Group: Bortolin Samuel, Grassi Alessandro

Department of Information Engineering and

Computer Science

Project of Signal, Image and Video:

Computer Vision applied to sprout detection:

Detection of sprouts and flowers

Academic year 2020/2021

Contents:

1. Introduction

The scope of the project is to isolate sprouts and flowers from images. We have been provided with 88 images from professor De Natale, such images regard mostly branches and often they are clearly in the foreground. Indeed the main challenge of this work is given by the fact that the sprouts that have to be detected have colors similar to the object in the background. Due to this fact edges detection and isolation of the color are techniques hard to use.
Here is the link of the provided images: Robinia

1.1. Used materials:

In order to accomplish our goal we relied on popular libraries to speed up the work, the complete list is reported below:

- Copy: Shallow and deep copy operations — Python 3.9.3 documentation
- NumPy: NumPy Documentation — NumPy v1.20 Manual
- OpenCV: OpenCV: OpenCV modules 4.5.1
- Typing: Support for type hints — Python 3.9.3 documentation

1.2. Structure of the project:

To make the work easier we decided to put some attention to the structure of the project and to the cleanliness of the code. In particular, we decided to develop a library and split the code in different scripts so as to keep a modular setup.

```
sprout-detection
├── (images)           [ignored folder where you can add your images and then change the "image.extension" in the files with the name of your images]
└── src
    ├── image_utils              [package that contains the utils for images]
    │   └── standard_image_operations              [library which contains the main operations on images]
    ├── color_picker             [script to pick HSV colors and define ranges]
    ├── hsv_filter               [script to try our standard HSV filter on different kind of images]
    └── main                     [script to extract the relevant edges of the image using canny and sobel, also using an HSV filtering approach]
```

Figure 1.1: Graphical representation of the structure of the project.
Source: sprout-detection project structure

image_utils package containing standard_image_operations.py where there is our library: **StandardImageOperations**.
We implemented there all the main operations on images that we needed in order to perform the detection.

- rescale_image(image_to_rescale, target_number_of_pixels)
- grab_contours(contour_tuple)
- find_canny_best_threshold(greyscale_image)
- canny_edges(greyscale_image, low_threshold)
- canny_elements(greyscale_image, low_threshold)
- canny_on_image(greyscale_image, low_threshold, original_image)

- sobel_edges(greyscale_image)
- remove_percentile(greyscale_image)
- apply_hsv_mask(original_image, hsv_image, lower_bound, upper_bound)
- get_hsv_mask(original_image, hsv_image, color)

The list of the developed scripts is provided below:

- **color_picker**.py: in the first phase it allows you to pick pixels from an image and then clicking “q” it returns the range of HSV values that contains all the selected pixels; in the second phase it allows you to build an HSV filter using trackbars to modify the value of hue, saturation and value, clicking a button it updates the filter on the image and clicking “q” it returns the range of HSV values selected with the trackbars. This has been particularly useful to pick the right HSV values for the detection of flowers, and in particular for the removal of the sky.
- **hsv_filter**.py: script that performs an isolation of the color for different types of images containing flowers, leaves or branches using our standard HSV filter.
- **main**.py: script that extracts the relevant edges of the image using canny and sobel. It also tries an HSV color filtering approach and extracts the relevant edges of the HSV filtered image.

2. Methodology

This section will describe the methodologies used in realizing this project.

In particular we mainly used an edge detection approach and an HSV color filtering approach and tried to unify the benefits of both the techniques.

Here is the link for the version 1.2.3 of the developed code: sprout-detection-1.2.3
Here is the link for the GitHub repository used for this project: sprout-detection

2.1. Edge detection:

To isolate the sprouts we assumed that they are the most focused elements in the image, and if it is true we assume that the most relevant edges are the ones belonging to the sprouts. To detect the edges we relied on Canny, which is well known for its good results. The only problem with Canny is that it has two thresholds that need to be tuned. To pick the right threshold we've been inspired by the gradient descent approach. First of all we discard all values below 25 because they kept too many edges, even the least relevant ones. After that the algorithm computes the edges with threshold 25 and with threshold 24 and we compute the difference between the number of edges detected with both thresholds. After that we iterate computing the Canny algorithm with increasing number as threshold until the sum of the two previous differences of edges is below 50.

Just like that we obtained good results, but some edges that are not part of the sprout are still present. To delete such edges we relied on the sobel algorithm, we compute first the x and y derivative obtaining a grayscale image where each pixel can be interpreted as the intensity of the edge in that position. The two derivatives are summed together. After that we apply the close morphological operator in order to connect the edges together and the open in order to remove noise. We also tried to apply first open and then close but it had the effect of disconnecting too much of the edges. To the result of these operations we remove the 85 percentile of the lowest pixel values so as to keep only the most relevant edges.

Finally we can retrive the final edges of the image just by computing the bitwise and between the two approaches above described. Once we have the edges we can reconstruct the sprouts by thickening the edges. This is done by finding the contours of the edges with cv2.drawContours and setting their thickness to 10 pixels.

2.2. HSV color filtering:

We tried to isolate the target object from the rest of the image. We have adopted a color filtering approach using the HSV color model.
The HSV is a coded system with which it is possible to define colors through the use of 3 parameters: color (hue), saturation and value. Following a series of attempts allowed by the “color_picker.py” that we developed it was possible to obtain three range values corresponding to the white in order to isolate flowers, green in order to isolate leaves and brown in order to isolate branches.

The obtained values are:

- for the white in order to better isolate the flowers and avoid to include the sky and the green we did a triple filtering operation and then use a or:
    - lower_bound_1 = [0, 0, 50]
    - upper_bound_1 = [35, 255, 255]
      For removing green
    - lower_bound_2 = [75, 0, 50]
    - upper_bound_2 = [90, 255, 255]
      For removing the sky
    - lower_bound_3 = [110, 0, 50]
    - upper_bound_3 = [179, 255, 255]
- for the green:
    - lower_bound = np.array([20, 0, 0])
    - upper_bound = np.array([80, 255, 255])
- for the brown:
    - lower_bound = np.array([0, 0, 0])
    - upper_bound = np.array([30, 255, 255])

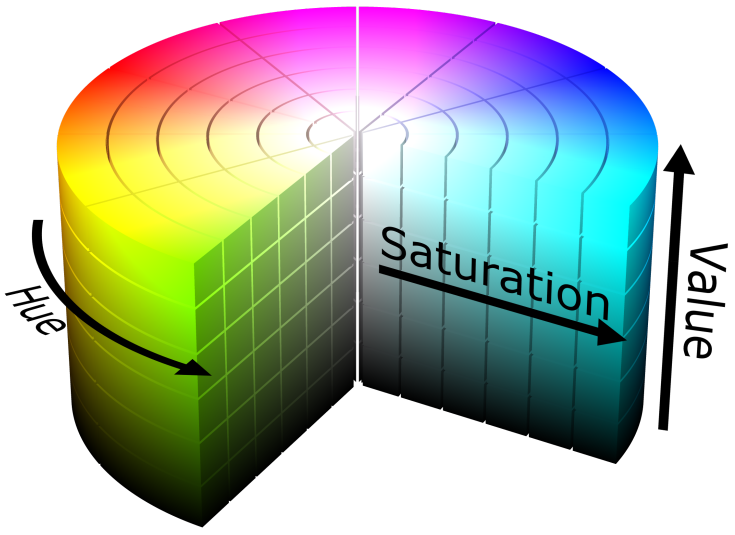

Figure 2.1: Pie chart representing the variation of the parameters Hue, Saturation and Value within the HSV domain. Source: https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png

2.3. Threshold:

One of the approaches tried is the thresholding of the image, the idea was that the sprotus could have different colors with respect to the background and mainly the sprouts are made of the same color with slightly different shades. Unfortunately the results weren't promising, the reason is mainly because the images are taken mostly in a countryside setting. In such a context the background of the images is mostly uniform and not really different from the sprouts. Better results were obtained when the main subjects of the picture are flowers that are white colored, so they are well distinguishable from the leaves and branches. Unfortunately not always flowers were the main part of the images, indeed it happened to have samples with flowers hovered with a lot of leaves, resulting in having a predominance of green instead of the white of the flowers. Another problem is raised by the sky, being bright it is clustered with the flowers, maybe with blooms of different colors this problem may be avoided. Due to these problems we decided to avoid this technique.

2.4. Standard Image Operation Documentation:

We developed an "image_utils" package containing standard_image_operations.py where there is our library: StandardImageOperations that contains all the operations that we use in the different developed scripts.

- rescale_image(image_to_rescale, target_number_of_pixels)
  Rescale the image to a target number of pixels

- grab_contours(contour_tuple)
  Grab a list contours from the tuple resulting from cv.findContours

- find_canny_best_threshold(greyscale_image)
  Find the best threshold for canny edge detector

- canny_edges(greyscale_image, low_threshold)
  Apply canny edge detector and return the edges

- canny_elements(greyscale_image, low_threshold)
  Apply canny edge detector and return the sum of elements in the mask generated by the edges

- canny_on_image(greyscale_image, low_threshold, original_image)
  Apply canny edge detector and return the edges on the original image

- sobel_edges(greyscale_image)
  Apply sobel edge detector and return the edges

- remove_percentile(greyscale_image)
  Remove percentile from a grayscale image

- apply_hsv_mask(original_image, hsv_image, lower_bound, upper_bound)
  Apply an hsv mask based on a range of hsv colors on the original image

- get_hsv_mask(original_image, hsv_image, color)
  Apply an hsv mask based on a color of interest on the original image

2.5. Description of the code:

We developed three scripts:

- “color_picker.py”
  We used it mainly to build our standard HSV filter.

- “hsv_filter.py”
  We used it mainly to see how the isolation of the color performs on different types of images containing flowers, leaves or branches.

- “main.py”
  This is the main code of our project, that given an image it extracts the relevant edges of the image using canny and sobel. It also tries an HSV color filtering approach and extracts the relevant edges of the HSV filtered image.
  The results of this code are presented in the next section on different kind of images to show how it performs and the different images it can extract.

3. Results & Conclusions

This section will describe the results that we have reached and also some conclusions, remarks and ideas for future works.

Results of the main code on an image containing a branch:

Figure 3.1.1: original branch image

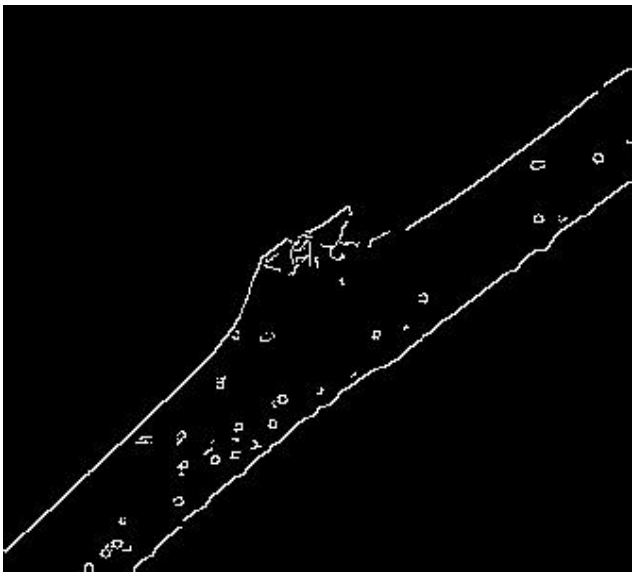

Figure 3.1.2: Sobel-Canny edges approach applied on a branch image

Figure 3.1.3: contours area approach on a branch image

Figure 3.1.4: Sobel-Canny edges on a branch image

Figure 3.1.5: hsv-filtered branch image

Figure 3.1.6: hsv-filtering edges + Sobel-Canny edges approach applied on a branch image

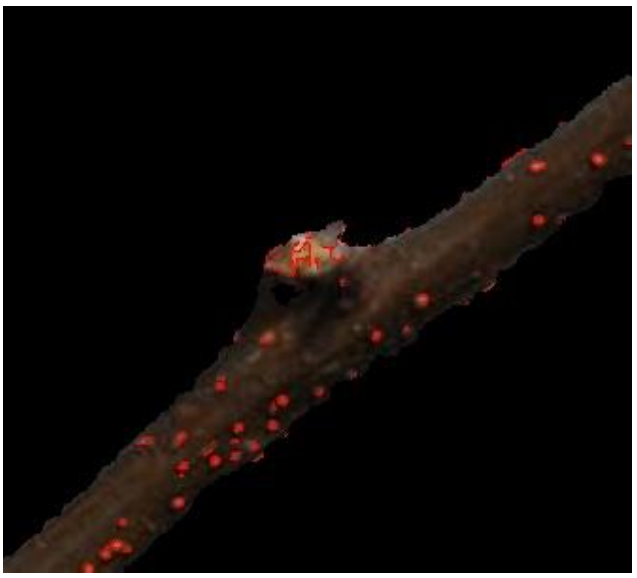

Figure 3.1.7: hsv-filtering edges + Sobel-Canny edges approach on the hsv-filtered branch image

Figure 3.1.8: hsv-filtering edges + Sobel-Canny edges approach on a branch image

Results of the main code on an image containing leaves:

Figure 3.2.1: original leaves image

Figure 3.2.2: Sobel-Canny edges approach applied on a leaves image

Figure 3.2.3: contours area approach on a leaves image

Figure 3.2.4: Sobel-Canny edges on a leaves image

Figure 3.2.5: hsv-filtered leaves image

Figure 3.2.6: hsv-filtering edges + Sobel-Canny edges approach applied on a leaves image

Figure 3.2.7: hsv-filtering edges + Sobel-Canny edges approach on the hsv-filtered leaves image

Figure 3.2.8: hsv-filtering edges + Sobel-Canny edges approach on a leaves image

Results of the main code on an image containing flowers:

Figure 3.3.1: original flowers image

Figure 3.3.2: Sobel-Canny edges approach applied on a flowers image

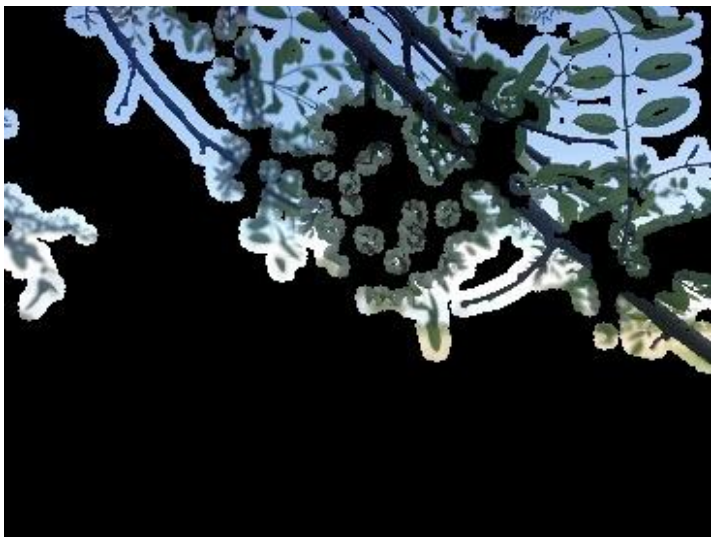

Figure 3.3.3: contours area approach on a flowers image

Figure 3.3.4: Sobel-Canny edges on a flowers image

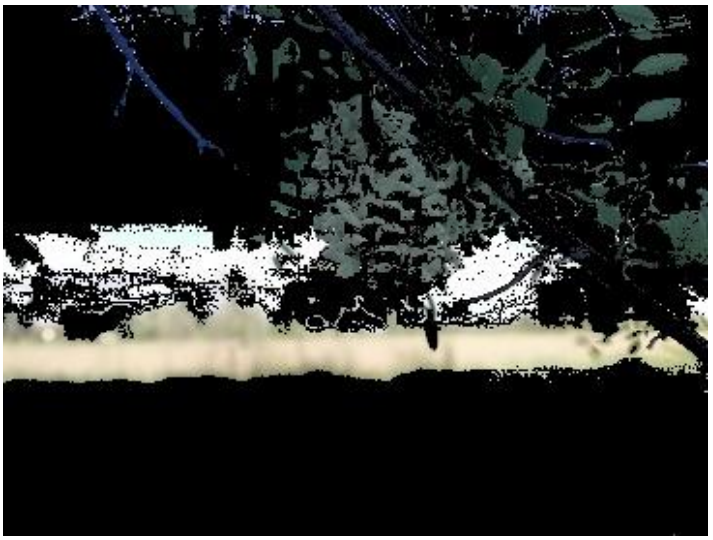

Figure 3.3.5: hsv-filtered flowers image

Figure 3.3.6: hsv-filtering edges + Sobel-Canny edges approach applied on a flowers image

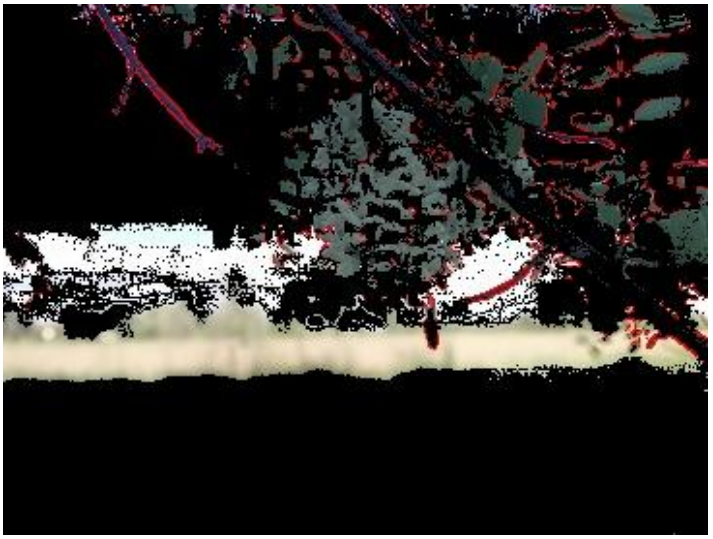

Figure 3.3.7: hsv-filtering edges + Sobel-Canny edges approach on the hsv-filtered flowers image

Figure 3.3.8: hsv-filtering edges + Sobel-Canny edges approach on a flowers image

From these image results we can see that even if the images are not optimal for this task, we detected in an effective way the main subject of the image.

In a lot of images the main subject is not on focus or in the foreground and our edge detection approach does not work so well. Instead, when the object of interest is in the foreground and clearly identifiable our edge detection approach works very well.

We also developed an area approach that tries to reconstruct the object through expanding the edges of 10 pixels.

A note on the hsv approach is that it is not valid in general for all kinds of images, this is due to different subjects. In fact our main code requires at the beginning to specify what do you want to analyze and type it.
When specified the subject for a specific image, the HSV approach works also well and isolates the subject. Sometimes there is an issue with that approach since most of the pictures are in a countryside setting. In such a context the background of the images could be of the same color of the object and not filtered out.

3.1. Future work:

Results are not bad but we think that the use of machine learning and deep learning can certainly improve the achieved results. In particular, these methods can understand the content of an image and isolate the object properly. Such methods usually require a lot of data to be properly trained, fortunately it could be possible to reach a good amount of images just with data augmentation.