Published in Image Processing On Line on 2016–07–10.
Submitted on 2015–12–14, accepted on 2016–04–15.
ISSN 2105–1232 © 2016 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2016.158

A C++ Implementation of Otsu's Image Segmentation Method

Juan Pablo Balarini, Sergio Nesmachnow

Facultad de Ingeniería, Universidad de la República, Uruguay
{jpbalarini,sergion}@fing.edu.uy

Communicated by Miguel Colom *Demo edited by* Juan Pablo Balarini

Abstract

This article presents an implementation of Otsu's segmentation method and a case study using multiple images. Otsu's method performs nonparametric and unsupervised image thresholding, usually used on image segmentation. The algorithm finds an optimal threshold of an image by minimizing the within-class variance, using only the gray-level histogram of the image. The proposed implementation is conceived emphasizing the role of mathematics as a source for algorithm design and the reproducibility of the research, according to the Image Processing On Line (IPOL) philosophy.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1]. Compilation should be performed using the provided makefile. The provided code enables running the algorithm using automatic threshold selection, or overriding this for manual thresholding.

Keywords: thresholding; segmentation; Otsu's method

1 Introduction

In the last twenty years, digital image processing has become of special interest in science and technology. The importance of analyzing data from images is very relevant in both critical systems and everyday applications, including satellite systems, automated medical analysis, text and face recognition, military and defense systems, and others. *Image processing* describes the process by which the information from an input image or a set of them (usually represented as bi-dimensional systems) are used (i.e. processed) to produce an output image or a set of output data, which is related with the input images [4, 8].

[1] https://doi.org/10.5201/ipol.2016.158

Image segmentation consists on separating an image into regions or contours, that generally correspond to boundaries or objects on images. Usually, segmentation is made by identifying common properties or finding differences between regions. This implies that pixels are grouped into regions or classes that share some common property (such as color, intensity, etc.).

Otsu's segmentation method (named after Nobuyuki Otsu) is a global image thresholding algorithm usually used for thresholding, binarization and segmentation [7, 6, 9]. It works mainly with the image histogram, looking at the pixel values and the regions that the user wants to segment out, rather than looking at the edges of an image. It tries to segment the image making the variance on each of the classes minimal. The algorithm works well for images that contain two classes of pixels, following a bi-modal histogram distribution.

Finding an open-source implementation that was peer-reviewed and verified for this algorithm can be difficult, therefore a working C++ implementation is presented that correctly segments the image using Otsu's method. The main contributions of this article are: i) a functional implementation of Otsu's thresholding algorithm for segmentation for IPOL and ii) a case study presenting the application of the implemented method on a group of test images. The article is organized as follows. Section 2 introduces image segmentation and describes Otsu's algorithm. The proposed implementation for Otsu's algorithm is described in Section 3. Section 4 presents a study of the application of the proposed method to a set of images. Finally, Section 5 summarizes the conclusions and formulates the main lines for future work.

2 Image Segmentation and Otsu's Algorithm

This section briefly describes image segmentation techniques, introduces Otsu's algorithm, and reviews some relevant related works about existing implementations of Otsu's algorithm.

2.1 Image Segmentation

Image segmentation is an umbrella term that includes a set of techniques for image processing based on applying a dividing strategy (i.e. a single image is divided in multiple parts) [3]. After the dividing process, each one of the image components is used for a specific purpose, for example, to identify objects or other relevant information. Several methods are used for image segmentation: thresholding, color-based, texture filters, clustering, among others. An effective approach to performing image segmentation includes using existing algorithms and tools, and integrating specific components for data analysis, image visualization, and the development and implementation of specific algorithms.

One of the most popular approaches for image segmentation is through *thresholding*. Thresholding takes a gray-scale image and replaces each pixel with a black one if its intensity is less than some fixed constant, or a white pixel if the intensity is greater than that constant. The new binary image produced separates dark from bright regions. Mainly because finding pixels that share intensity in a region is not computationally expensive, thresholding is a simple and efficient method for image segmentation.

Formally, a function $g(x, y)$ can be defined over every pixel value of the original image $f(x, y)$ that defines the new thresholded image.

$$g(x,y) = \begin{cases} 1, & f(x,y) \geq T \\ 0, & x < T \end{cases} . \tag{1}$$

Equation (1) defines that for every pixel on the original image, a new value of 0 or 1 will be assigned to the new image, depending if the current pixel value is greater than some defined threshold T.

Several approaches have been proposed to define thresholding methods. According to the categorization by Sezgin and Sankur [9], six strategies are identified:

- *Histogram shape methods*, which use information from the image histogram;
- *Clustering methods*, which groups objects in classes, e.g. background and foreground;
- *Entropy methods*, which make use of entropy information for foreground and background, or cross-entropy between the original and the binary image;
- *Object attribute methods*, which evaluate and use the similarities between the original and the binary images;
- *Spatial methods*, which apply higher-order probability distribution and/or correlation between pixels;
- *Local methods*, based on adapting the threshold value to locally defined characteristics.

The algorithm studied in this article (Otsu's algorithm) is a clustering-based method, which is described next.

2.2 Otsu's Algorithm

Otsu's algorithm is a simple and popular thresholding method for image segmentation, which falls into the *clustering* category.

The algorithm divides the image histogram into two classes, by using a threshold such as the in-class variability is very small. This way, each class will be as compact as possible. The spatial relationship between pixels is not taken into account, so regions that have similar pixel values but are in completely different locations in the image will be merged when computing the histogram, meaning that Otsu's algorithm treats them as the same.

Assuming that pixels are categorized in two classes, the algorithm tries to minimize the weighted within-class variance $\sigma_w^2(t)$, defined by the expression in Equation (2). The variable t is the threshold, which is typically a value between 0 and 255.

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t). \tag{2}$$

The process for computing $\sigma_w^2(t)$ is described next. A probability function P is obtained for every pixel value. First, the histogram distribution for the image is computed, and then a normalization is performed in order to guarantee it follows a probability distribution. After that, the pixel values are divided into two classes C_1 and C_2 by a threshold t, using the class probability functions $q_1(t)$ and $q_2(t)$, defined in equations (3) and (4).

$$q_1(t) = \sum_{i=1}^{t} P(i), \tag{3}$$

$$q_2(t) = \sum_{i=t+1}^{I} P(i). \tag{4}$$

Class C_1 represents those pixels with intensity levels in $[1, t]$, and class C_2 represents those pixels with levels in the interval $[t+1, I]$, where I is the largest pixel value (typically 255).

Then, the means for class C_1, $\mu_1(t)$, and class C_2, $\mu_2(t)$ are obtained:

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)}, \tag{5}$$

$$\mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}. \tag{6}$$

After that, the variances for class C_1, $\sigma_1^2(t)$, and class C_2, $\sigma_2^2(t)$ are computed

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)}, \tag{7}$$

$$\sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}. \tag{8}$$

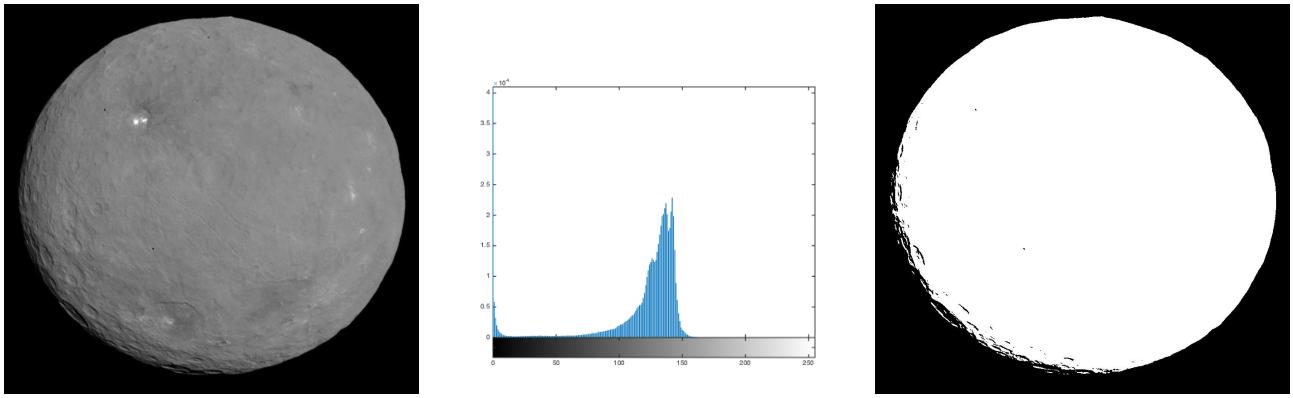
Equation (7) and Equation (8) define the weighted within-class variance for C_1 and C_2, respectively. These are the values that Otsu's algorithm tries to minimize. This variance is a measure of "how compact" each class is, meaning that if the method chooses a bad threshold, the variance for one of the classes will be large.

Using equations (2), (7) and (8), the total variance can be defined by Equation (9), as the sum of the within class and the between-class variance. The value σ^2 is constant, as it does not depend on the threshold (the variance of an image is always a constant value), meaning that the algorithm must focus on minimizing $\sigma_w^2(t)$, or maximizing $\sigma_b^2(t)$.

$$\sigma^2 = \sigma_w^2(t) + \sigma_b^2(t), \text{ where } \sigma_b^2(t) = q_1(t)q_2(t)[\mu_1(t) - \mu_2(t)]^2. \tag{9}$$

To better illustrate how the algorithm works, a brief example is presented next.

Figure 1a shows an image of the Ceres dwarf planet and Figure 1b presents the corresponding histogram for the image. From Figure 1b, the histogram clearly follows a bi-modal distribution, as the pixel gray-levels are distributed over two classes or *modes* of the histogram: one class corresponding to the dwarf planet itself and the other to the background (space). By taking advantage of this pixel distribution, Otsu's algorithm is able to find an appropriate threshold value, and the segmented output image is very sharp, as seen in Figure 1c. It is worth noting that images with a lot of noise will have an almost-uniform histogram distribution, meaning that the algorithm might not be able to find a proper threshold to segment. However, in those cases dealing with noisy images, the application of Otsu's algorithm can be preceded by a noise reduction [1, 5] and/or filtering algorithm, which allows transforming a flat (uniform) histogram into a one where two classes are clearly distinguishable.



(a) Ceres (b) Ceres histogram (c) Segmented Ceres

Figure 1: Ceres

A concrete implementation of Otsu's algorithm defines a procedure to automatically determine an appropriate value for the threshold, as it is described in the next section.

2.3 Related Work: Implementations of Otsu's Algorithm

On his seminal article "A threshold selection method from gray-level histograms", published in 1979, Nobuyuki Otsu only presented the theoretical background and the main ideas behind the segmentation algorithm. No implementation was provided then.

Nowadays, there are some implementations of Otsu's algorithm reported on the Internet. But some of them lack documentation, others do not specify usage rights and others were not peer-reviewed or verified. Only some of them explain the theoretical background behind the algorithm and how it actually works.

The implementation by Birdal[2], is available on the Code Project repository. Birdal developed an implementation of Otsu's algorithm using the C# language and presents a Graphical User Interface (GUI) for easy thresholding: it allows loading the input images, using Otsu's method to find the threshold, and finally saving the thresholded image to an output file.

Another implementation is presented on The Lab Book Pages[3]. This proposal introduces Otsu's thresholding method and provides a Java implementation of the algorithm. The implementation provides a method for obtaining the actual threshold for an input image and the thresholded output image. Also, an example code is presented, in which the provided method is used to open an input image, generate its histogram, segment the image using Otsu's method, and finally save the image.

The Walrus Vision Framework page presents a generic description of Otsu's algorithm, and provides a Java snippet[4]. This snippet only provides a basic idea on how the algorithm works but no explanation about how to use the tool is provided (details such as input image type, etc. are not commented). In addition, no code is provided to open and process input images, or to threshold and save the output image.

Diggins [2] presented an implementation of Otsu's thresholding method for the ARlib augmented reality toolkit. This implementation does not provide a method for opening the input image or saving the segmented image result. Another drawback of this implementation is that when contrasting the obtained threshold for an input image with the `graythresh` Matlab function, which provides a method to compute the Otsu's threshold, the two values differ.

3 The Proposed Implementation for Otsu's Algorithm in C++

This section describes the proposed C++ implementation for Otsu's algorithm and presents a brief analysis of the computational complexity of the method.

3.1 Implementation

The proposed implementation for Otsu's algorithm is based on the formulation presented in the previous section, applying an iterative method to compute and maximize the between-class variance, defined by Equation (9).

A straightforward approach for maximization is to explore all possible values, by iterating from the lowest value of t (typically 0) to the highest t (typically 255), computing $q_1(t)q_2(t)[\mu_1(t) - \mu_2(t)]^2$.

[2]Famous Otsu thresholding in C#, T. Birdal, 2009. http://www.codeproject.com/Articles/38319/FamousOtsu-Thresholding-in-C [Online; accessed 2-September-2015]

[3]Otsu thresholding, A. Greensted, 2010. http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html [Online; accessed 2-September-2015]

[4]How Otsu thresholder algorithm works, Walrus vision, 2014. http://www.walrusvision.com/wordpress/otsu-thresholder-algorithm-works/ [Online; accessed 2-September-2015]

The highest value computed is returned. The proposed implementation, described in the pseudocode in Algorithm 1, applies this idea.

The proposed algorithm was implemented in C++. The Image Processing Framework by Miguel Colom[5] is used to read the input images. The framework helps opening PNG and JPEG images and it allows working with the image as an unidimensional vector (one vector for each color channel). The length of each vector is $M \times N$, where M is the number of rows of the image and N is the number of columns of the image respectively; this is of special interest when accessing every pixel on the image and when computing the image histogram. This framework is also useful to validate the algorithm input parameters and options, using the capabilities of returning proper errors when input parameters or options are missing or misspelled.

Because Otsu's algorithm works with grayscale images, the algorithm assumes that the input image has one color channel; in case this requirement is not met, an error is returned. The first loop (lines 9-11 in Algorithm 1) iterates over every pixel, accumulating the counters for every gray-level pixel, in order to compute the image histogram. After that, the obtained histogram is used to compute Otsu's threshold value, by updating $q_i(t)$ and $\mu_i(t)$ (lines 15-24), which are used to obtain $\sigma_b^2(t)$ (line 25). Once the threshold value is found for the input image, a new output image is created, which applies a thresholding segmentation over the input image using the computed threshold value (lines 29-33). The new image, which correspond to the segmented version of the input image, is then returned (line 34). The algorithm allows overriding the threshold value (manual threshold), in order to see what the output would be if another threshold value is selected. The main advantage of the proposed work is that a functional Open-Source implementation is presented that contributes to the main lines of work of the IPOL project. Since it was implemented in C++ and was developed with efficiency in mind, it allows working with large images and obtaining faster thresholding times that when contrasted with more heavyweight implementations (i.e. Java, C#, etc.).

3.2 Computational Complexity

Regarding the computational complexity of the proposed implementation, the number of operations is given by the four loops in the code:

1. The first loop (lines 5–6) in Algorithm 1 corresponds to the histogram initialization, requiring $O(max\_intensity + 1)$ operations;
2. the second loop (lines 9–11) is performed to compute the histogram for the input image, it requires $O(N)$ operations, being N the product of the width and the height of the input image;
3. the third loop (lines 15–28) is used to compute the values of the weighted within-class variance $\sigma_b^2(t)$, it has a complexity of $O(max\_intensity + 1)$ operations; and finally,
4. the fourth loop, to define the output image (lines 29–33), requires $O(N)$ operations.

Thus, the whole C++ implementation for Otsu's algorithm involves $O(max\_intensity + N)$ operations, i.e. it is linear on the number of pixels in the input image.

4 Case Study

This section presents a case study demonstrating the application of the proposed implementation on a group of standard images used in image processing benchmarks. The problem instances consist on

[5]Image processing framework, M. Colom, 2015. https://github.com/mcolom/image_processing_framework [Online; accessed 30-July-2015]

Algorithm 1: Thresholding segmentation using Otsu's method or manual input

input : input_image (grayscale), overridden_threshold
output: output_image

```
1  read(input_image)
2  N = input_image.width × input_imnumberage.height
                                                    initialize variables
3  threshold, var_max, sum, sumB, q_1, q_2, μ_1, μ_2 = 0
4  max_intensity = 255
5  for i=0; i <= max_intensity; i++ do
6      histogram[value] = 0
                                                    accept only grayscale images
7  if num_channels(input_image) > 1 then
8      return error
                                                    compute the image histogram
9  for i=0; i<N; i++ do
10     value = input_image[i]
11     histogram[value] += 1
12 if manual threshold was entered then
13     threshold = overridden_threshold
14 else
                                                    auxiliary value for computing μ_2
15     for i=0; i<=max_intensity; i++ do
16         sum += i × histogram[i]
                                                    update q_i(t)
17     for t=0; t<=max_intensity; t++ do
18         q_1 += histogram[t]
19         if q_1 == 0 then
20             continue
21         q_2 = N − q_1
                                                    update μ_i(t)
22         sumB += t × histogram[t]
23         μ_1 = sumB/q_1
24         μ_2 = (sum - sumB)/q_2
                                                    update the between-class variance
25         σ_b^2(t) = q_1(t)q_2(t)[μ_1(t) − μ_2(t)]^2
                                                    update the threshold
26         if σ_b^2(t) > var_max then
27             threshold = t
28             var_max = σ_b^2(t)
                                                    build the segmented image
29 for i=0; i < N; i++ do
30     if input_image[i] > threshold then
31         output_image[i] = 1
32     else
33         output_image[i] = 0
34 return output_image
```

a set of 4 images with PNG format, 1 color channel (8 bits) and different sizes ranging from 256×256 pixels to 1024×765.

Figures 2a, 3a, 4a, and 5a show four input examples that have different histogram distributions.

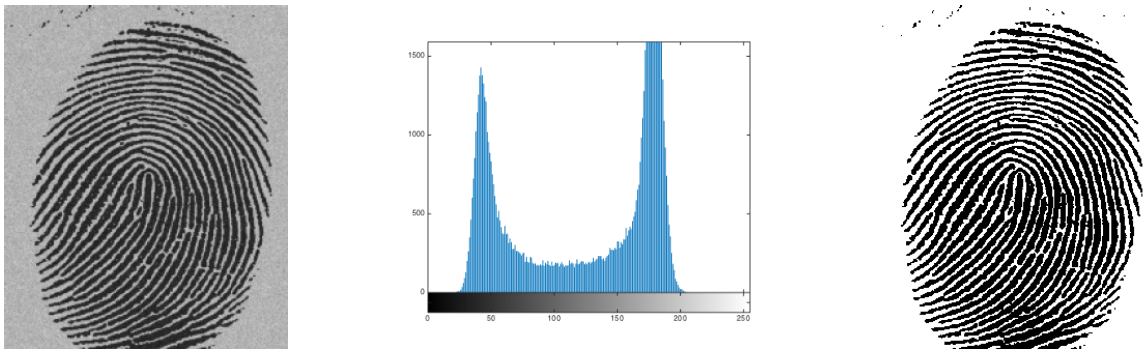
Regarding the first image (*Fingerprint*, in Figure 2a), the histogram in Figure 2b indicates that it does follow a bi-modal distribution, which means that the image pixels are distributed over two classes of the histogram. One class corresponds to the fingerprint itself and the other to the background.

The second image, in Figure 3a, shows a picture that has a clear distinction between the prominent elements on the image (the houses) and the background (the sky). This situation is clearly summarized in the image histogram in Figure 3b. Nonetheless, the segmented result (shown in Figure 3c) demonstrates that Otsu's algorithm is not able to separate properly the different parts of each house, mainly because they cover a large range of grayscale values (between 0 and T), which are widely distributed over the image histogram. All these values fall inside the "0" category when performing the image thresholding.

The third image, in Figure 4b, shows a distinction between the cameraman (with the camera equipment) and the background, and this information is clearly shown in the image histogram (in Figure 4b). In this case, pixels that correspond to the cameraman are distributed over the left part of the histogram (i.e. lower values of intensity) and there is a clear separation between those pixels and the rest of pixels of the image. Otsu's algorithm is able to find an appropriate threshold value, and it does a good job segmenting the cameraman with the camera from the image background.

The fourth image, Figure 5a, shows a clear distinction between the balls and the background, as shown in the image histogram (Figure 5b). In this example, pixels that correspond to the balls are distributed over the right part of the histogram (i.e. higher values of intensity) while the background is distributed over the lower left part of the histogram (i.e. lower values of intensity) meaning that Otsu's algorithm is able to perfectly segment the balls from the background as seen on Figure 5c.

From the presented examples it can be seen that the quality of the segmented output image depends highly on the image following a bi-modal histogram distribution or not. This is because Otsu's algorithm tries to separate image pixels into two distinct classes; if the image has more than two classes, the algorithm is not able to find a threshold that separates them.



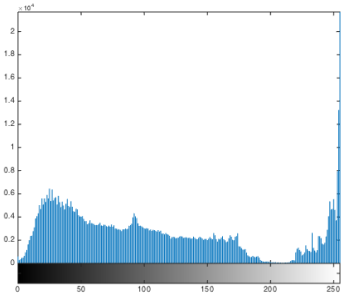
(a) Input image (b) Image histogram (c) Segmented image

Figure 2: Fingerprint

5 Conclusions

The main motivation behind Otsu's image thresholding algorithm is trying to find a threshold that separates the image histogram into two classes. For this reason, it is considered an image segmentation method. This article presents a functional implementation of Otsu's threshold selection

(a) Input image

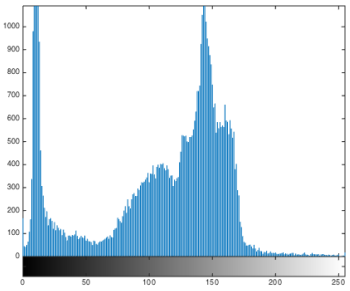


(b) Image histogram

(c) Segmented image

Figure 3: Houses

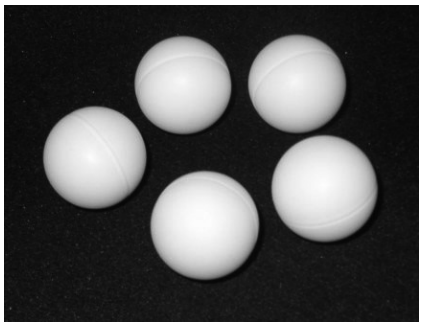
(a) Input image



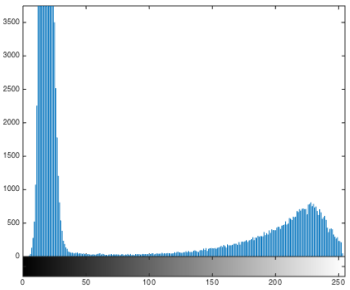
(b) Image histogram

(c) Segmented image

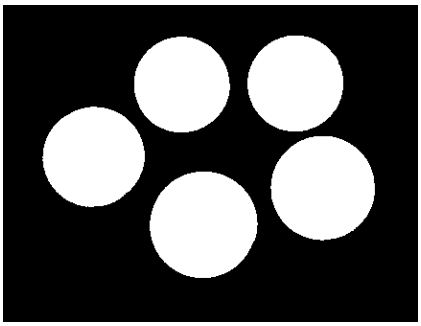
Figure 4: Cameraman



(a) Input image



(b) Image histogram



(c) Segmented image

Figure 5: Balls

algorithm, that it is used to segment an input image. A well commented implementation, along with a complete analysis of the algorithm is provided. A case study for the application of the proposed implementation is also presented. For the case study, multiple images are used as the algorithm input to better understand how the image histogram distributions affects the quality of the segmented output image. The obtained results indicate that the quality of the segmented image depends highly on the image following a bi-dimensional histogram distribution or not. Another important contribution of the article is an Open-Source implementation, following the criteria of reproducible research. The

code is publicly available also at GitHub[6]. The official code is the code at IPOL, which was submitted to a peer-review process; the code at GitHub is not official, but might contain useful updates.

Image Credits

 Ceres Dwarf Planet (NASA/JPL-Caltech/UCLA/MPS/DLR/IDA), Public Domain[7]

 Glenn J. Mason (Flickr), CC-BY-SA[8]

 Houses (Wikipedia), CC-BY-SA[9]

 Cameraman standard test image[10]

 Standard test image[11]

References

[1] A. Buades, B. Coll, and J-M. Morel, *On image denoising methods*, tech. report, CMLA (Centre de Mathematiques et de Leurs Applications, 2004.

[2] D. Diggins, *ARLib: A C++ Augmented Reality Software Development Kit*, master's thesis, MSc Computer Animation, N.C.C.A Bournemouth University, 2005. https://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/Msc05/ddiggins_msc_thesis.pdf.

[3] R.C. Gonzalez and R.E. Woods, *Digital image processing (3rd edition)*, Prentice-Hall, Inc., 2006. ISBN 013168728X.

[4] A.K. Jain, *Fundamentals of digital image processing*, Prentice-Hall, Inc., 1989. ISBN 0-13-336165-9.

[5] C. Liu, W. T. Freeman, R. Szeliski, and S.B. Kang, *Noise estimation from a single image*, in IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, 2006, pp. 901–908. http://dx.doi.org/10.1109/CVPR.2006.207.

[6] S. Malakar, D. Mohanta, R. Sarkar, N. Das, M. Nasipuri, and Basu D.K., *Binarization of the Noisy Document Images: A New Approach*, Springer Berlin Heidelberg, 2011, pp. 511–520. http://dx.doi.org/10.1007/978-3-642-22786-8_64.

[7] N. Otsu, *A threshold selection method from gray-level histograms*, IEEE Transactions on Systems, Man and Cybernetics, 9 (1979), pp. 62–66. http://dx.doi.org/10.1109/TSMC.1979.4310076.

[8] J.C. Russ, *Image processing handbook (4rd edition)*, CRC Press, Inc., 2002. ISBN 084931142X.

[6] Otsu's Segmentation Method, https://github.com/jpbalarini/otsus (05 December 2015)
[7] http://photojournal.jpl.nasa.gov/jpeg/PIA19562.jpg
[8] http://www.flickr.com/photos/glennji/3558118429/
[9] https://commons.wikimedia.org/wiki/File:Image_processing_pre_otsus_algorithm.jpg
[10] http://graphics.cs.williams.edu/data/images/cameraman.png
[11] http://www.fing.edu.uy/~sergion/balls.png

[9] M. Sezgin and B. Sankur, *Survey over image thresholding techniques and quantitative performance evaluation*, Journal of Electronic Imaging, 13 (2004), pp. 146–168. http://dx.doi.org/10.1117/1.1631315.