3D Face Reconstruction

Per Arne Kjelsvik, *p.a.kjelsvik@student.utwente.nl, s2049201, M-EE*
and Tiago Machado, *t.venturadasilvaribeiromachado@student.utwente.nl, s2041596, M-ME*

***Abstract*—This report presents an explanation and an algorithm for stereo-based 3D face reconstruction. The algorithm is based on depth estimation pairs of pictures. Pictures are taken from three different angles (left, middle, right) for three different subjects. The algorithm presented combined the left and middle images as a pair, and the middle and right images, producing two 3D surfaces meshes as the final result.**

***Keywords*—3D face reconstruction, 3D point cloud, depth estimation, stereo vision.**

I. Introduction

THE field of 3D reconstruction is a field that has grown rapidly in the age of information, with increasing accuracy and increasingly sophisticated techniques. In this project, the task was to use multiple 2D images of a human face to create a 3D representation of the face. Nowadays, several social media platforms use this kind of technology in real-time for entertainment purposes, for example applying complicated structures on top of a person's face. Another application is the use of Augmented Reality (AR) in computer systems and video games.

The challenge of 3D reconstruction is to correctly align data from 2D information and extrapolate it accurately in 3D space. There are multiple techniques and ways to do this. For example, find facial landmarks in a person's face and map them to a trained classifier's understanding of a face, and build the 3D model out from this. Or use parameters of your camera, calibrate it so that you know how images align with each other, and use mathematical theory to calculate points and their representation in 3D space based on camera models. It's also possible to track key points of an object (like a face) in video files to create the structure. There are other uses of these techniques than face reconstruction (which can be important to medical purposes), for example crime scene reconstruction so that investigator's can simulate the crime scene after the fact, and the aforementioned entertainment purposes.

For this report, the methods used for 3D face reconstruction will first be listed and explained in section II-B, before the results are presented in III with a short discussion afterwards in IV.

II. Methods and Materials

A. Materials

`MATLAB R2017b`:

- Image Processing and Computer Vision Toolbox.

(a) (b) (c)

Fig. 1. Three angles of subject 3

B. Methods

1) Analysis: There are different ways to reconstruct a face from 2D images. In this case, three pictures were taken of each subject at the same time, in sets of five different face expressions, from three cameras aligned around the subject. To see an example of such as set of images, see figure 1. Another resource made available for the project were calibration images, which is images of a checkerboard to find the cameras lens distortions and intrinsic and extrinsic parameters. For a full mathematical development of these models and their properties, see [1].

After the parameters has been obtained, `MATLAB` can be used to get the desired output in the end - namely the 3D surface meshes. The steps required to achieve this will be outlined in the following subsubsections, and are roughly broken down as: stereo rectify images, remove background of the images, create disparity maps based on the images, and finally make 3D point clouds and 3D surface meshes.

The set-up of two cameras of the same type that are aligned with each other has the most straightforward and simple epipolar geometry. Creating such a set-up (see figure 2) facilitates the processing of the stereo images (see figure 1). This is called stereo rectification. When the two cameras are aligned with each other (the baseline is oriented horizontally, x-direction), the horizontal direction of both images planes are parallel to the baseline. With the multi-camera stet-up is possible to estimate a 3D depth map from the face using triangulation (see figure 3).

2) Camera Calibration: To start, as stated, identification of the camera parameters with a special checkerboard is necessary, so that the cameras can be calibrated. Besides having the camera calibrated, it is important to rectify the images so that the images can be related to each other in the same system. The rectification of the images is done in pairs and the aim of this process is to have the same features in the face from the two images aligned horizontally with each other (along epipolar lines). The cameras are virtually rotated and their calibration matrices are changed as well to bring the

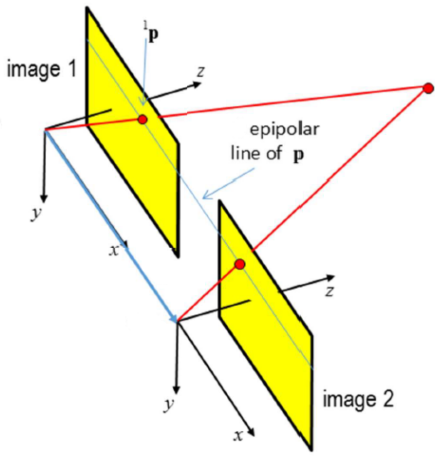


Fig. 2. Camera set-up

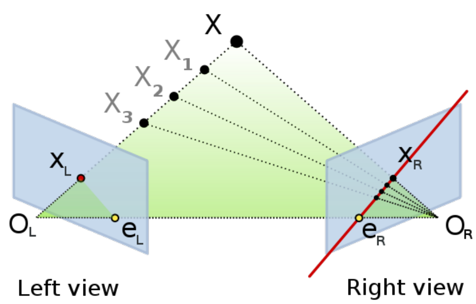


Fig. 3. Corresponding points between two cameras (figure from website Stackoverflow by user Max Allan)

epipolar geometry to its easiest state. This process was done with the `MATLAB`-function `stereoCameraCalibrator` from the `Image Processing & Computer Vision toolbox`. The function reads the images with a defined square size of 10mm in this case, and then calibrates the cameras. With the graphical interface, a threshold to remove outliers can be used, which in this case was used so that a suitable minimum error for each camera pair was chosen. Two different calibration sessions were given with the project.

3) Stereo Rectification: Now that the camera parameters are known, the image pairs can be rectified. This is done with the `MATLAB`-function *rectifyStereoImages*. The rectification were performed on the full images of pair left & middle, and middle & right. Stereo image rectification projects images into a common image plane in such a way that the corresponding points have the same row coordinate, epipolar geometry. This image projection makes the images appear as though the cameras that shot them are parallel.

4) Face Extraction: Since it's only the face that we're interested in, it's good to detect and remove the background of the image. There are several possible ways to do this, one of them is color based segmentation using *k-means clustering* [2]. Function `k_means_segment` (see appendix B) was written to perform this extraction, see below for how this algorithm works. The procedure was mostly based on the `MATLAB`-example: *Color-Based Segmentation Using K-Means Clustering*. The k-means algorithm is a clustering method implemented as a function in `MATLAB` that for our purpose tries to partition the image into three segments based on a rough assessment that there are 3 main colors in the image (wall, skin, shirt). Firstly, the images are converted from the *RGB* color space to *CIE 1976 L*a*b** color space. After the clustering has been done, a binary mask can be made and then several morphological operations to make it fit the face. One of the underlying assumption for the whole project is the *global constant brightness* assumption, which states that corresponding points of two images have identical color or gray level. This is however mostly not the case, for example due to reflection of a light source in the forehead of the subjects. A normalization can be applied to make the assumption more valid however.

Algorithm 1 Function K-means_clustering

Input: image
Output: image mask
convert image to L*a*b* color space
run kmeans algorithm
label pixels in image using result from kmeans
segment image into multiple images based on color
segment color image of face into separate image, convert to binary mask
apply morphological operations to image mask
return *image mask*

Algorithm 2 Algorithm k-means

Input: k the number of clusters
D (a set of lift ratios)
Output: a set of k clusters
METHOD:
Arbitrarily choose k objects from as the initial cluster centers;
REPEAT:
(re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
Update the cluster means, i.e., calculate the mean value of the objects for each cluster
Until no change;

5) Stereo Matching: The task of the *Stereo Matching* is close to a 2D search problem. There are several ways to perform stereo matching, which is to find the corresponding points of images that will have depth information encoded into them. One way to do it, which was performed in this project to verify that the images were correctly rectified, i.e. on the same horizontal lines, is the `Registration Estimator` app from the `MATLAB`-toolbox. Using Maximally Stable Extremal Regions (MSER), quality and number of points can be fine-tuned, which gives results as seen in figure 4. As we can see,

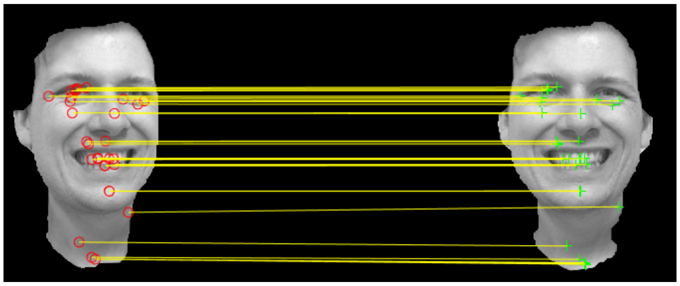
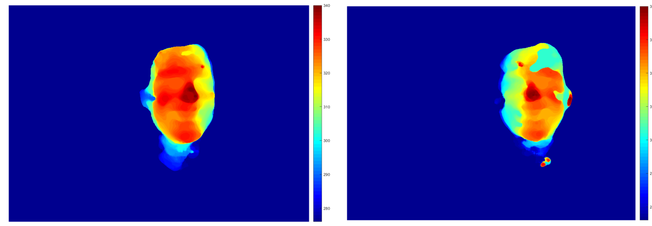


Fig. 4. Example of rectified images with epipolar lines after point matching with `Registration Estimator` app. Green crosses and red circles are the matched points in each image.



(a) (b)

Fig. 5. Disparity map of subject 1. (a) is left and middle images, (b) is middle and right images.

Fig. 6. Masked image obtained after k-means clustering method

the images are correctly rectified, as the corresponding points are on epipolar lines.

If a point *(n,m)* and a point *(u,v)=(u,m)* are corresponding points, then the pixel shift:

$$D \overset{\text{def}}{=} n - u \tag{1}$$

is the disparity of *(n,m)*. Assuming that each pixel has a corresponding point, the disparity is called for each pixel and the so-called disparity map can be made, which decodes the depth for each pixel. The built-in `MATLAB`-function `disparity` can be used for this purpose, with several options. A function `disparityMapAndUnreliable` was made with a switch-case inside for the different possibilities of the images used, to optimize the results, but this could easily be made more general. See Appendix C for the implementation of this function. The general procedure of this function, is as follows:

Algorithm 3 disparityMapAndUnreliable

Input: left image, image right, image mask, subject number
Output: disparity map, unreliable map
PROCEDURE:
Define disparity range
Create disparity map
Mask out the background
Median filter the disparity map to smoothen the results
Fill out holes in the disparity map
return disparity map, unreliable map

The first step is defining the disparity range. This range is really important, as it defines the depth relation to each other, for example that the nose is closest and the neck is far behind. The minimum value of the disparity range should be the farthest point of the given image, while the maximum range should be the closest point to the camera (nose, temple or cheek). This was manually measured in the separate images, but there are methods to estimate this range based on the maximum and minimum values of the pixels of the rectified images. After, the disparity map is created (see Figure 5) plus some additional parameters to enhance the results (see appendix C). Next, the image mask is applied to remove the background of the rectified images, in order to have just the face of the subject (see Figure 6). The disparity map is a little sharp due to the various contrasts and unreliable points of the map, so a median filter is applied to smooth out the results. Then any remaining holes in the mask is removed with the function `imfill`. Finally, points that are estimated to be unreliable by the function, as well as point with the value 0, are set to 1 in another binary mask, the `unreliable` mask.

6) Point Cloud: Knowing the disparity at a given pixel position *(n,m)* the goal is to finally find the 3D position of the 2D images. Applying the function *reconstructScene*, which has as input the disparity map and camera parameters. This function returns coordinates of a point cloud. The two point clouds can be combined together, following a procedure where they first are denoised, downsampled, then using the ICP algorithm, they are matched to each other, before one is transformed to the other and they can be merged (see Figure 7).

7) 3D Surface mesh: Now, the final step of the project is here. Using the code from the Syllabi, a function `create_3D_mesh` can be written, which transforms the disparity map, unreliable map, point cloud, and the images to a 3D surface mesh. This is done by forming a set of adjacent triangles, called faces, and corners which are called vertices. The vertices are the 3D points that constitute the 3D point cloud. A connectivity structure is made, the unreliable points

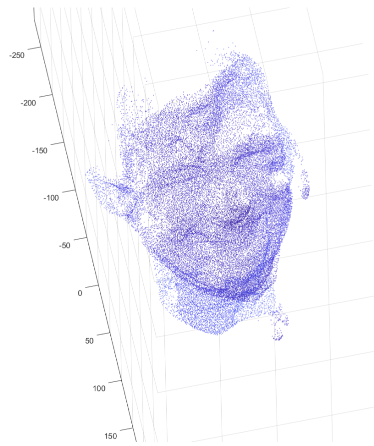


Fig. 7. Combined 3D point clouds

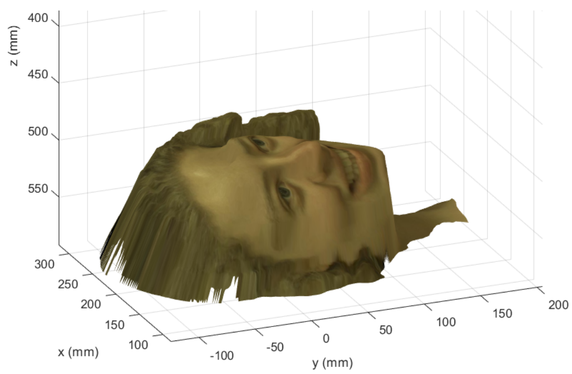


Fig. 8. 3D surface mesh of subject 2

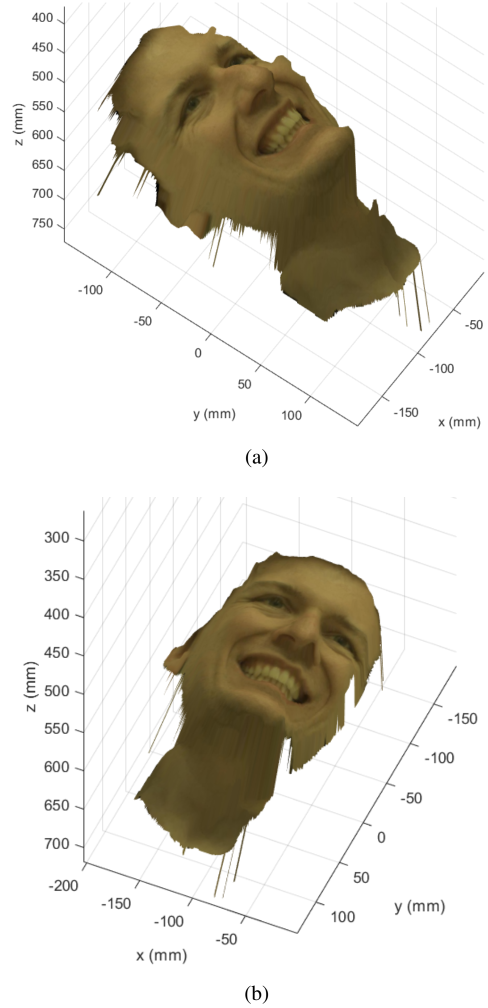


Fig. 9. 3D surface mesh of subject 1

and associated triangles are removed, before the 3D mesh is finally created with the function `triangulation` (see figure 8).

III. Results

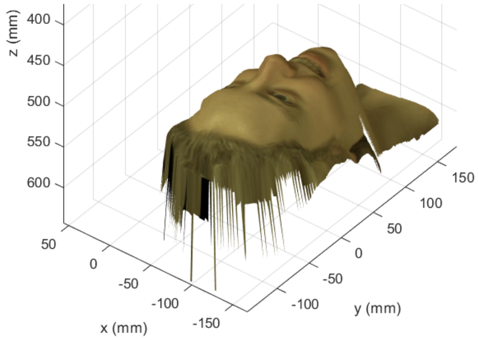
The results of the 3D surface mesh creation can be seen in figures 8 - 11.

IV. Discussion

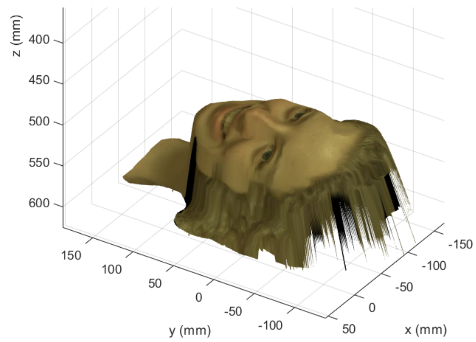
Interestingly, the *constant brightness assumption* seemed to not matter much in our solution of removing the background. We expected, for example, the forehead of the subjects (where reflection of a light source is visible) to bring problems as it would be hard for the k-means algorithm to separate them, but morphological operations allowed us to fill out these areas without filling in additional unwanted areas outside of the faces. However, the masks have very sharp edges, and are not very refined. Less morphological operations, or normalizing the brightness of the images, could perhaps improve the masks further. However, we can see that the forehead is a problem for the 3D surface, especially in subject 3, where the forehead is very flat and far back.

Another potential point of improvement is the matched points between the images that were found with the `Registration Estimator` app. These could for example be used to estimate disparities more accurately using a more fine-tuned algorithm than the `disparity`-function from `MATLAB`. Aligning the images through matching like this could also perhaps improve the disparity map and give less unreliable points in the most vital areas.

Before processing, steps could have been taken to improve the images of the subject. Using a background with a more natural color that doesn't almost match the skin tone of the subjects (so the contrast between the face and the background could increase) for example, using more uniform light to light up the subject's face, and to have a neutral shirt, like a tight white shirt with some space from the neck to allow more of the skin to come through. Or laying really close to the neck to have the whole face and parts of the neck like in this project. Another improvement that could have been done before the pictures were taken is the usage of a mob-cap, to facilitate the face extraction (since the main goal is to extract the face,



(a)



(b)

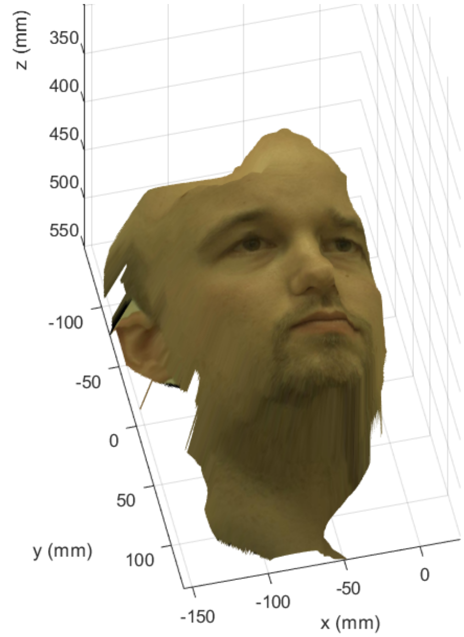
Fig. 10. 3D surface mesh of subject 2

not the hair; or in the case of subject 3, the disparities in the forehead will decrease with the usage of a mob-cap).

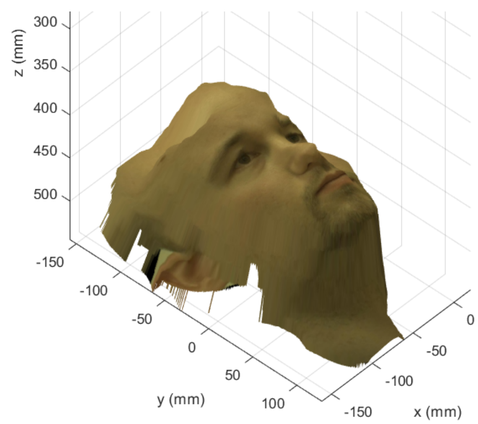
The results are not great, as our disparity maps are not accurately reflecting the depth of the face. For example images of the left side of the face will have very little information in the top right of the face, making these parts almost flat in the surface mesh while the rest of the face is lifted up more. Features like the chin and nose are however pretty well represented in most subjects, but the areas in the forehead and on the the outside of the face is not well represented. An attempt was made to median filter the disparity maps to increase the uniformity of the faces, so that they would not be like a rough texture. In the process however, some depth information have been wiped out, so that parts of the face become more flat than they otherwise would be.

V. Conclusion

In this project, a procedure was developed to handle a set of 3 images of a face to reconstruct a 3D surface of that face. The images were taken at the same time from around the face, and by using stereo rectification and matching, a disparity map of the faces were made, so that a point cloud and accordingly a 3D surface mesh could be made. The results are not as impressive in the field in general where more modern algorithms and approaches has been developed, such as using a trained classifier to reconstruct faces based on facial features that can be extracted, even from only 1 image. However, the results do give insight into the subject's geometry, so that a human can interpret information from the structure. In that regard, sufficient results have been achieved, though the procedure could easily be improved.



(a)



(b)

Fig. 11. 3D model of the face from subject 3

Appendix A
MATLAB main procedure script

```
% 3D Face Reconstruction
% Course: Image Processing & Computer Vision
% University of Twente
% April, 2018

% Authors: Per Arne s2049201 & Tiago Machado s2041596

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(genpath('functions'));
```

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

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

%% Camera Calibration

% Use the Stereo Camera Calibrator from the Computer Vision Toolbox
% to find Extrinsic and Intrisic parameters for the cameras
%{
stereoCameraCalibrator('data/calibration/calibrationSession1_LM.mat')
stereoCameraCalibrator('data/calibration/calibrationSession1_MR.mat')
%}

% Load stereo camera calibrated parameters
load('data/stereoParams_subject1_calib1.mat');
%load('data/stereoParams_subject1_calib2.mat');

figure; showExtrinsics(stereoParams_LM); % show extrinsic parameters of LM
figure; showExtrinsics(stereoParams_MR); % show extrinsic parameters of MR

%% Stereo rectification
% Rectify the images of the subject Left -> Middle -> Right
[imLeft_rect_full, imMiddleLeft_rect_full] = rectifyStereoImages(...
    im_left, im_middle, stereoParams_LM,'OutputView','full');

[imMiddleRight_rect_full, imRight_rect_full] = rectifyStereoImages(...
    im_middle, im_right, stereoParams_MR, 'OutputView','full');

% Use K-means clustering to extract a mask of the face of the subject
% This is based mostly on the Matlab Example:
%       'Color-Based Segmentation Using K-Means Clustering'
% The masks found with the method has been saved to .mat-files that we load
%{
[mask_left] = k_means_segment(im_left);
[mask_middle] = k_means_segment(im_middle);
mask_middle2 = mask_middle;
[mask_right] = k_means_segment(im_right);
%}

% Load masks of the subject found through k-means clustering
load('data/masks_subject1_calib1.mat'); % load left, middle, and right mask
%load('data/masks_subject1_calib2.mat'); % load left, middle, and right mask
%load('data/masks_subject2_calib1.mat'); % load left, middle, and right mask
%load('data/masks_subject3_calib1.mat'); % load left, middle, and right mask

mask_left = cat(3, mask_left, mask_left, mask_left);
im_left_masked = im_left;
im_left_masked(imcomplement(mask_left)) = 0;

mask_middle = cat(3, mask_middle, mask_middle, mask_middle);
im_middle_masked = im_middle;
im_middle_masked(imcomplement(mask_middle)) = 0;

mask_right = cat(3, mask_right, mask_right, mask_right);
im_right_masked = im_right;
im_right_masked(imcomplement(mask_right)) = 0;

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

[imMiddleRight_rect, imRight_rect] = rectifyStereoImages(im_middle_masked, ...
    im_right_masked, stereoParams_MR, 'OutputView','full');

%% Stereo matching between two images using Registration Estimator App
% Used the Registration Estimator app to verify that matched points are on
% epipolar lines. Exported the results as a function (for subject 1). Not
% used further.
[movingReg_LM]= registerImages_LM(imLeft_rect, imMiddleLeft_rect);
[movingReg_MR]= registerImages_MR(imMiddleRight_rect,imRight_rect);

%% Disparity map
% Create disparity map for the two image pairs
[map_LM, unreliable_LM] =  disparityMapAndUnreliable(imLeft_rect_full, ...
    imMiddleLeft_rect_full, imLeft_rect, 11 +subNum*10);
[map_MR, unreliable_MR] =  disparityMapAndUnreliable(imMiddleRight_rect_full, ...
imRight_rect_full, imMiddleRight_rect, 12 +subNum*10);

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

%% Create 3D meshes from point clouds
% Function based on UT syllabi
TR_LM = create_3D_mesh(map_LM, xyzPoints_LM, ...
    unreliable_LM, imLeft_rect);
TR_MR = create_3D_mesh(map_MR, xyzPoints_MR, ...
    unreliable_MR, imMiddleRight_rect);
```

APPENDIX B
MATLAB FUNCTION FOR K-MEANS CLUSTERING ALGORITHM

```
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/KMeansSegmentationExample')

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

%% Classify the Colors in 'a*b*' Space Using K-Means Clustering
ab = lab_face(:,:,2:3);  % Extract tha a*b* values
nrows = size(ab,1);
ncols = size(ab,2);
ab = reshape(ab,nrows*ncols,2);
nColors = 3;             % Skin colour, hair colour, background colour
n = 3;                   % Repeat clustering n times to avoid local minima

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

%% Label Every Pixel in the Image Using the Results from KMEANS
pixel_labels = reshape(cluster_idx,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_center,2);
[~, idx] = sort(mean_cluster_value);
skin_cluster_num = idx(3);

L = lab_face(:,:,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==false)) = 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

% Perform morphological operations to isolate 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
MATLAB FUNCTION FOR DISPARITY MAP AND UNREALIBLE MAP

```
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,'DisparityRange',...
            disparityRange, 'ContrastThreshold',0.9, ...
            'UniquenessThreshold',5,'DistanceThreshold',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,'DisparityRange',...
            disparityRange, 'ContrastThreshold',0.9, ...
            'UniquenessThreshold',5,'DistanceThreshold',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,'DisparityRange',...
            disparityRange, 'ContrastThreshold',0.6, ...
            'UniquenessThreshold',1,'DistanceThreshold',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,'DisparityRange',...
            disparityRange, 'ContrastThreshold',0.6, ...
            'UniquenessThreshold',1,'DistanceThreshold',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,'DisparityRange',...
            disparityRange, 'ContrastThreshold',1, ...
            'UniquenessThreshold',5,'DistanceThreshold',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,'DisparityRange',...
            disparityRange, 'ContrastThreshold',1, ...
            'UniquenessThreshold',5,'DistanceThreshold',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
MATLAB FUNCTION FOR 3D MESH SURFACE CREATION (FROM COURSE SYLLABI)

```
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
```

REFERENCES

[1] Ferdi van der Heijden, *Stereo: from pixels to 3D surface mesh*, University of twente - RAM , Netherlands: 2016.

[2] Oge Marques, *PRACTICAL IMAGE AND VIDEO PROCESSING USING MATLAB*, Florida Atlantic University, Wiley Press US: 2011