In the preceding exercises, several image processing methods have been introduced. The output of the pipeline (two processed images showing a cube from different points of view) shall now be used in order to reconstruct a 3d wire frame version of that cube. In the theoretical part, the position of the right camera has to be derived. The practical part is about looking for correspondences between the two images and about triangulation.

1 Theoretical Exercises

For stereo reconstruction, both the internal and external parameters of each camera must be known. The internal parameters are given: The focal length is 35 mm (1.378 in). The width of the film back equals to 1.417 in and its height is 0.945 in. The external parameters are provided only partly. The missing information has to be derived.

1.1 External Camera Parameters

The camera setup is shown in Fig. 1(a). Determine the position $C'$ of the right camera. In the general case, this task requires the coordinates of three points in space and the image coordinates of these points with respect to the right camera. However, since there is no rotation involved two points are sufficient. In the world coordinate system, point $P_1$ is located at $(-0.023, -0.261, 2.376)$ and point $P_2$ at $(0.659, -0.071, 2.082)$. In the image of the right camera (640x480), $P_1$ has coordinates $(52, 163)$ and $P_2$ is positioned at $(218, 216)$ - see Fig. 1(b).

Figure 1: (a) The world coordinate system is defined to be the same as the coordinate system of the left camera ($C$ lies in the origin). Both cameras look in the direction of the positive $Z$-axis. Neither is rotated. (b) The cube observed by the right camera $C'$.

As a starting point, look at the relevant equation in the script. You can obtain $k_x$ and $k_y$ by using the image resolution and the size of the film back. Do not forget that the center of each image at this scale is at $(320, 240)$. Round the calculated camera position with a precision of half an inch.
Solution

Starting from that equation, one can derive \( u = f \frac{X - C_1}{Z - C_3} \) and \( v = f \frac{Y - C_2}{Z - C_3} \) because no rotation is involved. Solving the first equation with respect to \( C_1 \) delivers

\[
C_1 = X - \frac{u(Z - C_3)}{f}. \tag{1}
\]

Using two points \( P_1 \) and \( P_2 \) we get

\[
X_1 - \frac{u_1(Z_1 - C_3)}{f} = X_2 - \frac{u_2(Z_2 - C_3)}{f}. \tag{2}
\]

Thus,

\[
C_3 = \frac{(X_1 - X_2)f - u_1 * Z_1 + u_2 * Z_2}{u_2 - u_1}. \tag{3}
\]

\( u_1 \) and \( u_2 \) refer to image coordinates in inch and for their computation \( k_x \) and \( k_y \) are needed. \( k_x \) is the amount of pixels per inch in horizontal direction. \( k_x = 640 \) pixel /1.417 in. \( k_y = 480 \) pixel /0.945 in.

\[
u_1 = \frac{x_1 - x_{\text{center}}}{k_x} = (52 - 320) * \frac{1}{451.658} = -0.5934. \tag{4}
\]

\[
u_2 = \frac{x_2 - x_{\text{center}}}{k_x} = (218 - 320) * \frac{1}{451.658} = -0.2258. \tag{5}
\]

The result of Equation 3 is \( C_3 = -0.0004 \approx 0 \). Then, using \( P_1 \), Equation 1 gives

\[
C_1 = X_1 - \frac{u_1(Z_1 - C_3)}{f} = -0.023 + 1.0233 = 1.0003 \approx 1 \tag{6}
\]

Correspondingly,

\[
C_2 = Y_1 - \frac{v_1(Z_1 - C_3)}{f} = -0.261 + 0.2614 = +0.0004 \approx 0, \tag{7}
\]

with

\[
v_1 = \frac{y_1 - y_{\text{center}}}{k_y} = -77 * \frac{1}{507.937} = -0.152. \tag{8}
\]

Result: the position of right camera is \((1.0, 0.0, 0.0)\)
2 Practical Exercises

The goal of the practical exercise is to compute a 3d reconstruction for a given image pair. In this exercise we will be dealing with 3 examples, ie the tsukuba image pair, the sawtooth image pair, and the cube from the previous exercises. Copy the respective image files from `~cvcourse/pics/` to a new directory. Also place the template `4_stereo_template.cpp` from `~cvcourse/materials/` in this directory.

3d reconstruction from two views can be split into two sub task: (1) finding corresponding points in the two images and (2) recovering the 3d world coordinates (of the “physical” point) given such corresponding points. The latter is called triangulation and requires fully calibrated cameras, ie, the internal and external camera parameters must be known. Calibration of cameras has been addressed in the theoretical exercise (see Fig. 1(a)). Assume now that the two optical camera centers are 1.0 inch apart from each other. The optical axes of the cameras are parallel and their image planes are coplanar, with coincident x-axes.

Given our special camera setup, we know that corresponding points must have the same \(y\) coordinate which simplifies finding point correspondences. Hence, we can process each line of the image (ie, each scan line) separately. In order to identify corresponding points we need a measure of similarity. In this exercise you should use the normalised cross-correlation which will be defined later.

2.1 Triangulation

Implement the function `PointThreeD triangulate(float xLeft, float xRight, float y)`. The first two parameters correspond to the \(x\)-coordinate of a projected 3d-point in the left and the right gray scale image. \(y\) is the common \(y\)-coordinate. Note that this method uses the (globally defined) camera parameters (including image width and height). The function must return the world coordinates in inch.

Check your code with \(P_1\) shown in Fig. 1(b): `triangulate(314,52,163)` should return \((-0.023,-0.261,2.376)\). Try the other test cases provided in the template. What is the smallest and the largest possible distance?

Solution:

- `triangulate(m_width, 0, m_height/2) = 0.5, 0, 0.972477`
- `triangulate(m_width/2+1, m_width/2-1, m_height/2) = 0.5, 0, 311.193`
- `triangulate(m_width/2 , m_width/2, m_height/2) = nan, nan, inf`

Remark: `nan` stands for ‘not a number’ and `inf` for ‘infinity’. The depth of the first example corresponds to the smallest possible distance.

2.2 Correlation Coefficient

Implement `float computeCorrelation(...)`. The method is supposed to compute the similarity between a patch from the left and a patch from the right image during correspondence search. Check the code with the test case provided in the template.

The normalised cross-correlation of patches surrounding two points (in the two views), is defined as

\[
NCC(p_L,p_R) = \frac{1}{|\Delta|\sigma_L\sigma_R} \sum_{\Delta} (I(p_L + \Delta) - \mu_L) \cdot (I(p_R + \Delta) - \mu_R),
\]

where \(\Delta\) defines the (square) neighbourhood of a pixel, \(\mu_R = \frac{1}{|\Delta|} \sum_{\Delta} I(p_R + \Delta)\) and \(\sigma_R = \sqrt{\frac{1}{|\Delta|} \sum_{\Delta} (I(p_R + \Delta) - \mu_R)^2}\).
2.3 Stereo Reconstruction

The final step is to implement the function `definePointsThreeD(grayLeft, grayRight, cannyLeft, cannyRight)` which does the actual 3d reconstruction. It makes use of the above function to identify corresponding points and to compute the 3d world coordinates. The output, i.e., one 3d point for each pair of corresponding points, should be stored in a vector called `points` which is globally defined. This vector of 3d points will then be visualised by OpenGL.

Task: In each scan line, find one corresponding pixel in the right image for each pixel in the left image (note that the x-coordinate of the pixel in the right image always has to be smaller than the x-coordinate of the pixel in the left image - explain why). Use the correlation coefficient as a similarity measure for the correspondence search. Compute a 3d-point by triangulation for every pixel pair and add it to the display list.

Test your code with the `tsukuba` and `sawtooth` image pairs. You can then examine the resulting 3d-structure by left-clicking on the OpenGL window, holding the button and moving the mouse. Two shell-scripts are provided (`4_stereo_tsukuba.sh` and `4_stereo_sawtooth.sh` found in `cvcourse/materials/`) which pass the correct command line argument to the program. Your results should look like Fig. 2(a) and (b) for a correlation mask of 11 × 11.

![Figure 2: 3d reconstructions. (a) Tsukuba (b) Sawtooth](image)

2.4 Cubes

As you may have noticed, computing point correspondences is quite time consuming. The reason is that one has to compare each point in the left image to many points of the corresponding scan line in the right image. You should now implement a faster method (for the `cube` image) which makes use of the canny edge maps. Adapt the method `definePointsThreeD` to take advantage of the canny images. This allows to limit the correspondence search to only a few pixels per scan line.

Do not forget to set `--cube` to 1 and provide the canny edge maps. The two spheres that appear in the OpenGL window correspond to $P_1$ and $P_2$ (see Fig. 1(b)) and are meant as guidance. Again, a little script (`4_stereo_cube.sh`) is provided which passes the correct parameters to the program.