FACULTY OF COMPUTER ENGINEERING

PROJECT

FOR

IMAGE PROCESSING

Leaf detection
-using OpenCV-

Coordinating teacher:
Rednic Ana

Student name:
Pal Tudor Ovidiu
Group: 30234, semestrul II

Academic year: 2022-2023

Cuprins

1. Introduction

For the final project for the “Image Processing” class, I chose to implement, build, and test an algorithm for detecting leaves. The purpose of this documentation is to provide an overview and explanation of the code implementation for leaf detection in digital images.

I wanted to build this project because I found in my research several implementations using python and python libraries, but I wanted to try my own implementation in C++ using the OpenCV library.

Leaf detection plays a crucial role in various fields such as agriculture, botany, and environmental research. By accurately identifying and analyzing leaves, researchers and professionals can gain insights into plant health, species classification, and environmental conditions.

2. Bibliographic research

As mentioned above, I found multiple algorithms that try to detect leaves using python and OpnCV (fig. 1). The main idea of this implementation is to look at green/yellow/brown color ranges and extract only specific HSV values from the image.

The results are mixed because of the varied lighting and noise in the images but this it is intersting nonetheless.

Fig. 1 https://www.kaggle.com/code/trushk/leaf-detection-using-opencv/notebook

Another approach would be trying to classify the images using machine learning and neural networks. The following figure (fig. 2) shows the steps the algorithm would follow in order to detect leaf diseases. The model is developed based on the IP and ML approaches for detection of leaf disease in presented in this section. The proposed model (DWT+PCA+GLCM+CNN) using computer vision and machine learning approaches for leaf disease detection.

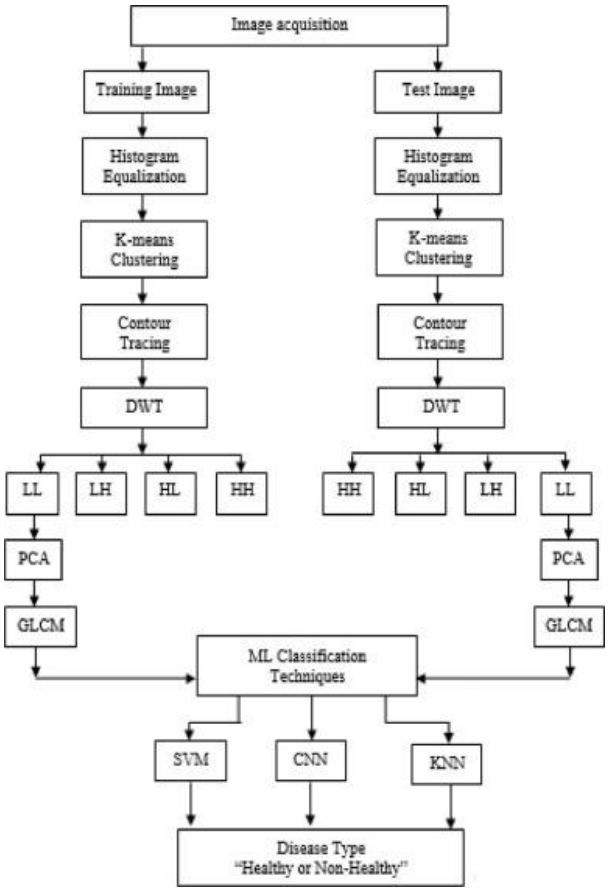


Fig. 2 https://www.sciencedirect.com/science/article/pii/S2666285X22000218#fig0006

3. Proposed solution and implementation

The project is implemented in an OpenCV application that has multiple functions built previously. In order to run the leaf detection algorithm, a menu pops up where the user is prompted to introduce a number. If the user types the number 45, then, the user can choose an image for the algorithm to run on.

After opening the image, the algorithm follows the next steps of the Canny algorithm for edge detection:

1. Filters the image with a **Gaussian filter**.

2. After applying the Gaussian filter, the code proceeds to apply the **Sobel filter** to compute the gradient magnitude and direction of the filtered image. These are computed using the formulas modul = sqrt(gx^2 + gy^2) / (4 * sqrt(2)) and directie = atan2(gy, gx) + CV_PI.
3. **Non-maximum suppression** is performed to thin the edges obtained from the gradient magnitude and direction. This step helps to keep only the local maximum values along the edges and suppress the non-maximum values.
4. **Adaptive binarization** is applied to the thinned edge image. The code calculates a threshold value based on a histogram analysis of the image. The histogram is computed, and the threshold value is determined such that a certain percentage of pixels fall below the threshold.
5. The edges are further enhanced by extending them using **hysteresis thresholding**. The code iterates over each pixel in the binarized image and assigns a value of 255, 128, or 0 based on different threshold conditions. This step helps to connect weak edges to strong edges and fill gaps in the contour.

We use the “Canny” algorithm (fig. 3) in order to only keep the edges of the image. The next steps are:

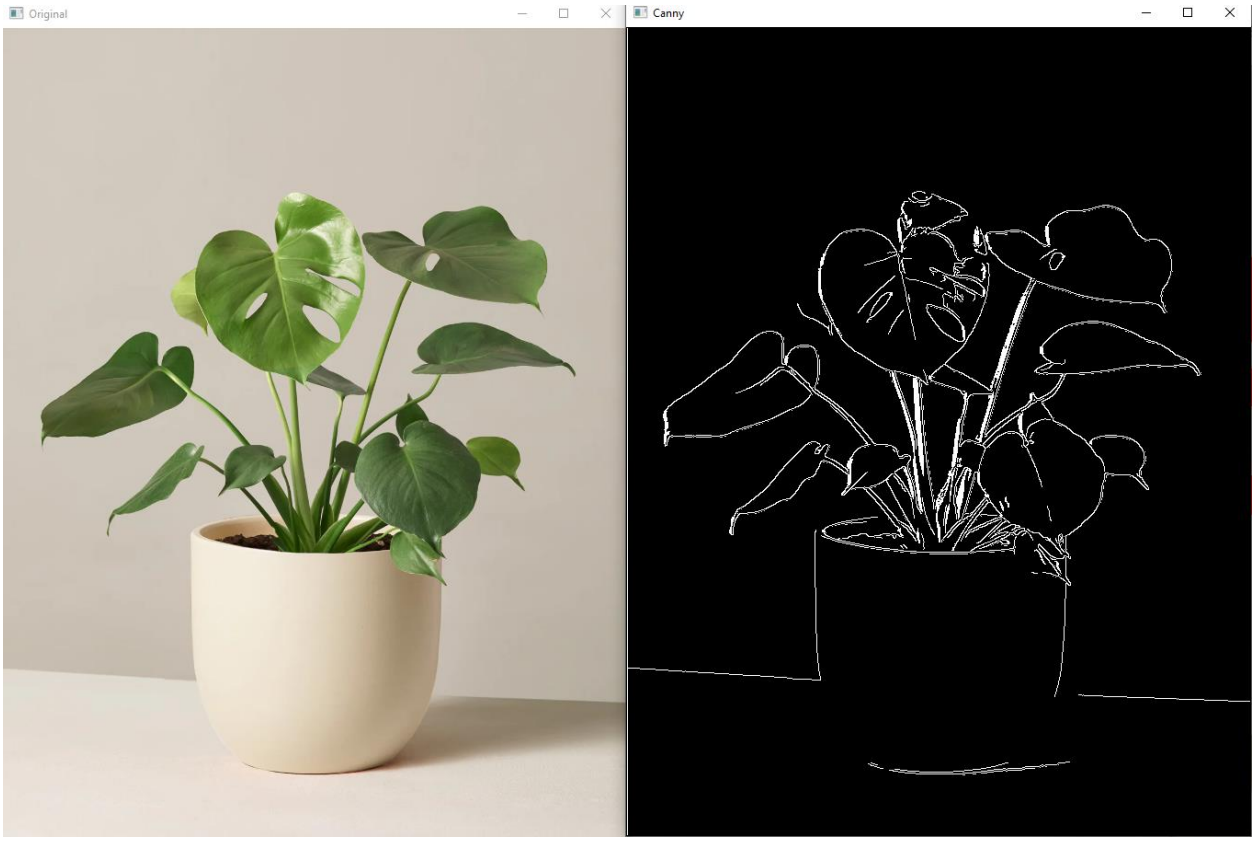


Fig. 3 Canny visualisation

1. **Open** the image to reduce the number of holes in the contour. Opening the image is an erosion and a dilatation performed on the source image.
2. **Contour detection** is performed on the canny image.
3. **Leaf properties** such as minimum area, maximum area, minimum circularity, and maximum circularity are calculated using the calculateLeafProperties function.
4. The code filters the detected contours based on **area and shape criteria**. Contours with areas within a specified range and circularity within a specified range are considered as leaf contours.
5. A mask image is created to keep only the regions **inside the leaf contours**. The mask is drawn using the drawContours function.
6. The **final result** is obtained by applying the mask to the original color image using the bitwise AND operation.

After these steps are finished, the program displays both the grayscale image with contour around the leaves and the color image with the leaves colored red (fig. 4).

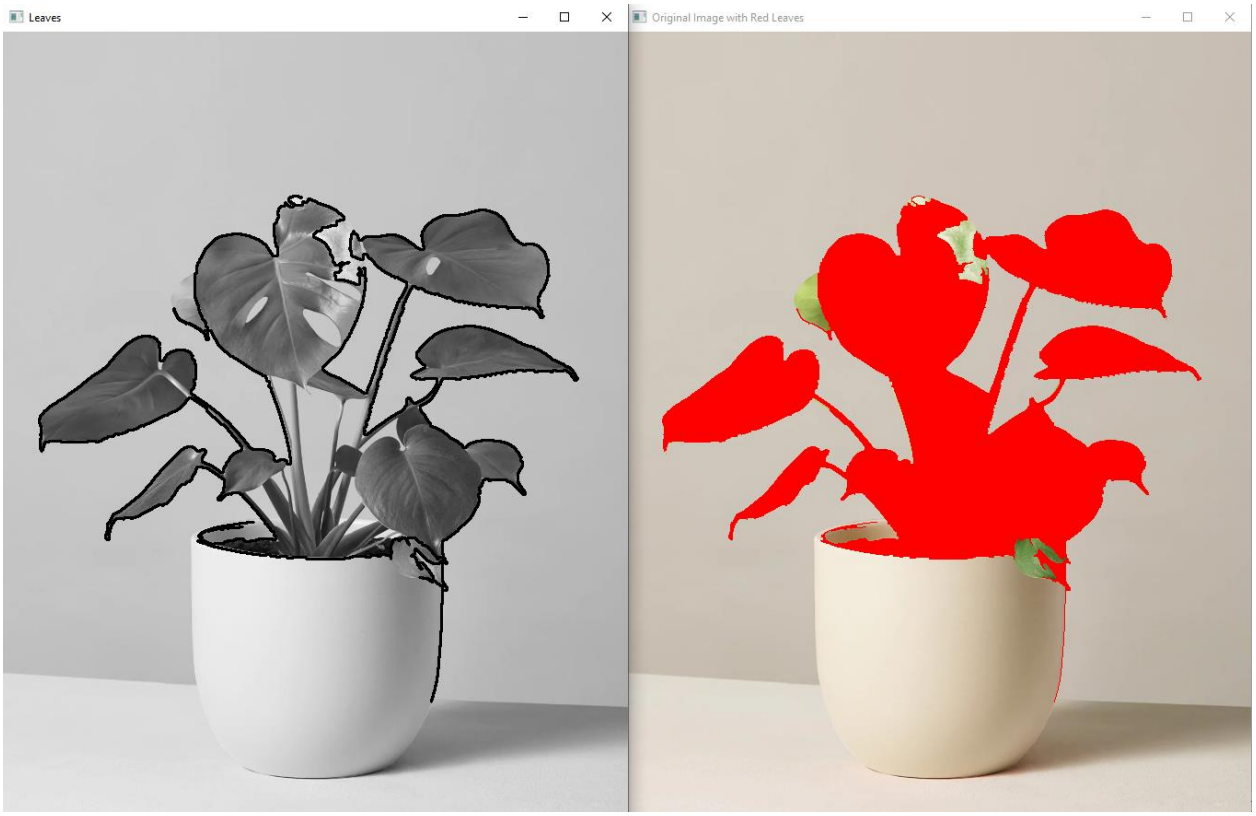
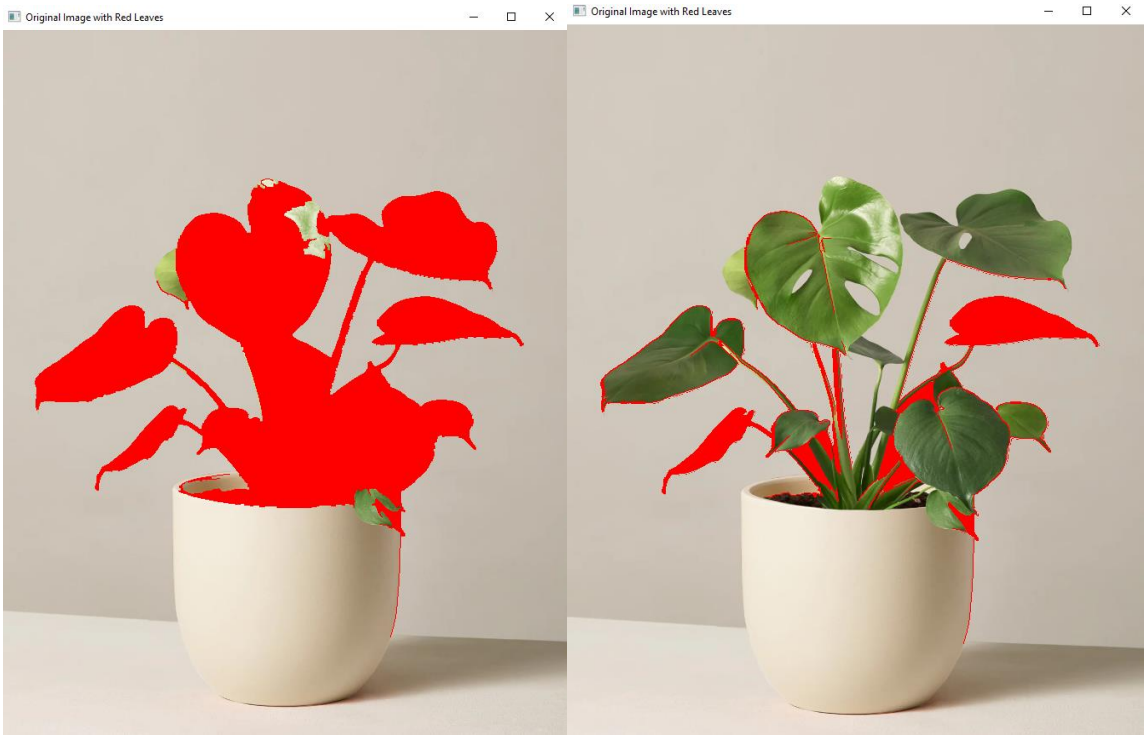


Fig. 4 Final result

4. Testing and validation

Before putting everything together, I tested every step and function individually. For the "Canny" part of the project I chose values that would fit the image I decided to work on (p=0.20, thLow = 0.6*threshold). The algorithm works well on detecting singular leaves, but when applied to a part of the image that has many leaves and stems, it seems to detect everything as one object.

I tried two different implementations by changing the steps of the algorithm. On one implementation, I tried opening the image in order to get rid of the white spaces between contour lines that were not connected, thus not counted as leaves (fig. 5.) and on the other one I tried doing it without opening the image(fig. 6).



5. Conclusion

I would like to continue working on this project and making everything work better. The next step I see in completing this project is to change the contour traversal algorithm so it could recognize the difference between objects.