3D Face Reconstruction

Bishoy Essam Samir Yassa Gerges, bishoy.gerges@student.utwente.nl, s2921073, M-ROB
Panteleimon Manouselis, p.manouselis@student.utwente.nl, s3084493, M-ROB
Daniel Ciulei, d.ciulei@student.utwente.nl, s3017680, M-ROB

***Abstract* — This report presents the fundamental theory and necessary algorithms for stereo based 3D face reconstruction. The key process of the algorithm is depth estimation from pairs of images. The upper bodies of three different subjects were photographed from three angles, namely, left, middle and right. The presented algorithm combines the left and middle images, the middle and right images and the left and right images, producing three 3D meshes for each subject. The three newly created 3D meshes are then aligned, before different combinations of 3D mesh pairs are merged to form a complete mesh of the face as the final result**

***Keywords* —3D face reconstruction, 3D meshes, depth estimation, stereo vision, disparity maps.**

I. INTRODUCTION

Researchers around the world are increasingly interested in the problem of 3D facial reconstruction, as it has multiple applications in various fields. In the field of defense and security, 3D facial reconstruction is used to produce a biometric template for authentication, which can then be used to identify a criminal or provide access to authorized personnel. In addition, 3D reconstruction can be used for entertainment purposes, such as virtual reality video games, interactive video chatting and enhanced content creation.

The challenge of 3D reconstruction is to accurately extract data from 2D information (images/frames), into 3D space. In recent years, a plethora of techniques have been developed that tackles this problem. Deep learning methods that utilize Convolutional Neural Networks (CNN) or 3D Generative Adversarial Networks (3DGANs) and One-Shot Learning-based Reconstruction where a single image of an individual is used to recreate a 3D model are some of those techniques. In this project, we evaluated the parameters of the cameras, aligned the images and used well known formulas and algorithms to calculate the depth of points and their representation in 3D space.

The fundamental challenge presented in this paper is the reconstruction of a 3D face from a set of images taken from different angles. This problem proved to be difficult for several reasons: the background of the images had a similar color to the faces of the subjects, some 3D surface parts were visible in a single image but hidden in other images (occlusion), and the corresponding points in different images did not have identical color or gray level (the constant brightness assumption was violated).

In this report we will describe our solution to the aforementioned problem. Firstly, the methods used will be analyzed and explained in *section II*, and then the results will be presented in *section III*. Afterwards, a short discussion will follow in *section IV* before reaching our conclusion in section *V*.

II. METHODS AND MATERIALS

A. Materials

Software used throughout the project:

- `MATLAB R2022a`

Libraries/toolboxes used within the software:

- `Image Processing and Computer Vision toolbox`

B. Methods

To reconstruct a face from 2D images we were given as inputs a set of images containing three different subjects. Each subject has been photographed with five different facial expressions. Images of the subjects were taken from three different cameras that were positioned around the subjects, thus having the faces of the subjects seen from the left, middle and right sides (Figure 1).

Figure 1: Images of subject 1 seen from the three angles.

Furthermore, we were given sets of images for camera calibration, where each image contained a checkerboard which was placed in different positions within the images. Just like the pictures of the subjects, the pictures containing the checkerboards were also taken from the left, middle and right side.

We start off by obtaining the intrinsic and extrinsic camera parameters of the stereo cameras with the help of the checkerboard images. After the parameters have been found we need to remove the background and keep only the faces of the subjects. We apply stereo rectification on the image pairs of the subjects, as this will enable us to create a disparity map between two image pairs. With the disparity map created we can generate point clouds from these maps. Finally, we create 3D meshes from the point cloud to reconstruct the faces of the subjects.

We have divided the process of creating the 3D model of a

subject from 2D images in 6 steps:

a. Camera calibration

Camera calibration is the process of finding the intrinsic and extrinsic parameters of the camera. Knowing that we need to reconstruct a 3D model of the faces of the subjects, who's images were taken from three different positions, we have grouped the checkerboard images into two groups. The first group being the left and middle view, and the second is the middle and right view of the checkerboards. This results in obtaining the intrinsic and extrinsic camera parameters of three stereo pairs of cameras.

To do this, we used the `Stereo Camera Calibrator` found within `MATLAB`'s `Image Processing and Computer Vision toolbox`. Here, we import the left and right sets of images containing the checkboards, define the length of the checkboards – in our case these were of $10mm$ in length – and let the application load the environment. After this has been done, we calibrate the virtual camera to get the intrinsic and extrinsic camera parameters. The algorithm detects the connecting edges of the checkerboard squares and by knowing the length of the squares it calculates the parameters. At the end of the calibration process, the app also allows us to view the mean errors in pixels. Outliers can be removed to decrease the mean error. Recalibrating after the removal of the outliers showed significantly better results in overall mean reprojection error. While calibrating, a model of two radial distortion coefficients and zero tangential distortion coefficients were chosen, as that gave us the best results.

The algorithm which estimates the intrinsic parameters (focal length in x – f_x, and y – f_y direction, optical center in x – c_x and y – c_y direction, and the skew coefficient - s) stores the values in a matrix named K. Equation (1) shows the format of this calibration matrix.

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The calibration matrix represents a transformation from the camera's 3D coordinate system to the 2D image coordinates of the image. Equation (2) describe this transformation. Here x_c, y_c, z_c are the camera's coordinates and u, v are the coordinates in the image.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = [K|O_{3x1}] \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = M_{int} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \quad (2)$$

The extrinsic parameters are essentially a rotation and translation from the world's 3D coordinate system to the camera's 3D coordinate system, which is described in (3). Here R_{3x3} represents the rotation matrix, t the translation vector and x, y, z the 3D coordinates in the world's coordinate frame.

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R_{3x3} & t \\ O_{1x3} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = M_{ext} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (3)$$

This way we can map every pixel from the image into the world's coordinate system and calculate these parameters. Equation (4) describes this process.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = M_{int} M_{ext} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (4)$$

b. Background removal

To recreate a 3D face model of a person, we want to avoid both the background and any unnecessary information. Observing closely the images of the subjects, one can see that the images were taken in a room where the color of the subject's skin is closely related to the color of the walls. This increases the difficulty of the problem as we do not want to include the walls in the final result.

Saturation is related with how colorful an area is. With this concept in mind, we have resolved the aforementioned problem by firstly transforming the RGB image into the HSV color space. Here we have split the channels and threshold the S channel – i.e., the saturation of the image – in such a way that the pixels representing the background are differentiated from the pixels representing the subject's face. Figure 2 shows the histogram of the S channel in an image of a subject. This caption was taken with the help of `MATLAB`'s `Color Thresholder` app. As we can see, there are two peaks in the saturation channel. Up until the first peak almost all the bins are related to the background of the image, and consequently to the walls behind the person. After the first peak and from the next local minimum to the second peak and forward, one can find the pixels that relate to the subject's face. Consequently, we can write a simple algorithm to remove the background by thresholding the S channel (threshold value located in the local minimum, found between the two peaks of the histogram).

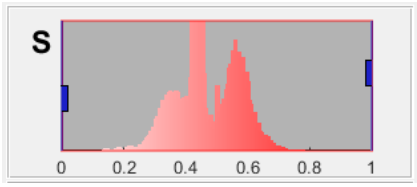

Figure 2: Histogram of the S channel of a subject. The caption shows two peaks, the first relating to the background of the image and the second one relates to the pixels of the subject's face.

Next step of our process is to perform color-based segmentation using the K-Means algorithm to cluster the pixels of the image in three different groups, namely, background, face and outfit of the subject. To cluster the pixels of the images we transform yet again the RGB image into the L*a*b* color space as it is easier to cluster the colors in two channels rather than in three. The same color information present in RGB color space can be found in the

a* and b* channels. Here we take the a* and b* channels and ignore the third dimension. The L channel, which only accounts for the perceived lightness level, does not contribute to clustering of pixels. We used `MATLAB`'s `imsegkmeans` built-in function to solve the clustering of the pixels.

Algorithm 1: K-means clustering

Input:

$D = d_1, d_2, \dots, d_n$	Set of elements
k	Number of desired clusters

Output:

k	Set of clusters

K-Means algorithm:

Arbitrarily choose k data-items from D as initial centroids;

repeat

assign each item d_i to the clusters which has the closest centroid;

calculate new mean for each cluster;

until convergence criteria is met

After the clustering of the pixels has been completed, we select the pixels which represent the subject's face and create a mask for the original image where only the subject's face is visible. On the mask we've also performed several morphological operations mainly to ensure edge connectivity, fill in the holes and smoothen the outline of the mask. Figure 3 shows an example of a mask applied onto the original image of the subject.

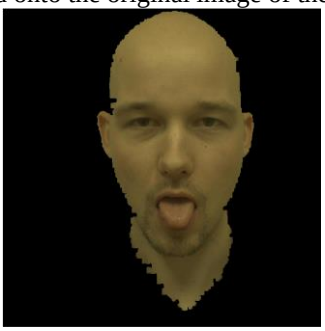

Figure 3: Masked image of subject 4.

c. Stereo rectification

The set-up of two identical aligned cameras facilitates the processing of stereo images. For a fully aligned stereo set-up, the following should be achieved: (a) the orientations of the cameras are equal ${}^1R_2 = I$, (b) the base line is horizontal ${}^1t_2 = [t_x 0\ 0]^T$, (c) the two optical axes are orthogonal to the base line, and the two calibration matrices are equal $K_1 = K_2 = K$. An aligned stereo camera set-up is shown in Figure 4. The rectification process was done in `MATLAB` with the function `rectifyStereoImages` from `MATLAB`'s `Computer Vision Toolbox`. The rectification process performed by this function is explained here. The first step in rectification is de-skewing, which rotates both cameras to look in the same direction. ${}^2R_1$, obtained during stereo calibration, rotates CCS_1 to CCS_2. This can also be done by rotating around an axis of rotation $\boldsymbol{e}$ by and angle ϕ written as $(\boldsymbol{e}, \phi)$. Aligning the cameras is obtained by rotating camera 1 with half the angle, $(\boldsymbol{e}, \frac{1}{2}\phi)$, and rotating camera 2 with the same angle in the opposite direction, $(\boldsymbol{e}, -\frac{1}{2}\phi)$. The rotation $(\boldsymbol{e}, \frac{1}{2}\phi)$ is represented by the rotation matrix $R_{de-skew}$. The next step is to align the cameras with the baseline by rotating the cameras by an angle ϕ_{align} around a normalized rotation axis defined as:

$$e_{align} = \frac{t_d \times e_x}{\|t_d \times e_x\|} \tag{5}$$

with $t_d = R_{de-skew}\,{}^1t_2$ the baseline after de-skewing and $e_x = [1\ 0\ 0]^T$ is the x-direction. The angle ϕ_{align} can be obtained from the dot product:

$$e_x^T t_d = \|e_x\|\|t_d\|\cos(\phi_{align}) \tag{6}$$

The alignment rotation is corresponding to the rotation matrix R_{align}.

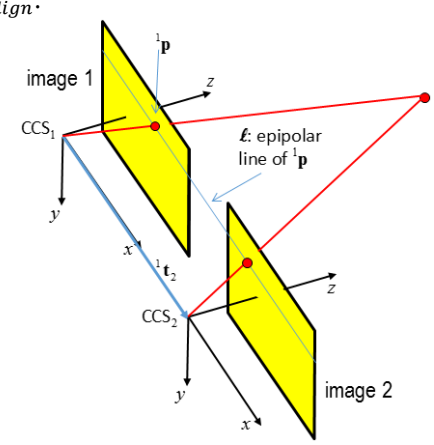

Figure 4: Rectified stereo camera set-up (figure from 3D Computer Vision, Ferdinan van der Heijden, 2016)

The final step is to unify the calibration matrices of both cameras. MATLAB assigns the following calibration matrix to both cameras:

$$K = \begin{bmatrix} d & 0 & K_1(1,3) \\ 0 & d & \frac{1}{2}(K_1(2,3) + K_2(2,3)) \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

with $d = \min(K_1(1,1),\ K_2(1,1))$. The rectification step can be applied by performing the following homographies to the stereo images:

$$\begin{aligned} H_1 &= K R_{align} R_{de-skew} K_1^{-1} \\ H_2 &= K R_{align} R_{de-skew}^T K_2^{-1} \end{aligned} \tag{8}$$

The function `rectifyStereoImages` also undistorts the image pairs. In our case the intrinsic stereo camera parameters were estimated with two radial and zero tangential distortion coefficients. Equation (9) describes the relation between the distorted and undistorted pixel location. Here u, v are the input camera pixel coordinates, x, y are the undistorted pixel locations and k_1, k_2 are the radial coefficients which were found in the previous step of stereo camera calibration.

$$\begin{cases} u = x(1 + k_1 r^2 + k_2 r^4) \\ v = y(1 + k_1 r^2 + k_2 r^4) \end{cases}, \quad where\ r^2 = x^2 + y^2 \qquad (9)$$

Algorithm 2: Stereo image pair rectification

Input:

$I_1,\ I_2$ Input images

${}^1t_2,\ {}^1R_2$ Known rotation and translation between cameras obtained from calibration

Output:

$I_{1_{rect}}, I_{2_{rect}}$ Rectified images

Rectification algorithm:

1. Rotate the cameras to look at the same direction.
2. Rotate the cameras again such that the row direction coincides with that of the baseline
3. Unify the virtual calibration matrices of both cameras.
4. Undistort images using calculated intrinsic parameters.

Figure 5: Pair of rectified images of left and middle view of subject 1 with matching keypoints lying on horizontal epipolar lines.

d. Stereo matching and disparity map

The aim of the stereo matching is to find corresponding points in a pair of rectified images. Corresponding points, after rectification, lie on the same row but with different column position in each image as shown in Figure 5.

The disparity is calculated as the difference in column positions. A pixel ${}^1p = [n\ m]^T$ in the first image will have a corresponding matched point in the second image ${}^2p = [u\ m]^T$. ${}^1p$ will then have a disparity $D = n - u$. A disparity map $D_i(n, m)$ maps a disparity value to each pixel in image i.

The stereo matching is highly dependent on the so-called Constant Brightness Assumption (CBA). For CBA to hold true, an intensity transformation was applied to the RGB channels of the second stereo image to have the same mean and standard deviation of that of the first image. For each channel of the second image, the following intensity transformation was applied:

$$g_2(x, y) = A f_2(x, y) + B \qquad (10)$$

where $g_2(x, y)$ is the output channel of the second image and $f_2(x, y)$ is the input channel of the second image. The coefficients A and B are calculated as follows:

$$\begin{aligned} A &= \frac{\sigma_1}{\sigma_2} \\ B &= \mu_1 - A\mu_2 \end{aligned} \qquad (11)$$

where μ and σ are the mean and standard deviation respectively. The subscript 1 and 2 is for image 1 and image 2 respectively.

A semiglobal matching (SGM) method [1] is used to match points and obtain the disparity map. This method takes into consideration disparity consistency along different directions (scan lines) which gives better results than row-by-row dynamic programming and is less computationally expensive than the global method. A custom-made function `calculateDisparityMap` is created which uses the built-in MATLAB function `disparitySGM` to obtain the disparity map of two rectified images using semiglobal matching. Additionally, the custom-made function outputs a reliability map which marks all the non-`NaN` disparities calculated by `disparitySGM`. The custom-made function also gives the flexibility to select the reference image (1 or 2) that the disparity map will be calculated with respect to.

After creating the disparity maps, they are further enhanced by detecting and filtering out most of the unreliable points. This was achieved and implemented using the custom-made function `enhanceDmap` which applies a number of procedures:

1. A consistency check has been applied based on the 'Symmetry Constraint'. That is, disparities calculated with reference to image 1, $D_1(n, m)$, and disparities calculated with reference to image 2, $D_2(u, m)$, should satisfy the following relation:
   $$D_1(n, m) = D_2(n - D_1(n, m), m) \qquad (12)$$
   Points that do not satisfy the above relation are marked as unreliable.
2. Using the created mask in background removal stage, any point that lies on the background is marked as unreliable.
3. A median filter has been applied to smooth the map and decrease the noise.

The results before and after enhancement are shown in Figure 6 and Figure 7 respectively.

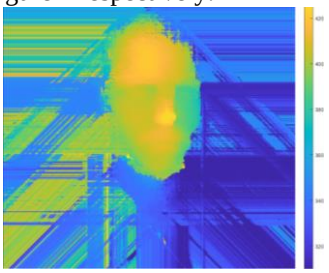

Figure 6: Disparity map before enhancement (disparity in pixel unit)

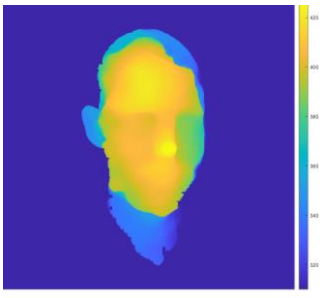

Figure 7: Disparity map after enhancement (disparity in pixel unit)

e. Point clouds

By utilizing the obtained disparity maps one can now calculate the 3D position of the images. More specifically, the function `reconstructScene` returns an array of 3D world point coordinates that reconstruct a scene from a disparity map. From this array, a `point cloud object` can be constructed, which one can then easily process, using build in `MATLAB` functions. We proceed to combine three pairs of images, namely, left-middle, middle-right, and left-right, into three point clouds which are then denoised. The resulting point clouds are then aligned and matched together by employing the ICP algorithm (`pcregistericp` function) and using as reference the point cloud created by the left and middle images. Finally, pairs of point clouds can be merged using the function `pcmerge`. Figure 8 shows the result of the ICP algorithm for cloud points left-middle and middle-right. Figure 9 depicts the merged cloud points. For the convenience of the reader, the cloud points have been colored based on the subject. Additionally, the transformation matrices that align the point clouds together are obtained and will be used later in aligning and merging the 3D meshes.

Algorithm 3: Iterative Closest Point

Input:

C_1, C_2 Input Cloud points (reference/fixed and source/moving)

Output:

$C_{2_{matched}}$, $RMSE$ Transformed point cloud that aligns with C_1 and root mean square error of the Euclidean distance between the inlier aligned points.

Iterative closest point algorithm:

repeat

1. For each point in the moving point cloud, match the closest point in the reference point cloud.
2. Estimate the combination of rotation and translation using a root mean square point to point distance metric minimization technique which will best align each source point to its match found in the previous step.
3. Transform the source points using the obtained transformation.

until

Maximum number of iterations over which the function attempts to make the two point clouds converge has been reached.

or

The average difference between estimated rigid transformations in the three most recent consecutive iterations is less than the specified tolerance values (i.e., define tolerance for the absolute difference in translation and in rotation, respectively, estimated in consecutive ICP iterations)

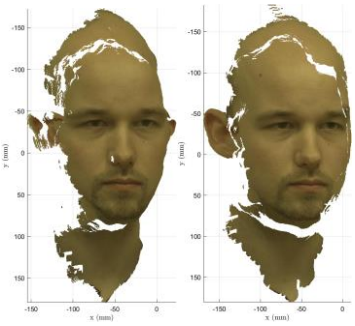

Figure 8: Aligned MR (left) and LM (right) Point Clouds

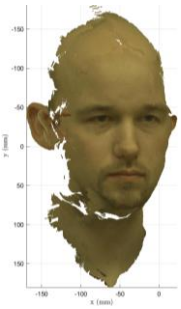

Figure 9: Merged LM and MR Point Clouds

f. Creating 3D meshes

The last step of our project is to create 3D surface meshes and merge them. By utilizing the code provided in the syllabi, we created a function named `make_3D_surface_mesh` where the disparity map, point cloud, unreliable map and provided images are transformed into a 3D surface mesh. The mesh is formed by a set of triangles, called faces, and corners, called vertices. A connectivity structure is defined, the triangles which contain an unreliable point are removed and the 3D points from the point cloud are associated with the points in the connectivity structure. Finally, the 3D surface mesh is created using the built-in function `triangulation`. Figure 10 shows the LM and MR 3D surface meshes, the 3D meshes have been colored based on the subject. The transformation matrices, obtained from the ICP algorithm in aligning the point clouds, have been used to merge the 3D meshes together. The final mesh after merging LM and MR meshes together is shown in Figure 11.

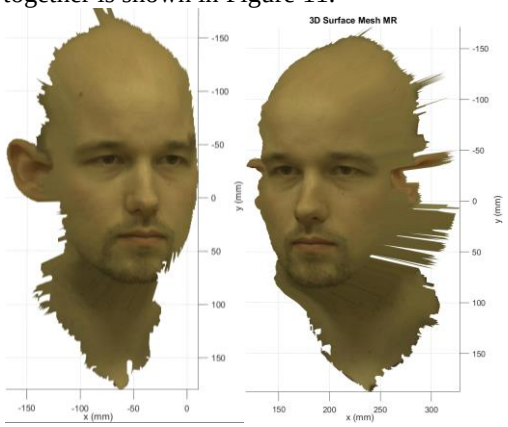

Figure 10: LM mesh (left) and MR mesh (right)

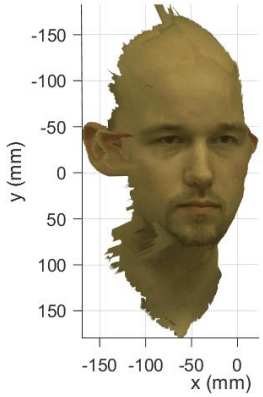

Figure 11: Merged 3D mesh

III. Results

In addition to the results shown in the previous section, the final results for different subjects are shown below for various merged meshes in Figure 12 to Figure 14. To understand the quality of the formed meshes, accuracy and robustness are measured and collected in Table I and Table II respectively. RMSE values between matched points of two meshes (e.g.: left-middle and middle-right) before and after alignment, are used as a measure of accuracy. The 3D position of the pixel at the center of the left eye of the subject was recorded in each mesh. To measure robustness, distance between this point at one mesh and the matching point in another mesh was recorded before and after alignment.

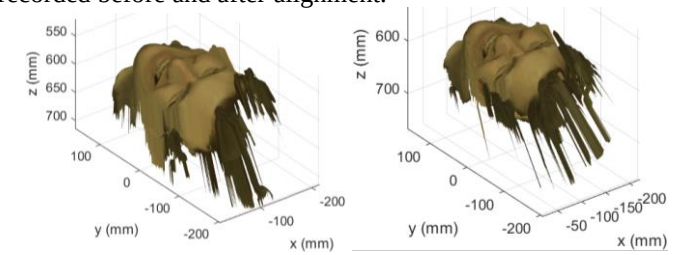

Figure 12: Subject 1 with left-middle and left-right meshes merged (left) and middle-right and left-right meshes merged (right)

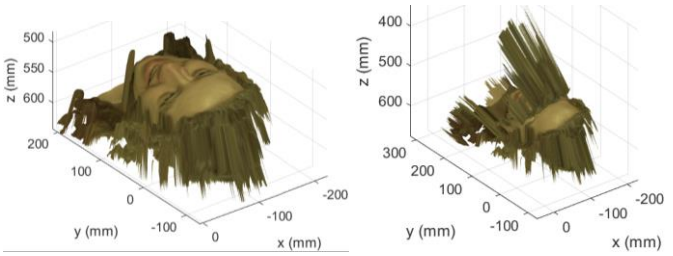

Figure 13: Subject 2 with left-middle and left-right meshes merged (left) and with left-middle and middle-right meshes merged (right)

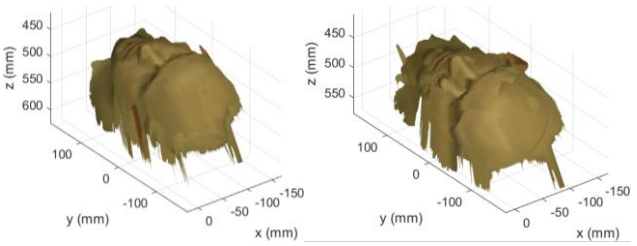

Figure 14: Subject 4 with left-middle and middle-right meshes merged (left) and with left-middle and left-right meshes merged

TABLE I
ACCURACY BETWEEN MESHES
(RMSE OF INLIER POINTS IN MM)

	Subject 1	Subject 2	Subject 4
Before alignment			
Middle Right to Left Middle	281.8	243.3	207.3
Left Right to Left Middle	154.5	133.7	117.0
After alignment			
Middle Right to Left Middle	6.1	31.0	5.4
Left Right to Left Middle	5.2	9.6	12.1

TABLE II
ROBUSTNESS BETWEEN MESHES
(DISTANCE (ERROR) OF A SELECTED POINT IN MM)

	Subject 1	Subject 2	Subject 4
Before alignment			
Middle Right to Left Middle	317.2	275.8	232.5
Left Right to Left Middle	203.7	184.7	164.7
Left Right to Middle Right	113.7	91.0	67.9
After alignment			
Middle Right to Left Middle	4.0	68.0	4.5
Left Right to Left Middle	2.9	7.6	27.0
Left Right to Middle Right	2.5	61.8	29.7

IV. Discussion

By examining Table I and Table II before and after alignment, one can observe that the ICP algorithm successfully decreases the mean error between point clouds.

Furthermore, it can be seen from the results that occlusion affects the quality of the mesh which is apparent at the edges of the face (e.g.: the ears).

A shortcoming of the result of the ICP is observed in Figure 13 where the left-middle and middle-right point clouds were aligned. The ICP succeeded in estimating an acceptable translation to align the middle-right point cloud to the left-middle point cloud but the rotation was completely wrong as the middle-right is shown to be reflected along the touching plane with the left-middle point cloud. An explanation for this could be that the ICP algorithm got stuck in an undesired local minimum. The fact that the middle-right point cloud for subject 2 is incorrectly aligned, agrees with the results in Table I and Table II where, for subject 2 after alignment, meshes containing the middle-right mesh have the largest error.

Another factor that in some cases decreased the quality of the merged 3D meshes, is the fact that the left-right stereo pair has a large disparity range (difference between maximum and minimum disparity). This large disparity range is greater than

128 which is the maximum disparity range supported by the MATLAB function `dispartiySGM`. This leads to wrongly estimating part of the disparity range which the `dispartiySGM` does not cover in its search space. This can be observed in Table I and Table II for subject 4 after alignment, where meshes having the left-right mesh have the highest error. It is also depicted in Figure 14 (right).

While addressing the background removal process, we noticed that the background color is very close to the skin color of the subjects which made it difficult to remove the background. Saturation thresholding might not be necessary, and can be completely avoided, if the background had a distinguishable color from the skin tone of the subjects. In this case, color-based segmentation with morphological transforms might be enough. Using better lighting conditions could also be used to preserve the Constant Brightness Assumption as much as possible.

A future recommendation could be to use feature matching between stereo pairs to align the point clouds more accurately. That is, only meaningful and important points (such as that of the face), should be considered.

V. Conclusion

This paper has presented a detailed procedure for 3D face reconstruction. The upper bodies of three people were photographed from three angles, namely, left, middle and right, and the resulting images were used to construct a 3D surface mesh of each person's face. By employing stereo rectification and stereo matching the disparity and unreliability maps of pairs of images were created and consequently we were able to create 3D point clouds. The point clouds were then aligned using the ICP algorithm, before being merged into 3D surface meshes. Because each stereo pair of images results in a partial representation of the face, several combinations of 3D meshes were merged together to obtain a complete 3D representation of the face. Furthermore, an educated attempt was made to tackle the Constant Brightness Assumption problem before creating the point clouds in order to improve the final meshes. Although the resulting 3D surface meshes are not as impressive as similar meshes created by modern algorithms that utilize deep learning architectures, one can still observe and extract useful information from the created 3D face such as relative depths or distances between different parts of the subject’s face.

APPENDIX A

MAIN MATLAB SCRIPT

```
clear variables
close all
clc
%% Preparing the environment
% Get the images of the subject.
subject = 2; % Get the images from suject (1, 2, 4)
folder = "subject" + num2str(subject); % Name of the folder
facial_expression = 1; % Select facial expression (1, 2, 3, 4, 5)

im_set_left = imageSet(convertStringsToChars(folder + "/" + folder + "_Left"));
im_set_middle = imageSet(convertStringsToChars(folder + "/" + folder + "_Middle"));
im_set_right = imageSet(convertStringsToChars(folder + "/" + folder + "_Right"));

im_left = im2double(read(im_set_left, facial_expression)); % Left image of the subject
im_middle = im2double(read(im_set_middle, facial_expression)); % Middle image of the subject
im_right = im2double(read(im_set_right, facial_expression)); % Right image of the subject

%% Camera calibration
% The images were taken with a stereo camera.
% For calibration purposes we have pairs of two pairs of images to get a
% more accurate description of the camera model. These are left-middle
% and middle-right.
% To get the intrisic and extrinsic parameters of the cameras we used the
% stereoCameraCalibrator app.

% Load in the parameters which were obtained from the app
load("Camera Data\calib2StereoParamsLM.mat");
load("Camera Data\calib2StereoParamsMR.mat");
load("Camera Data\calib2StereoParamsLR.mat");
% figure; showExtrinsics(stereoParamsLM); % show extrinsic parameters of LM
% figure; showExtrinsics(stereoParamsMR); % show extrinsic parameters of MR

%% Background remover
% Remove the background from the photo and extract only the face of the
% subject
face_left_mask = background_remover(im_left);
face_middle_mask = background_remover(im_middle);
face_right_mask = background_remover(im_right);

% Plot the three masked images (Left, middle, right)
figure;
subplot(1,3,1);imshow(face_left_mask.*im_left);title('Left');
subplot(1,3,2);imshow(face_middle_mask.*im_middle);title('Middle');
subplot(1,3,3);imshow(face_right_mask.*im_right);title('Right');
%% Stereo Rectification
% Reproject image planes onto a common plane parallel to the line between
% camera center
% Rectify the images of the subject in this order: left -> middle,
% middle -> right
[im_left_rect, im_middle_left_rect] = rectifyStereoImages(...
    im_left, im_middle, calib2StereoParamsLM,'OutputView','full');

[im_middle_right_rect, im_right_rect] = rectifyStereoImages(...
    im_middle, im_right, calib2StereoParamsMR, 'OutputView','full');

% Do the same operation for the masks Left -> Middle -> Right
[mask_left_rect, mask_middle_left_rect] = rectifyStereoImages(...
    face_left_mask, face_middle_mask, calib2StereoParamsLM,'OutputView','full');

[mask_middle_right_rect, mask_right_rect] = rectifyStereoImages(...
    face_middle_mask, face_right_mask, calib2StereoParamsMR, 'OutputView','full');

[im_LR_l_rect, im_LR_r_rect, reprojection_mat_LR] = rectifyStereoImages(...
    im_left, im_right, calib2StereoParamsLR,'OutputView','full');

[im_LR_l_rect_mask, im_LR_r_rect_mask] = rectifyStereoImages(...
    face_left_mask, face_right_mask, calib2StereoParamsLR,'OutputView','full');

% Unify brightness
im_middle_left_rect = mimic_colorspace(im_middle_left_rect ,im_left_rect);
```

```
im_right_rect = mimic_colorspace(im_right_rect ,im_middle_right_rect);
im_LR_r_rect = mimic_colorspace(im_LR_r_rect ,im_LR_l_rect);

% Create rectified images cell arrays (for ease of access afterwards):
% column1: first rectified image, column2: second rectified image
LM_rec = {im_left_rect im_middle_left_rect};   % Left-Middle
MR_rec = {im_middle_right_rect, im_right_rect}; % Middle-Right
LR_rec = {im_LR_l_rect, im_LR_r_rect};          % Left-Right

% Create rectified masks cell arrays (for ease of access afterwards):
% column1: mask of first image, column2: mask of second image
LM_mask = {mask_left_rect, mask_middle_left_rect};      % Left-Middle
MR_mask = {mask_middle_right_rect, mask_right_rect};    % Middle-Right
LR_mask = {im_LR_l_rect_mask, im_LR_r_rect_mask};       % Left-Right

% Create rectified masked imgs cell arrays (for ease of access afterwards):
% column1: first masked image, column2: second masked image
LM_masked = {LM_rec{1,1}.*LM_mask{1,1}, LM_rec{1,2}.*LM_mask{1,2}};% Left-Middle
MR_masked = {MR_rec{1,1}.*MR_mask{1,1}, MR_rec{1,2}.*MR_mask{1,2}};% Middle-Right
LR_masked = {LR_rec{1,1}.*LR_mask{1,1}, LR_rec{1,2}.*LR_mask{1,2}};% Left-Right

%% Stereo Matching and disparity maps
ref = 1; % dmap to be calculated wrt 1:first image, 2:second image

% Minimum disparity for each subject was manually calculated. This was
% done by inputing the Anaglyph to imtool, then measuring the distance
% between the deepest two corresponding points (of the neck for example).
% mdisp -> minimum disparity
mdisp_LM = [250 280 0 310]; %[subject1 subject2 dummy subject4]
mdisp_MR = [260 265 0 325]; %[subject1 subject2 dummy subject4]
mdisp_LR = [530 560 0 691]; %[subject1 subject2 dummy subject4]

% Disparity range can be then calculated. (maxD-minD) should be <= 128 and
% divisible by 8. So, to achieve max range, maxD = minD +128.
drange_LM = [mdisp_LM(subject), mdisp_LM(subject) + 128];
drange_MR = [mdisp_MR(subject), mdisp_MR(subject) + 128];
drange_LR = [mdisp_LR(subject), mdisp_LR(subject) + 128];

% Left-middle disparity map
[dmap_LM, reliability_map_LM]= calculateDisparityMap(LM_masked, drange_LM, ref);
% Left-middle enhance disparity map
[dmap_LM, reliability_map_LM] = enhanceDmap(dmap_LM, reliability_map_LM, ...
    LM_masked, drange_LM, LM_mask, ref);
% show_pc(dmap_LM, drange_LM,calib2StereoParamsLM, LM_masked{ref});

% Middle-right disparity map
[dmap_MR, reliability_map_MR]= calculateDisparityMap(MR_masked, drange_MR, ref);
% Middle-right enhance disparity map
[dmap_MR, reliability_map_MR] = enhanceDmap(dmap_MR, reliability_map_MR, ...
    MR_masked, drange_MR, MR_mask, ref);
% show_pc(dmap_MR, drange_MR,calib2StereoParamsMR, MR_masked{ref});

% Left-right disparity map
[dmap_LR, reliability_map_LR]= calculateDisparityMap(LR_masked, drange_LR, ref);
% Left-right enhance disparity map
[dmap_LR, reliability_map_LR] = enhanceDmap(dmap_LR, reliability_map_LR, ...
    LR_masked, drange_LR, LR_mask, ref);
% show_pc(dmap_LR, drange_LR,calib2StereoParamsLR, LR_masked{ref});

%% Create point clouds
[pc_LM, pc_LM_denoised] = Point_Clouds(dmap_LM, calib2StereoParamsLM, LM_masked{ref});
[pc_MR, pc_MR_denoised] = Point_Clouds(dmap_MR, calib2StereoParamsMR, MR_masked{ref});
[pc_LR, pc_LR_denoised] = Point_Clouds(dmap_LR, calib2StereoParamsLR, LR_masked{ref});
%% Point clouds merging
pc_LM_aligned = pc_LM_denoised; % For convention (aligned)

% % Align point clouds by using ICP algorithm
% [tform_MR_to_global,pc_MR_aligned,rmseMR_LM,originalrmseMR_LM] =
pcregistericp_err(pc_MR_denoised,pc_LM_denoised,MaxIterations=150,verbose = true);
% [tform_LR_to_global,pc_LR_aligned,rmseLR_LM,originalrmseLR_LM] =
pcregistericp_err(pc_LR_denoised,pc_LM_denoised,MaxIterations=150,verbose = true);

% Load saved point clouds (for time optimization)
pc = "PointClouds" + num2str(subject);
pc_MR_aligned = pcread(pc +"/pc_MR_aligned.ply");
pc_LR_aligned = pcread(pc +"/pc_LR_aligned.ply");
```

```
% Load save transformation matrices (for time optimization)
tforms = "tforms" + num2str(subject); % Name of the folder
load(tforms+"/tform_LR_to_global.mat");
load(tforms+"/tform_MR_to_global.mat");

% Show Point Cloud LM and new Point Cloud MR (aligned with Point Cloud LM)
figure; subplot(1,2,1);pcshow(pc_MR_aligned, BackgroundColor=[1,1,1]);
xlabel('x (mm)','interpreter','latex');ylabel('y (mm)','interpreter','latex');zlabel('z
(mm)','interpreter','latex');
subplot(1,2,2);pcshow(pc_LM_aligned,BackgroundColor=[1,1,1]);
xlabel('x (mm)','interpreter','latex');ylabel('y (mm)','interpreter','latex');zlabel('z
(mm)','interpreter','latex');
sgtitle("$$New$$ $$Point$$ $$Cloud$$ $$MR$$ $$and$$ $$Point$$ $$Cloud$$ $$LM$$",'interpreter','latex');
% Show Point Cloud LM and new Point Cloud LR (alligned with Point Cloud LR)
figure; subplot(1,2,1);pcshow(pc_LR_aligned, BackgroundColor=[1,1,1]);
xlabel('x (mm)','interpreter','latex');ylabel('y (mm)','interpreter','latex');zlabel('z
(mm)','interpreter','latex');
subplot(1,2,2);pcshow(pc_LM_aligned, BackgroundColor=[1,1,1]);
xlabel('x (mm)','interpreter','latex');ylabel('y (mm)','interpreter','latex');zlabel('z
(mm)','interpreter','latex');
sgtitle("$$New$$ $$Point$$ $$Cloud$$ $$LR$$ $$and$$ $$Point$$ $$Cloud$$ $$LM$$",'interpreter','latex');

%% Merge point clouds
merge_size = 0.02; % unit: meter

% Merging LM and MR
pc_global = pcmerge(pc_LM_aligned, pc_MR_aligned, merge_size);
figure;pcshow(pc_global, BackgroundColor=[1,1,1]);
xlabel('x (mm)','interpreter','latex');ylabel('y (mm)','interpreter','latex');zlabel('z
(mm)','interpreter','latex');
title('Merged point clouds LM and MR', 'Interpreter','latex');
%% Create 3D surface meshes
[Tri_LM, img_LM_line] = make_3D_surface_mesh(dmap_LM, pc_LM, ...
    ~reliability_map_LM, LM_rec{ref});
title('3D Surface Mesh LM')

[Tri_MR, img_MR_line] = make_3D_surface_mesh(dmap_MR, pc_MR, ...
    ~reliability_map_MR, MR_rec{ref});
title('3D Surface Mesh MR')

[Tri_LR, img_LR_line] = make_3D_surface_mesh(dmap_LR, pc_LR, ...
    ~reliability_map_LR, LR_rec{ref});
title('3D Surface Mesh LR')
%% Merge 3D surface meshes
% Align each mesh to the left-middle 3D mesh
P_MR_aligned = transformPointsForward(tform_MR_to_global,Tri_MR.Points);
Tri_MR_aligned = triangulation(Tri_MR.ConnectivityList, P_MR_aligned);

P_LR_aligned = transformPointsForward(tform_LR_to_global,Tri_LR.Points);
Tri_LR_aligned = triangulation(Tri_LR.ConnectivityList, P_LR_aligned);

% Merge MR mesh to LM mesh to obtain LMMR mesh
merged_points = [Tri_LM.Points;Tri_MR_aligned.Points];
merged_conn = [Tri_LM.ConnectivityList;...
    Tri_MR_aligned.ConnectivityList+ size(Tri_LM.Points,1)];
Tri_LMMR = triangulation(merged_conn, merged_points);
merg_color_line_LMMR = [img_LM_line;img_MR_line];

% Merge LR mesh to LM mesh to obtain LMLR mesh
merged_points = [Tri_LM.Points;Tri_LR_aligned.Points];
merged_conn = [Tri_LM.ConnectivityList;...
    Tri_LR_aligned.ConnectivityList+ size(Tri_LM.Points,1)];
Tri_LMLR = triangulation(merged_conn, merged_points);
merg_color_line_LMLR = [img_LM_line;img_LR_line];

% Merge LR mesh to MR mesh to obtain MRLR mesh
merged_points = [Tri_MR_aligned.Points;Tri_LR_aligned.Points];
merged_conn = [Tri_MR_aligned.ConnectivityList;...
    Tri_LR_aligned.ConnectivityList+ size(Tri_MR_aligned.Points,1)];
Tri_MRLR = triangulation(merged_conn, merged_points);
merg_color_line_MRLR = [img_MR_line;img_LR_line];

% Merge LR mesh to LMMR mesh to obtain LMMRLR mesh
merged_points = [Tri_LMMR.Points;Tri_LR_aligned.Points];
merged_conn = [Tri_LMMR.ConnectivityList;...
```

```
    Tri_LR_aligned.ConnectivityList+ size(Tri_LMMR.Points,1)];
Tri_LMMRLR = triangulation(merged_conn, merged_points);
merg_color_line_LMMRLR = [merg_color_line_LMMR;img_LR_line];

% Visualize merged mesh
figure;
subplot(2,2,1);
TM = trimesh(Tri_LMMR);
title('Merged LMMR mesh');
set(TM,'FaceVertexCData',merg_color_line_LMMR); % 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

subplot(2,2,2);
TM = trimesh(Tri_LMLR);
title('Merged LMLR mesh');
set(TM,'FaceVertexCData',merg_color_line_LMLR); % 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

subplot(2,2,3);
TM = trimesh(Tri_MRLR);
title('Merged MRLR mesh');
set(TM,'FaceVertexCData',merg_color_line_MRLR); % 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

subplot(2,2,4);
TM = trimesh(Tri_LMMRLR);
title('Merged LMMRLR mesh');
set(TM,'FaceVertexCData',merg_color_line_LMMRLR); % 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


% %% Accuracy
% % Initial RMSE
% disp('RMSE before alignment')
% disp(originalrmseMR_LM(1))
% disp(originalrmseLR_LM(1))
```

```
%
%
% % After alignment
% disp('RMSE after alignment')
% disp(rmseMR_LM)
% disp(rmseLR_LM)

%% Robustness
% Load eye location in each point cloud
%[LM MR LR]
load("Robustness/subj1_eye_after.mat");
load("Robustness/subj1_eye_before.mat");
load("Robustness/subj2_eye_after.mat");
load("Robustness/subj2_eye_before.mat");
load("Robustness/subj4_eye_after.mat");
load("Robustness/subj4_eye_before.mat");
% Distances before
%[LM-MR MR-LR LM-LR]
subj1_distance_before = [...
    sqrt(sum((subj1_eye_before{1}-subj1_eye_before{2}).^2));
    sqrt(sum((subj1_eye_before{2}-subj1_eye_before{3}).^2));
    sqrt(sum((subj1_eye_before{1}-subj1_eye_before{3}).^2));];

subj2_distance_before = [...
    sqrt(sum((subj2_eye_before{1}-subj2_eye_before{2}).^2));
    sqrt(sum((subj2_eye_before{2}-subj2_eye_before{3}).^2));
    sqrt(sum((subj2_eye_before{1}-subj2_eye_before{3}).^2));];

subj4_distance_before = [...
    sqrt(sum((subj4_eye_before{1}-subj4_eye_before{2}).^2));
    sqrt(sum((subj4_eye_before{2}-subj4_eye_before{3}).^2));
    sqrt(sum((subj4_eye_before{1}-subj4_eye_before{3}).^2));];

% Distances after
%[LM-MR MR-LR LM-LR]
subj1_distance_after = [...
    sqrt(sum((subj1_eye_after{1}-subj1_eye_after{2}).^2));
    sqrt(sum((subj1_eye_after{2}-subj1_eye_after{3}).^2));
    sqrt(sum((subj1_eye_after{1}-subj1_eye_after{3}).^2));];

subj2_distance_after = [...
    sqrt(sum((subj2_eye_after{1}-subj2_eye_after{2}).^2));
    sqrt(sum((subj2_eye_after{2}-subj2_eye_after{3}).^2));
    sqrt(sum((subj2_eye_after{1}-subj2_eye_after{3}).^2));];

subj4_distance_after = [...
    sqrt(sum((subj4_eye_after{1}-subj4_eye_after{2}).^2));
    sqrt(sum((subj4_eye_after{2}-subj4_eye_after{3}).^2));
    sqrt(sum((subj4_eye_after{1}-subj4_eye_after{3}).^2));];
```

MATLAB FUNCTION FOR BACKGROUND REMOVAL (K-MEANS)

```
function face_mask = background_remover(im)
%% Step 1 - Saturation thresholding for background detection
% The subjects have been photographed in a room where the color of the walls
% are close to the color of their skin
% For better color segmentation firstly we convert the RGB image to an HSV
% image and threshold the saturation level
% Convert from RGB to HSV color space
im_hsv = rgb2hsv(im);

% Split the channels
s = im_hsv(:, :, 2);

% Get number of bins and values for them
[N, edges] = histcounts(s);
idx = find(N == max(N));

% Get the first local minimum and it's index
lmin = islocalmin(N(:, idx:size(N, 2)));
lmin_idx = find(lmin == 1);
% Don't get the first value, have some offset
```

```
lmin_idx = lmin_idx(5);

% Get the threshold value
threshold_value = edges(:, idx + lmin_idx);

% Create mask based on chosen histogram thresholds
s_mask = (im_hsv(:,:,1) >= 0 ) & (im_hsv(:,:,1) <= 1) & ...
    (im_hsv(:,:,2) >= threshold_value ) & (im_hsv(:,:,2) <= 1) & ...
    (im_hsv(:,:,3) >= 0 ) & (im_hsv(:,:,3) <= 1);

% Initialize output masked image based on input image
im_masked = im;

% Set background pixels where BW is false to zero
im_masked(repmat(~s_mask,[1 1 3])) = 0;
% figure; imshow(im_masked);

%% Step 2 - Color based segmentation using K-means to find the person
% The first color based segmentation is used to find the person, meaning
% face + outfit
% Convert the input image into L*a*b* space color
im_lab = rgb2lab(im);

% Extract a*b*
ab = im_lab(:, :, 2:3);
ab = im2single(ab);

% Get the pixel labels for 3 colors (background, outfit and face)
pixel_labels = imsegkmeans(ab, 3);

% The pixel label with the second highest ocurances should be the face, the
% third one is the outfit
counted_pixel_labels = [sum((pixel_labels(:) == 1));
                        sum((pixel_labels(:) == 2));
                        sum((pixel_labels(:) == 3))];

sorted_pixel_labels = sort(counted_pixel_labels,'descend');

% Extract the face and outfit pixels
face_pixels = find(counted_pixel_labels == sorted_pixel_labels(2));
outfit_pixels = find(counted_pixel_labels == sorted_pixel_labels(3));

face_mask = (pixel_labels == face_pixels);
outfit_mask = (pixel_labels == outfit_pixels);

% Create a binary mask to for the person
face_mask = face_mask | outfit_mask;

% figure; imshow(labeloverlay(im,pixel_labels));

%% Step 3 - Perform morphological operations to get the mask of the person
% Extract the greatest area from the opened face mask which is the face
face_mask = bwareafilt(face_mask, 1);
% figure; imshow(face_mask);

% Fill in the holes and gaps in the mask
face_mask = imfill(face_mask,'holes');
% figure; imshow(face_mask);

%
face = im;
face(repmat(face_mask,[1,1,3])==0) = 0;
% result = labeloverlay(im,face_mask);
% figure; imshow(result);
im(repmat(face_mask,[1,1,3])==0)=0;
% figure; imshow(im);

%% Step 4 - Color based segmentation using K-means for face extraction
% Now that the blackground is completly black, the face and outifit of the
% subjects can be separted more easily
% Convert the input image into L*a*b* space color
im_lab = rgb2lab(im);

% Extract a*b*
ab = im_lab(:, :, 2:3);
ab = im2single(ab);
```

```
% Get the pixel labels for 3 colors (background, outfit and face)
pixel_labels = imsegkmeans(ab, 3);

% The pixel label with the second highest ocurances should be the face
counted_pixel_labels = [sum((pixel_labels(:) == 1));
                        sum((pixel_labels(:) == 2));
                        sum((pixel_labels(:) == 3))];

sorted_pixel_labels = sort(counted_pixel_labels,'descend');

% Extract the face pixels
face_pixels = find(counted_pixel_labels == sorted_pixel_labels(2));
face_mask = (pixel_labels == face_pixels);

% figure; imshow(labeloverlay(im,pixel_labels));

%% Step 5 - Perform morphological operations to get the mask of subject's face
% Extract the greatest area from the opened face mask which is the face
face_mask = bwareafilt(face_mask, 1);
% figure; imshow(face_mask);

se = strel('disk', 5);
face_mask = imerode(face_mask, se);
% figure; imshow(face_mask);

% Fill in the holes and gaps in the mask
face_mask = imfill(face_mask,'holes');
% figure; imshow(face_mask);

% Dilate the image with horizontal and vertical lines to ensure
% connectivity
se = strel('line', 10, 90);
se_horizontal = strel('line', 10, 0);
face_mask = imdilate(face_mask, [se se_horizontal]);
% figure; imshow(face_mask);

% Just to be sure
face_mask = imfill(face_mask,'holes');
end
```

MATLAB FUNCTIONS FOR CONSTANT BRIGHTNESS ASSUMPTION (INTENSITY TRANSFORMATION)

```
function adjustedimage = mimic_colorspace(img,reference_img)
adjustedimage = zeros(size(img),'like', img);
    for i=1:1:3
        adjustedimage(:,:,i) = mimic_layer(img(:,:,i),reference_img(:,:,i));
    end
end


function adjustedlayer = mimic_layer(img,reference_img)
mean_reference = mean2(reference_img);
std_reference = std2(reference_img);
mean_img= mean2(img);
std_img= std2(img);

sigma_desired = std_reference;
mean_desired = mean_reference;

a = sigma_desired/std_img;
b = mean_desired - a*mean_img;

adjustedlayer = a*img + b;
end
```

MATLAB FUNCTION FOR DISPARITY AND RELIABILITY MAPS

```
function [disparityMap, reliability_map] = calculateDisparityMap(img_pair,...
    disparityRange, reference)
% Convert images to greyscale
img_1_grey = rgb2gray(img_pair{1});
img_2_grey = rgb2gray(img_pair{2});

% If reference = 2 (disparity map to be calculate wrt 2nd image), then:
if reference == 2
```

```
    % disparity range has to be negative
    disparityRange = -disparityRange;
    disparityRange([1 2]) = disparityRange([2 1]);

    % the order of images has to be swaped
    img_1_grey = rgb2gray(img_pair{2});
    img_2_grey = rgb2gray(img_pair{1});
end

% Calculate disparity map
disparityMap = disparitySGM(img_1_grey, img_2_grey,...
    'DisparityRange',disparityRange,'UniquenessThreshold',0);

% If disparity map was calcualted wrt 2nd image, then disparities are
% negative. They are to be multiplied by -1 to convert them to positive:
if reference == 2
    disparityMap = -disparityMap;
end
reliability_map = ~isnan(disparityMap);
reliability_map(disparityMap<0) = 0;
end
```

MATLAB FUNCTION FOR ENHANCING THE RELIABILITY MAP

```
function [D_map_out, reliability_map] = enhanceDmap(D_map, reliability_map,...
    img_pair, disparityRange, mask,reference)
D_map_out = D_map;
%% First enhancement: Calculate second disparity map
% To filter out points that have different disparities in both maps

if reference == 1
    % If the provided D_map is wrt first image, then calculate the
    % D_map wrt the second image
    D_map2 = calculateDisparityMap(img_pair, disparityRange,2);
    % For a point in disparity map1, calculate its disparity in disparity map2
    corresponding_disparities = D_map;
    for y=1:size(D_map,1)
        for x=1:size(D_map,2)
            if floor(x-D_map(y,x)) <= 0 || isnan(D_map(y,x))
                corresponding_disparities(y,x) = -realmax;
                continue
            end
            % notice the -ve sign in x-D_map(y,x)
            corresponding_disparities(y,x) = D_map2(y,floor(x-D_map(y,x)));
        end
    end

else % reference == 2
    % If the provided D_map is wrt second image, then calculate the
    % D_map wrt the first image
    D_map2 = calculateDisparityMap(img_pair, disparityRange,1);
    % For a point in disparity map1, calculate its disparity in disparity map2
    corresponding_disparities = D_map;
    for y=1:size(D_map,1)
        for x=1:size(D_map,2)
            if floor(x-D_map(y,x)) <= 0 || isnan(D_map(y,x))
                corresponding_disparities(y,x) = -realmax;
                continue
            end
            % notice the +ve sign in x+D_map(y,x)
            corresponding_disparities(y,x) = D_map2(y,floor(x+D_map(y,x)));
        end
    end
end
% Mark points of same disparities in both maps as reliable
reliability_map(abs(D_map - corresponding_disparities)<1) = 1;

%% Second enhancement: Any point that lie on the background is unreliable
reliability_map = reliability_map.*mask{reference};

%% Third enhancement: Remove holes in the face
reliability_map = imfill(reliability_map,'holes');
D_map_out(~reliability_map) = -realmax;

%% Fourth enhancement: Using Median Filtering
D_map_out = medfilt2(D_map_out, [40 40],'symmetric');
```

```
% Update reliability_map:
reliability_map(D_map_out==-realmax) = 0;
reliability_map(D_map_out==-Inf) = 0;
reliability_map(D_map_out==Inf) = 0;
reliability_map(isnan(D_map_out)) = 0;
D_map_out(~reliability_map) = NaN;
end
```

MATLAB FUNCTION THAT CREATES AND DENOISES POINT CLOUDS

```
function [ptCloud_pair1, ptCloud_pair1_new] = Point_Clouds(disparityMap_pair1,reprojectionMatrix_pair1,
im_rect)
point3d_pair1 = reconstructScene(disparityMap_pair1,reprojectionMatrix_pair1); % Based on whether we return
reprojection Matrix from Stero Rectification (i.e., rectifyStereoImages)
%or reconstructScene(disparityMap_pair1,StereoParameters_pair1)
ptCloud_pair1 = pointCloud(point3d_pair1,"Color",im_rect);

% Remove noise (outliers) from pointCloud
%PreserveStructure=true in order to keep the correct dimension. Correct
%dimensions will be needed in a subsequent function provided by IPCV
%lectures that creates the 3D meshes
ptCloud_pair1_new = pcdenoise(ptCloud_pair1,PreserveStructure=true);
figure; subplot(1,2,1);pcshow(ptCloud_pair1, BackgroundColor=[1,1,1]);% Compare before and after noise
removal
xlabel('x (mm)','interpreter','latex');ylabel('y (mm)','interpreter','latex');zlabel('z
(mm)','interpreter','latex');
subplot(1,2,2);pcshow(ptCloud_pair1_new, BackgroundColor=[1,1,1]);
xlabel('x (mm)','interpreter','latex');ylabel('y (mm)','interpreter','latex');zlabel('z
(mm)','interpreter','latex');
sgtitle("$$Before$$ $$and$$ $$after$$ $$noise$$ $$removal$$",'interpreter','latex');
end
```

MATLAB FUNCTION THAT CREATES THE 3D SURFACE MESHES (CODE ADAPTED FROM STEREO: FROM PIXELS TO 3D SURFACE MESH)

```
function [TR, J1l] = make_3D_surface_mesh(disparityMap, ptCloud_pair1, unreliable, J1)
%% create a connectivity structure
% Code adapted from Stereo: from pixels to 3D surface mesh by Ferdi van der Heijden
[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(ptCloud_pair1.Location,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
%% Unionizing the array of unreliable points used previously with any new unreliable points of the point
cloud
ind_unreliable = find(unreliable(indI));% get the linear indices of unreliable 3D points
new_unreliable = find(isnan(pcl(:,1)));
new_unreliable = union(ind_unreliable,new_unreliable);
%% remove the unreliable points and the associated triangles
imem = ismember(TRI(:),new_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
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] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and Mutual Information," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005 .