Assignment 4

Viljar Bergstøl

IKT213

Machine Vision

Department of Information and Communication Technology

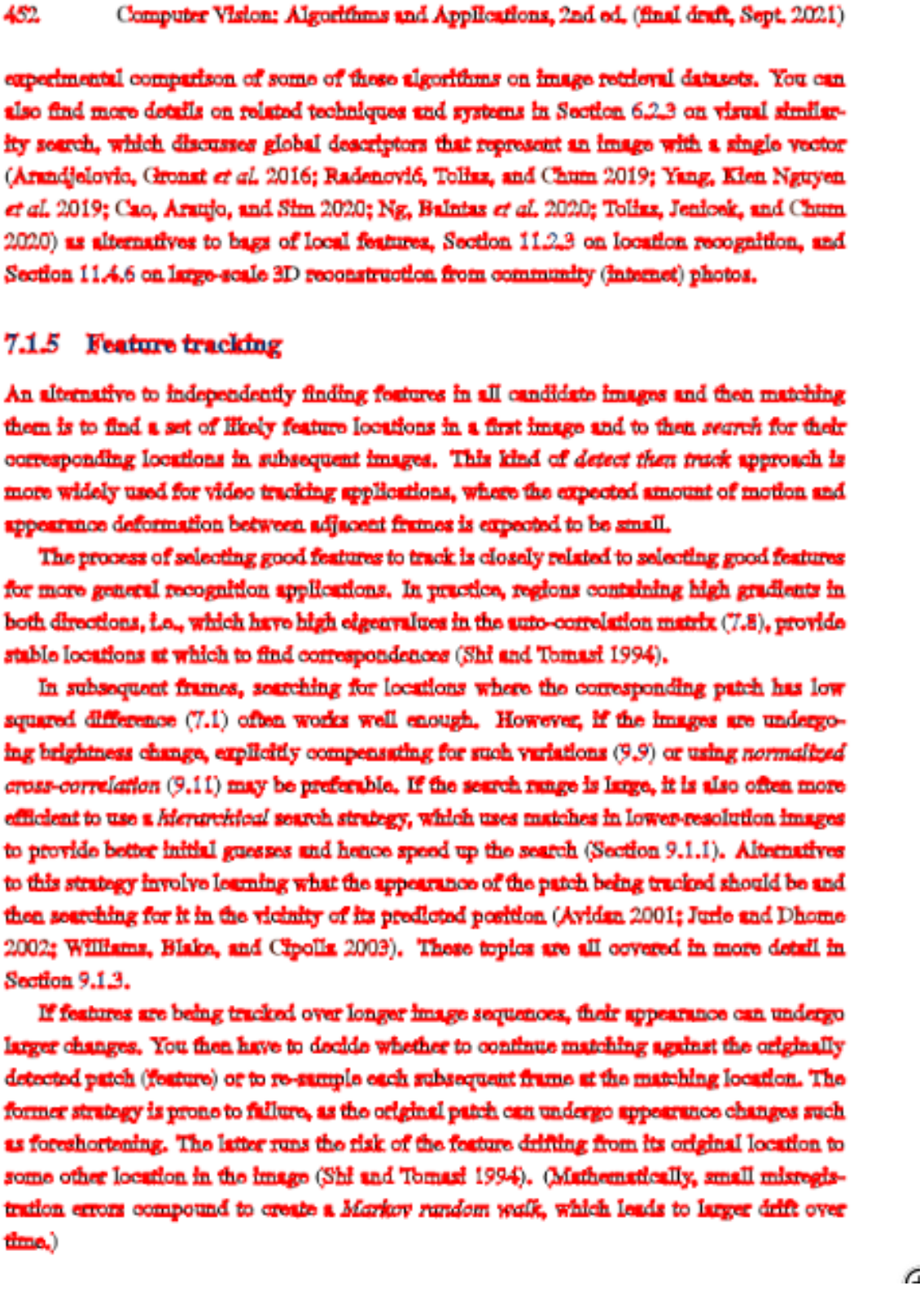
Faculty of Engineering and Science

November 11, 2025

URL to repository: https://github.com/viljarb0/ikt213_bergstol

Task 1

The goal in this task is to detect and highlight all edges of shapes in an image. To test our program, we use an image of a page of text which was provided in the task description. Our program uses SIFT to detect the edges, because SIFT is more accurate than SURF. SURF is significantly faster than SIFT; however, since we are dealing with only a few small images in this case, we prioritize accuracy over performance.



452 Computer Vision: Algorithms and Applications, 2nd ed. (final draft, Sept. 2021)

experimental comparison of some of these algorithms on image retrieval datasets. You can also find more details on related techniques and systems in Section 6.2.3 on visual similarity search, which discusses global descriptors that represent an image with a single vector (Arandjelovic, Gronat *et al.* 2016; Radenović, Tolias, and Chum 2019; Yang, Kien Nguyen *et al.* 2019; Cao, Araujo, and Sim 2020; Ng, Balntas *et al.* 2020; Tolias, Jenicek, and Chum 2020) as alternatives to bags of local features, Section 11.2.3 on location recognition, and Section 11.4.6 on large-scale 3D reconstruction from community (Internet) photos.

7.1.5 Feature tracking

An alternative to independently finding features in all candidate images and then matching them is to find a set of likely feature locations in a first image and to then *search* for their corresponding locations in subsequent images. This kind of *detect then track* approach is more widely used for video tracking applications, where the expected amount of motion and appearance deformation between adjacent frames is expected to be small.

The process of selecting good features to track is closely related to selecting good features for more general recognition applications. In practice, regions containing high gradients in both directions, i.e., which have high eigenvalues in the auto-correlation matrix (7.8), provide stable locations at which to find correspondences (Shi and Tomasi 1994).

In subsequent frames, searching for locations where the corresponding patch has low squared difference (7.1) often works well enough. However, if the images are undergoing brightness change, explicitly compensating for such variations (9.9) or using *normalized cross-correlation* (9.11) may be preferable. If the search range is large, it is also often more efficient to use a *hierarchical* search strategy, which uses matches in lower-resolution images to provide better initial guesses and hence speed up the search (Section 9.1.1). Alternatives to this strategy involve learning what the appearance of the patch being tracked should be and then searching for it in the vicinity of its predicted position (Avidan 2001; Jurie and Dhome 2002; Williams, Blake, and Cipolla 2003). These topics are all covered in more detail in Section 9.1.3.

If features are being tracked over longer image sequences, their appearance can undergo larger changes. You then have to decide whether to continue matching against the originally detected patch (feature) or to re-sample each subsequent frame at the matching location. The former strategy is prone to failure, as the original patch can undergo appearance changes such as foreshortening. The latter runs the risk of the feature drifting from its original location to some other location in the image (Shi and Tomasi 1994). (Mathematically, small misregistration errors compound to create a *Markov random walk*, which leads to larger drift over time.)

Figure 1: The output image from our program.

In the output image we see that all the letters have been highlighted in red.

Task 2

In this task we implement a program for feature-based image alignment. The program takes a target image and warps it so it matches a reference scene. The program loads both inputs provided in the task description, extracts up to 10 SIFT features per image, and uses a FLANN KD-tree matcher to pair descriptors. It filters matches with Lowe's 0.7 ratio test, falls back to the best subset when matches are sparse, and then estimates a homography via RANSAC (with a direct fallback) to align the target image. Finally, it saves the warped result (aligned.jpg) and a diagnostic visualization (matches.jpg) so we can confirm the quality of the alignment.

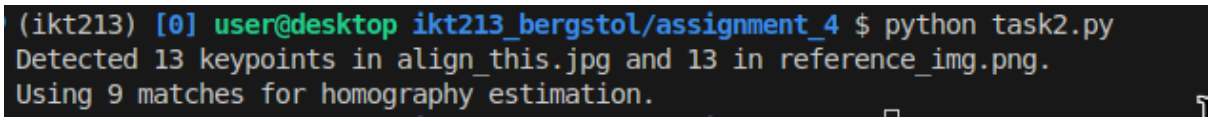


Figure 2: Running the program.

We see from the output of our program that it found 13 keypoints, and used 9 matches for homography estimation.

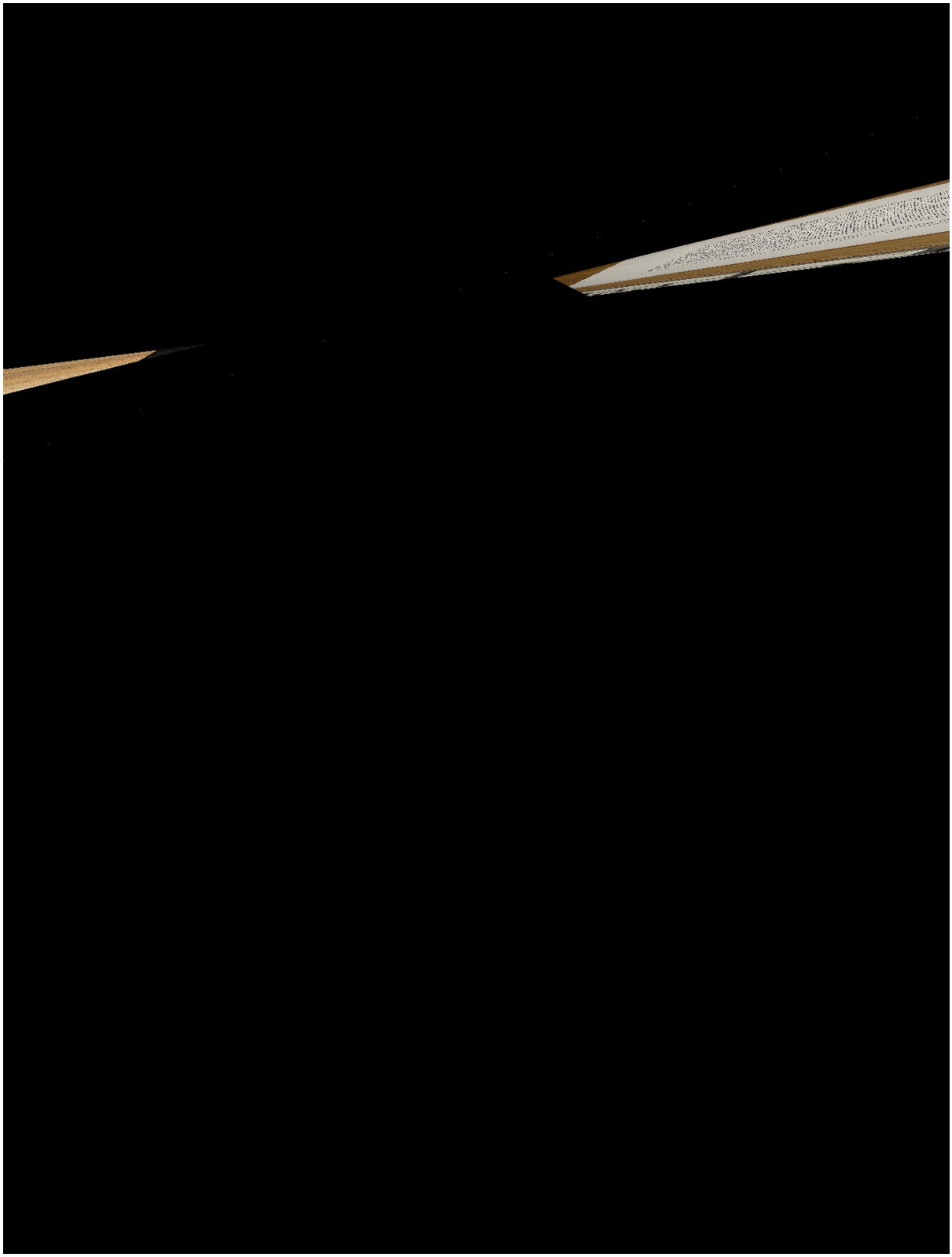


Figure 3: The **aligned** image.

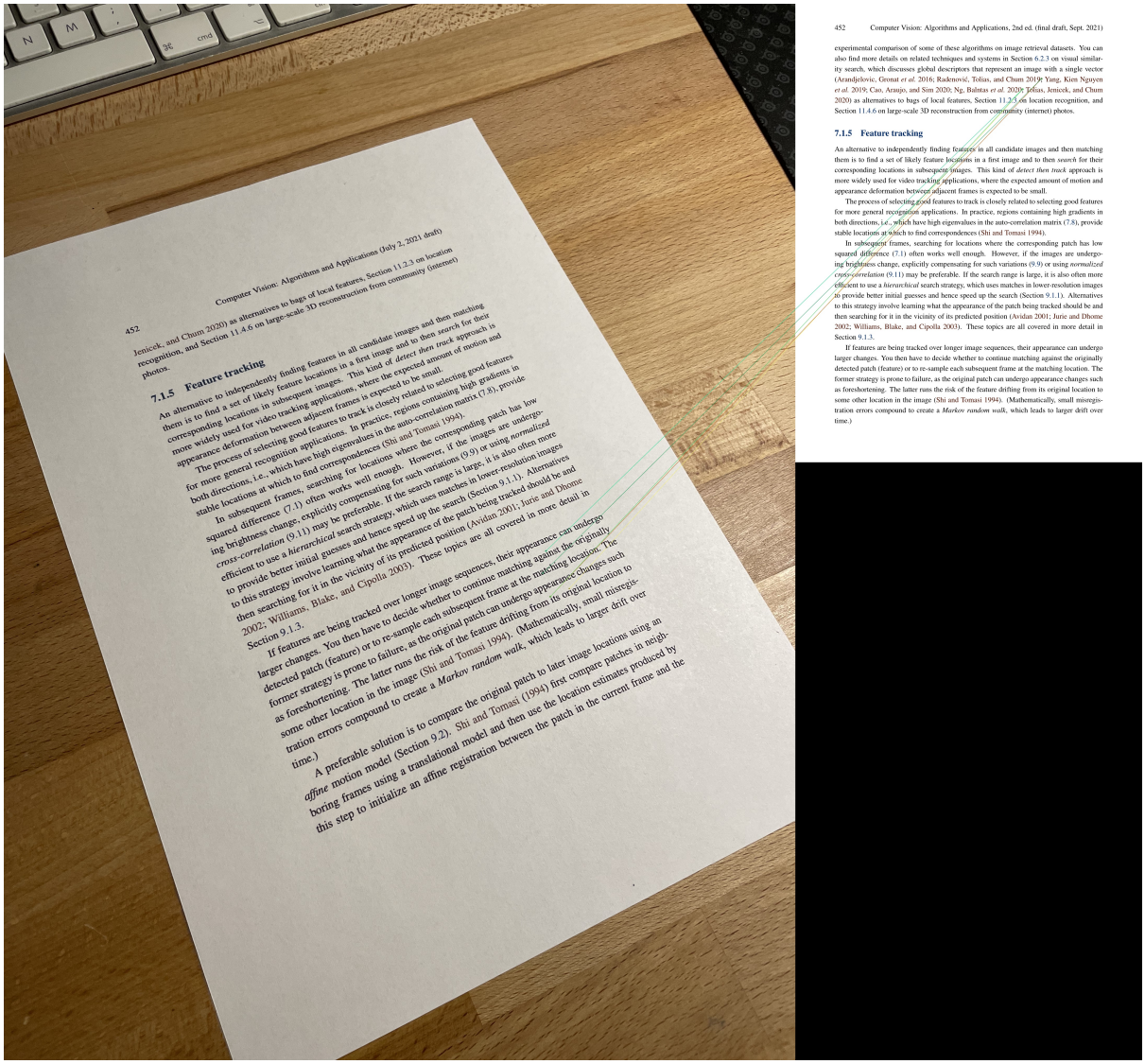


Figure 4: The **matches** image.