
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 calibration¹

- 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.

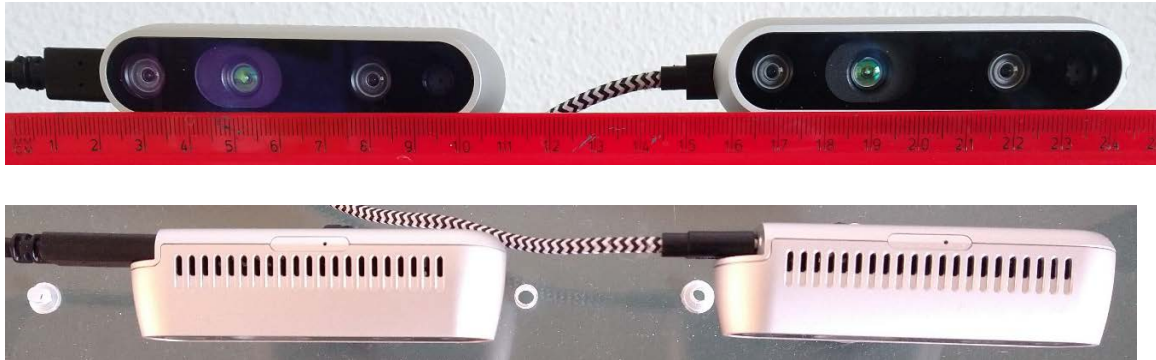
Camera 1:

principle point px:		uncertainty px	
principle point py:		uncertainty py	
focal length dx		uncertainty dx	
focal length dy		uncertainty dy	

Camera 2:

principle point px:		uncertainty px	
principle point py:		uncertainty py	
focal length dx		uncertainty dx	
focal length dy		uncertainty dy	

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.



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:

--

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

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

--

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.

--

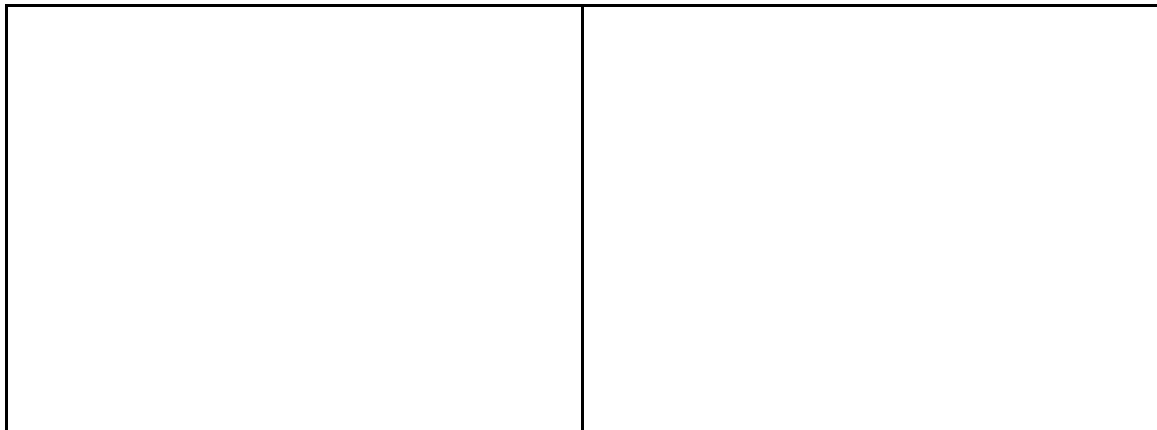
3. Data preparation

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)

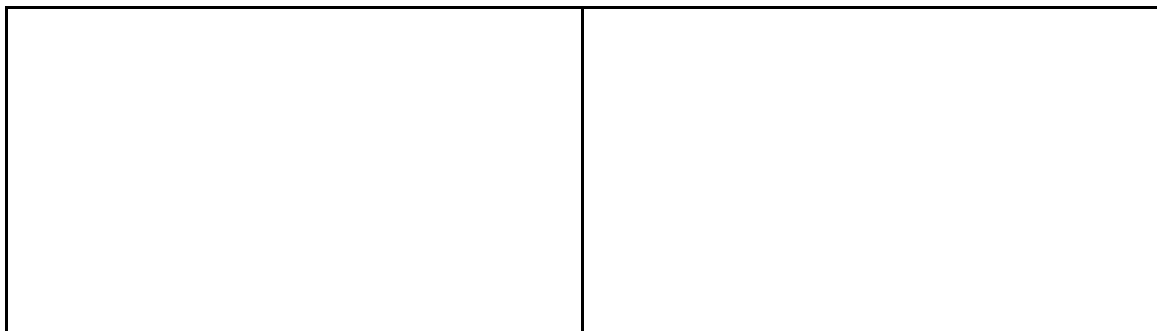


4. Stereo rectification

- 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 wrong², 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: rms:

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
<input type="text"/>	<input type="text"/>
What is the rotation matrix 1R_2 after rectification?	What is the rotation matrix 2R_1 after rectification?
<input type="text"/>	<input type="text"/>

calibration matrix \mathbf{K}_1 after rectification	calibration matrix \mathbf{K}_2 after rectification

baseline vector ${}^1\mathbf{t}_2$

after rectification
(include units)

--	--	--

baseline vector ${}^2\mathbf{t}_1$

after rectification

--	--	--

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:

maximal disparity:

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

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
```

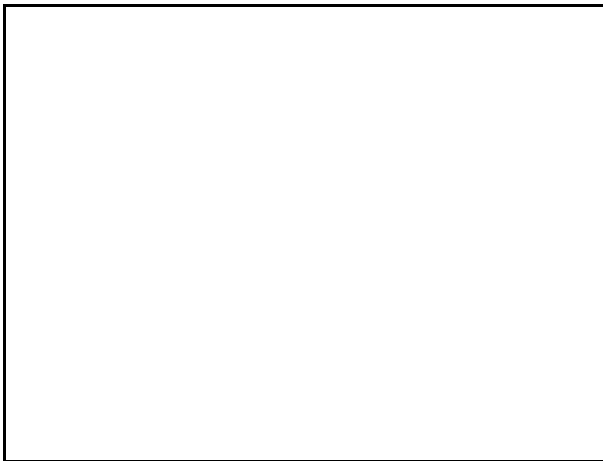
- 5.2. The other option that can be added is '**UniquenessThreshold**'. Read the help of the function. Then apply this option such that the resulting disparity map looks best.

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.

- 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:



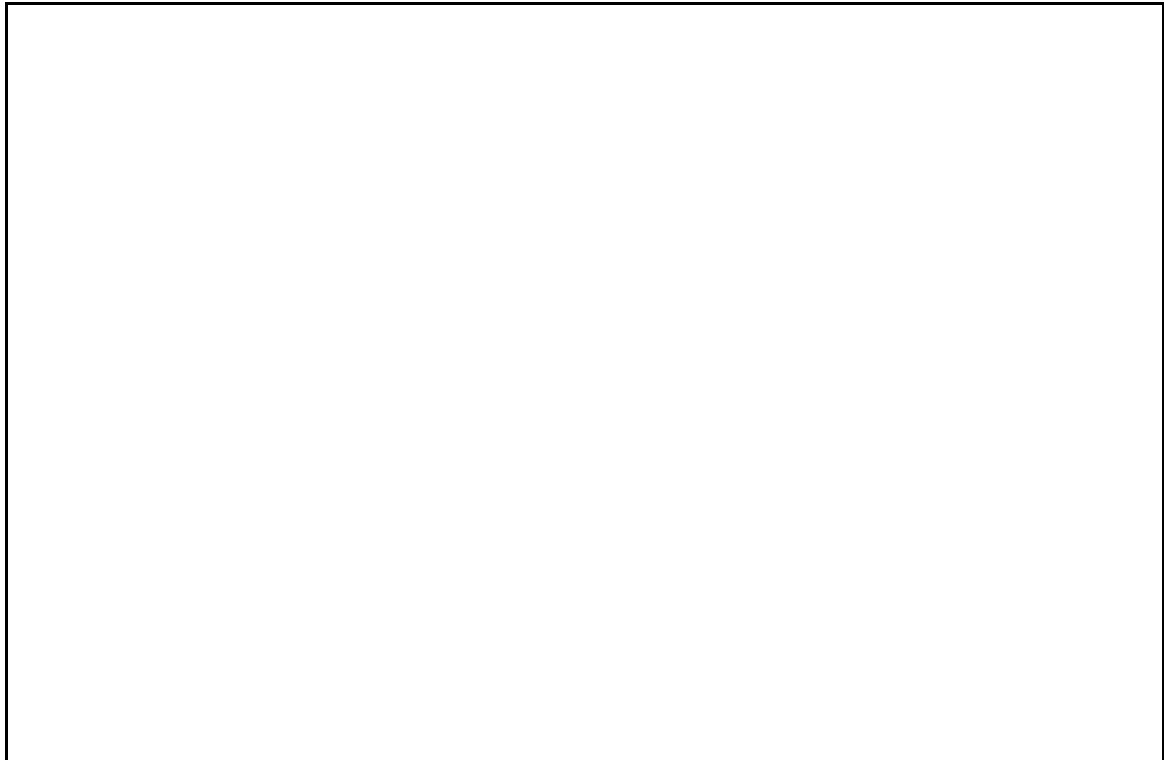
6. Point cloud

- 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 show³ the result with the function **pcshow**.

What is the size of the resulting point cloud array?

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?

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.

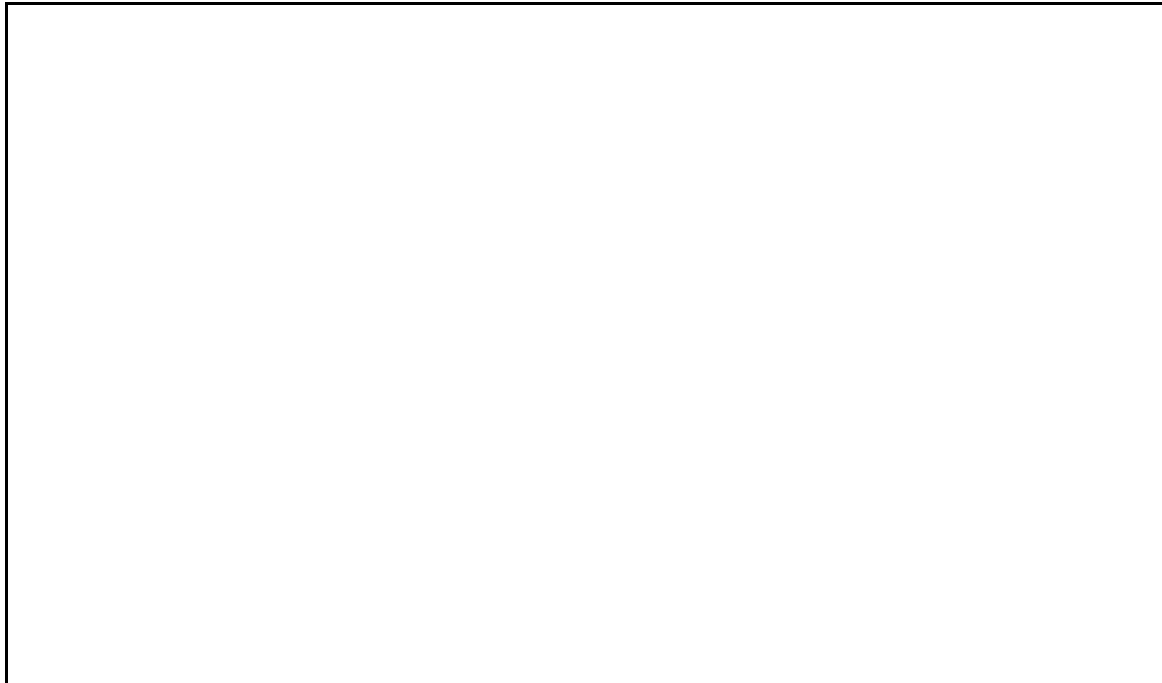
- Rigid geometrical transform.
- Estimate normal vectors of point cloud.
- Fit a plane to the point cloud.
- Fit a cylinder to the point cloud.
- Fit a cylinder to the point cloud

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:

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:

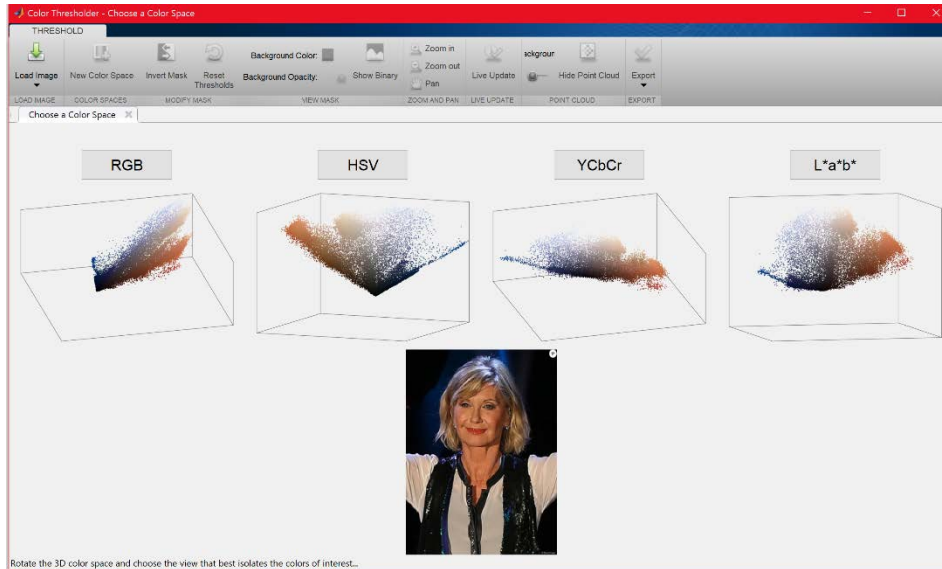
- | | |
|---|---------------------------|
| 1. What is an epipole? What are epipolar lines? | level
knowledge |
| 2. What is an essential matrix? And what is a fundamental matrix? What is the difference? | understanding |
| 3. What do we want to achieve with stereo rectification? What is/are the benefit(s) 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:

Appendix: segmentation of the head

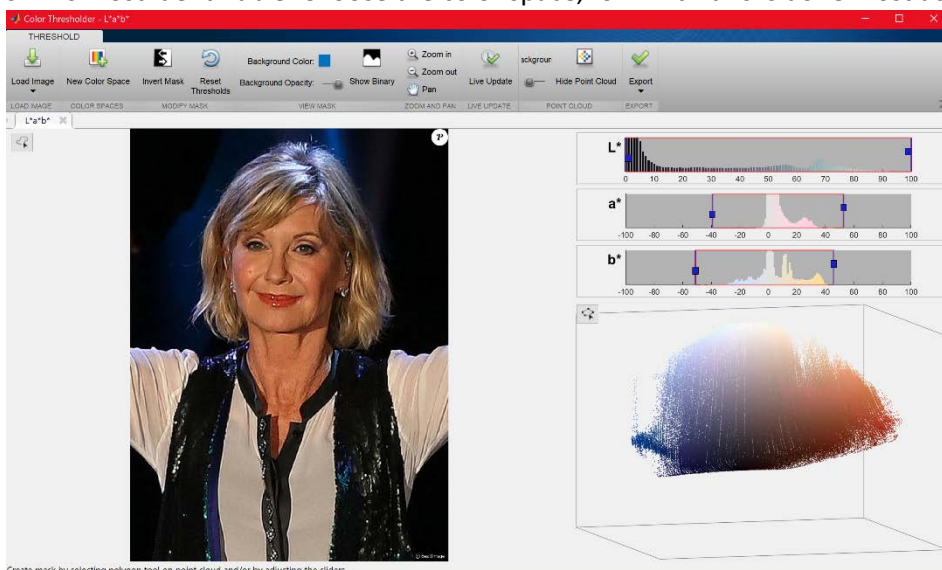
Matlab's Color Thresholder is an app that enables quick, manual segmentation of color images. This appendix outlines its use for segmentation of the head.

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

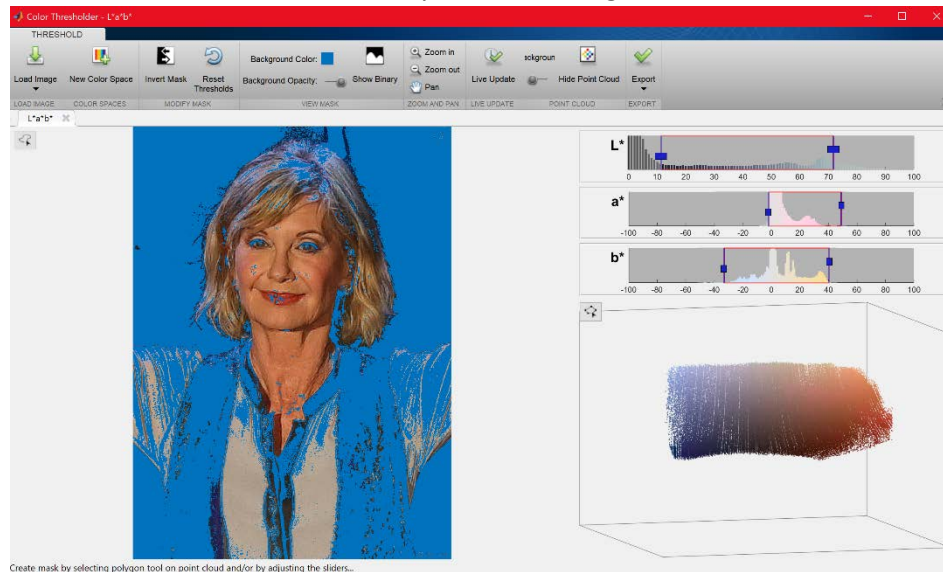


2. You have to choose a color space. There are 4 different color space:
 - The RGB color space is the most often used color representation: red, green and blue in separate channels. But it can be converted into other representations.
 - HSV is a very intuitive one: Hue, Saturation, and Value. The latter 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.

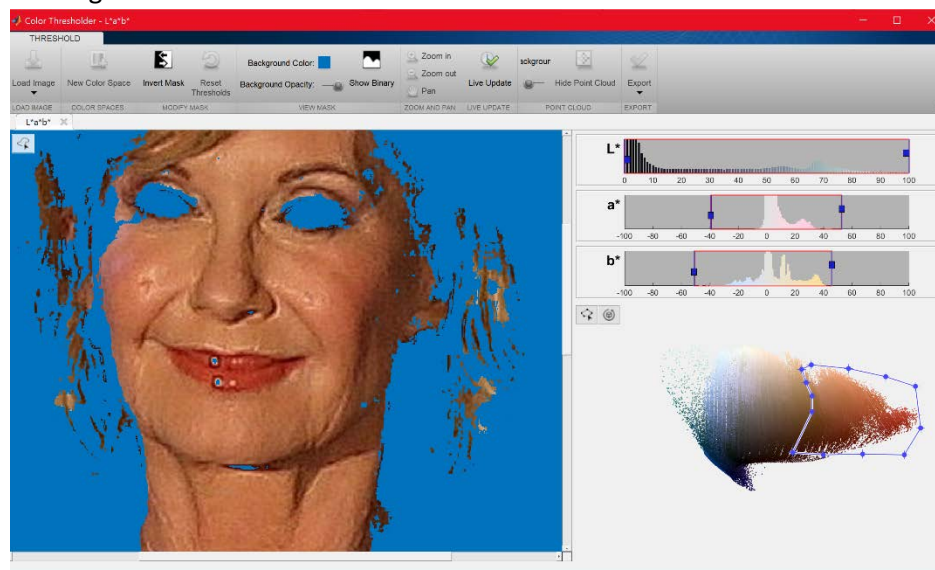
You can rotate the visualization of each color space. Rotate them such that the color of the skin is most identifiable. Choose the color space, for which this is done most accurately.



- 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.



- 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.



- 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.

