AUTOMATED RECOGNITION OF ROAD LINES IN AERIAL IMAGES

CARLOS CONTRERAS, KERAN LI, YI SUI, LI WANG, TINGZHOU YU, AND JUNJIE ZHU

Abstract. Training of object detection models requires a large number of images with identification of the regions of interest. Currently, the selection of regions of interest is done manually, which could be inefficient and inaccurate. In this project, we developed an automated recognition and labeling tool for road lines in aerial road and parking lot images, based on image processing techniques and unsupervised machine learning methods. The tool we created here contains three major steps, including color filtering, edge detection, and objects recognition and labeling. We demonstrate how this tool works by presenting an example where the yellow parking lot lines are successfully identified and labeled. We conclude this project with a summary of future work could be done to improve this tool.

1. Introduction

Identifying road lines and marks is an important task in monitoring traffic and parking lot occupancy. For instance, determining the real-time occupancy percentage of a parking lot. One of the challenges here is to identify objects in the images, such as yellow lines to delimit parking spaces. This process can be done manually using computing vision programs, although this is not time- efficient and robust.

Machine learning methods can help automate the recognition process. If objects were previously classified for some images, we could use supervised machine learning to classify the same objects in different images. In order to obtain a set of images with identified objects, such as yellow lines, we can use unsupervised machine learning in combination with image processing techniques. The difficult task is to distinguish yellow lines from other yellow marks (such as handicap marks and no parking yellow stripes) and yellow patterns in the ground or vehicles. The goal of this project is to identify yellow and white lines in images of parking lots and roads using image processing techniques and unsupervised machine learning.

2. Methods

In this section, we introduce the method we used for automated road lines recognition. The method contains 3 main steps, including color filtering, edge detection, and Hough line detection and shape classification with box labeling. Note that we only mention yellow line recognition here for demonstration purpose; however, the method could be applied to any color line recognition. We implement this method on Python with the help of various libraries, such as OpenCV [1], Scikit-Learn [7], and Pillow [3]. We will present some of the results in Section 3.

2.1. **Color Filtering.**

Since our goal is to recognize the yellow lines, it would be beneficial to identify all yellow pixels first. We select a standard RGB (Red, Green, Blue) value for the yellow color and look for all pixels in the images with RGB values close to that of the standard yellow. Here are some methods we have attempted.

2.1.1. *HSV Filter.*

One way to extract the yellow pixels is to convert the RGB values to HSV (hue, saturation, value), and we identify yellow pixels with predefined range of hue, saturation, and value/brightness.

Carlos Contreras: carlos.contreras@ualberta.ca
Keran Li: keran.li1@ucalgary.ca
Yi Sui: ysui@sfu.ca
Li Wang: lwang@math.ubc.ca
Tingzhou Yu: tingzhouyu@uvic.ca
Junjie Zhu: jzhu@math.ubc.ca

2.1.2. *Color Quantization with K-Means.* K-Means is an unsupervised machine learning algorithm that groups data into clusters. Given a hyper-parameter K, K-Means can assign rows of data to K clusters, such that each row is assigned to its closest cluster center in Euclidean space, and each cluster center is the mean of data assigned to it.

We applied the K-means algorithm from Scikit-Learn to RGB values of the pixels from all images. To extract the yellow pixels, we only keep the pixels that are in the same cluster as the standard yellow. Compared to the HSV filter, we did not need to specify the range of yellow, which allows the potential that pixels with noise can still be classified as yellow.

2.1.3. *Color Quantization with DBSCAN and HDBSCAN.* DBSCAN (Density-based spatial clustering of applications with noise) and HDBSCAN (Hierarchical density-based spatial clustering of applications with noise) are two other clustering algorithms. Given a hyper-parameter ϵ and *min_samples*, DBSCAN declares two pixels to be in the same cluster if their distance is less than ϵ, and we ignore clusters with less than *min_samples* data points. Compared to K-Means, not every data point needs to be in a cluster, and DBSCAN ensures that pixels with similar values are in the same cluster. HDBSCAN is an improvement of DBSCAN.

However, the run time of these two algorithms is much higher than K-Means. We adapted by resizing the images to 224×224. Also, other pixels are in the same cluster as the standard yellow. Especially with resizing, the RGB values of the pixels space out evenly in the color space, and there is not a large gap between the cluster of yellow pixels and other pixels. Thus, these two algorithms do not perform well. Setting $\epsilon = 5.3$, they excluded almost all pixels. If $\epsilon = 5.4$, they included most of the pixels, so they did not help with extracting the yellow pixels only.

2.1.4. *Color Quantization with Gaussian Mixture Models.* We also applied Gaussian Mixture models to study the color quantization on the images. The Gaussian Mixture model is a probabilistic unsupervised model, and models are learned by using the maximum likelihood estimates (MLE). Given a pixel K-means would classify it as one color only, where the mixture of Gaussian could say with certain probability it is yellow, and with certain probability chance it is green, allowing us the view the color clustering with uncertainty.

We implemented the Gaussian Mixture Models from Scikit-learn with co-variance types "tied" and "diag" and different number of components. Type "tied" showed better results for the image set we have. To identify the yellow pixels from the parking lines, we changed the color modes to RGB for images and follow the same procedures as K-means. We compared the results from K-means and Gaussian Mixture models, and see that Gaussian Mixture models can capture more relevant yellow parking lines pixels from the shadow see Figure 5, which is consistent with our expectations. However, Gaussian mixture was not able to solve the problem with shadow completely.

2.2. **Edge Detection.** Based on the color clustering information, we can now get a masked image with only yellow objects left and then we can move on to do edge detection. To obtain the best edge detection result, we first change the masked image to be grayscale and then apply a Gaussian filtering [8, Section 3.1] to remove noise in the masked image. Following this, Canny edge detection [2] is applied to find the edges for the yellow lines we are interested in. Note that to apply Gaussian filtering, we need to specify the size of the kernel to be used. Moreover, for Canny edge detection, we also need to specify two threshold values, `minVal` and `maxVal`, to classify edge lines and non-edge lines. The value of these parameters used in our experiments will be specified in Section 3. For more information on Canny edge detection along with Gaussian filtering and how to perform it on OpenCV in Python, see [9].

2.3. **Hough Line Detection, Bounding Boxes and Shape Classifier.** Upon obtaining the Canny edge detection result, we can identify objects by creating either line detection based on Hough transform [8, Section 7.4.2] or the bounding boxes based on contours.

Experiments presented in Section 3 apply the probabilistic Hough transform function (`HoughLinesP`) [6] in OpenCV for line detection, and then the detected lines are used as a mask over the original image. Note that, in order to apply the probabilistic Hough transform, we need to specify the values of threshold, the minimal line length, and the maximum line gap. The values used in our experiments will be specified in Section 3. For more information on how to perform Hough transform using OpenCV, see [9].

Bounding boxes are obtained using firstly the function `findCountours` to obtain contours delimiting the detected edges (given by delimiting point), and secondly the function `minAreaRect` to obtain the smallest

rectangle that encloses each contour (given by the corner point of the rectangle). This gives a number of bounding boxes enclosing each of the regions detected in the previous steps.

Note that those rectangles will include lines as well and other identified objects, which are not lines. To distinguish lines from non-lines objects, we classify them based on geometric properties of the bounding boxes using the function `boundingRect`. In particular, we compute the maximum length and aspect ratio (the longest side over the shortest side) of the bounding box since lines will be long and thin (large aspect ration and maximum side). We then select the shapes with aspect ratio largest than 4. Alternatively, we can use clustering methods, such as hierarchical clusters or K-means, to classify long and thin rectangles based on their geometrical properties.

3. Results

In Section 2, we introduced the method we used for automated line recognition. Now we are ready to present the results for some experiments we have done. All experiments presented in this section are performed on the resized 512 by 512 parking lot image shown as Figure 1. Unless specified, we worked on the standard RGB color space with the reference yellow color taken to be $(210, 200, 0)$. If the color quantization is performed, it is always done on all the data images we have. Moreover, a 7 by 7 kernel for the Gaussian filter is applied before Canny edge detection. The minimal and maximal value used in Canny edge detection are 150 and 280, and the threshold, the minimal line length, and maximum line gap for Hough line detection are taken to be 20, 10, and 10 respectively. Finally, it is worth pointing out that all experiments are performed on Python version 3.7.7 with OpenCV version 3.4.2.

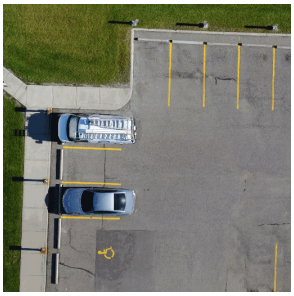
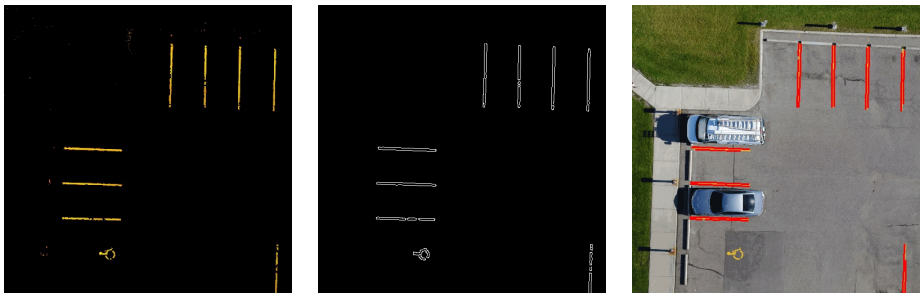


Figure 1. Resized 512 by 512 parking lot image.

In the first experiment, we apply a fixed range yellow color filter defined in the HSV space. Here, we take the lower bound of the yellow filter to be $(22, 60, 140)$ and the upper bound of the yellow filter to be $(60, 255, 255)$. Shown as Figure 2, with applying this color filter, we are able to keep only the yellow objects in the parking lot in the masked image. Following by Canny edge detection and Hough line transform, we can detect the yellow lines in the parking lot accurately (shown as panel (c) of Figure 2). However, a clear drawback for this approach is that we need to predefine a yellow filter in the HSV space, which can be hard to determine and have dependence according to the image. This motivates us to do the next trial, which applies color quantization to filter out the color other than yellow.

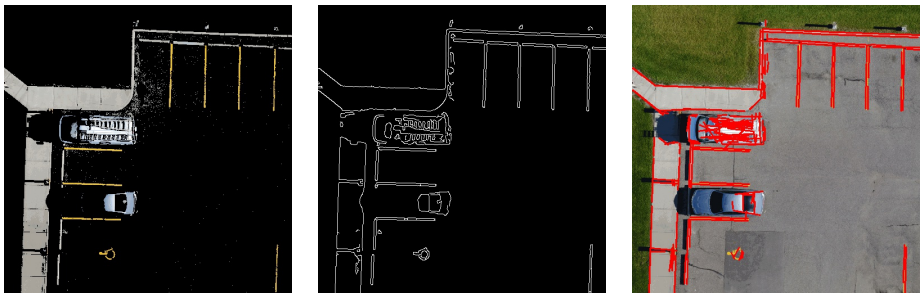
In this experiment, we apply color quantization with K-means clustering of group $K = 5$ in order to filter out the color other than yellow. The corresponding result is shown in Figure 3. In Figure (a), we see that, in addition to the yellow lines in the parking lot, the sidewalk, cars and some other non-yellow objects remain in the masked image. It leads to a noisy Canny edge detection result as shown in panel (b), so that we end up with a very inaccurate detection of the yellow lines of the parking spots. It suggests us that $K = 5$ may be too small for the yellow color filtering, and a bigger number of K should be applied in the clustering to see whether it can improve the result.

Finally, we present the result when either $K = 6$ for K-means or 5 components for Gaussian mixture model is applied for the color quantization in Figure 4. Here, for Gaussian mixture model, we apply the "tied" co-variance. Comparing the K-means with Gaussian mixture model, we see that, by applying either of the clustering method, yellow parking lines can be detected accurately. However, we have used one less cluster group for Gaussian mixture model compared to K-means in this example. After obtaining the edge



(a) (b) (c)

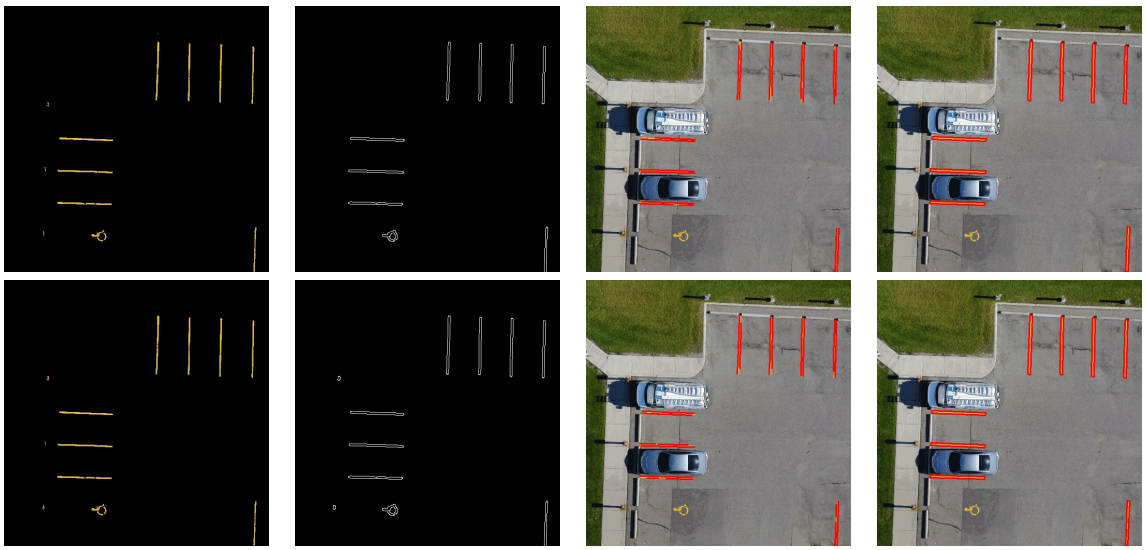
FIGURE 2. Result for a fixed range yellow filter in HSV. (a) Color filtering result, (b) Canny edge detection result, (c) Hough line detection result.



(a) (b) (c)

FIGURE 3. Result for K-means with $K = 5$. (a) Color filtering result, (b) Canny edge detection result, (c) Hough line detection result.

detected image, we can then also apply the shape classification method (described in Section 2.3) to label the yellow lines we are interested in with boxes.



(a) (b) (c) (d)

FIGURE 4. (a) color filtering results, (b) Canny edge detection results, (c) Hough line detection results, (d) box labelling results. Results for K-means are shown on the top and for Gaussian mixture on the bottom.

4. Conclusion and Future Work

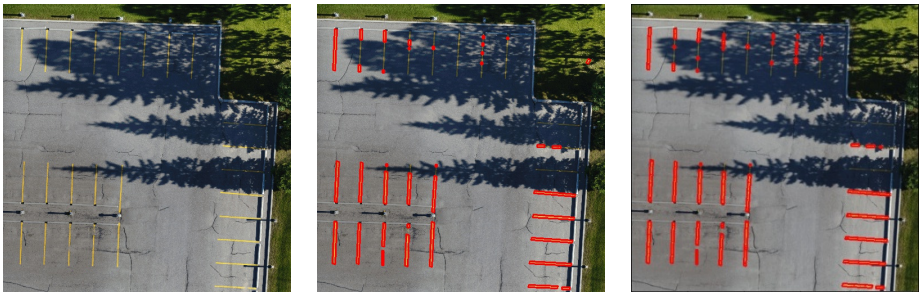
In this work, we use unsupervised machine learning methods and image processing techniques to identify yellow and white lines in aerial images of roads and parking lots. The general pipeline consists of the following sequential steps:

(1) Color quantization. Use unsupervised machine learning (K-means or Gaussian mixture models) to classify colors into clusters. The clustering can be applied to a number of images concatenated together, in which case a larger spectrum of colors is considered and the classifier can be saved for future use.
(2) Color filtering. Keep only yellow (white) pixels based on the color clusters. The output is a copy of the original image (or transformed color model) where all pixels not belonging to the cluster have been removed. This step assumes that all yellow (white) pixels of interest are very similar.
(3) Image smoothing. Apply a filter to eliminate isolated or faint pixels that remain after the color filter.
(4) Edge detection. Apply Canny edge detection to identify all enclosed regions that remain after color and smooth filters. The output is a binary image delimiting all filtered pixels.
(5) Contours. Identify the contour enclosing each of the group of pixels that has been edge detected. The output is a list of contour points delimiting each group of filtered pixels.
(6) Bounding boxes. Determine the minimal rectangle that encloses the contour. The output is a list of four (corner) points for each group of filtered pixels.
(7) Shape recognition. Use unsupervised machine learning (hierarchical clusters or K-means) or heuristic rules (aspect ratio larger than four) to classify and select those rectangles that are thin and long. The output is a subset of the bounding boxes.
(8) Labeling. Identify yellow (white) lines by plotting bounding boxes over the original image.

We believe the pipeline above can be applied outside the data set that we considered. However, more experiments would be required with images including curved or angled road lines, etc. From our experiments with the available data set, we identified a number of issues that we couldn't address due to time constrains. These issues and potential solutions/improvements are explained in the rest of this section.

4.1. **Future Work.**

4.1.1. *Shadow.* Shadow is the main obstacle in this project. For example, from Figure 5 (b) we can see the K-means clustering with Hough line detector cannot find all the yellow lines in the shadow area. This is because K-means can only pick up one yellow color while there is an obvious color difference between the normal yellow color in the bright area and dark yellow color in the shaded area. This can be partially solved by Gaussian Mixture model as shown in Figure 5 (c), in which dark yellow color in shaded area is recognized as the normal yellow color in the bright area with a certain probability. However, we can see some yellow lines in shaded area are still missing from line detection.



(a) (b) (c)

Figure 5. (a) resized original image, (b) K-means result, (c) Gaussian Mixture model result.

One possible way to address the shadow issue is to modify the part of the image inside the shadow area while keeping the part of the image outside of the shadow unchanged. We tried two different types of modifications, to increase the brightness of the shadow area by a certain value, or to modify the RGB color values of the shadow area and make them closer to the ones in the bright area. Both methods start with

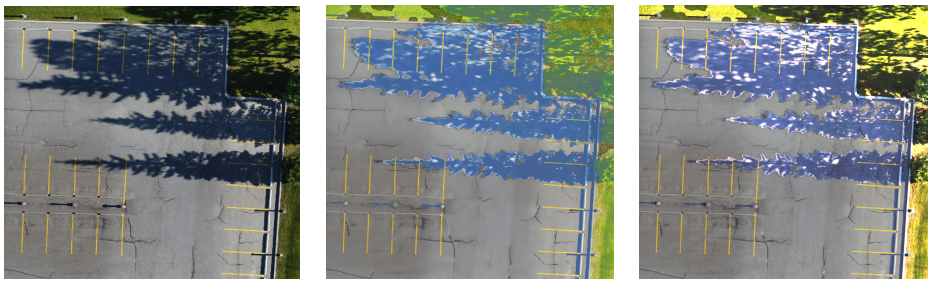
removing the texture of the image by a mean shift filter. Then threshold the image and the shadow area can be picked up. Now change the color space of the shadow area from RGB to HSV so we can increase the brightness of the shaded area. (Or we can multiply the RGB color values of the shadow area to make the shadow area and the bright area have the same means of color values). Finally, we combine the images of the modified shadow area with the normal bright area to obtain the output. We can also use the mean shift filter again or inpainting tool to make the gap between the shadow area and bright area smoother. Figure 6 shows the result of brightening and color modification (the white spots inside shadow in panel (c) come from small components removal after thresholding the original image, which is optional).

Another way to address with shadow parts is to apply the Homomorphic filter [4] to all images in the image process procedures, in order to improve the light exposure. This part is done before the color quantization. To be more specific, we first changed the color modes to HSV. After that, we applied the Homomorphic filter on the low energy frequency on the brightness V channel, because the shadow part is related to the low energy frequency compared to the non-shadow parts. Then, we shifted the mean value of the V channel of the isomorphic filter images back to the original images. We can see some results in Figure 7.

Together with other tools, one of the possible procedures to detect all lines in image with shadow is

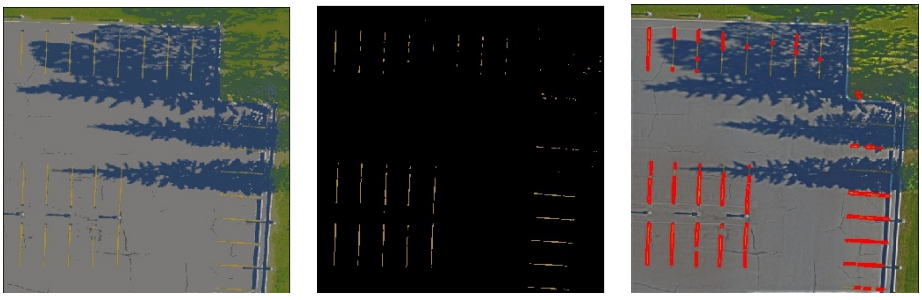
(1) Shadow Modification
(2) Color Quantization
(3) Line Detection

where we can change the order of the first two steps to compare the performance.



(a) (b) (c)

FIGURE 6. (a) original image, (b) brightened shadow, (c) color modified shadow.



(a) (b) (c)

FIGURE 7. (a) Homomorphic filtered image (b) Color filtered image (c) Line detection image with Homomorhpic filter

4.1.2. *Unwanted White Lines.* When detecting white lines on roads, usually curbs are also detected at the same time. This creates problems if we want to distinguish white dash lines or marks in the middle of roads from curbs. As an example, Figure 8 (b) shows curb affecting the detection of white dash lines if we only combine K-means clustering with Hough line detection. Figure 8 (c) shows if we apply one iteration of mean shift filter before K-means clustering, the influence of curbs becomes much smaller. It leaves us future work

to do with further investigation on how to properly apply mean shift filter to only detect the white dash lines in the middle of the road.

(a) (b) (c)

FIGURE 8. (a) original image, (b) direct K-means result, (c) mean shift filtering before K-means result.

4.1.3. *Discontinued Lines.* Some images have discontinued lines due to washed out paint. In this case, two or more objects are identified which may be filtered out in the shape recognition step. We think that such lines can be reconstructed using machine learning based on complete lines present in the same image.

4.1.4. *Shape Recognition.* Currently, the shape recognition uses a fixed number of two clusters to classify lines from non-lines. However, this leads to sub-classifications of lines when there are only lines in the image, and misclassification of non-lines as lines when there are too many non-line objects in the image. We suggest using an unsupervised machine learning method with a variable number of clusters.

4.1.5. *Automated Parameter Selection.* Recall that the goal for this project is to create an automated recognition tool for road lines. When the color filtering step could be done automatically through machine learning techniques, we should not forget the fact that there are many parameters in this method which need to be pre-selected, such as the number of groups in clustering, the minVal, and MaxVal threshold in Canny edge detection, and three parameters in Hough line detection. Moreover, in the newly proposed shadow modification method shown as Figure 6 (b), we also need to manually increase the HSV brightness value. Future work includes using machine learning to automate optimal parameter selection.

ACKNOWLEDGMENTS

We want to thank our industrial mentor Heather Vooys, Brian Bullas, Daniel McReynolds, the rest of the team at AERIUM Analytics, and our academic mentor Matthew Greenberg for their guidance, feedback, and support during the development of this project. We are also very grateful to Kristine Bauer, James Colliander, Ian Allison, and the rest of the PIMS organizing committee and instructors of the Math$^{\text{Industry}}$ 2020 workshop.

REFERENCES

1. G. Bradski, *The OpenCV Library*, Dr. Dobb's Journal of Software Tools, (2000).
2. John Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, **PAMI-8** (1986), no. 6, 679–698.
3. Alex Clark, *Pillow (PIL Fork) Documentation*, 2015.
4. Glasgio, *Homomorphic filter*, `https://github.com/glasgio/homomorphic-filter`.
5. Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, Lawrence Zitnick C., and Poitr Dollár, *Microsoft COCO: Common Objects in Context*, 2014.
6. Jiri Matas, Charles Galambos, and Josef Kittler, *Robust detection of lines using the progressive probabilistic hough transform*, Computer Vision and Image Understanding, **78** (2000), no. 1, 119–137.
7. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research **12** (2011), 2825–2830.
8. Richard Szeliski, *Computer Vision: Algorithms and Applications, 2nd ed.*, 2020.
9. OpenCV Team, *Image processing in OpenCV*, `https://docs.opencv.org/3.4.2/d2/d96/tutorial_py_table_of_contents_imgproc.html`, 2018.