Exercise 2: 3D surface reconstruction

The final project will be about the design of computer vision application for the quantitative assessment of knee mobility. Such a quantification is useful for the assessment of Anterior Cruciate Ligament injury (ACL). As a preparation for the final project, this exercise introduces to 3D surface reconstruction using the images from a stereo camera.

Available data and software:

The data and software that will be used for the exercise are:

- Images that were acquired with a stereo camera and a projector. The images show the upper and lower leg of a volunteer, while bending his knee. The projector is just to cast a red random pattern on the surface. This is needed to have a sufficient amount of texture for the stereo matching.
- Attached to the surface of the upper and lower leg there are also some markers. These markers are needed for key point detection and matching, needed in exercise 3.
- For this exercise, there is an image from a left camera: **`ACLtestL05.png`** and a corresponding image from a right camera: **`ACLtestR05.png`**.
- Camera calibration images. A number of calibration images are given in the files **`caliL%.png`**, and **`caliR%.png`**. These images are acquired with a checkerboard hold in different positions and orientations before the stereo camera.

1. Camera calibration[1]

1.1. Use the stereo camera calibration app from MATLAB to calibrate the stereo set-up. The size of the checkers on the board is 40 mm. Apply the calibration. Inspect the reprojection errors. If one image shows large errors relative to others, discard the image and calibrate again. Export the object with the stereo parameters to the work space (give it the name **`camp`**), and export also the object with estimation errors. Save these objects to file for later use.

Camera 1:

principle point px:	633.2384	uncertainty px	1.3073
principle point py:	360.8340	uncertainty py	0.9870
focal length dx	919.4129	uncertainty dx	1.8543
focal length dy	915.0867	uncertainty dy	1.8976

Camera 2:

principle point px:	624.2837	uncertainty px	1.4743
principle point py:	358.9291	uncertainty py	1.0231
focal length dx	919.3448	uncertainty dx	1.7615
focal length dy	914.8480	uncertainty dy	1.7930

[1] Note about the notation in mathematical symbols: the left camera will be indexed by 1, and the right camera by 2. Thus, camera 1 denotes the left camera, and camera 2 denotes the right camera.

Extrinsic parameters (= inter-camera geometry)

Rotation matrix ${}^{2}\mathbf{R}_{1}$ • Don't forget to correct for the transposed form of matlab! • See the footnote on page 13 of the syllabus			base line vector ${}^{2}\mathbf{t}_{1}$	uncertainty base line vector
0.992	-0.0036	0.0410	-137.8552	0.2875
0.0030	0.9999	0.0146	-1.8875	0.1701
-0.0410	-0.0145	0.9991	4.2540	0.8859

1.2. Matlab provides ${}^{2}\mathbf{R}_{1}$ and ${}^{2}\mathbf{t}_{1}$ (camera 1 expressed in coordinates of camera 2). But actually, we need camera 2 expressed in coordinates of camera 1, i.e. ${}^{1}\mathbf{R}_{2}$ and ${}^{1}\mathbf{t}_{2}$. See section 2.1 of the syllabus "3D surface reconstruction". Calculate this matrix and vector. Assuming that the uncertainties in the rotation matrix are negligible, determine also the uncertainty in ${}^{1}\mathbf{t}_{2}$.

Rotation matrix ${}^{1}\mathbf{R}_{2}$			base line vector ${}^{1}\mathbf{t}_{2}$	uncertainty base line vector
0.9992	0.0030	-0.0410	137.9186	-0.2514
-0.0036	0.9999	-0.0145	1.4596	-0.1562
0.0410	0.0146	0.9991	1.4297	-0.8993

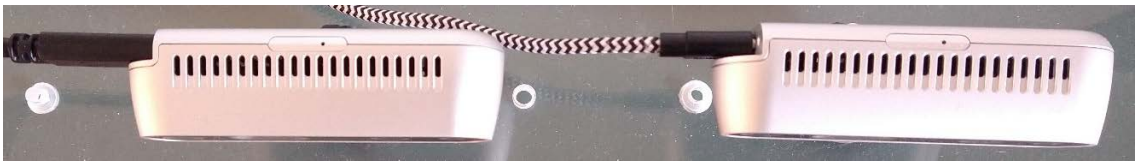
What is the physical unit in which the elements of ${}^{1}\mathbf{t}_{2}$ are expressed?	mm
What is the physical unit in which px, py, dx, and dy are expressed?	pixels

1.4. The function `rot2axang` converts the rotation matrix into its rotation axis/angle representation. Apply this function to the rotation matrix ${}^{1}\mathbf{R}_{2}$ (remember: should you transpose it first, or not?).

Rotation axis			angle (degrees)
0.3339	-0.9397	-0.0745	2.5022

1.5. The pictures below show a front and top view of the stereo camera. Does the found baseline vector and the rotation axis/angle make sense? Explain.

The calculation makes sense according to the picture below. As we can see, the baseline vector from the edge of camera 1 (around 2 cm) to the edge of camera 2 (around 16 cm) gives us approximately 14 centimeters, or 140mm. When we calculate the magnitude of baseline vector calculated above (by taking the norm), the distance is around 137.9 mm, closely equal to our value.
For the rotation axis, from the top view, we can observe that camera 2 is not in the same orientation as camera 1. Camera 2 needed to be rotatate in the y-axis as the left side of camera 2 is behind comparing with the right side and rotated in the positive direction of the x-axis as camera 2 view a bit higher than 1. The angle of 2.5 degree is also appropriate as there are not significant different in rotation.



2. Epipolar geometry

2.1. Calculate the essential matrix $\mathbf{E}$ using one of the found rotation matrices and the base line vector. Next, calculate the fundamental matrix $\mathbf{F}$. See Section 2.2.
Fundamental matrix:

-7.6807e-08	5.0235e-06	3.5495e-04
1.6742e-06	2.4075e-06	-0.1527
-0.0022	0.1466	3.1785

2.2. Calculate the positions of the two epipoles expressed in pixel coordinates. See Section 2.1 and 2.2. Hint: to calculate the null space of matrix you can use the MATLAB function **null**.

epipole 1: $^{1}\mathbf{q}$ epipole 2: $^{2}\mathbf{q}$

89324e+04	-2.9168e+04
1.2951e+03	-46.9971

2.3. Do the results of 2.2 make sense? Explain.

Yes the values are appropriate. As we can see the x axis of camera 1 epipole being positive while the camera 2 in the negative direction, this shows a certain skewness between 2 cameras leaning inside and not parallel to the baseline vector. The positive in y axis of epipole of camera 1 and negative in camera 2 showing camera 2 might be tilted upward. However, the high values indicate that the epipoles lay outside the image plane and the angle of rotation is not too high (2.5 degree as we found above).

2.4. Is the stereo camera set-up approximately aligned such as described in Section 3.1? Motivate by using the calculated positions of the epipoles.

No, the camera is not fully alligned as describe in Section 3.1. The epipoles (not 0 in y direction) appears hat they are not positioned at infinity. This proves that the baseline vector is not parallel to the image plane. Therefore, the stereo camera rectification is necessary. Besides epipoles, we can also observe that the rotation matrices do not closely equal identity matrix and the baseline vector still has y and z component

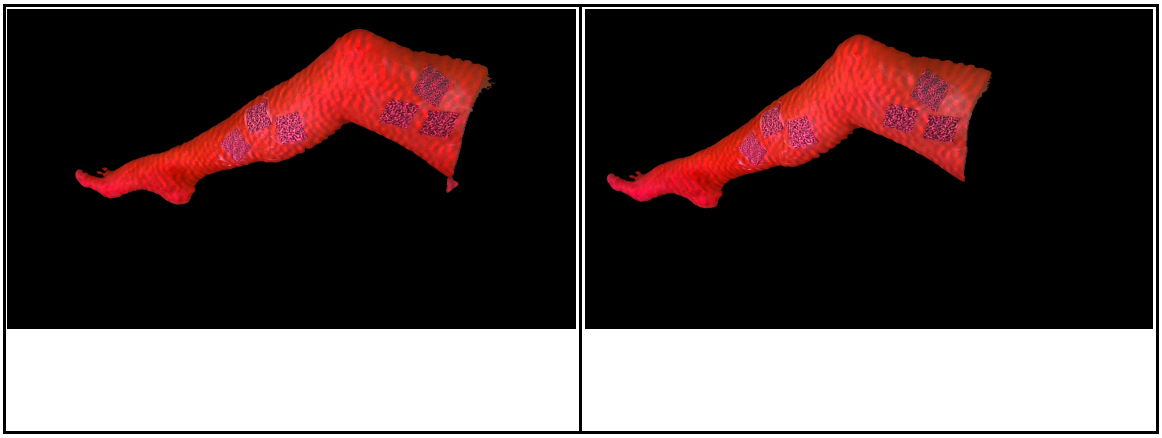
3. Data preparation

Before diving into the topics of stereo rectification, stereo matching, and surface reconstruction, the images need some preprocessing: segmentation to get the relevant parts in the images.

3.1. Read the left and right stereo image. Since we need image processing, the first step is to convert the images to type **`double`** using the function **`im2double`**. In this text, the resulting images will be referred to by **`im1`** (=left image) and **`im2`** (= right image).

3.2. **Segmentation.** We are only interested in the surface of the leg. To get rid of the background, it would be advantageous to make this background fully black, so that during disparity estimation these background pixels are invalidated. Use the **`color thresholder`** app of MATLAB to create a mask for the detection of the foreground pixels in the **`im1`** image. The Appendix of the current document contains the procedure to do so. In that procedure a function **`createMask.m`** will be created. The function was designed for segmenting **`im1`**, but you can apply the same function to segment **`im2`** as well. The function creates masks for each of the two images. Use these masks to get RGB images, **`im1`** and **`im2`**, with all background pixels turned into black. Show the resulting images below.

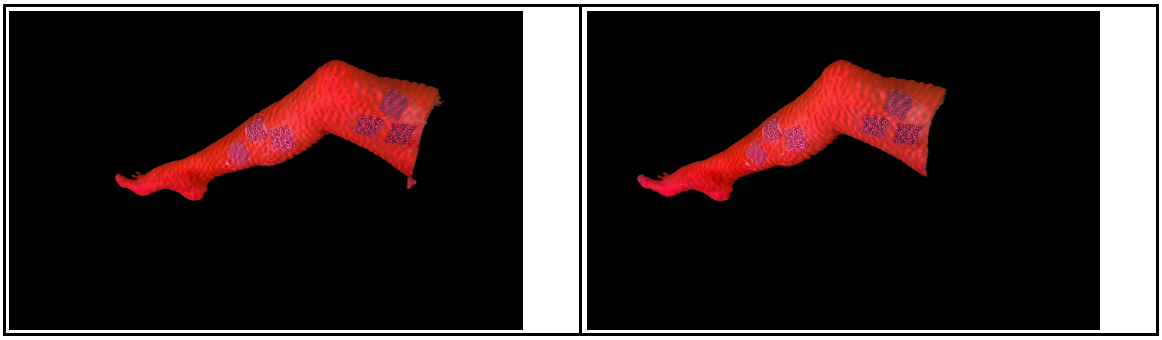
`im1` (left) **`im2`** (right)



4. Stereo rectification

4.1. Apply lens undistortion and rectification by using the function **`rectifyStereoImages`**. Use the output view '**`full`**'. Write the resulting images to file, and show them below.

Image 1 (left) Image 2 (right)



Check visually whether the images are really rectified. You can use, for instance, the tool **`cpselect`**. If they are not rectified something went wrong[2], and you have to debug your code.

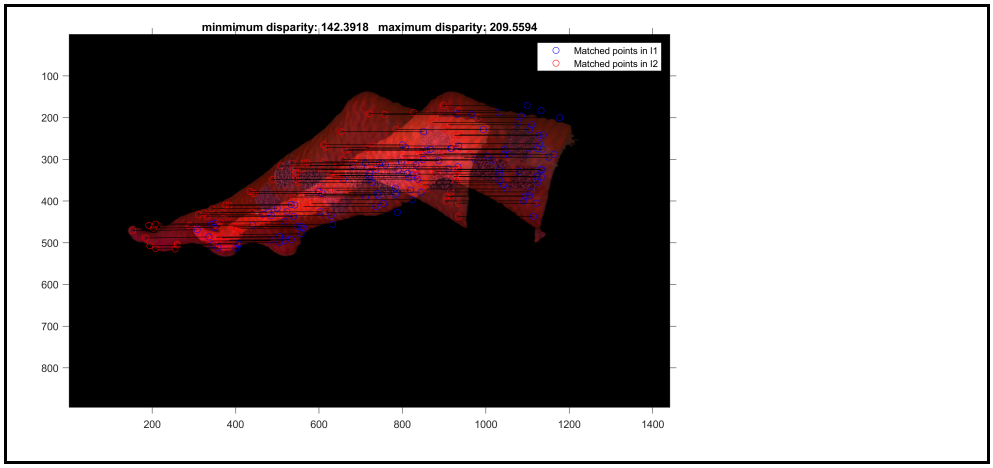
[2] Maybe you swapped left and right?

4.2. The matlab function **`ut_check_rect`** checks whether the rectification was successful. First, the function finds two sets of surf key points in both images, and then matches them. Each pair of corresponding points is connected by a displacement vector. The directions of these vectors, expressed as the angle between the vector and the horizontal axis, are calculated. The mean and root means square (rms) of these angles are estimated so as to provide a measure of rectification. In case of a perfect rectification, all angles should be zero. Angles are expressed in degrees.
Use **`ut_check_rect`** to find the mean and rms of the angles. Visualize the result (see the help of the function).

Statistics of the angles of displacement vectors (degrees):

mean: `-0.2375` rms: `0.4005`

Save the figure window that popped up after calling **`ut_check_rect`**, and insert it below:



4.3. The function **`ut_getStereo_rect_parms`** calculates the two rotation matrices for the virtual rotation of the left and right cameras. It also calculates the new calibration matrix and the new baseline vector. Apply the function.

Rotation matrix for rectifying camera 1	Rotation matrix for rectifying camera 2
0.9999 0.0106 0.0104 -0.0107 0.9999 0.0071 -0.0103 -0.0072 0.9999	0.9994 0.0137 -0.0308 -0.0139 0.9999 -0.0070 0.0307 0.0074 0.9995

What is the rotation matrix ${}^{1}\mathbf{R}_{2}$ **after** rectification?	What is the rotation matrix ${}^{2}\mathbf{R}_{1}$ **after** rectification?
1 0 0 0 1 0 0 0 1	1 0 0 0 1 0 0 0 1

calibration matrix $\mathbf{K}_1$ **after** rectification	calibration matrix $\mathbf{K}_2$ **after** rectification
919.3448 0 633.2384 0 919.3448 359.8816 0 0 1.0000	919.3448 0 633.2384 0 919.3448 359.8816 0 0 1.0000

baseline vector ${}^1\mathbf{t}_2$ **after** rectification (include units)	137.9337mm	-4.9636e-13mm	4.1589e-13mm
baseline vector ${}^2\mathbf{t}_1$ **after** rectification	-137.9337mm	4.9636e-13mm	-4.1589e-13mm

5. Disparity

The MATLAB function **`disparitySGM`** takes as an input two rectified images and calculates the disparity map. The first argument in the function is the reference image. Two options can be added to make a reliable disparity map and to identify unreliable disparities in it.

5.1. One of the options is to adjust the range of allowable disparities. Inspect the result of question 4.1 to find the maximal disparity. For that, use the tool **`cpselect`**, and pinpoint the surface patch on the limb that is closest to the camera, and a surface patch on the limb (still visible in both images) with has the largest depth. Note: in question 5.2, you have to iterate a number of times to finetune the range.

minimal disparity: 164.3750 maximal disparity: 206.500

Does this meet your expectation? Explain. Hint: see eq 45 in the syllabus.

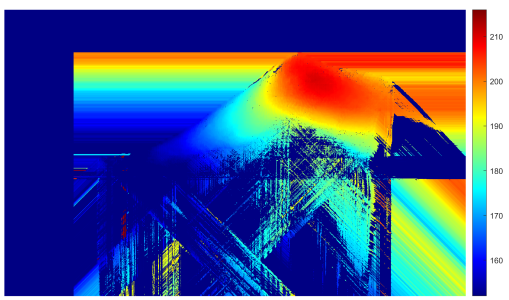
Using the focal length dx=919 from part 1 and distance T between 2 camera = 138mm from previous section, we substitute these values to equation 45 in the syllabus. As we calculated that the object is between 614mm=61.5cm to 770mm=77cm away the pinholes, these number are expected base on our visual impressions of the picture.

With the option `'DisparityRange'` the range of disparity can be specified. However, there are some limitations on this range. Read the help of the function to check this out. Then choose the range that is suitable for this application. Apply the function and visualize the result with:

```
figure; imagesc(disparityMap,[Dmin Dmax]); % display map with scaled colors
colormap(jet); colorbar;                   % choose a nice color scale
```

5.2. The other option that can be added is `'UniquenessThreshold'`. Read the help of the function. Then apply this option such that the resulting disparity map looks best.

Disparity map:



5.3. There will be pixels for which the disparity cannot be calculated, or for which this would be unreliable. These pixels are indicated in the disparity map by **NaN**'s (not a number). Mention at least 4 reasons/situations for which this would be the case.

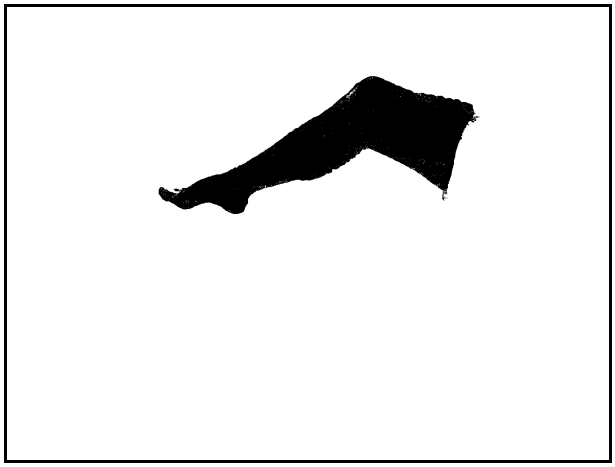
1. Occlude pixels which is only appeared in one picture and not in other picture are considered unreliable. Same situation happens with the background
2. The sum of absolute difference (SAD) between a point in the image 1 and 2 is not inside the range of the disparity.
3. Texture in the window is too weak for local matching.
4. The contrast (estimated by the Sobel edge operator) is too large , such as on the body edge, for reliable disparity

5.4. To identify the pixels that are unreliable, we create a new mask image, called: **unreliable**. This is a logical image the same size as the disparity map, and in which each pixel indicates whether the corresponding disparity is not reliable. Obviously, the disparities that are outside the allowable range are unreliable. And, as said before, pixels that are **NaN** are not reliable. Thus, these pixels should also be set in the map **unreliable**. Create the map **unreliable**.

5.5. A second step to identify unreliable disparities is to use the mask of the first image, as obtained in question 3. All pixels that are not in the foreground can be regarded as not reliable. Extend the code such that these pixels are also added to the mask **unreliable**. Note: to perform a logical 'or' operation on arrays, use the **|** operator.

5.6. If the processing steps of 5.4 and 5.5 are sufficiently successful, then the largest reliable region (connected component) corresponds to the limb region. This largest reliable region can be found by application of the MATLAB function **bwareafilt** to the inverse of the unreliable mask, i.e. to **~unreliable**. All pixels not belonging to this largest reliable region can be set to unreliable, and thus added (that is, logical or) to the mask **unreliable**.

The mask **unreliable** after application of the process in question 5.6:



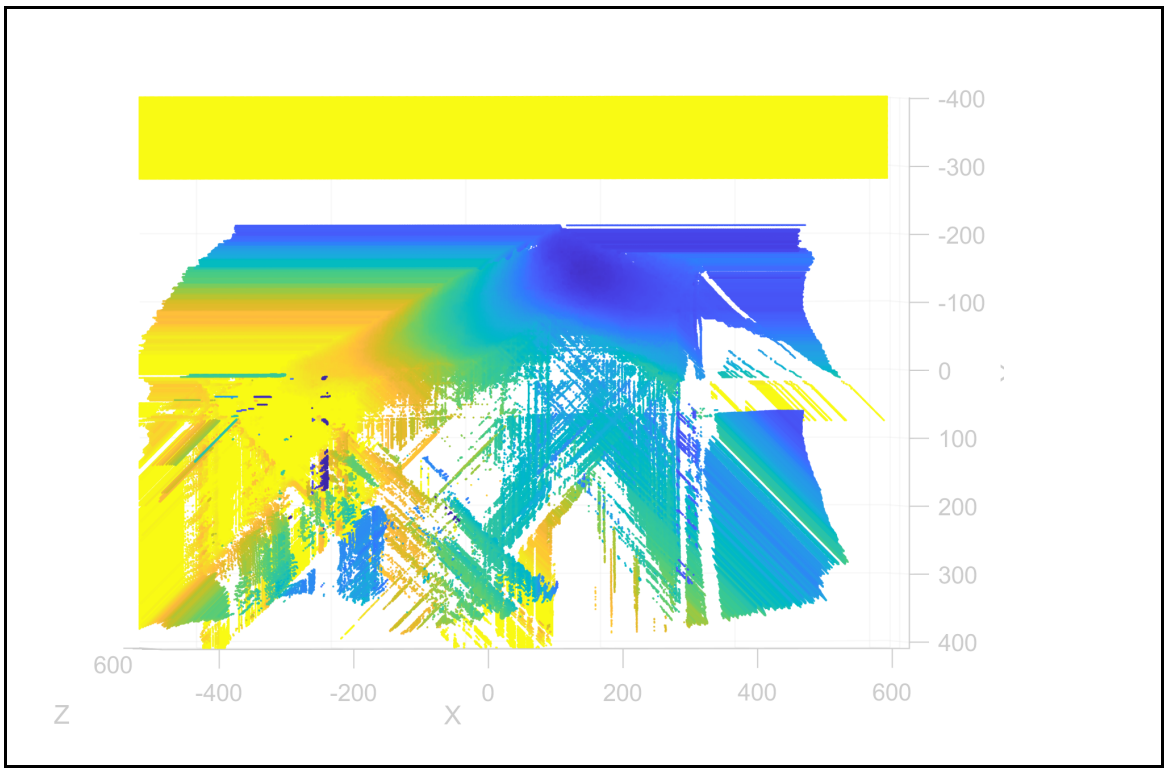
6. Point cloud

6.1. The Matlab function **reconstructScene** takes as input a disparity map and the calibration parameters and returns an array of 3D world point coordinates of the surface. Apply this function and show[3] the result with the function **pcshow**.

What is the size of the resulting point cloud array? `894x1441x3`

[3] Useful functions to control the view are **campos** and **camup**. Don't forget the labels at the axes, and their units.

Resulting point cloud:



6.2. The function **`reconstructScene`** returns an array of 3D world point coordinates, i.e. xyz points. This set of 3D points is called a *point cloud*. Apart from this, MATLAB also defines a *class* **`pointCloud`**. Objects from this class also holds such a set of xyz points, but they can also hold more information (properties), like the colors of the points. In addition, the object holds specific functions (methods) such as **`removeInvalidPoints`**.

Create an instance of the class **`pointCloud`** which contains the point cloud from 6.1, but without the unreliable pixels. For this, use the mask **`unreliable`** to remove these points from the set. Hint: to do so, first reshape the point cloud to linear arrays likewise the example on page 15 of the syllabus.

What is the size of the resulting point cloud array? `137665x3`

The advantage of the class **`pointCloud`** is that it provides access to a number of properties of the point cloud, like:

- Find the nearest neighbors of a point in the cloud.
- Find the points within a radius of a point in the cloud.
- Find the points within a ROI.

Besides, there are also a number of functions that can be applied to the point cloud:

- Merge two point clouds
- Remove noise.
- Downsample the point cloud.
- Register two point clouds using ICP.

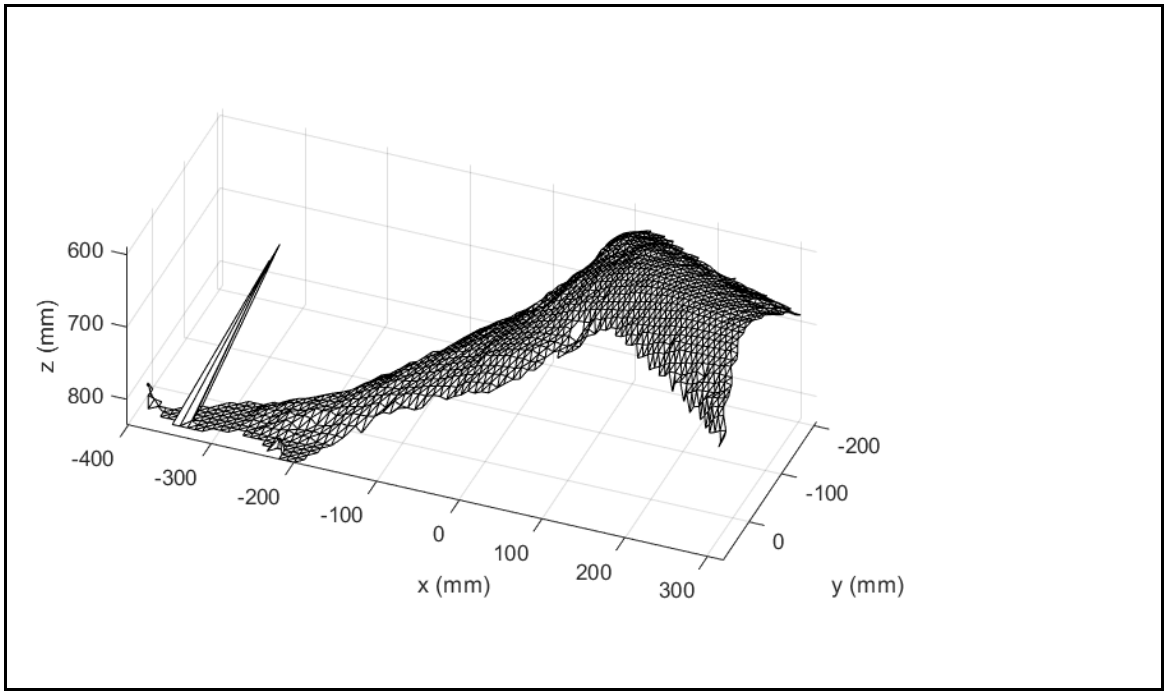
- Rigid geometrical transform.
- Estimate normal vectors of point cloud.
- Fit a plane to the point cloud.
- Ffit a cylinder to the point cloud.
- Fit a cylinder to the point cloud

7. 3D Surface mesh

7.1. Create a 3D surface mesh such as shown in the last figure of the syllabus (**`facecolor`** must be white, and **`edgecolor`** must be black). Select the resolution such that the triangles are not too large, but still visible. You can recycle the code given on page 15 of the syllabus.

Selected resolution: `10`

Resulting mesh



8. Take home messages = preparation for the oral exam

Prepare yourself for the oral exam by making a list of new concepts that you have learned in this exercise. Make sure that you know these concepts, and that you understand them. Examples of questions that may be asked during the oral are:

		level
1.	What is an epipole? What are epiplor lines?	knowledge
2.	What is an essential matrix? And what is a fundamental matrix? What is the difference?	understanding
3.	What do we want to achieve with stereo rectification? What is/are the benefit(s) of stereo rectification? How do we achieve stereo rectification? What will be the rotation matrix and baseline vector after stereo rectification?	understanding
4.	What is the essence of stereo matching and what are the difficulties?	understanding
5.	What is the principle of triangulation?	understanding

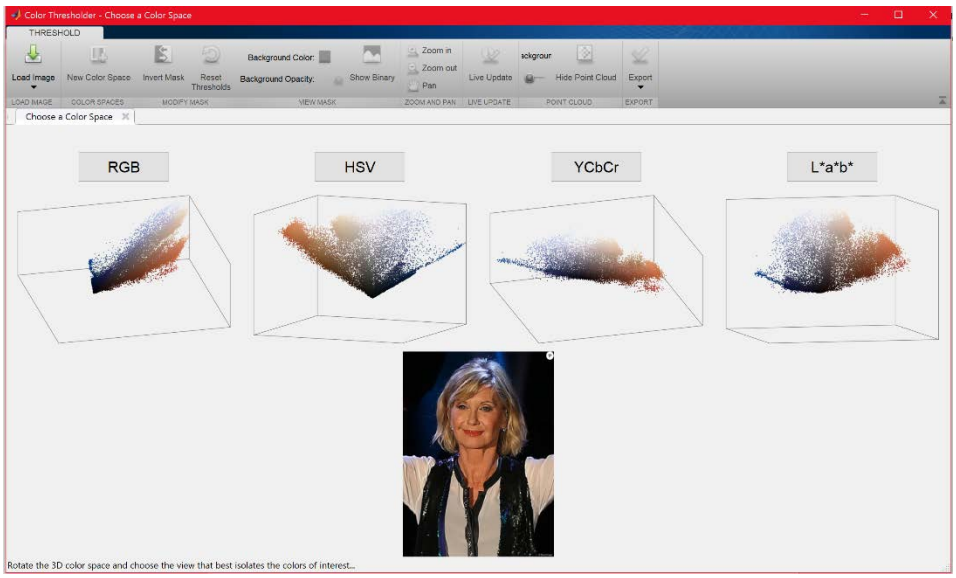
Matlab listing:

```
%%%%% 3DCV Exercise 2: 3D surface reconstruction %%%%%%%%%%%%%

%% Summary
% 3D surface reconstruction using:
% - left and right camera images of bending leg with patterns
% - Camera calbiration for left and right camera
clear all; close all

%% Part 1: Camera calibration
% 1.1 Using stereo camera calibration app in Matlab
% The size of chessboard is 40mm
% save camp camp
% save estimationErrors estimationErrors
load camp
load estimationErrors

% 1.2 Calculate 1R2 and 1t2
R21=(camp.RotationOfCamera2)'; % transpose bcz of Matlab default
R12=R21';
t21=(camp.TranslationOfCamera2)';
t12=-R21'*t21; % inverse 2R1 as formula

% Uncertainty
t21_unc=estimationErrors.TranslationOfCamera2Error';
t12_unc=-R21'*t21_unc;

% 1.4 Finding rotation angles and axis
axang = rotm2axang(R12);
ang_degree=rad2deg(axang(:,4));

%% Part 2: Epipolar Geometry
%2.1 Find Essential matrix and Fundamental Matrix
T=[0 -t12(3) t12(2);t12(3) 0 -t12(1); -t12(2) t12(1) 0 ];
E=R12'*T;
%fundamental matrix
K1=camp.CameraParameters1.Intrinsics.IntrinsicMatrix;
K2=camp.CameraParameters2.Intrinsics.IntrinsicMatrix;
F=(inv(K2'))'*E*inv(K1');

%2.2 Finding epipoles
q1=null(F);
q2=null(F');
q1=hom2cart(q1');
q2=hom2cart(q2');



%% Part 3: Data Preparation
% 3.1 Change to double data
im1=im2double(imread('ACLtestL05.png'));
im2=im2double(imread('ACLtestR05.png'));

%3.2 Segmentation: separate the background
% Using Color Threshold to create mask for the image im1 and im2
[BWmaskL,maskedRGBImage1] = createMask(im1);
[BWmaskR,maskedRGBImage2] = createMask(im2);

%Visualizing mask
figure (1);
imshow(BWmaskL)

Mask1=bwareafilt(BWmaskL,1);
Mask2=bwareafilt(BWmaskR,1);
Mask1=imfill(Mask1,'holes');
Mask2=imfill(Mask2,'holes');

figure(2);
imshow(Mask1)
figure(3);
imshow(Mask2)

% Applying mask

imL_noB=im1.*double(Mask1);
imR_noB=im2.*double(Mask2);

figure(4);
imshow(imL_noB);
imwrite(imL_noB,'3-2-im1.png');
figure(5);
imshow(imR_noB);
imwrite(imR_noB,'3-2-im2.png');

%% Part 4: Stereo
% 4.1  Apply lens rectifying
[J1, J2]=rectifyStereoImages(imL_noB,imR_noB,camp,'OutputView','full');
```

Appendix: segmentation of the head

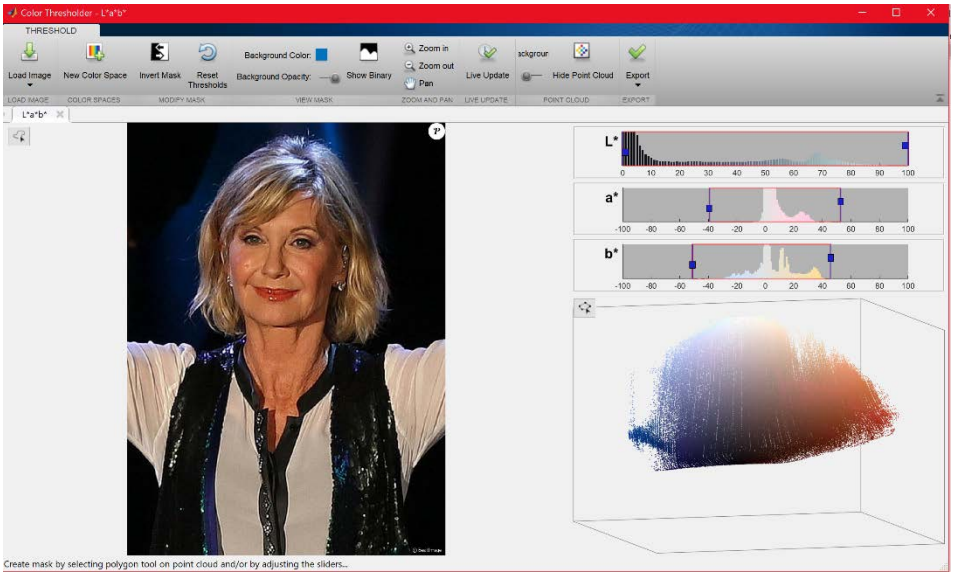
Matlab's Color Thresholder is an app that enables quick, manual segmentation of color images. This appendix outlines its use for segmentation of the head.

1. Open the app, and load an image, either from file or from the workspace. The window looks like this:

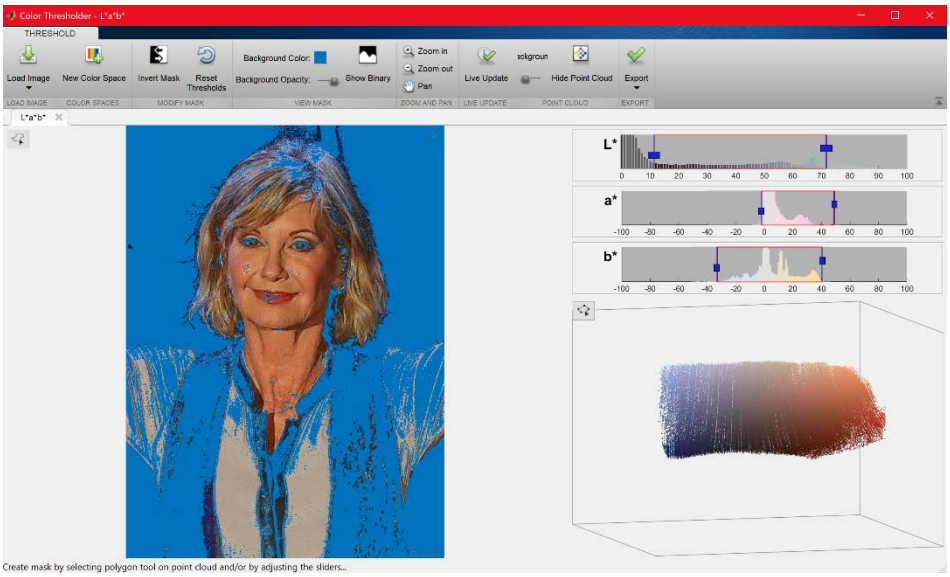


2. You have to choose a color space. There are 4 different color space:
   - The RGB color space is the most often used color representation: red, green and blue in separate channels. But it can be converted into other representations.
   - HSV is a very intuitive one: Hue, Saturation, and Value. The latter is the intensity of a color which will correlate with the grey level, i.e. the brightness.
   - In the YCbCr representation, the Y also encodes intensity. Cb and Cr are related to the difference between blue and Y, and red and Y.
   - The L in the L*a*b representation also correlates with brightness. The *a channel discriminates between red and green. The *b channel differentiates between blue and yellow, i.e. between blue and R+G.

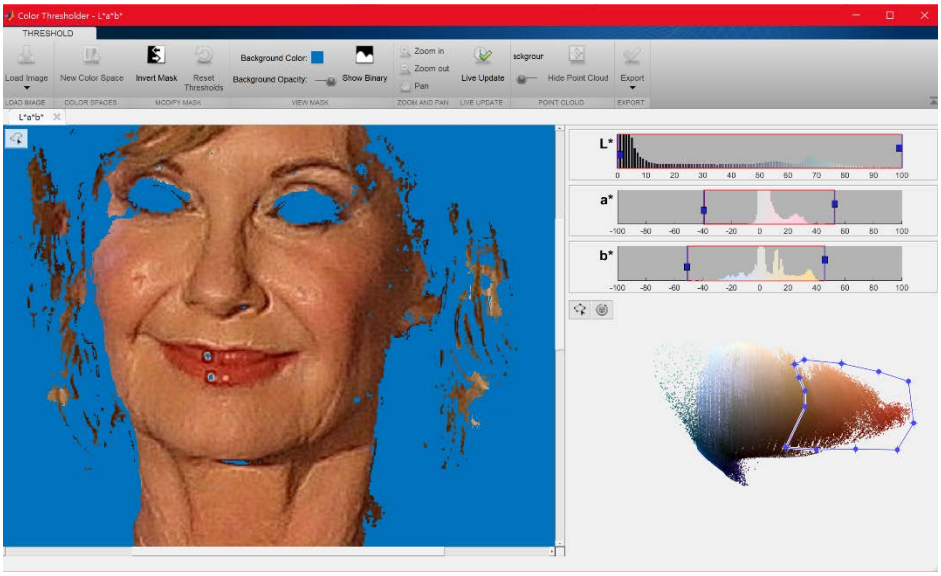
   You can rotate the visualization of each color space. Rotate them such that the color of the skin is most identifiable. Choose the color space, for which this is done most accurately.



3. With the sliders a first selection of colors can be made. Do this conservatively. Pixels that moved to the background cannot be retrieved easily in the next steps. Small gaps in the skin of the head are not harmful as they can be filled again later.



4. Rotate the color space again, so that the cluster of 'head pixels' becomes most isolated. Then define a polygon in the color space that cuts out this cluster. Adjust the vertices of the polygon such that the head pixels are in the foreground, and hair pixels are not. False-positives in the background are not harmful as long as they are isolated, i.e. islands, and not too large.



5. Export the binary mask to the workspace: **`BWmask`**. Also, for later usage, export the whole procedure as a function **`createMask.m`**.

6. Use the matlab function **`bwareafilt`** to select the largest area in the mask, which should be the head area. Use **`imfill`** to fill the holes in the foreground. You can use other morphological operations, like closing and opening, to smooth the boundaries of the object. Finally, use the created mask to suppress all background pixels in the original RGB image by turning them into black.