Parking Spot Detection - Signal Image and Video

Karthik Govindarajan
Department of Information Engineering and Computer Science
University of Trento
Trento, Italy
karthik.govindarajan@studenti.unitn.it

***Abstract*—Real-time parking availability information is important in modern cities suffer from a parking problem, which could be real time improvement in decreasing pollution and confusions. In this paper, I present software solution for detecting the availability of on-street parking spaces using computer vision techniques. The algorithms are evaluated using real-time and real world street parking data. The implementation is based on the aerial view of the vision from camera for information of the parking zone by-which the data is transformed into dynamic pixel change in the frames. The algorithm is carried over by adaptive thresholding, dynamic thresholding. The output of the binary image representing the segmentation helps in finding the count of pixels in the specific location by-which the available parking slots are highlighted. Wherein the results are robust on comparison with the complex deep-learning methods.**

***Index Terms*—Real Time Parking, Parking slot application, Image Processing**

I. Introduction

While real-time parking slot detection plays a critical role in valet parking systems, existing methods have limited success in real-world application. There are several approach's for real-time slot detection parking. The rise of car ownership has created an imbalance between parking demand and supply. Computer vision enables the accurate inspection and reading of information using images and high dimensional data. Application can also be used automation, data gathering, and data decoding process. In the current situation, a parking management system that can track parking spots [1] has become a necessity for all major cities. The system has to be scalable, efficient, reliable, and affordable at the same time. Computer vision powered algorithms have shown very promising results in a variety of tasks. Similar techniques can be used to address the problem of parking space detection.

In this paper, detection of the parking space system is visioned from aerial view by-which the layout of the parking lot are segmented and help in providing the count. The camera which gives the feed for the input is fixed wherein the frames generated from the video should be of the same angle. The information of each frames of the feed are obtained and thresholding "Fig. 1" is used to segment an image by setting all pixels whose intensity values are above a threshold to a foreground value and all the remaining pixels to a background value. In our case the provided input to the thresholding operation [2] is typically a grayscale image. As an output we get a binary image representing the segmentation. Black pixels correspond to background and white pixels correspond to foreground (or vice versa). Here we are not determining the segmentation by a single parameter to get the intensity threshold.

Multiple thresholds [3] can also be specified, so that a band of intensity values can be set to white while everything else is set to black. As not all images can be neatly segmented into foreground and background using simple thresholding for this case I approaching the adaptive threshold. In this case, I expect to see a distinct peak in the histogram corresponding to foreground objects such that thresholds can be chosen to isolate this peak accordingly. In this case, adaptive thresholding [4] "Fig. 1" may be a better answer.

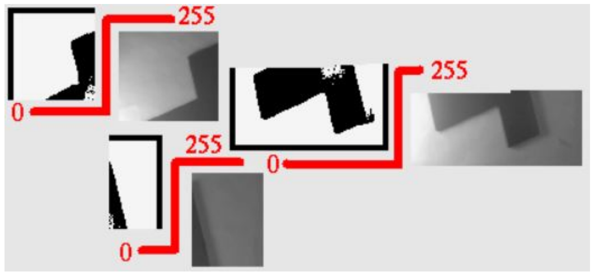


Fig. 1. Adaptive thresholding, Dynamic thresholding

II. Implementation

The feed for this project is based on aerial view of the camera. The video input is read for obtaining the data filtering and following to the counting of available space. The video input which was taken from iStock library, it was further modified before it was feeded to the algorithm. In this case as I explained most recommended screen resolution used in overall monitors. As the main objective is to find the pixel count of the used and unused space in the parking lot the resolution of the video is important as the fact for obtaining the leverage results. I took a mid ranged resolution for working in most case of the monitors, the width of 1366 and height of 768. It can also be of more resolution wherein the pixel count is increased, which is reflecting a quality output in other hand I examined it totally depends on the density of the parking lot. It can also be less resolution video which can be scaled to obtain the output. As stepping to the object detection part it can be obtained in different approaches, here I use a simple technic with a robust results. The most commonly used approach for parking lot detection deep learning combined with Mask-RCNN object

[5] , most commonly used edge detection algorithm (Canny edge detector) followed with extracting the co-ordinates and manipulating it to detect the parking space. In first approach the video feed can be of different angle and it has to be trained according to the visual in the other hand the edge detection of the lot is made and segmentation [6] is done.

In my case I capture the space of the parking zone by a rectangular shaped object from the feed. The process is of two main steps the first is followed by capturing a frame (image) of the video and use it for positioning the rectangular objects as shown from the “Fig. 2”

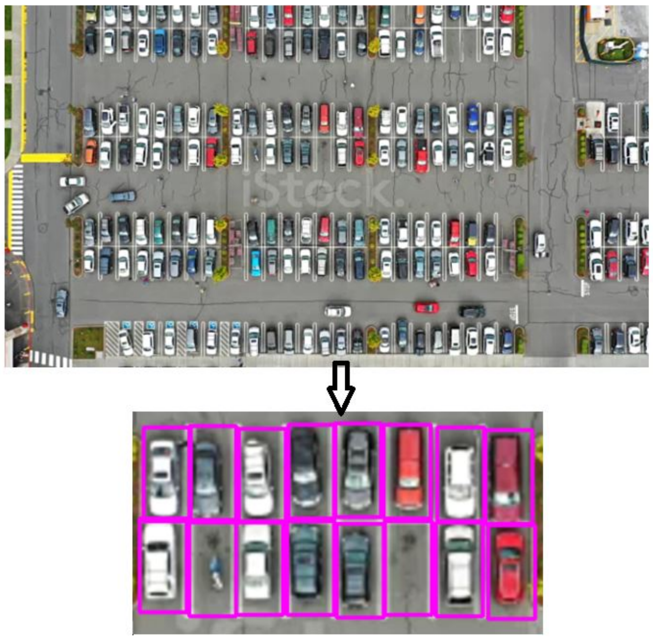


Fig. 2. Parking lot in aerial view (Input feed for the algorithm)

Library used here for image processing is OpenCV. OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV [7] was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV [8] makes it easy for businesses to utilise and modify the code. The working flow is saving the rectangular space as an object in the picture (frame which was taken from the video) and pickle library create a file with the information of the objected created. The object-position file is looped inside to detect the live usage of the rectangular space which was captured from the static picture. The flow chart of the above is shown in the “Fig. 3”

The captured frame of the video is taken as a feed and with the help of mouse click event the slot of the parking lot is marked. As the picture is of resolution 1366 x 768 the slots which are marked are occupying the pixels in the form of

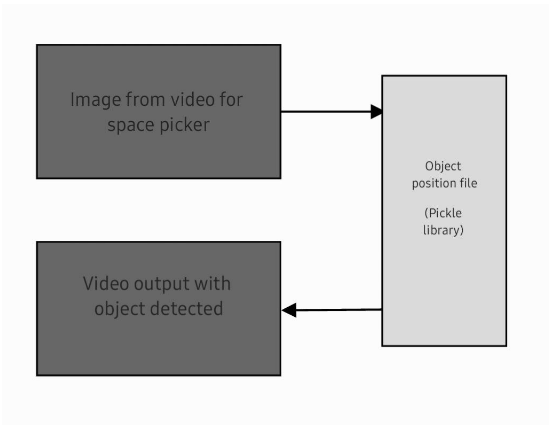
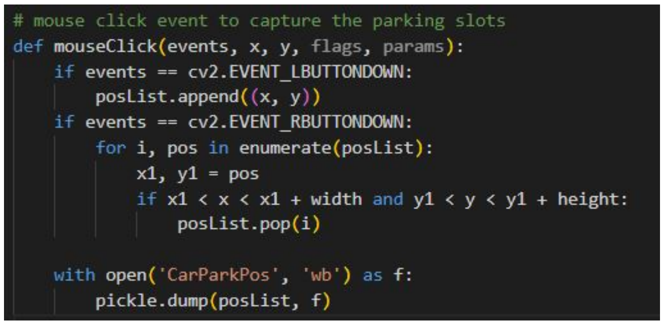


Image from video for space picker
Object position file (Pickle library)
Video output with object detected

Fig. 3. Flowchart for marked slots parameter through pickle library

width and height (rectangle) in the picture which in-return is going to help define the pixel count while the objects in the video move. Code for selecting the rectangular mark is shown in the figure “Fig. 4”



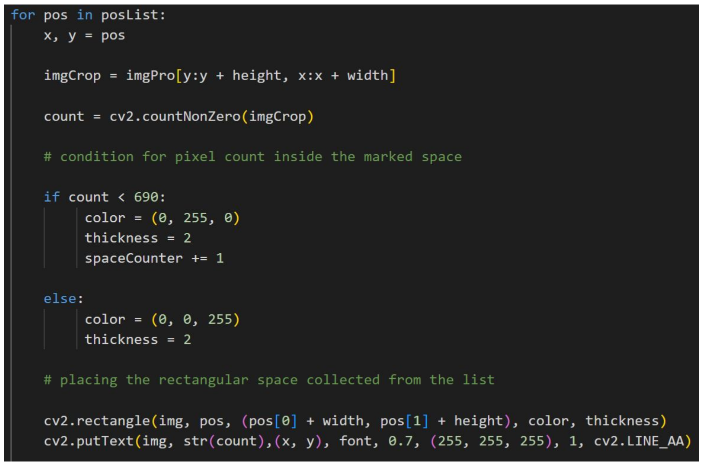
```
# mouse click event to capture the parking slots
def mouseClick(events, x, y, flags, params):
    if events == cv2.EVENT_LBUTTONDOWN:
        posList.append((x, y))
    if events == cv2.EVENT_RBUTTONDOWN:
        for i, pos in enumerate(posList):
            x1, y1 = pos
            if x1 < x < x1 + width and y1 < y < y1 + height:
                posList.pop(i)

    with open('CarParkPos', 'wb') as f:
        pickle.dump(posList, f)
```

Fig. 4. Code for selection of the rectangular mark by click event

The file which is generated for the park position is called by pickle.load to use the information of the marked rectangular mark. A position coordinates are taken as list and looped with the same width and height of the rectangle. A loop is made to run to find the change in count of pixel “Fig. 5” inside the rectangle which reflects the change of colour on the rectangle. This makes the change in count of the available space.

Applying noise reduction [9] was required in my case, as the objective is to count the pixel of the rectangular area. Also applying threshold to the smoothened image was the idealogy for my parking lot case. Getting back the position from the pickle object file and positioning the marked shape in the video. By this we loop the marking on every single frame which makes them stick with the video pixels. The colour difference of the filled and available slot is shown in the “Fig. 6”

Low pass filter is applied to smoothen the image. As smoothing technique is to eliminate noises in images and videos. In this technique, the image should be convolved with a Gaussian kernel [10] to produce the smoothed image. Before gaussian blur Often, the grayscale intensity is applied as the they are stored as an 8-bit integer giving 256 possible different



```
for pos in posList:
    x, y = pos

    imgCrop = imgPro[y:y + height, x:x + width]

    count = cv2.countNonZero(imgCrop)

    # condition for pixel count inside the marked space

    if count < 690:
        color = (0, 255, 0)
        thickness = 2
        spaceCounter += 1

    else:
        color = (0, 0, 255)
        thickness = 2

    # placing the rectangular space collected from the list

    cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color, thickness)
    cv2.putText(img, str(count),(x, y), font, 0.7, (255, 255, 255), 1, cv2.LINE_AA)
```

Fig. 5. Code for positioning the marked shape in the video by looping the pickle object file

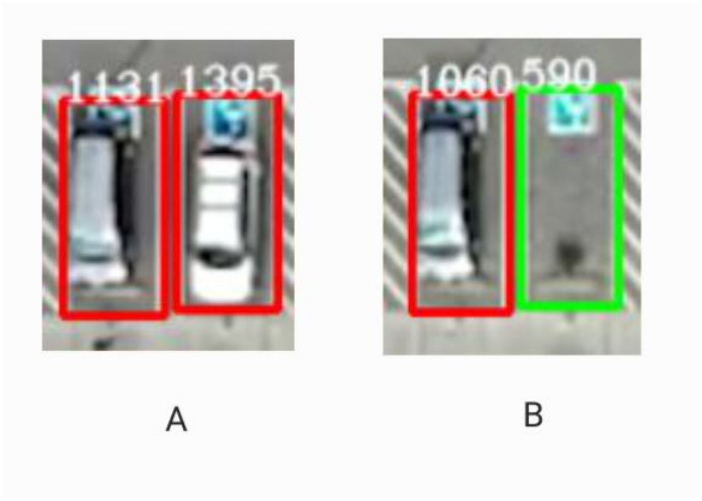
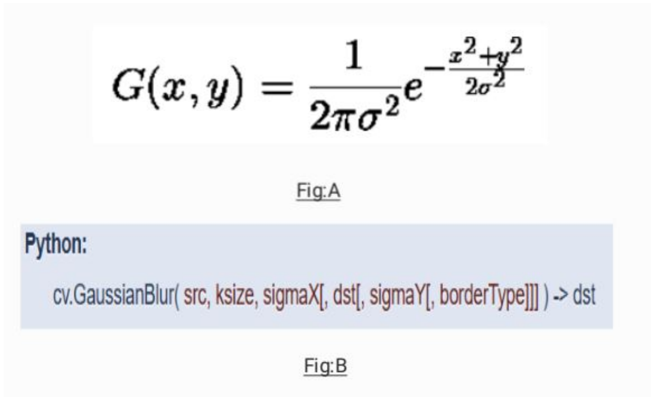


Fig. 6. Group A: Filled slot in parking & Group B: Available slot in parking with the count of pixel on top

shades of gray from black to white. If the levels are evenly spaced then the difference between successive graylevels is significantly better than the graylevel resolving power of the human eye. Grayscale images are very common, in part because much of today's display and image capture hardware can only support 8-bit images. In addition, grayscale images are entirely sufficient for many tasks and so there is no need to use more complicated and harder-to-process color images. Convolution is simply the process of taking a small matrix called the kernel and running it over all the pixels in an image. At every pixel, I perform some math operation involving the values in the convolution matrix and the values of a pixel and its surroundings to determine the value for a pixel in the output image.Process of using Gaussian Filter on an image we firstly define the size of the Kernel/Matrix that would be used for demising the image wherein I am using 3x3 matrix for kernel. The formula for 2 Dimensional gaussian function is implemented via opencv library via python language in my approach "Fig. 7". Wherein the σ Standard Deviation is kept 1 in my case.



$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Fig:A

Python:

```
cv.GaussianBlur( src, ksize, sigmaX[, dst[, sigmaY[, borderType]]] ) -> dst
```

Fig:B

Fig. 7. Fig A: Gaussian Blur formula & Fig B: Implementation with opencv library

You may define the size of the kernel according to your requirement. But the standard deviation of the Gaussian distribution [11] in X and Y direction should be chosen carefully considering the size of the kernel such that the edges of the kernel is close to zero. Here I have shown the 3 x 3 and 5 x 5 Gaussian kernels. A 3×3 Gaussian Kernel Approximation(two-dimensional) with Standard Deviation = 1, appears as follows "Fig. 8"

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

Fig. 8. 3×3 Gaussian Kernel

Followed Gaussian function and we'll generate an n x m matrix. Using this matrix and the height of the Gaussian distribution [12] at that pixel location, we'll compute new RGB values for the blurred image. Heading towards removing the salt and pepper in the specific space the parking lot is bit complicated when has the data which can get changed when more pixels in come is done. Adaptive Threshold is one of the good options is obtaining the results in a leverage. Adaptive thresholding typically takes a grayscale or color image as input and, in the simplest implementation, outputs a binary image representing the segmentation. For each pixel in the image, a threshold has to be calculated. If the pixel value is below the threshold it is set to the background value, otherwise it assumes the foreground value. However,the methods are global thresholding techniques, implying that the same value of T is used to test all pixels in the input image, thereby segmenting them into foreground and background.

The problem here is that having just one value of T may not suffice. Due to variations in lighting conditions, shadowing, etc., it may be that one value of T will work for a certain part of the input image but will utterly fail on a different segment.

Instead of immediately throwing our hands and claiming that traditional computer vision and image processing will not work for this problem (and thereby immediately jumping to training a deep neural segmentation network like Mask R-CNN or U-Net), we can instead leverage adaptive thresholding.

Depending on your project, leveraging adaptive thresholding can enable you to:

1. Obtain better segmentation than using global thresholding methods, such as basic thresholding and Otsu thresholding [13]

2. Avoid the time consuming and computationally expensive process of training a dedicated Mask R-CNN or U-Net segmentation network

III. Mathematics underlying adaptive thresholding

As I mentioned above, our goal in adaptive thresholding is to statistically examine local regions of our image and determine an optimal value of T for each region. The question which pressured here is - Which statistic do we use to compute the threshold value T for each region?

It is common practice to use either the arithmetic mean or the Gaussian mean of the pixel intensities in each region (other methods do exist, but the arithmetic mean and the Gaussian mean are by far the most popular).

In the arithmetic mean, each pixel in the neighborhood contributes equally to computing T. And in the Gaussian mean, pixel values farther away from the (x, y)-coordinate center of the region contribute less to the overall calculation of T.

The general formula to compute T is thus:

$$T = mean(IL) - C$$

where the mean is either the arithmetic or Gaussian mean, IL is the local sub-region of the image, I, and C is some constant which we can use to fine tune the threshold value T. As we use opencv library for the implementation of the adaptive threshold the following function is called from cv library.
Function used from opnecv library:

$$cv.adaptiveThreshold(src, maxValue, adaptiveMethod, \\ thresholdType, blockSize, C[, dst])$$

The resulted output blurred using the median filter has the third step. The function smooths an image using the median filter with the ksize×ksize aperture. Each channel of a multi-channel image is processed independently. Following to that the a kernel is created by numpy library for making the dilation. A kernel(a size(3,3) is convolved with the image. A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel are 1, otherwise, it is eroded (made to zero). Thus all the pixels near the boundary will be discarded depending upon the size of the kernel. So the thickness or size of the foreground object decreases or simply the white region decreases in the image. The objective of dilation [14] which is added in last of the line is to increase the object area and is Used to accentuate features. "Fig. 9"

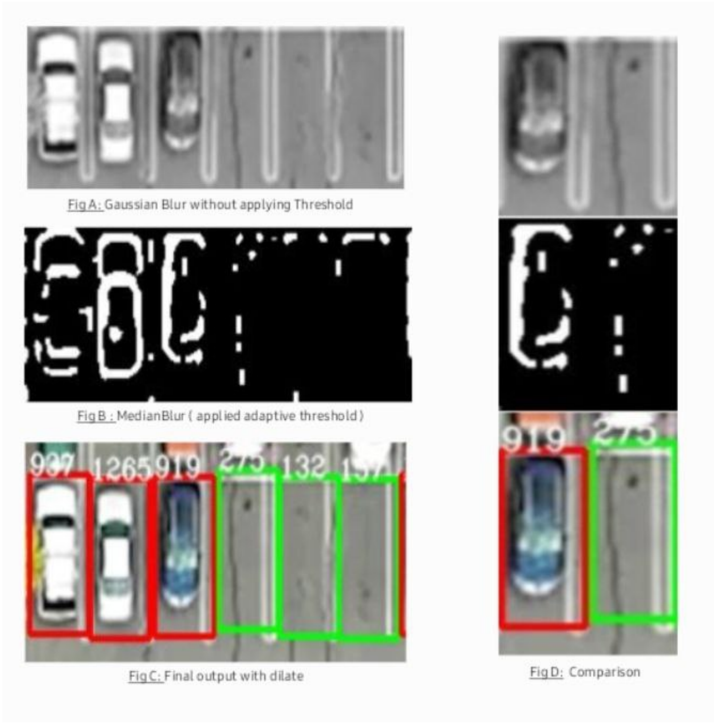


Fig A: Gaussian Blur without applying Threshold
Fig B : MedianBlur (applied adaptive threshold)
937
1265
919
275
132
157
Fig C: Final output with dilate
919
275
Fig D: Comparison

Fig. 9. Gaussian blur, Median blur applied effect on pixel count in rectangular area

Real time count of the available slot of the parking zone is defined by examining the count of the pixel. If the certain number of the pixel count is increasing the slot is changed in colour Red wherein while count is decreasing the slot change in green. The total number of parking slots is 136 in number "Fig. 10"

.

The free slots are taken out live from the reading and printed on the top right of the video. This methodology can be implemented in different UI (User Interface) the information of the positioning can be chained with the database.

IV. Conclusion

In this work, a robust approach for detecting and counting the free vehicle parking lots is implemented. Initially, all the available parking lots are manually marked in rectangular bounding box. Then image processing techniques such as RGB to Gray image conversion, Gaussian Blur, Adaptive

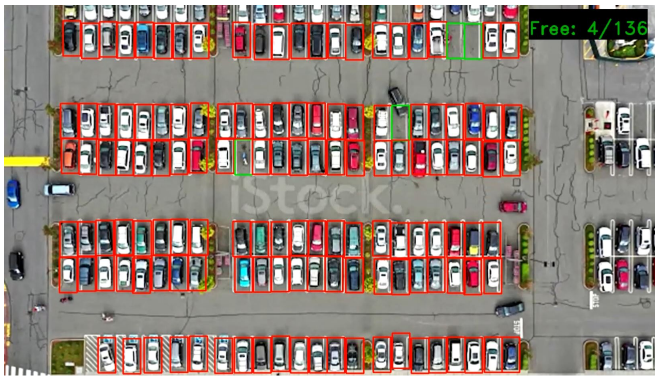


Fig. 10. Display of available slots in parking lot (current image has 4 free slots out of 136 slots)

thresholding, Median Blur [15] and dilation are performed to find the required pixels for locating the free parking lot. Based on the thresholding values the occupied lot are marked in red color, the available parking lots are marked in green color. The proposed methodology is out performed in correctly detecting and count the free vehicle parking lots.

References

[1] Rahat Iqbal, Tomasz Maniak, and Charalampos Karyotis. "Intelligent Remote Monitoring of Parking Spaces Using Licensed and Unlicensed Wireless Technologies". In: *IEEE Network* 33.4 (2019), pp. 23–29. DOI: 10.1109/MNET.2019.1800459.

[2] Johan Wahlström et al. "Zero-Velocity Detection—A Bayesian Approach to Adaptive Thresholding". In: *IEEE Sensors Letters* 3.6 (2019), pp. 1–4. DOI: 10.1109/LSENS.2019.2917055.

[3] Wei Lu et al. "Binary Image Steganalysis Based on Histogram of Structuring Elements". In: *IEEE Transactions on Circuits and Systems for Video Technology* 30.9 (2020), pp. 3081–3094. DOI: 10.1109/TCSVT.2019.2936028.

[4] Bala Venkateswarlu Isunuri and Jagadeesh Kakarla. "Fast brain tumour segmentation using optimized U-Net and adaptive thresholding". In: *Automatika* 61.3 (2020), pp. 352–360. DOI: 10.1080/00051144.2020.1760590.

[5] Bill Yang Cai et al. "Deep Learning-Based Video System for Accurate and Real-Time Parking Measurement". In: *IEEE Internet of Things Journal* 6.5 (2019), pp. 7693–7701. DOI: 10.1109/JIOT.2019.2902887.

[6] Zaiwang Gu et al. "CE-Net: Context Encoder Network for 2D Medical Image Segmentation". In: *IEEE Transactions on Medical Imaging* 38.10 (2019), pp. 2281–2292. DOI: 10.1109/TMI.2019.2903562.

[7] Sunila Gollapudi. "OpenCV with Python". In: *Learn Computer Vision Using OpenCV: With Deep Learning CNNs and RNNs*. Berkeley, CA: Apress, 2019, pp. 31–50. ISBN: 978-1-4842-4261-2. DOI: 10.1007/978-1-4842-4261-2_2.

[8] Malti Bansal et al. "Palmistry using Machine Learning and OpenCV". In: *2020 Fourth International Conference on Inventive Systems and Control (ICISC)*. 2020, pp. 536–539. DOI: 10.1109/ICISC47916.2020.9171158.

[9] Zhang Yunjun, Heresh Fattahi, and Falk Amelung. "Small baseline InSAR time series analysis: Unwrapping error correction and noise reduction". In: *Computers Geosciences* 133 (2019), p. 104331. ISSN: 0098-3004. DOI: https://doi.org/10.1016/j.cageo.2019.104331. URL: https://www.sciencedirect.com/science/article/pii/S0098300419304194.

[10] Dongyun Wang et al. "Color Edge Detection Using the Normalization Anisotropic Gaussian Kernel and Multichannel Fusion". In: *IEEE Access* 8 (2020), pp. 228277–228288. DOI: 10.1109/ACCESS.2020.3044341.

[11] Muhammad Fahad Khan, Adeel Ahmed, and Khalid Saleem. "A Novel Cryptographic Substitution Box Design Using Gaussian Distribution". In: *IEEE Access* 7 (2019), pp. 15999–16007. DOI: 10.1109/ACCESS.2019.2893176.

[12] Noorbakhsh Amiri Golilarz, Hui Gao, and Hasan Demirel. "Satellite Image De-Noising With Harris Hawks Meta Heuristic Optimization Algorithm and Improved Adaptive Generalized Gaussian Distribution Threshold Function". In: *IEEE Access* 7 (2019), pp. 57459–57468. DOI: 10.1109/ACCESS.2019.2914101.

[13] Hasnae El Khoukhi et al. "A hardware Implementation of OTSU Thresholding Method for Skin Cancer Image Segmentation". In: *2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*. 2019, pp. 1–5. DOI: 10.1109/WITS.2019.8723815.

[14] Oana Săman, Robert Muscaliuc, and Loredana Stanciu. "Music panel: An application for creating and editing music using OpenCV and JFugue". In: *2020 24th International Conference on System Theory, Control and Computing (ICSTCC)*. 2020, pp. 379–383. DOI: 10.1109/ICSTCC50638.2020.9259705.

[15] H Muthu Mariappan and V Gomathi. "Real-Time Recognition of Indian Sign Language". In: *2019 International Conference on Computational Intelligence in Data Science (ICCIDS)*. 2019, pp. 1–6. DOI: 10.1109/ICCIDS.2019.8862125.