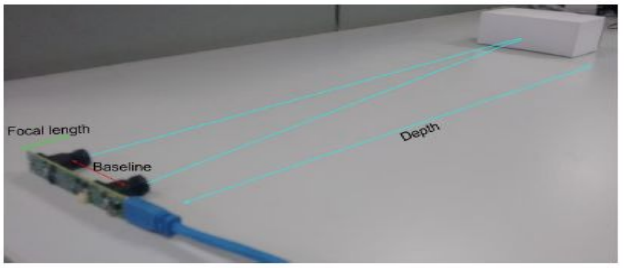


Figure 2

d represents the disparity of a particular point (spatial shift) of a particular point in the two images. The disparity is reduced with with increased distance between the camera and the point.

Depth is estimated by finding the common pixel in both the images that represents the same point in the real world coordinate system. The process is known as stereo matching which can be done using MATLAB app, *stereo camera calibration* and results in giving the parameters of the two cameras used in taking the pictures. Using these parameters the depth can be estimated using the equation mentioned above.

2D images of three different individuals with 5 expressions each were provided. The images were captured from 3 different angles (L- left, M- middle, R- right) at the same instant for each expression for each person. For processing, as shown in Figure 3, one set of L,M,R images were used at a time, in order to get a 3D representation of the person-expression combination.

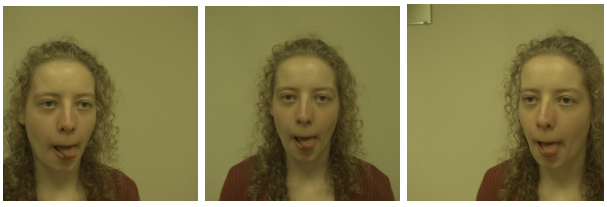


Figure 3: Example of set of images provided of one person from different angles.

2.2 Face Segmentation

The images provided have a background which is not desirable in 3D creation of the face, hence to eliminate the background a mask was created (containing face, neck and hair) of the subject. As the image has a clear background, so it was easy to segment the image. For the segmentation the *Color Thresholder app* from MATLAB was used but it did not provide good results as the forehead was being masked along with the background. Hence, a function was created, called *k_means_clustering* (Appendix B) taking the Lab *Color Segmentation Example* and *K Means Segmentation Example* that are provided in MATLAB as reference. K means clustering segments the image into the desired number of clusters. In this case the image was divided into 3 categories mainly face, background and hair. For this, the image is first converted to L*a*b colour space. From this, the a*b values were extracted as the luminous variations were causing a problem and proper image masking was being difficult. By neglecting the luminous values we were able to extract workable masks from the images. After this the clustering was performed resulting in a binary mask which was later used to get a better mask as the mask obtained simply from k means was not quite sufficient to extract the face from the image. Next morphological operations were used on the binary mask obtained to get a more appropriate mask that is able to provide a sufficient amount of face area, which will be used later for rectification, disparity map generation and point cloud generation. After performing the above tasks a fairly useful mask was obtained from the images which was used to mask the original images and obtain the masked images as shown in figure 4.

Figure 4: Left, Middle and Right mask

Algorithm K-means_clustering

Input - image

Output - Binary mask

1. convert to L*a*b color space.
2. extract the a*b values.
3. run k means (inbuilt function).
4. label pixels in the image obtained from kmeans.
5. segment image into multiple images based on colors.
6. segment the color face into separate image and convert it into a binary mask.
7. apply morphological operation to eliminate any false segmentation within the face.

The obtained binary masks were used to overlay over the original image and generate a mask containing only the face without the background.

2.3 Stereo Rectification and Matching

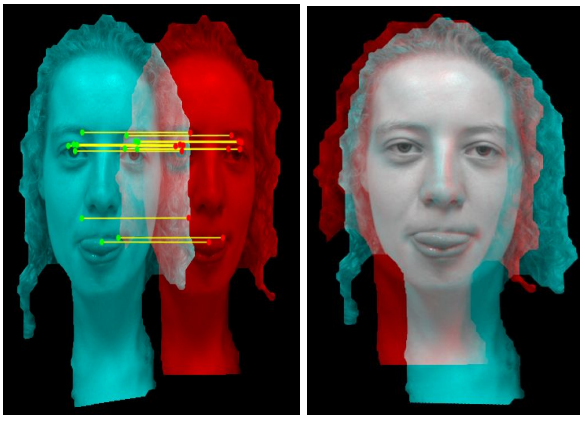


Figure 5: Stereo rectification and matching

Next the images obtained so far (the original images of left, right and middle along with the masked images). This is done using the MATLAB function *rectifyStereoImages*. Stereo rectification projects images into a common image plane in a way that the corresponding points have the same epipolar geometry.

Next from the obtained rectified images stereo matching was done to check the results of stereo rectification. This was done by using *Registration Estimator app* from MATLAB toolbox.

2.4 Disparity Map

After stereo rectification the disparity of the images can be calculated. This is done when the images are in the same epipolar geometry.

If 2 points (x,y) and a point (u,v) = (u,y) and corresponding points, then the shift of pixel can be calculated using :

$$D \overset{\text{def}}{=} n - u$$

D is disparity of (x,y). This is based on the assumption that each pixel has a corresponding pixel in the other image. The disparity helps in estimating the depth for each pixel. The built-in function, *disparity*, from the MATLAB can be used and with thresholding parameters to optimize the results. A function *DisparityAndUnreliability* (Appendix C) was built with a switch case scenario for different subjects.

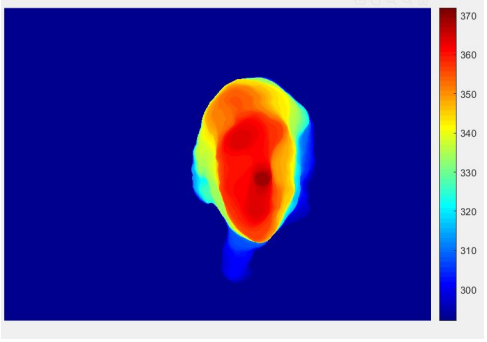
Algorithm

Input - image1,image2,mask,subject number

Output - disparity map, unreliability

1. disparity range is defined.
2. create disparity map.
3. the background is masked.
4. the image is smoothed by using a median filter.
5. the holes created are filled using imfill.

First the disparity range is defined, as it defines the depth relation to each other. The minimum value is the farthest point and the maximum value is the closest point of the image to the camera. After the disparity map is generated some additional parameters are used to enhance the result. Next the mask is applied over the image to just get the face without the background. The disparity map is still a little sharp even after applying the parameters, so a median filter was applied to smooth out the results. The remaining holes are removed using imfill. The disparity maps are shown in Figure 6.



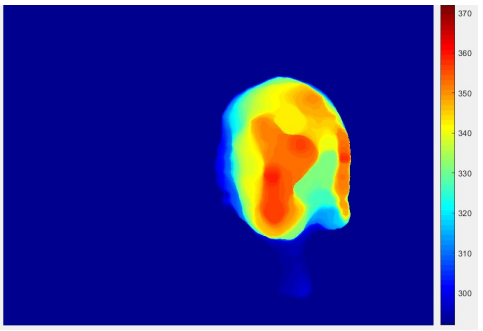


Figure 6: Disparity maps

2.5 Point Cloud and 3D mesh

With the disparity known at given pixel position, a point cloud can be generated from the obtained disparity map and the camera parameters using the function *reconstructScene*. It returns a point cloud from the stereo pair of images. The corresponding point clouds are obtained. Next, the obtained point cloud is denoised and downsampled. Following this, the obtained point clouds are matched to each other using ICP algorithm where one is transformed to the other and finally merged together. The point cloud after denoising and matching as explained above is shown in Figure 7.

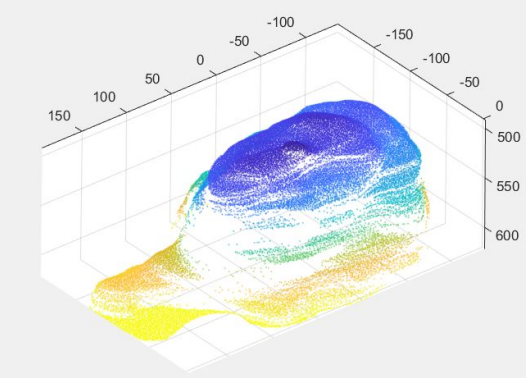


Figure 7: Point cloud generation

Now from the obtained point clouds a 3D surface mesh was generated using the provided function from Syllabi. The function *3D_surface_mesh* (Appendix D) transforms the obtained disparity map, unreliable points (outliers), point clouds and the images to create a 3D surface mesh. It is done by forming triangles that are the face structures and corners that are the vertices. The vertices are the 3D points that correspond to the 3D point cloud. From this, a connecting structure is built, from which the unreliable points and the triangles associated with those vertices are removed. The final 3D surface is created with the function *triangulation*, shown in Figure 8.

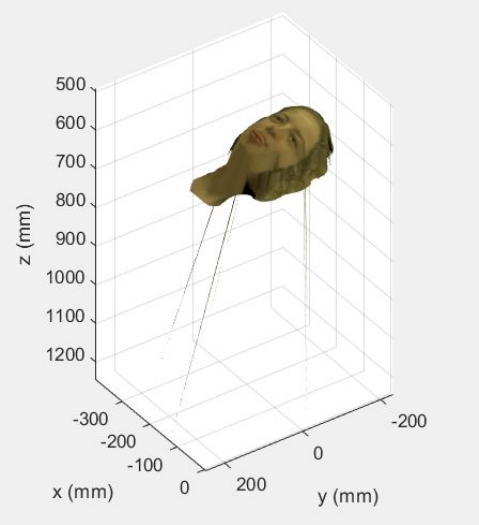


Figure 8: Final 3D mesh

3. Discussion

The brightness in the images caused a problem to extract a mask from the images, but it was rectified by eliminating the luminous values in the L*a*b color space and a workable mask of the face was extracted from the images. This we were unable to do while using the color threshold app from MATLAB toolbox.

Another point of improvement is optimizing the disparity mapping which can further reduce the disparity between the images and reduce the unreliable points between images. This can however also be improved by preprocessing the image. For example, the background and the skin colour were quite similar that caused a problem for masking. Having a brighter colour which does not resemble other colours in the image would be ideal for mask extraction. This could even reduce the unreliable points when generating disparity map. The shirt of the person was also quite similar in color for subject 1 which also caused a problem, so this can also be changed by wearing a shirt that does not resemble the skin colour as is the case for subject 2.

Tackling these problems and enhancing results can aid in various applications like virtual reality simulations, plastic surgery simulations, face recognition, face morphing, 3D games, human computer interaction and animations, etc.

4. Conclusion

In this project, a procedure was developed through which a 3D face can be reconstructed using two stereo pair images. The images provided were taken from different sides for the same subject. To create the 3D mesh of the face first the stereo calibration of the cameras was done to obtain the parameters, next the images were used to extract a mask and obtain the face from the images which will later be used for disparity mapping and getting the unreliable points. From the obtained images the images were stereo rectified. Next, the masked and the original images were used to create the disparity map and get the unreliable points, using the disparity map point cloud was generated. Using the point cloud, disparity map and unreliable points 3D mesh was created. However the 3D mesh obtained is satisfactory the mesh can be further improved using better clustering algorithm and by optimizing the algorithm used to generate the disparity map.

Appendix A

```
% 3D Face Reconstruction from stereo images
% Course: Image Processing & Computer Vision
% University of Twente
% 18 April, 2019

% Authors: Gunish Alag s2148439 & Brolin
Fernandes s2107112

clear variables % Clear variables in
workspace
close all       % Close all
images/graphs/plots
clc             % Clear Command Window

% Initialize paths to data, functions and
images
addpath('data');
addpath('test');
addpath('functions');

% LM represent Left -> Middle
% MR represent Middle -> Right

%% Reading images
images = imageSet('test'); % Create a indexed
imageSet of the subject
Sub = 1; %0, 1, 2 represents subject 1, 2, 3
im_left = im2double(read(images,1+3*Sub));
im_middle = im2double(read(images,2+3*Sub));
im_right = im2double(read(images,3+3*Sub));

%% Camera Calibration

% matlab app stereo camera calibration was
used to calibrate the camera and
% the parameters are stored and loaded.

% Load stereo camera calibrated parameters
load('CalibrationData/stereoParams_subject1_c
alib1.mat');

%% k means clustering and image segmentation

% the image was broken into L*a*b category
and then k means clustering aws
% used to extract a mask from the left middle
and right image.
% This is based mostly on the Matlab Example:
%
openExample('images/LabColorSegmentationExamp
le')
```

```
%
openExample('images/KMeansSegmentationExample
')
% but the results were not satisfactory
enough so next we used the colour
% thresholder app from the matlab to segment
the images and it was further
% used as the mask for the images. the data
from the app was stored and
% loaded later for use.

%
% [mask_left] = k_means_segment(im_left);
% [mask_middle] = k_means_segment(im_middle);
% [mask_right] = k_means_segment(im_right);

im_left_masked = im_left;
im_middle_masked = im_middle;
im_right_masked = im_right;

% Load masks of the desired subject found
through color thresholder  clustering
% load('Masks/masks_subject1_calib1.mat'); %
load left, middle, and right mask
load('Masks/masks_subject2_calib1.mat'); %
load left, middle, and right mask
%load('Masks/masks_subject3_calib1.mat'); %
load left, middle, and right mask

% the obtained binary masks after using the k
means segment is used to
% overlay on the original image and generate
the masked images
mask_left = cat(3, mask_left, mask_left,
mask_left);
im_left_masked(imcomplement(mask_left)) = 0;
mask_middle = cat(3, mask_middle,
mask_middle, mask_middle);
im_middle_masked(imcomplement(mask_middle)) =
0;
mask_right = cat(3, mask_right, mask_right,
mask_right);
im_right_masked(imcomplement(mask_right)) =
0;


%% Stereo rectification
% rectification of the original images with
stereo parameters obtained from
% stereo camera calibration
[im_left_rect, im_middleleft_rect] =
rectifyStereoImages(...
    im_left, im_middle,
stereoParams_LM,'OutputView','full');

[im_middleright_rect, im_right_rect] =
rectifyStereoImages(...
    im_middle, im_right, stereoParams_MR,
'OutputView','full');

% Stereo rectify the masked images as well
for disparity maps later
[im_left_rect_mask, im_middleleft_rect_mask]
= rectifyStereoImages(im_left_masked, ...
    im_middle_masked,
stereoParams_LM,'OutputView','full');

[im_middleright_rect_mask,
im_right_rect_mask] =
rectifyStereoImages(im_middle_masked, ...
    im_right_masked, stereoParams_MR,
'OutputView','full');


%% Disparity map
% Create disparity map for the two image
pairs
[disp_map_LM, unreliable_LM] =
disparityMapAndUnreliable(im_left_rect, ...
    im_middleleft_rect, im_left_rect_mask, 11
+Sub*10);
[disp_map_MR, unreliable_MR] =
disparityMapAndUnreliable(im_middleright_rect
, ...
im_right_rect, im_middleright_rect_mask, 12
+Sub*10);

%% 3D point clouds
% Create point clouds for LM and MR
xyzPoints_LM =
reconstructScene(disp_map_LM,stereoParams_LM)
;
xyzPoints_MR =
reconstructScene(disp_map_MR,stereoParams_MR)
;

% Complete PointCloud Registration Algorithm
%{
    Denoise(LM)     &   Denoise(MR)     -
pcdenoise
    Downsample(LM)  &   Denoise(MR)     -
pcdownsample
    Rigid Transformation LM & MR        -
pcregrigid
        Match Points between LM & MR
        Remove incorrect matches (Outlier
```

```
filter)
        Recover rotation and translation
(minimize error)
        Check if algorithm stops
    Align LM and MR                 -
pctransform
    Merge LM and MR                 -
pcmerge
%}

pc_LM = pointCloud(xyzPoints_LM);
pc_LM = pcdenoise(pc_LM);
pc_LM = pcdownsample(pc_LM,
'nonuniformGridSample', 15);
pc_MR = pointCloud(xyzPoints_MR);
pc_MR = pcdenoise(pc_MR);
pc_MR = pcdownsample(pc_MR,
'nonuniformGridSample', 15);
[tform,pc_MR,rmse]= pcregrigid(pc_MR, pc_LM,
'Extrapolate',true);
 pc = pcmerge(pc_LM, pc_MR, 1);
figure;
pcshow(pc_LM);

%% Create 3D meshes from point clouds
% function is used from the one provided in
UT_sylabi
meash_LM = create_3D_mesh(disp_map_LM,
xyzPoints_LM, ...
    unreliable_LM, im_left_rect_mask);
```

Appendix B

```
function [segmentedImage] =
k_means_segment(im)
% This function is based on the Matlab
Example:
%   'Color-Based Segmentation Using
K-Means Clustering'
% This example can be opened with the
command
%
openExample('images/KMeansSegmentationE
xample')

%% Convert Image from RGB Color Space
to L*a*b* Color Space
lab_im = rgb2lab(im);  % Convert image
to L*a*b* color space

%% Classify the Colors in 'a*b*' Space
Using K-Means Clustering
ab = lab_im(:,:,2:3);  % Extracting the
a*b values from the image
nrows = size(ab,1);
ncols = size(ab,2);
ab = reshape(ab,nrows*ncols,2);
nColors = 3;              % segmenting
the image into 3 categories Skin
colour, hair colour, background colour
n = 3;                    % Repeating
the process to avoid local minima

% Run k-means clustering algorithm
[cluster_index, cluster_centeroid]=
kmeans(ab,nColors,'Distance', ...
    'sqeuclidean', 'Replicates', n);

%% Label Every Pixel in the Image Using
the Results from KMEANS
pixel_labels =
reshape(cluster_index,nrows,ncols);
%imshow(pixel_labels,[]), title('image
labeled by cluster index');

%% Create Images that Segment the Face
Image by Color.
segmented_images = cell(1,nColors);
rgb_label = repmat(pixel_labels,[1 1
3]);

for k = 1:nColors
    color = im;
    color(rgb_label ~= k) = 0;
    segmented_images{k} = color;
    %figure;
    %imshow(segmented_images{k}),
title(['objects in cluster '
num2str(k)]);
end

%% Segment the Face into a Separate
Image
% Sort the clusters based on mean
values
mean_cluster_value =
mean(cluster_centeroid,2);
[~, idx] = sort(mean_cluster_value);
skin_cluster_num = idx(3);

L = lab_im(:,:,1);
% Extract the L* values
skin_idx = find(pixel_labels ==
skin_cluster_num);
L_skin = L(skin_idx);
is_light_skin =
```

```
imbinarize(rescale(L_skin));

face_labels = repmat(uint8(0),[nrows
ncols]);
face_labels(skin_idx(is_light_skin==fal
se)) = 1;
face_labels = repmat(face_labels,[1 1
3]);

face = im;
face(face_labels ~= 1) = 0;
% Enhancing face
face_masked = face |
segmented_images{3}; % Make binary mask
face_masked = face_masked(:,:,1);
% Resize to two-dimensional

% morphological operations are done to
open and close the mask to get a
% suitable mask that can be used to
filter the image and extract the face.
% The operations performed here were
found by trial-and-error
% They gave generally acceptable
results for the different subjects
seed =
imerode(face_masked,strel('disk',12));
face_masked =
imreconstruct(seed,face_masked);
face_masked = imopen(face_masked,
strel('disk',10));
face_masked = imclose(face_masked,
strel('disk',40));
seed =
imerode(face_masked,strel('disk',24));
face_masked =
imreconstruct(seed,face_masked);
segmentedImage = face_masked;

figure; imshow(face_masked),
title('face_b');
end
```

Appendix C

```
function [disparityMap, unreliable] =
disparityMapAndUnreliable(im_left,
im_right, mask, subNumPair)
% This is a function to create
disparity maps for the different
subjects
% It is specalised for the different
image pairs possible, but they all
% have similiar operations performed to
them
im_left = rgb2gray(im_left);
im_right = rgb2gray(im_right);
mask = rgb2gray(mask);

% Procedure:
%{
    Define disparity range
    Make disparity map
    Mask out the background
    Median filter the disparity map to
smoothen the results
%}

switch subNumPair
    case 11 % Subject 1, LM image pair
        disparityRange = [276,340];
        disparityMap =
disparity(im_left,im_right,'DisparityRa
nge',...
            disparityRange,
'ContrastThreshold',0.9, ...

'UniquenessThreshold',5,'DistanceThresh
old',15,'BlockSize',5);
        disparityMap(imcomplement(mask
> 0)) = 0;
        disparityMap =
medfilt2(disparityMap, [30
30],'symmetric');
    case 12 % Subject 1, MR image pair
        disparityRange = [276,340];
        disparityMap =
disparity(im_left,im_right,'DisparityRa
nge',...
            disparityRange,
'ContrastThreshold',0.9, ...

'UniquenessThreshold',5,'DistanceThresh
old',15,'BlockSize',5);
        disparityMap(imcomplement(mask
> 0)) = 0;
        disparityMap =
medfilt2(disparityMap, [30
30],'symmetric');

    case 21 % Subject 2, LM image pair
        disparityRange = [292 372];
        disparityMap =
disparity(im_left,im_right,'DisparityRa
nge',...
            disparityRange,
```

```
'ContrastThreshold',0.6, ...

'UniquenessThreshold',1,'DistanceThresh
old',5,'BlockSize',11);
        disparityMap(imcomplement(mask
> 0)) = 0;
        disparityMap =
medfilt2(disparityMap, [100 100],
'symmetric');
    case 22 % Subject 2, MR image pair
        disparityRange = [292 372];
        disparityMap =
disparity(im_left,im_right,'DisparityRa
nge',...
            disparityRange,
'ContrastThreshold',0.6, ...

'UniquenessThreshold',1,'DistanceThresh
old',10,'BlockSize',9);
        disparityMap(imcomplement(mask
> 0)) = 0;
        disparityMap =
medfilt2(disparityMap, [100 100],
'symmetric');

    case 31 % Subject 3, LM image pair
        disparityRange = [430-16*7
430];
        disparityMap =
disparity(im_left,im_right,'DisparityRa
nge',...
            disparityRange,
'ContrastThreshold',1, ...

'UniquenessThreshold',5,'DistanceThresh
old',100,'BlockSize',15);
        disparityMap(imcomplement(mask
> 0)) = 0;
        disparityMap =
medfilt2(disparityMap, [100
100],'symmetric');
    case 32 % Subject 3, MR image pair
        disparityRange = [426-16*5
426];
        disparityMap =
disparity(im_left,im_right,'DisparityRa
nge',...
            disparityRange,
'ContrastThreshold',1, ...

'UniquenessThreshold',5,'DistanceThresh
old',100,'BlockSize',17);
        disparityMap(imcomplement(mask
> 0)) = 0;
        disparityMap =
medfilt2(disparityMap, [80
80],'symmetric');
end

% Remove holes in the disparity map
disparityMap =
imfill(disparityMap,'holes');

% Plot Disparity map
figure;
imshow(disparityMap,disparityRange);
colormap(gca,jet);
colorbar;

% Remove unreliable points from the
disparity map
unreliable = ones(size(disparityMap));
unreliable(find(disparityMap~=0)) = 0;
unreliable(find(disparityMap==-realmax(
'single'))) = 1;

end
```

Appendix D

```
function [TR, pcl2]=
create_3D_mesh(disparityMap, pc,
unreliable,J1)
%% create a connectivity structure
[M, N] = size(disparityMap);
% get image size
res = 2;                              %
resolution of mesh
[nI,mI] = meshgrid(1:res:N,1:res:M); %
create a 2D meshgrid of pixels, thus
defining a resolution grid
TRI = delaunay(nI(:),mI(:));          %
create a triangle connectivity list
indI = sub2ind([M,N],mI(:),nI(:));    %
cast grid points to linear indices

%% linearize the arrays and adapt to
chosen resolution
pcl = reshape(pc,N*M,3); % reshape to
(N*M)x3
J1l = reshape(J1,N*M,3); % reshape to
(N*M)x3
pcl = pcl(indI,:);       % select 3D
points that are on resolution grid
J1l = J1l(indI,:);       % select
pixels that are on the resolution grid
```

```
%% remove the unreliable points and the
associated triangles
ind_unreliable =
find(unreliable(indI));% get the linear
indices of unreliable 3D points
imem = ismember(TRI(:),ind_unreliable);
% find indices of references to
unreliable points
[ir,~] = ind2sub(size(TRI),find(imem));
% get the indices of rows with refs to
unreliable points.
TRI(ir,:) = [];              % dispose
them
iused = unique(TRI(:));      % find the
ind's of vertices that are in use
used = zeros(length(pcl),1); %
pre-allocate
used(iused) = 1;             % create a
map of used vertices
map2used = cumsum(used); % conversion
table from indices of old vertices to
the new one
pcl = pcl(iused,:);      % remove the
unused vertices
J1l = J1l(iused,:);
TRI = map2used(TRI);     % update the
ind's of vertices

%% create the 3D mesh
%pcl(~isfinite(pcl)) = 0;
TR = triangulation(TRI,double(pcl)); %
create the object

%% visualize
figure;
TM = trimesh(TR);              % plot
the mesh
set(TM,'FaceVertexCData',J1l); % set
colors to input image
set(TM,'Facecolor','interp');
% set(TM,'FaceColor','red');   % if you
want a colored surface
set(TM,'EdgeColor','none');    %
suppress the edges
xlabel('x (mm)')
ylabel('y (mm)')
zlabel('z (mm)')
axis([-250 250 -250 250 400 900])
set(gca,'xdir','reverse')
set(gca,'zdir','reverse')
daspect([1,1,1])
axis tight
end
```