Binarization and Vanishing Points Detection

Computer Vision Project

Davide Ligari, Andrea Alberti

Department of Computer Engineering - Data Science

University of Pavia, Italy

Email: davide.ligari01@universitadipavia.it - andrea.alberti01@universitadipavia.it

GitHub: https://github.com/AndreaAlberti07/Binarization-and-vanishing-points

March 11, 2024

Abstract

In this project, two image processing programs have been developed.
The first, the **Binarization Program**, employs a developed Histogram-based thresholding technique to convert images into binary form. It offers both automatic and manual tuning methods, allowing users to find the ideal threshold for image segmentation. The program calculates the optimal threshold by minimizing a loss function and the results are accurate as shown by reported examples.
The second program, the **Vanishing Point and Lines Detection Program**, focuses on detecting vanishing points and lines within images, a critical task in computer vision, augmented reality, and architectural design. It uses a series of techniques including the Canny edge detector, probabilistic Hough transform, and RANSAC algorithm to achieve this. The program is versatile and robust, as evidenced by real-world examples.
Both programs offer command-line interfaces for easy integration into various workflows, making them valuable tools for complex image processing challenges in the field of computer vision.

CONTENTS

1. Requirements

To run these programs successfully, you must have the following packages installed:

- **Python 3.9** or higher
- **OpenCV:** pip install opencv-python
- **Numpy:** pip install numpy
- **Matplotlib:** pip install matplotlib
- **Scipy:** pip install scipy
- **Tkinter:** see Documentation

2. Binarization Program

This program is designed to perform image binarization using a specifically designed Histogram based thresholding technique. It offers both automatic and manual tuning methods to determine the optimal threshold for converting a given image into a binary image.

2.1. Introduction

Image binarization is a common image processing technique used to separate objects or regions of interest from the background. The goal is to find an optimal threshold value that divides the pixel values into two classes: foreground and background.

2.2. Reasoning Behind the Program

The program aims to find the best threshold value minimizing a loss function. Specifically, a loss is computed for each possible value of the threshold and the minimum is selected. Additionally, this program provides an option for manual tuning, allowing users to adjust the threshold values to suit their specific needs.

2.3. Functions Explanation

2.3.1. Loss Function

The `get_loss` function calculates a loss function based on the provided parameters, such as the histogram values, bin values, threshold, tuning method, and tuning values. The fundamental principle underlying this approach is that when a threshold is established, the pixels falling below it are assigned a value of 0, and those surpassing it are designated as 255. This behavior inherently introduces errors, particularly pronounced for pixels in close proximity to the threshold. To illustrate this concept further, consider a threshold set at 128. A pixel with a value of 127 will be forcefully assigned a value of 0, incurring a more substantial error compared to a pixel with a lower initial value, such as 10.

Moreover the magnitude of error amplifies proportionally with the number of pixels close to the threshold value. Consequently, the design of an effective loss function necessitates consideration of both the quantity of pixels affected and their distance from the threshold. This requirement is encapsulated in the following formula:

$$L = \sum_{i=0}^{T} num_i \cdot dist\_u\_thresh_i + \sum_{i=T+1}^{255} num_i \cdot dist\_o\_thresh_i$$

where:

- **num**: Histogram values.
- **T**: Threshold value for binarization.
- **dist_u_thresh**: Distance from the threshold for pixels under the threshold.
- **dist_o_thresh**: Distance from the threshold for pixels over the threshold.

Note that the distance is an 'inverted distance' due to the reasons explained above. Thus the closest pixel to the threshold has the highest distance value, thereby is the most impacting on the loss function.

If the tuning method is 'Auto', the distance is calculated as:

IF $mean > 128$ **THEN** $dist\_o\_thresh += tuning\_val$

IF $mean < 128$ **THEN** $dist\_u\_thresh += tuning\_val$

where:

- **tuning_val** = abs(mean - 128)

This tuning is necessary to deal with cases in which the histogram is highly unbalanced (e.g. most values are concentrated in the left part of the histogram). In this case, due to how the distance is computed, the program would tend to select a threshold located on the extreme right, because all the pixels on the left would have a very high distance from it which translates into a very low weight. This is solved by adding a tuning value to the distance, which is proportional to the distance of the mean from the middle of the histogram (128).

If the tuning method is 'Manual', the tuning value is inserted by the user and divided in two:

- **under_tuning**: Tuning value for pixels under the threshold.
- **over_tuning**: Tuning value for pixels over the threshold.

The distance is calculated as:

$$dist\_o\_thresh += under\_tuning\_value$$
$$dist\_u\_thresh += over\_tuning\_value$$

2.3.2. Finding the Best Threshold

The `get_best_thresh` function finds the best threshold value for binarization using a brute force approach, trying all possible threshold values and retaining the one that minimizes the loss function. It can operate in either 'Auto' mode, which automatically determines the optimal threshold, or 'Manual' mode, where users can specify under-tuning and over-tuning values. This function returns the best threshold value and the corresponding minimum loss value. It also offers the option to plot the loss function for analysis.

2.3.3. Applying Threshold

The `apply_thresh` function applies the calculated threshold to the given image. It returns a binary image with values of 0 or 1, where 0 represents the background and 1 represents the foreground.

2.4. Command Line Usage

The program accepts the following command line arguments:

- `-i` or `-img_path`: Specify the path to the input image + name.
- `-t` or `-tuning_method`: Choose the tuning method, either 'Auto' or 'Manual.'
- `-u` or `-under_tuning`: Set the tuning value for pixels under the threshold (only for 'Manual' tuning). It generally results in a left-shift of the threshold.
- `-o` or `-over_tuning`: Set the tuning value for pixels over the threshold (only for 'Manual' tuning). It generally results in a right-shift of the threshold.
- `-s` or `-storing_path`: Specify the path to store the output image + name.
- `-show_all`: Set to 'False' only for serial script execution of multiple images.

For 'Auto' tuning method:

```
python binarization.py -i [
    path_to_input_image] -t Auto -s [
    path_to_output_image] -show_all
    True
```

For 'Manual' tuning, you can use the following command:

```
python binarization.py -i [
    path_to_input_image] -t Manual -u [
    under_tuning_value] -o [
    over_tuning_value] -s [
    path_to_output_image] -show_all
    True
```

2.5. GUI Usage

The program is also provided with a basic Graphic User Interface (GUI) that allows users to select the input image, tuning method, tuning values, and storing path. The GUI also displays the loss function and the resulting binary image. To run the GUI, simply run the following command in the program directory:

Command line input

```
python binarization_GUI.py
```

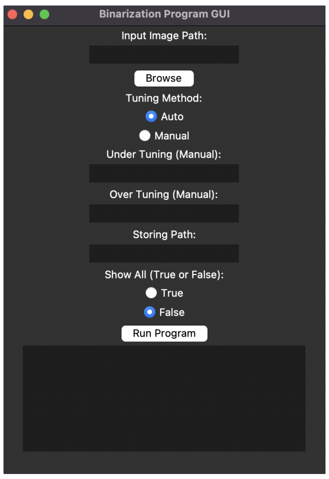
Binarization Program GUI
Input Image Path:
Browse
Tuning Method:
[✓ Auto]
Manual
Under Tuning (Manual):
Over Tuning (Manual):
Storing Path:
Show All (True or False):
True
[✓ False]
Run Program

Fig. 1: Graphical User Interface

2.6. Examples

These are some examples of the program's output:

Command line input

```
python binarization.py -i ../Images/
    binarization/27img.jpg -t Auto -s
    27img_bin.jpg -show_all True
```

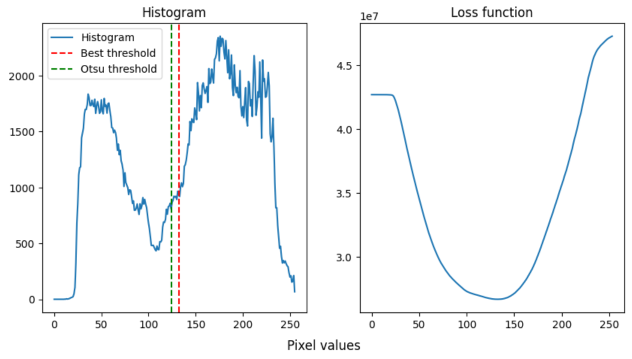

Fig. 2: Lake Loss Function

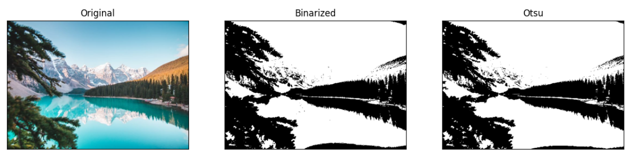

Fig. 3: Lake Binary Image

Command line input

```
python binarization.py -i ../Images/
    binarization/5img.jpeg -t Auto -s 5
    img_bin.jpg -show_all True
```

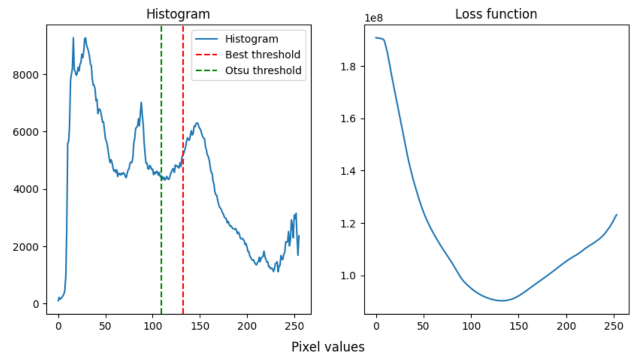

Fig. 4: Cars Loss Function

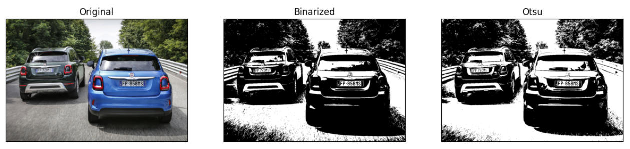

Fig. 5: Cars Binary Image

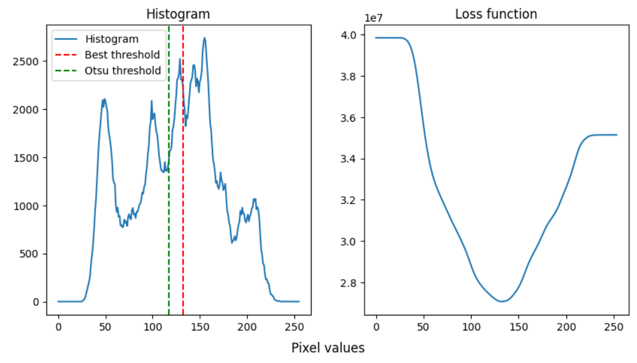

Fig. 6: Lena Loss Function

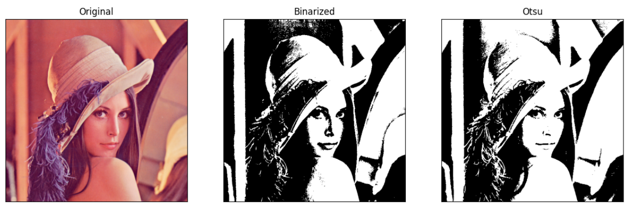

Fig. 7: Lena Binary Image

2.7. SSIM Computation

Comparing the binarization program with Otsu's method, the SSIM index was computed for each image. The results are shown below:

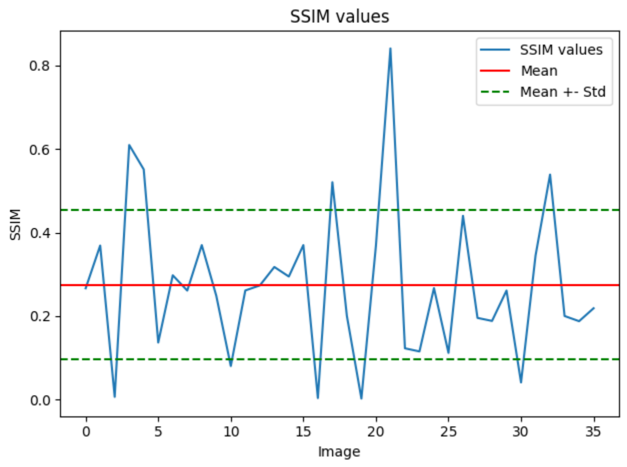

Fig. 8: SSIM Index

As expected, the SSIM index is low, this doesn't imply one method to overcome the other but simply that the two methods produce different results, indeed they are different algorithms. Looking at the proposed example we can notice that, despite its simplicity, the implemented algorithm performs visually well across different images.

3. Vanishing Point and Lines Detection Program

This program is designed to detect vanishing points and vanishing lines within images. It is implemented in Python and is organized into multiple scripts, each dedicated to a specific aspect of the task.

3.1. Reasoning Behind the Program

The development of the Vanishing Point and Lines Detection Program is driven by the compelling need to extract valuable geometric insights from images, particularly the identification of converging lines towards a vanishing point. This information holds substantial significance across various domains, including computer vision, augmented reality, and architectural design.
The program is designed to provide a robust and efficient solution for the detection of vanishing points and lines without the need for manual parameter specification. The program's functionality can be divided into four main stages:

- **Preprocessing:** This initial step involves transforming the input image into a grayscale format and reducing noise. It prepares the image for subsequent analysis by enhancing its clarity and reducing unwanted artifacts.
- **Edge Detection:** Following preprocessing, the program detects edges within the image.
- **Lines Detection:** In this phase, the program identifies and extracts straight line segments from the image.
- **Vanishing Point Detection:** The final and most critical stage involves identifying the vanishing point and vanishing lines within the image.

3.1.1. Edge Detection

Edge detection is a fundamental component of the program, aimed at identifying significant transitions in intensity within an image. These transitions often correspond to object boundaries or structural elements, providing essential cues for further analysis. To achieve edge detection, the program employs the Canny edge detector, a multi-stage algorithm that consists of several steps:

- **Noise Reduction**
- **Gradient Computation**
- **Non-Maximum Suppression**
- **Hysteresis Thresholding**

The values of the "low threshold" and "high threshold" parameters for the Canny edge detector are not fixed but rather computed dynamically based on the image's intensity distribution. Specifically, the program calculates these thresholds using the median of the image's intensity distribution, and the thresholds are set to the computed median ± 0.22. This adaptive approach ensures that edge detection remains effective across a wide range of images, adapting to variations in lighting and content.

3.1.2. Lines Detection

The lines detection component of the program is responsible for identifying straight line segments within the image. These lines can represent a variety of linear structures, including architectural elements, road markings, and other prominent linear features.
To accomplish this task, the program employs the probabilistic Hough transform, a widely-used technique for line detection in images. The Hough transform works by mapping image points to a parameter space where lines are represented as points. However, the Hough transform involves several parameters that require careful tuning to achieve accurate results.
Due to the sensitivity of the Hough transform to parameter settings and the potential for returning different lines based on these parameters, our approach involves running the Hough transform multiple times, each with a different set of parameters. By varying the parameters and accumulating the resulting lines, we increase the program's ability to capture a wide range of potential vanishing lines present in the image.
To reduce the number of lines, a decision was made to consider only the ten longest lines for each combination of parameters. Additionally, since parallel and vertical lines are unlikely to be vanishing lines, they are removed.

3.1.3. Vanishing Point Detection

Before discussing the implementation, it is important to understand the concepts of the vanishing point and vanishing lines.
The vanishing point refers to the point where most of the lines in the image converge. Vanishing lines are the lines that intersect at the vanishing point. Therefore, the challenge of finding the vanishing point can be reduced to identifying the lines that intersect at a common point.
To achieve this, the RANSAC algorithm is employed. Since there are often a large number of detected lines, it is not feasible to consider all of them for efficiency reasons. Therefore, the algorithm iterates 500 times, randomly selecting two lines in each iteration to calculate their point of intersection.
For each intersection point, the algorithm counts the number of lines passing within a distance of 5 pixels from it. The point with the highest number of lines passing within this 5-pixel distance is considered the vanishing point. The lines that pass within 5 pixels of this point are identified as the vanishing lines.
To make the results more comprensible to the user, the program draws the vanishing point and the 15-longest vanishing lines on the image.

3.2. Command Line Usage

The program accepts the following command line arguments:

- `-h` or `-help`: show the help message and exit.
- `-p` or `-path`: Specify the path to the input image or a folder containing images for batch processing.
- `-t` or `-tuning_method`: Choose the tuning method,

either 'Auto' or 'Manual.'

- `-l` or `-lowThreshold`: Set the low threshold for the Canny edge detector (only for 'Manual' tuning).
- `-r` or `-highThreshold`: Set the high threshold for the Canny edge detector (only for 'Manual' tuning).
- `-a` or `-theta`: Set the theta value for the Hough transform (only for 'Manual' tuning).
- `-d` or `-threshold`: Set the threshold value for the Hough transform (only for 'Manual' tuning).
- `-m` or `-minLineLength`: Set the minimum line length for the Hough transform (only for 'Manual' tuning).
- `-g` or `-maxLineGap`: Set the maximum line gap for the Hough transform (only for 'Manual' tuning).
- `-s` or `-storingPath`: Specify the path to store the output image + name.

For 'Auto' tuning method:

```
python vanishingPointDetection.py -p
    ../Images/vanishing_points  -t Auto
    -s ../Images/vanishing_points/
    results
```

For 'Manual' tuning, you can use the following command:

```
python3 vanishingPointDetection.py -p
    ../Images/vanishing_points  -t
    Manual -l 30 -r 55 -a 1 -d 100 -m
    20 -g 16
```

3.3. Examples

These are some examples of the program's output:

Command line input

```
python3 vanishingPointDetection.py -p
    ../Images/vanishing_points/
    van_points.jpg -t Auto -s ../Images
    /examples/van_points.jpg
```

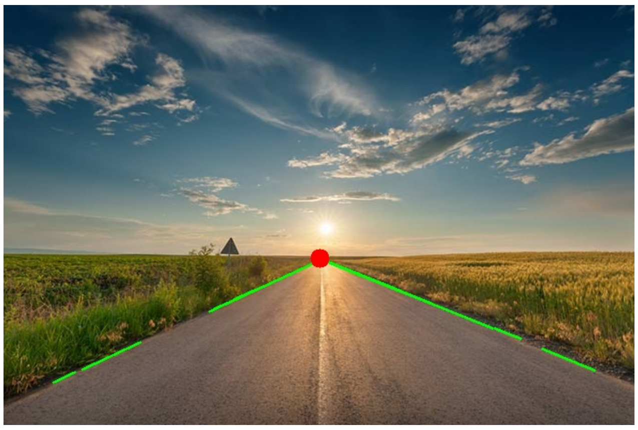

Fig. 9: Result of the vanishing point detection on the image van_points.jpg

Command line input

```
python3 vanishingPointDetection.py -p
    ../Images/vanishing_points/
    van_points8.jpg -t Auto -s ../
    Images/examples/van_points8.jpg
```

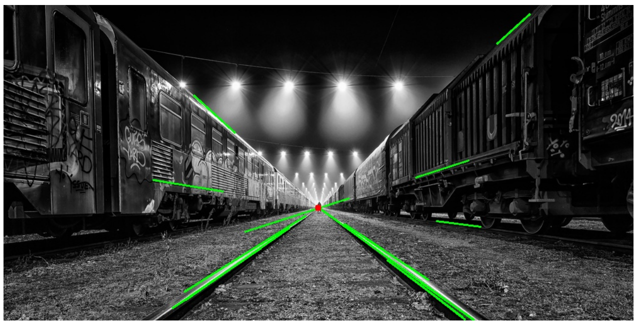

Fig. 10: Result of the vanishing point detection on the image van_points8.jpg

Command line input

```
python3 vanishingPointDetection.py -p
    ../Images/vanishing_points/
    van_points13.jpg -t Auto -s ../
    Images/examples/van_points13.jpg
```

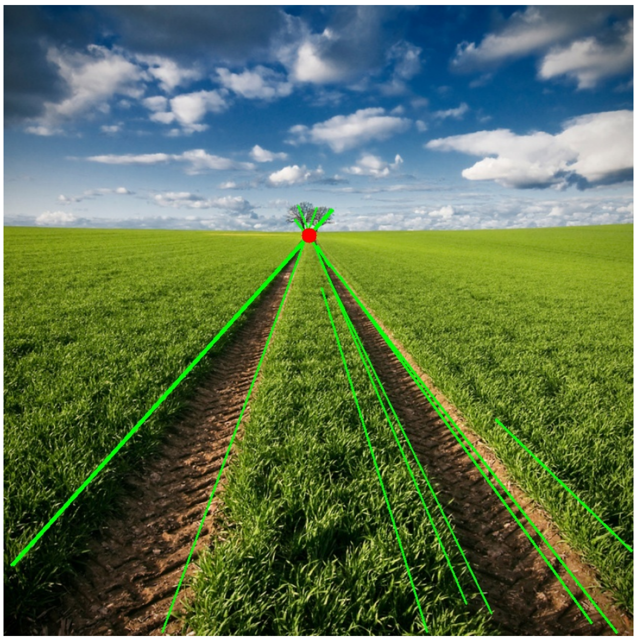

Fig. 11: Result of the vanishing point detection on the image van_points13.jpg

Command line input

```
python3 vanishingPointDetection.py -p
    ../Images/vanishing_points/
    van_points6.jpg -t Auto -s ../
    Images/examples/van_points6.jpg
```

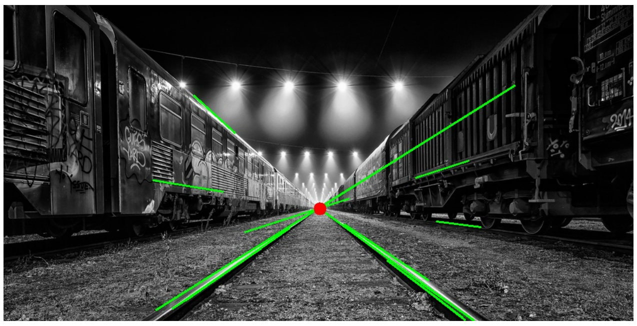

Fig. 12: Result of the vanishing point detection on the image van_points6.jpg

2