UNIVERSITY OF TRENTO

140266 - Computer Vision

Sports - Project 1

Authors:
Mari Hetlesaeter - mari.hetlesaeter@studenti.unitn.it - 249595
Eline Lilleboe Karlsen - eline.karlsen@studenti.unitn.it - 249597
Anne Marthe Hvammen Sikkerboel - anne.sikkerboel@studenti.unitn.it - 245997

May, 2024

Contents

1 Introduction

In this project the aim was to create a top view picture of a volleyball court, by stitching together two top view cameras. This is done by calculating the homograhpy matrix and using the RANSAC algorithm. In this paper our solution to the problem will be explained, alongside some theory and explanation of the code.

2 Theory

2.1 Camera Calibration

Camera calibration involves capturing multiple images of a known pattern, like a chessboard, to determine a camera's intrinsic parameters (focal length, optical center) and lens distortion coefficients. The camera matrix obtained from the calibration is used to map 3D points of the world to the 2D image plane. A good calibration is important to generate good stitching.

2.2 Key points and descriptors

Key points are distinctive locations in an image that can be detected and matched across different images. A key point needs to have rich image content. This can be achieved through for example high brightness variations or high color variation (Nayar, 2024c). Edges are usually not distinctive enough, and therefore the key points are often blobs. Blobs are objects or parts of objects with distinct features or shapes. The key points needs to be invariant to image rotation and scaling. When a key point is selected, the key point needs an identification. The identifiers are called descriptors, and are numerical representations of the local image information around each key point. Because the descriptor describes the unique features of the region surrounding each key point, it is suited to match the key points throughout different pictures.

2.3 Homography

A match should be points in the two pictures projecting the same point in the 3D world. This is a valid match and is called an inliners. However some matches are invalid, meaning that they do not represent the same point in the 3D world. This matches are called outliners (Nayar, 2024b).

Homography is a rotation matrix through a point of projection. This means that using the hompography matrix between different images you can map these images onto the same plane (Nayar, 2024a). In equation 1 the connection between the destination and the source picture is visible.

$$\begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} \tag{1}$$

For now there is no known way to determined which matches are inliners and which are outliners before you calculate the homography matrix. This is where the RANSAC algoirithm comes in. Ransac - random sample consencus is an algorithm that randomly chooses the minimum amount of samples necessary from a dataset, and then fits the model based on these random samples. Then the algorithm counts the number of remaining dataponits fitting with this model within a given error rate, these are called the inliners. The number of these inliners determines the score for this specific version of the model. This procedure is repeated N times, and then you chose the model with the highest score. This algorithm works as long as the outliners are less than 50% of the total matches.

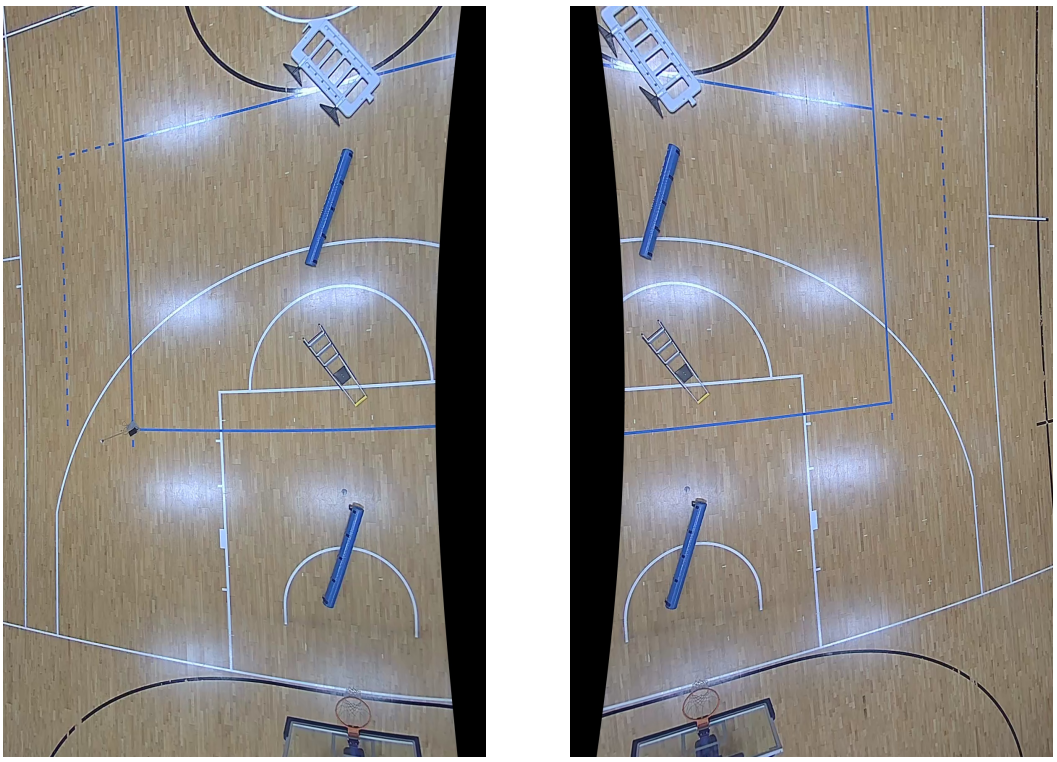
When calculating the homography the minimum number of samples needed from the dataset is four, since each coordinates has one X and one Y value. This is enough to model the homography since it has 8 DOF.

3 Camera Calibration

In the beginning of the project we tried to calibrate each camera of the stereo top view camera using a chessboard video and in addition using manually selected points. None of our calibrations outperformed the mapping matrix found by Morini, 2023. For the rest of the project we used this mapping to obtain the calibrated versions of the cameras.

4 Stitching images

The second part of the project involves to stitch the rectified images together. Since we were unable to correct them ourselves, we used the pre-rectified images that were provided to us. Image stitching entails merging overlapping images to generate a seamless, high-resolution coherent image. With the two images presented in figure 1, an algorithm was developed to generate a composite image. The algorithm was developed by using several functions from OpenCV, which is an open source computer vision library for image processing.



(a) Image 1 (b) Image 2

Figure 1: The images to be stitched together.

4.1 Implementation

Stitching images together involves several steps. Figure figure 2 shows the main steps, which is now going to be explained in more detail below.

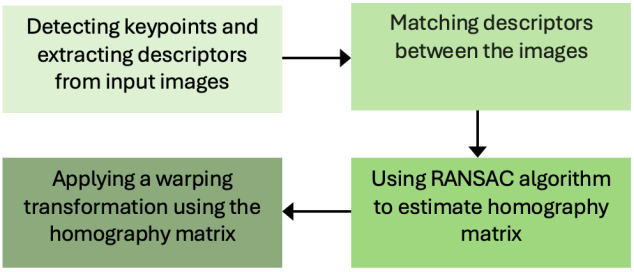


Figure 2: Procedure of stitching images

4.1.1 Calculating key points and descriptors

The first part of the implementation is to compute the key points and descriptors in the two input images, as discussed in the theory part. The key points and descriptors is found by using a SIFT (Scale Invariant Feature Transform) detector. When the SIFT alorithm detects a key point in an image, it calculates the descriptor for the given key point.

The SIFT-algorithm is first initialized using the function cv2.SIFT.create(). Further, the function detectAndCompute() is used to find the keypoints and descriptors for each image, as presented in figure 3.

```
#Find the keypoints and descriptor using SIFT
sift = cv2.SIFT.create()
keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
keypoints2, descriptors2 = sift.detectAndCompute(image2, None)
```

Figure 3: Detecting keypoints and extracting descriptors

4.1.2 Matching descriptors

Now, the descriptors extracted from one image need to be matched and identified within the other image as well. A brute force matcher is created using the cv2.BFMacther() function. Further, this object will be used to perform nearest neighbor matching between the descriptors of the key points in the two images. Using a kNN-matcher with parameter k=2 gives the 2 best matches of each descriptor, which is saved in 'matches'.

```
#Getting the 2 best matches for each descriptor
bf = cv2.BFMatcher()
matches = bf.knnMatch(descriptors1, descriptors2, k=2)
```

Figure 4: Finding matching descriptors

'Matches' is a list of lists, where each sublist contains 'k' objects. Since we have chosen k=2, each sublist will consist of the two best matches corresponding to each key point. Because a feature can occur at multiple locations within an image, we need to iterate through each sublist corresponding to the different key points to identify the best one. Finding the best match is done by applying the ratio test. If the distance to the first match is less than the distance to the second match multiplied by a specific ratio, the match is considered as good, and appended to the lists 'good_points' and 'good_matches'.

```
#Applying the ratio test
good_points = []
good_matches=[]
for m in matches:
    m1,m2 = m[0], m[1]
    if m1.distance < ratio * m2.distance:
        good_points.append((m1.trainIdx, m1.queryIdx))
        good_matches.append([m1])
```

Figure 5: Ratio test, with ratio = 0.70

The list of matches saved in 'good_matches' is used to create a visual representation of the behavior of the matching algorithm. Figure 6 illustrates the outcome of the matching key points between image 1a and image 1b.

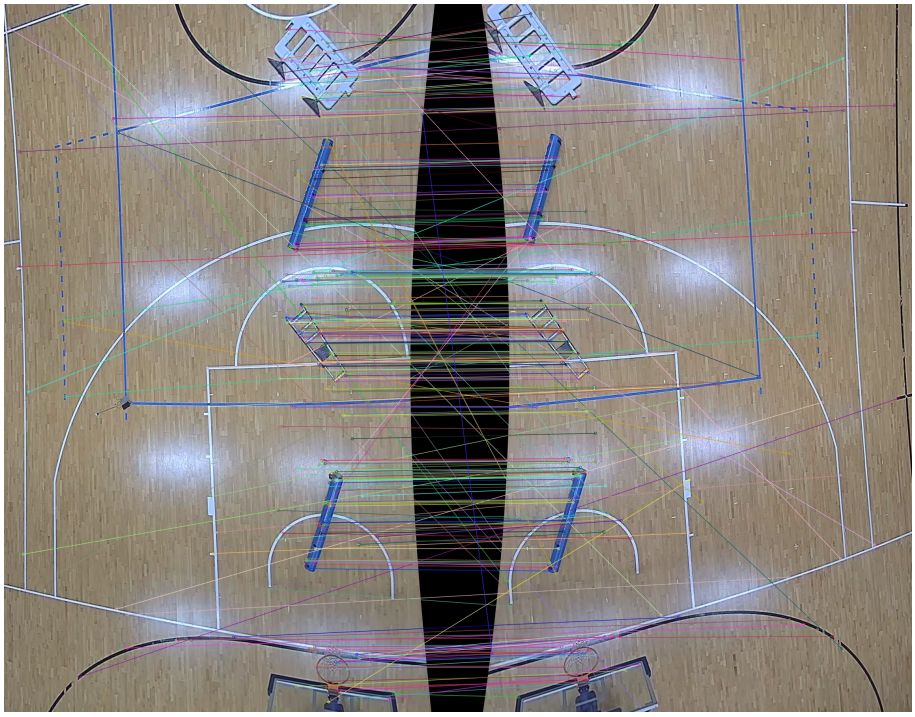


Figure 6: Top matches between the two images

4.1.3 Calculating homography matrix

For calculation the homography matrix the openCV function findHomography is used as seen in 7. First it is essential that there are over 4 good matches between the two pictures for the homography to be calculated. Afterwards the location of the good matches are extracted as ImageKeypoints of picture 1 and 2. These Imagekeypoints are used in the findHomography function. In addition it is specified that the algorithm for finding the homography is the previously mentioned RANSAC algorithm. The number 5.0 is the treshold used by RANSAC algorithm to determine if the match is an inliner match or an outliner match. Based on the result of the warping it can be concluded that over 50% of the good matches are inliner matches.

```
#Finding the coordinates of the good matches and computing the Homography matrix
if len(goodPoints) >= 4:
    image1Keypoints = np.float32([keypoints1[m1.queryIdx].pt for [m1] in goodMatches])
    image2Keypoints = np.float32([keypoints2[m1.trainIdx].pt for [m1] in goodMatches])
    H, _ = cv2.findHomography(image2Keypoints, image1Keypoints, cv2.RANSAC,5.0)
else: print("Not enough matches to compute homography")
return H
```

Figure 7: The code to calculate the homography

4.1.4 Warping transformation

The last part of the algorithm is to stitch the images together. First, the function createMasks() is used to specify which parts of the picture that should be processed, based on if it is the image to the left or to the right. The function creates an empty mask that has the same dimensions as the final image (i.e. the dimensions of image 1 plus image 2), with all pixel values set to 0. Figure 8 illustrates how the empty mask is divided into different regions.

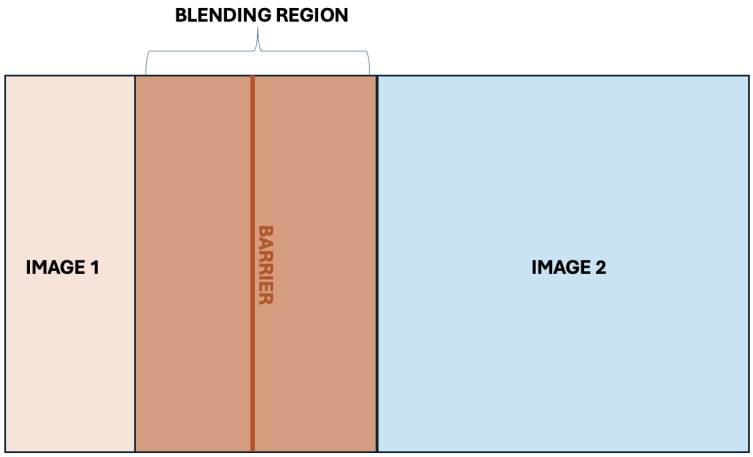
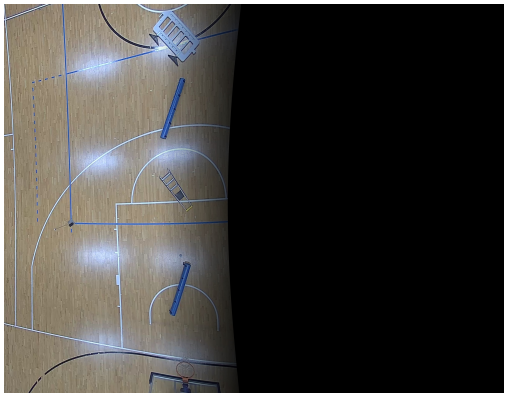
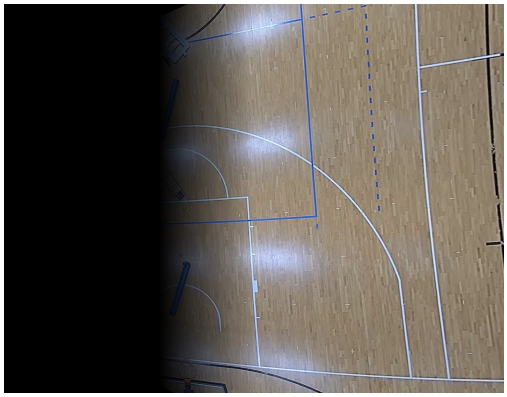


Figure 8: Overview over the masks

In the middle of the blending region there is a barrier positioned, where the two images are blended together. The pixels outside the blending region are set to the respective pixel values that the left and right images originally have in this area. Inside the blending region, each pixel is assigned a blending weight between 0 and 1, which is calculated using a linearly spaced array between 0 and 1. In figure 9 it is presented how the images blend around the barrier.



(a) Image 1 blended



(b) Image 2 perspective transformated and blended

Figure 9: The blended images to be stitched together.

Further, the function stitchingImages(), which blends the two input images together and creates a stitched image, is applied. Because the two pictures is taken from different view point, we have to apply a perspective transformation to make it seems like they are taken from the same viewpoint. This is done by the function warpPerspective(), which uses the homography matrix H and warps image 2 to align with image 1. In the end, the result saved in the variable perspectiveTransformation is multiplied with the existing pixel values of the right part, to make the blending smooth. After finding the pixel values for both the left and right part of the image, these are added together, and the images are blending together into the final result.

4.2 Result

The final result of the stitching is shown in figure 10 below. After testing various values for the size of the blending region, we found that the results were best with a width of 80, meaning 40 pixels extending from each side of the barrier.

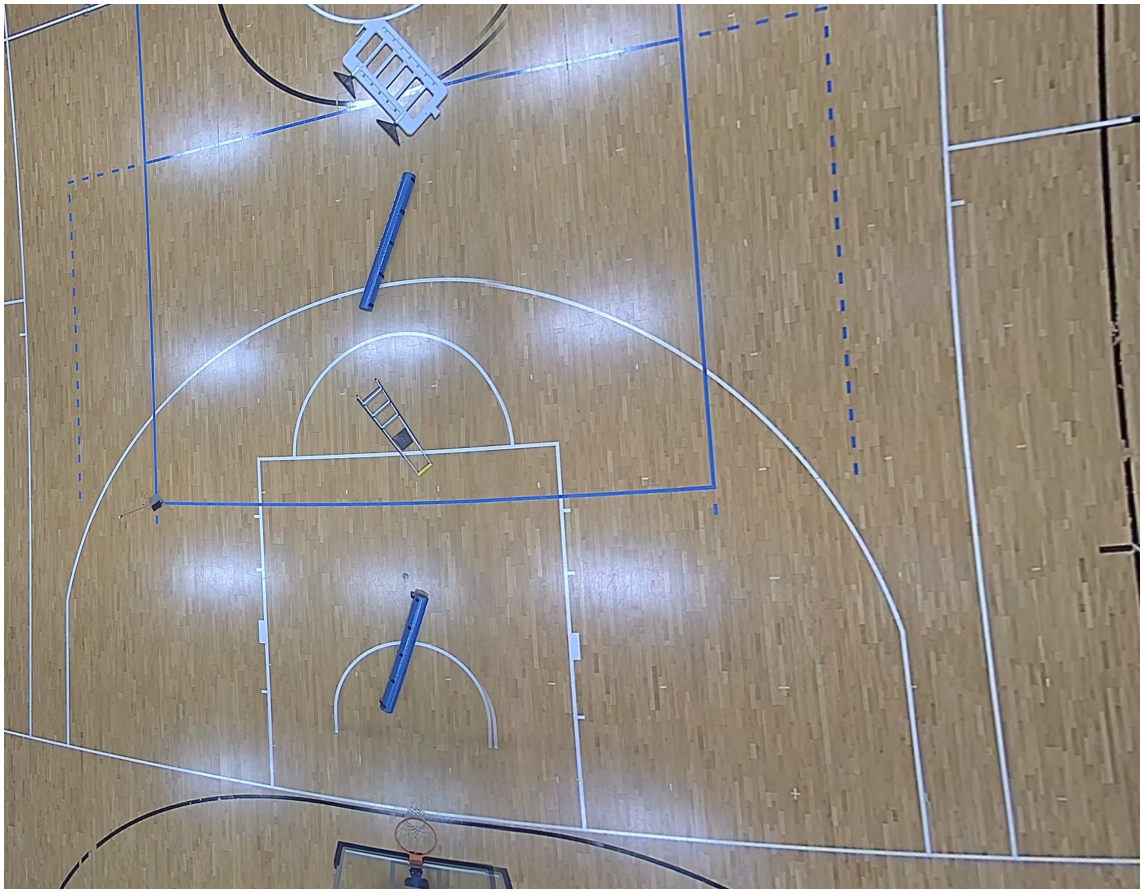


Figure 10: Final result, with blending region equal to 40*2

References

Morini, Martino (2023). 'CALIBRAZIONE DI UNA CAMERA MULTI-OTTICA SENZA OGGETTO DI RIFERIMENTO'. In.

Nayar, Shree (2024a). *Computing Homography — Image Stitching*. URL: https://www.youtube.com/watch?v=I_qjO4cM74o (visited on 21st May 2024).

— (2024b). *Dealing with Outliers: RANSAC — Image Stitching*. URL: https://www.youtube.com/watch?v=EkYXjmiolBg (visited on 21st May 2024).

— (2024c). *What is an Interest Point? — SIFT Detector*. URL: https://www.youtube.com/watch?v=wcqbiHonfbo (visited on 21st May 2024).