IM520/MC505 Computer Vision
Term Report

David Krög

June 17, 2020

Contents

Guidelines for authoring lab reports

Cumulative lab report

The lab report is a **single, cumulative document** which should contain a concise and well-structured summary of the work you did in this course. Also, you are asked to demonstrate and discuss your “report in progress” at any time throughout the semester. If help or advice is needed, please ask in class or use the course’s online forum.

Weekly and final submissions

You are asked to upload a snapshot of your worked-out assignments weekly (i.e., prior to the next lab unit). These submissions are not graded but randomly checked to verify your progress. The final (complete) documentation for all assignments must be turned in at the end of the semester, prior to the (oral) exam. Thus you can pace your work individually and turn back to previous assignments for improvements at any later point.

> **Note** that this **freedom** puts a lot of **responsibility** on yourself. Make sure that you start to write immediately, make steady progress and nothing important is left behind!

Document structure and content

This document should help you to get started with the report. It is strongly suggested to use the final format right away to avoid surprises at a later point. Also, you will discover that writing and documenting your findings can help you in developing good and understandable solutions from the very beginning. Here are a few hints for writing your reports:

- One **chapter** should be dedicated to each **assignment** (note that chapter names have been modified for this).
- Make notes and write down your concepts immediately, that is, **before** you start coding!
- Describe each given task in your own words (do not just replicate the assignment). Then describe your approach, explain the main difficulties, clearly outline your solution, finally provide illustrative and meaningful results.
- Try to go beyond the material you find elsewhere, use and extend formal (mathematical) descriptions in a creative way. Also, try to keep your notation simple and

consistent, which is not always easy to do. Look at good examples and consider this part of the learning process.

- Be careful and creative when it comes to designing meaningful tests and selecting examples. Do not make screenshots but save the relevant images with ImageJ (usually as PNGs).
- Always give appropriate references to literature, figures and other work you used!
- Get used to work with formal and concise descriptions (math, symbols, relations, algorithms, ...) and train yourself in "getting the notation right".
- Write in complete sentences and try to use a "professional" language.

The bad and the ugly

- **Don't just show program code!** Use prose with mathematical and algorithmic notation wherever appropriate (use the assignments and lecture notes for guidance). Insert actual code sparingly and only to show particularly interesting or critical parts of your implementation.
- **Do not explain details that are trivial** or elementary (such as Pythagoras' law, for example). Otherwise, make a reference to the *all* sources you used (including school books, blogs, WikiPedia etc.).
- **Do not just replicate** equations and figures from the lecture materials, but – as said above – describe the task in your own words. In particular, you will be **executed** (i.e., beheaded, drawn and quartered) if you ever copy/paste equations from the assignment or any other sources. Make sure you write these things yourself (that's what LaTeX is famous for)!

Assignment 1

Circle detection in binary dot images

Humans can detect all kinds of shapes instinctively without even thinking about it while computers only *see* a collection of pixels with certain values instead. Computer Vision tries to let computers see the same way humans do. Sometimes this can be achieved with more or less simple mathematical concepts and sometimes complex algorithms are required.

The task in the first assignment was to detect a circle which is embedded in a binary dot image. As humans we can easily recognize the circle and connect the points in our head but the computer does not know which image points actually belong to the circle. To detect the circle we are using a rather simple algorithm called RANSAC[6] created by Robert C.Bolles and Martin A.Fischer in the 1980s[1].

1.1 Algorithm

To detect the circle the following steps were performed:

1. Collect the coordinates of all points.
2. Randomly choose 3 of them.
3. Calculate the parameters $(x_c, y_c, r)$of a circle which passes through all 3 points.
4. Count the points which are on/close enough to the circle.
5. Repeat the steps 2 to 4 a certain number of times and always remember the circle with the most points on it.
6. The "best-fitting" circle is the one which has the most points on it after a given number of iterations.

1.1.1 Collect all points

All black pixels were saved as points with their respective x/y - coordinates into a collection by iterating over all the pixels of the image.

1.1.2 Randomly choose 3 points

3 points of that collection were randomly selected while making sure that no point was selected more than once.

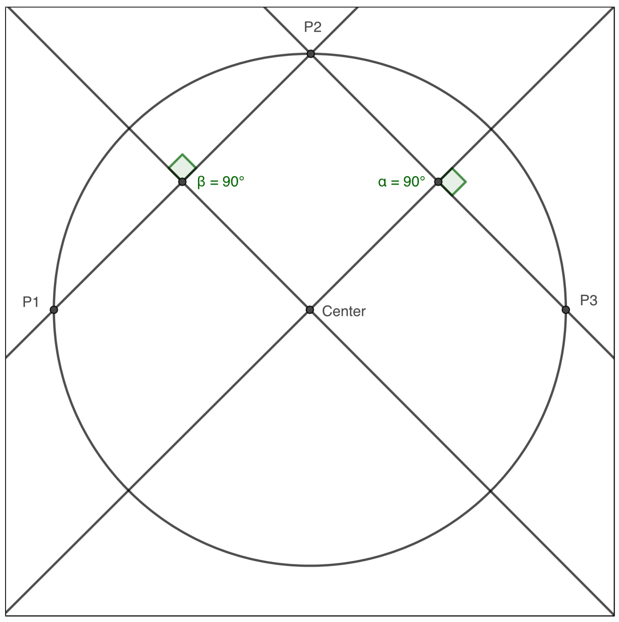

Figure 1.1: Determine the center of a circle with 3 points.

1.1.3 Calculate the circle parameters

With the help of these 3 points it is possible to determine the attributes of a circle which passes through them as shown in Figure 1.1 following an article from Paul Borke[5].

A line through the point P_1 and P_2 was formed and another line through P_2 and P_3. The equations for these two lines are:

$$y_a = m_a(x - x_1) + y_1$$
$$y_b = m_b(x - x_2) + y_2$$

These equations were rearranged to solve for their respective slopes m.

$$m_a = \frac{y_2 - y_1}{x_2 - x_1} \qquad m_b = \frac{y_3 - y_2}{x_3 - x_2} \tag{1.1}$$

Now two lines perpendicular to the lines $y_a(P_1P_2)$ and $y_b(P_2P_3)$ going through the center between each point pair were created. The perpendicular of a line with a slope m has a slope of $-1/m$. This results in following equations:

$$y_a' = -\frac{1}{m_a}(x - \frac{x_1 + x_2}{2}) + \frac{y_1 + y_2}{2} \tag{1.2}$$

$$y_b' = -\frac{1}{m_b}(x - \frac{x_2 + x_3}{2}) + \frac{y_2 + y_3}{2} \tag{1.3}$$

The center of the circle is the intersection of these two perpendiculars. The x-value of the center was calculated with the following equation.

$$x = \frac{m_a m_b(y_1 - y_3) + m_b(x_1 + x_2) - m_a(x_2 + x_3)}{2 \cdot (m_b - m_a)} \tag{1.4}$$

To calculate the y value of the center I substitute the x value into one of the two perpendiculars (1.2, 1.3). The circle radius is determined by calculating the distance between the center point and one of the 3 originally chosen points.

There are 3 situations where a circle can not be calculated with just 3 points:

- If all 3 points are collinear.
- If one of the 3 points was selected more than once. This gets checked while selecting the points.
- If one of the formed lines between 2 points is vertical the slope would be infinite. To avoid that the order of the 3 points is rearranged if this is the case.

1.1.4 Evaluate the circle

After the circle's parameters were calculated the number of points on the circle got determined. The distance of all detected points to the center point of the circle and its difference to circle radius was calculated. If this difference is smaller than a given threshold, the point is considered on the circle. The threshold can be defined when the plugin is started. The number of points on the circle describes how well the circle "fits".

1.1.5 Repeat

This whole process gets repeated for a given number of times which can be defined at the start of the plugin. After each iteration the number of points on the calculated circle is compared to the number of points on the current "best-fitting" circle. If the count is higher the "best-fitting" circle gets replaced by the newly calculated circle. After this process reached the defined number of iterations the "best-fitting" circle was considered as the detected circle in the image.

1.2 Results

Figure 1.2 shows the generated test image. As soon as the described process was repeated for 25 times. The detected circle got drawn on the image in blue as well as all the points on it in red (Figure 1.3). It worked pretty well even on noisier images.

1.3 Research Questions

1.3.1 Question A

Assume n is the total number of points, of which m points belong to a circle.

- *What is the probability of selecting 3 circle points in a random draw?*
- *How many random draws are needed such that the probability of finding 3 circle points is 99%*

The probability of selecting one point which lies on the circle is: $\frac{m}{n}$. Now if 3 random points are chosen in a row the formula to calculate the probability is:

$$P = \frac{m}{n} \cdot \frac{m-1}{n-1} \cdot \frac{m-2}{n-2} \tag{1.5}$$

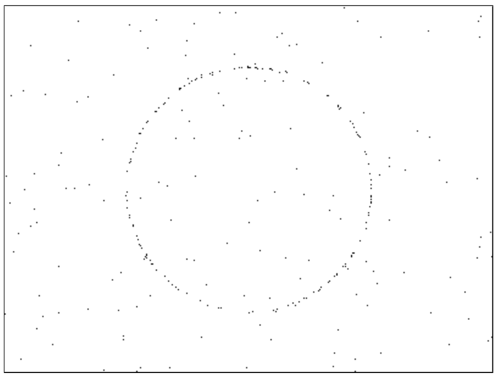
Figure 1.2: Generated noisy binary image.

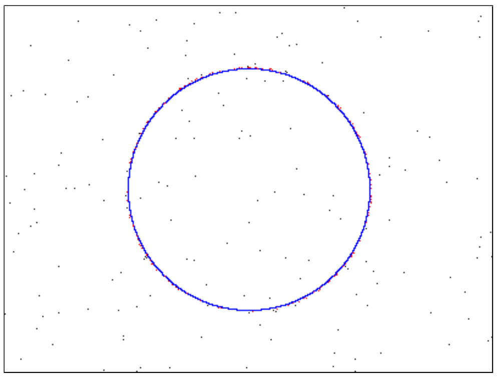
Figure 1.3: The Detected circle after 25 iterations and a threshold distance of 1 to the circle

To calculate the number of random draws n needed to select 3 circle points with a probability W of 99% we use following formula: $W = 1 - (1 - p)^n$ [4]. To calculate p the formula above (1.5) is used then solve the equation (1.8) to get the number of random draws n needed that the probability of finding a circle is 99%.

$$0.99 \geq 1 - (1 - p)^n \tag{1.6}$$
$$0.01 \geq (1 - p)^n \tag{1.7}$$
$$n \geq \frac{\ln(0.01)}{\ln(1 - p)} \tag{1.8}$$

1.3.2 Question B

What if the image contains more than 1 circle and we want to find all of them? Develop (but do not implement) a strategy for this case.

To detect multiple circles in one image a threshold for the number of points which needed to be on a circle to be detected as a valid circle needs to be defined. If detected the circle is added to a list. To make sure that the same circle is not detected multiple times it has to be checked that there is no circle with the same center point and radius already in the list before adding it.

Assignment 2

Affine Point Cloud Matching

In this assignment the task was to find the affine transformation between two point sets. At first a thought experiment should be done about how long it would take if to match the two point sets by brute force and if that would be a reasonable approach. In the second task the affine transformation between two point sets is given and it needs to be applied to check if the transformation is done right. The third task was about finding the affine transformation between two point sets with the help of a method called triangulation.

2.1 Matching point sets by brute force or RANSAC

As described in the last assignment (chapter 1) the RANSAC method can be an efficient way to find a solution. But can it also be used to find the correspondence between two point sets? How long would it take to find the affine transformation between two point sets by randomly or systematically picking three points in both point clouds? The following thought experiment discusses these questions:

Let's assume that we have two images with m points and we are picking systematically n points in each point cloud there would be

$$\frac{m!}{(m-n)!} \tag{2.1}$$

possible combinations. To find the best possible solution we need do same with the points in the second image which results in the following equation provided the number of points in both point clouds is the same:

$$\left(\frac{m!}{(m-n)!}\right)^2 \tag{2.2}$$

As we can see the growth is exponential and the number of possibilities would get really high really fast with an increasing number of points m.

2.1.1 Research Question 1

If your computer could perform 1 million such tests per second, how long would it take to examine all possible 3-point matches, if m = 10, 50, 1000?

$$\frac{\left(\frac{10!}{(10-3)!}\right)^2}{1000000} \approx 0.52s \tag{2.3}$$

$$\frac{\left(\frac{50!}{(50-3)!}\right)^2}{1000000} \approx 3.84h \tag{2.4}$$

$$\frac{\left(\frac{1000!}{(1000-3)!}\right)^2}{1000000} \approx 31520years \tag{2.5}$$

While 0.52 seconds for 10 points does not sound that bad, 3.84 hours for 50 points sounds already unusable and 31520 years by far exceeds any time limit for a human being. Considering that point clouds often exceed 1000 points by quite a bit nowadays proves that using a brute-force approach to find the correspondence between two point clouds is not an option.

2.1.2 Research Question 2

Why would the RANSAC approach (as used successfully in Assignment 1) not be of much help in this case?

Since the points also need to be chosen in the correct order in both point sets the chance to randomly select the correct 3 points in both sets is very low. The number of possibilities are way to high to use the RANSAC approach.

2.2 Implement and test the affine transformation

The true transformation between point set X and X' is given:

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{pmatrix} = \begin{pmatrix} 0.013 & 1.088 & 18.688 \\ -1.000 & -0.050 & 127.500 \end{pmatrix} \tag{2.6}$$

An affine transformation can be expressed as a matrix-vector multiplication:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.7}$$

The given transformation matrix was applied to each point of the first point set and then the closest point in the second point set was calculated which is considered the corresponding point. The residual error is the sum of the distance between the calculated position and the position of their respective corresponding point over all points.

2.2.1 Result

In Figure 2.1 both point sets are shown in one image. Figure 2.2 shows the result after applying the given transformation matrix to the red point set. The residual error resulted in 1.0 which was caused by the rounding error over time.

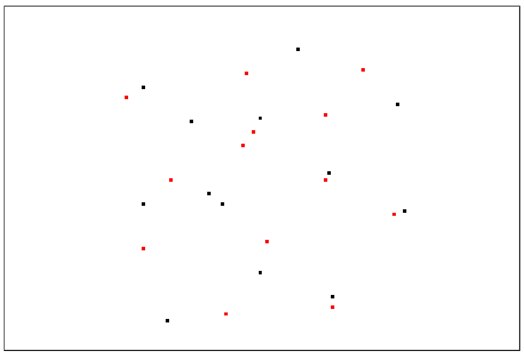

Figure 2.1: Both point sets.

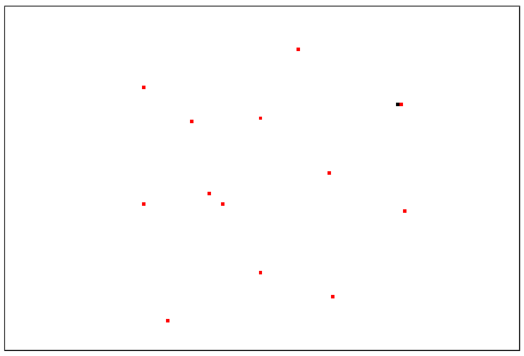

Figure 2.2: After applying the given transformation to the red point set.

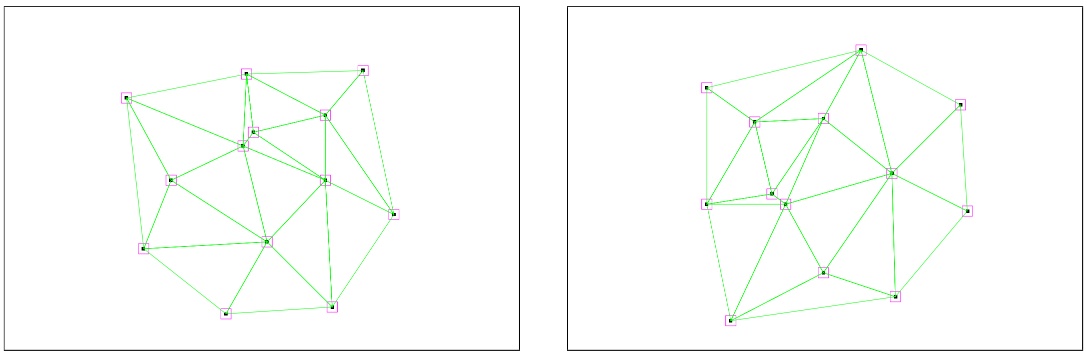

Figure 2.3: Delunay triangulation.

2.3 Structuring point sets by triangulation

As stated in the first Task using either brute-force or the RANSAC algorithm to find point correspondence is not a valid solution for bigger point sets. In this task the problem should be solved by applying the *Delaunay triangulation*[3]. This approach always connects 3 points of a point set to triangles. Figure 2.3 shows the two triangulated example point clouds.

2.3.1 Algorithm

1. Calculate the Delaunay triangulation for both point sets.
2. Iterate through all triangles from X and X' and for each pair perform the following steps:
    - Find the affine transformation A between them.
    - Apply A to all points in X and calculate the error of the projected points to the closest points in X'.
    - Check for permutation within the triangles.
    - Memorize the best-fit with the smallest error.

3. Find the best least-squares fit for the resulting point match.

2.3.2 Calculate the Delaunay Triangulation

A library for the triangulation was provided which returns a set of triangles for each point set.

2.3.3 Iterate through all triangles

For each triangle of the the first set we iterate over the triangles in the second set. Between each triangle pair the corresponding point pairs $(x_0, x_0^{'})$, $(x_1, x_1^{'})$, $(x_2, x_2^{'})$ are formed. The next step was to find the 6 needed parameters for the affine transformation. To find these parameter at least 6 equations were needed to find these 6 unknowns. With the help of the 3 point pairs a linear equation system was created:

$$\begin{aligned}
x_0^{'} &= a_{00} \cdot x_0 + a_{01} \cdot y_0 + a_{02}\\
y_0^{'} &= a_{10} \cdot x_0 + a_{11} \cdot y_0 + a_{12}\\
x_1^{'} &= a_{00} \cdot x_1 + a_{01} \cdot y_1 + a_{02}\\
y_1^{'} &= a_{10} \cdot x_1 + a_{11} \cdot y_1 + a_{12}\\
x_2^{'} &= a_{00} \cdot x_2 + a_{01} \cdot y_2 + a_{02}\\
y_2^{'} &= a_{10} \cdot x_2 + a_{11} \cdot y_2 + a_{12}
\end{aligned}$$

To be able to solve this system with the provided linear equation system solver the system needed to be rearranged into this general form:

$$M \cdot a = b \tag{2.8}$$

Which resulted in following matrix-vector form:

$$\begin{pmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 \\ x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \end{pmatrix} = \begin{pmatrix} x_0^{'} \\ y_0^{'} \\ x_1^{'} \\ y_1^{'} \\ x_2^{'} \\ y_2^{'} \end{pmatrix} \tag{2.9}$$

After solving that equation all parameters for the affine transformation were calculated $(a_{00}, a_{01} \ldots a_{12})$. With these parameters the transformation matrix A was created.

2.3.4 Find the best least-squares fit.

Once the Transformation A was known it was applied to all points in the point set X. The Euclidean squared distances were calculated between each point and it's assumed corresponding point and summed up. The resulting error was compared to the error of the last best-fitting transformation. The transformation with the smallest error was remembered. After iterating through both triangle sets the transformation resulting in the smallest error was known.

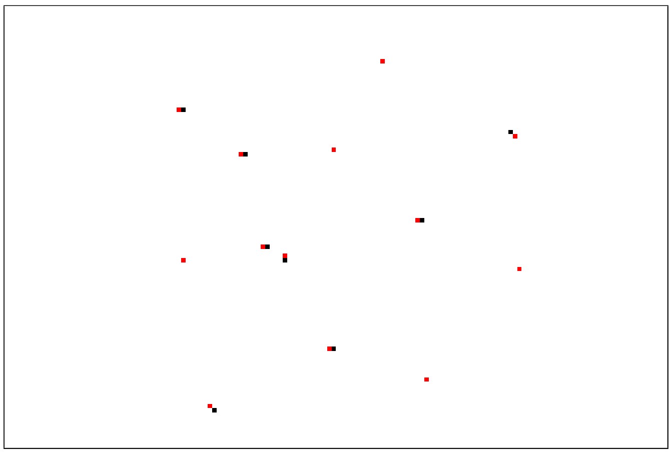
Figure 2.4: Black dots: The real result points. Red dots: the transformed points.

2.3.5 Result

The squared error of the best fit for the test images was 8.64 and the calculated transformation matrix A was:

$$A = \begin{pmatrix} 0.019 & 1.096 & 17.827 \\ -0.981 & -0.046 & 126.541 \end{pmatrix} \tag{2.10}$$

Which was pretty close to the originally provided matrix 2.6. The comparison between the provided result and the transformed points can also be seen in Figure 2.4.

2.3.6 Research Question

When using the Delaunay triangulation for structuring corresponding point sets we implicitly assumed that the triangulation is insensitive to affine transformations in 2D. Is this really true? What would be needed to validate this assumption? Make a quick internet search and document your findings.

In theory the Delaunay triangulation is invariant for translation, rotation and uniform scaling. However, triangulation of a point set is not stable and very sensitive to noise which can deliver different results.

Assignment 3

Iterative Closest-Point Algorithm

This assignment took a step further and instead of trying to determine a point correspondence in a binary dot image the task was to find a reasonable correspondence between a pair of stereo pictures. The *Harris Corner Detector*[7] was used to detect corners in the images. Then a point correspondence between the corners in the left and the right image needed to be determined. Both pictures were taken from a slightly different viewpoint and the ambient conditions may differ so the amount of detected corners may differ. This results in the possibility that not all corners of one picture have a matching corner in the other picture, which causes the problem of outliers.

3.1 Algorithm

Prepare images- part into two point clouds. (corner strength settings)

1. Use Corner Detector
2. Initial Transformation
3. ICP - Affine Fitter -> Transformation Matrix
4. Find point pairs
5. Transform back then connect point pairs with lines

```
User Corner Detector
Separate into two point clouds
Get centroids of both point clouds
Start ICP with vector connecting centroids as initial transformation
```

Assignment 4

Algebraic Lines, Straight Line Fitting

4.1 Only Started Working

Assignment 5

2D Shape Signatures

5.1 Not started yet!

Assignment 6

Texture Segmentation Using Laws Texture Filters

In this assignment the task was to perform a texture-based segmentation based on *Laws texture energy maps*[2]. A manually selected texture patch is used as reference texture. Since the calculation of Laws texture energy maps was already provided as an imageJ plugin it is not described in this assignment. The plugin creates 9 texture energy maps which were used to generate a 9-dimensional feature vector for every pixel.

6.1 Reference-base texture segmentation

The goal is to segment the image into regions which have a similar texture as the reference texture:

1. Select a region of interest R.
2. Create the 9 texture energy maps with the help of the plugin.
3. Calculate the reference feature vector x_R for the region of interest.
4. Create a distance map which holds the distance between the local feature vector and the reference feature vector.
5. Apply automatic thresholding to the distance map to obtain the final segmentation.

6.1.1 Calculate the reference feature vector

To create the reference feature vector x_R sum up all the local feature vector values of each pixel $x_k(u,v)$ in the marked region of interest and divide each value by the number of pixels in the region $|R|$ as stated in the following equation:

$$x_R = \frac{1}{|R|} \cdot \sum_{(u,v) \in R} x(u,v) \tag{6.1}$$

This results in the 9-dimensional reference feature vector.

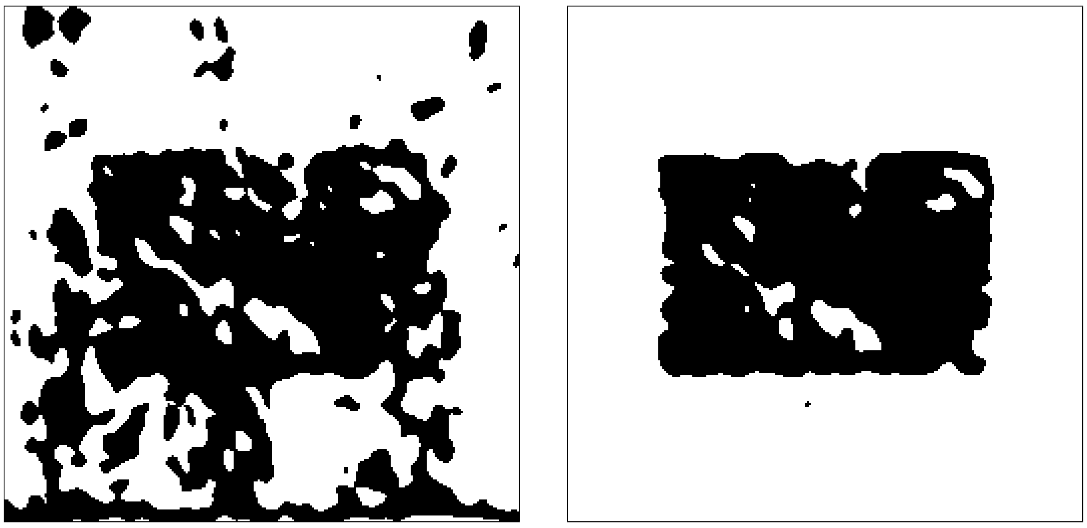

Figure 6.1: Texture Segmentation. Using the L1 norm on the left side and the L2 norm on the right.

6.1.2 Calculate the distance map

To setup a distance map D the distance between the local feature vector $x(u, v)$ and the reference feature vector x_R needs to be calculated:

$$D(u, v) = ||x(u, v) - x_R|| \tag{6.2}$$

The L1 as well as the L2 distance norm were used.

After the distance map is created automatic Otsu thresholding was applied to the distance map D to obtain the final segmentation.

6.1.3 Result

Figure 6.2, 6.3 and 6.4 show the results of some text images. The results show the original image on the left then the distance map and the distance map after thresholding on the right. In most cases the segmentation worked quite well. Overall the L2 distance norm showed better results. In some cases by quite a bit as shown in Figure 6.1.

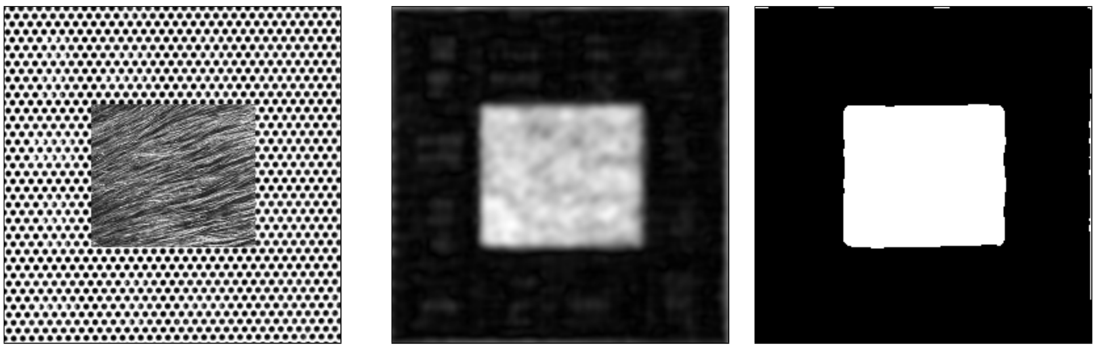
Figure 6.2: The outer texture was selected as reference texture. Image size: 256 x 256.

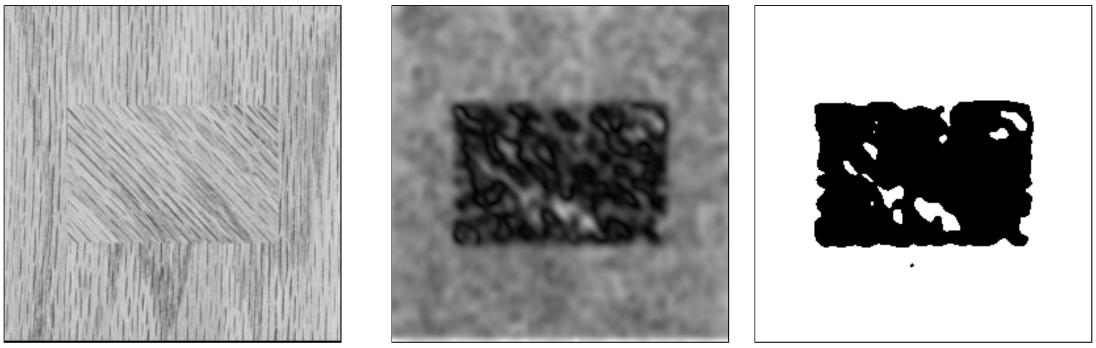
Figure 6.3: The inner texture was selected as reference texture. Image size: 256 x 256.

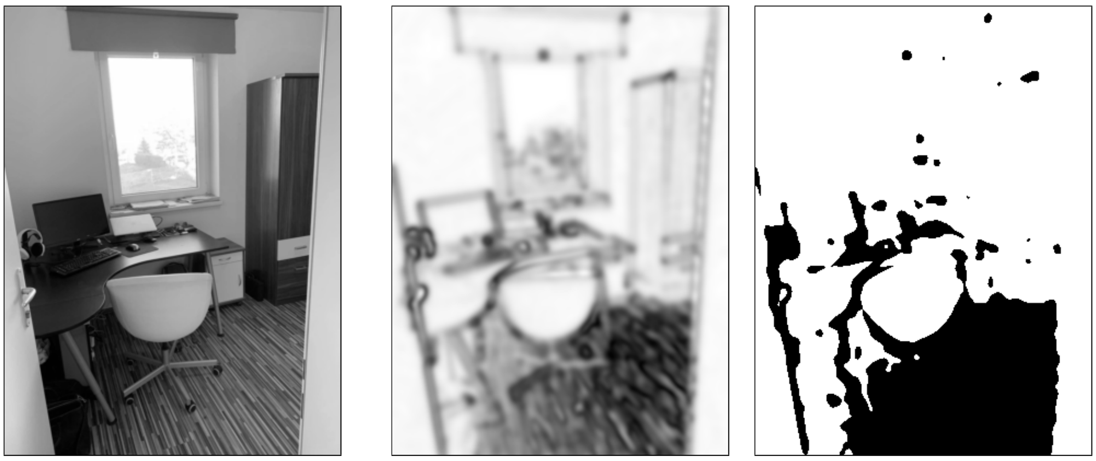
Figure 6.4: The floor was selected as reference texture. Image size: 307 x 410.

Assignment 7

Correlation-Based Subimage Matching

7.1 Not started yet!

Assignment 8

SIFT-Based Image Stitching

8.1 Not started yet!

Assignment 9

Optical Flow

9.1 Not started yet!

Summary

Finally, summarize what has been accomplished in this semester and what not. Point out topics that were instructive, confusing, too hard, too easy etc. Perhaps you even found problems that you would like to explore deeper (e.g., in a project).

References

[1] Martin A. Fischler and Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. *Commun. ACM* 24.6 (June 1981), pp. 381–395. URL: https://doi.org/10.1145/358669.358692 (cit. on p. 6).

[2] Kenneth Ivan Laws. “Textured image segmentation”. Dissertation. Los Angeles, California: University of Southern California, Jan. 1980 (cit. on p. 18).

[3] *Delaunay triangulation.* Mar. 2020. URL: https://en.wikipedia.org/wiki/Delaunay_triangulation (visited on 04/12/2020) (cit. on p. 12).

[4] *Diskrete Wahrscheinlichkeit.* Mar. 2020. URL: https://www.mathe-online.at/materialien/Daniela.Eder/files/Diskrete_WK/Zusammenstellung.html (visited on 03/12/2020) (cit. on p. 9).

[5] *Equation of a Circle from 3 Points (2 dimensions).* Mar. 2020. URL: http://paulbourke.net/geometry/circlesphere/ (visited on 03/12/2020) (cit. on p. 7).

[6] *Random sample consensus.* Mar. 2020. URL: https://en.wikipedia.org/wiki/Random_sample_consensus (visited on 03/12/2020) (cit. on p. 6).

[7] *Random sample consensus.* Mar. 2020. URL: https://en.wikipedia.org/wiki/Harris_Corner_Detector (visited on 04/15/2020) (cit. on p. 15).