**WORKFLOWS**

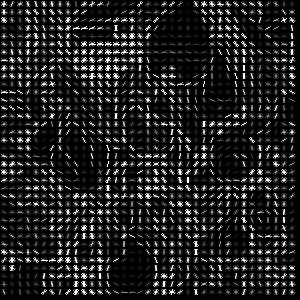
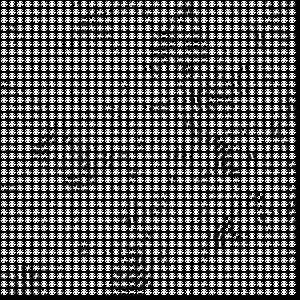
**WORKFLOW 1** (histogram of oriented gradients, hough circles)

1. PREPROCESSING: COLOR REMPAP, SIZING, BLURRING

For the HOG feature descriptor is necessary to choose between color scales and color normalization.

In the original paper by Dalal and Triggs[[1]](#footnote-1), the authors say that both RGB and LAB color spaces perform identically, while using grayscale images reduces performance meaning that HOG feature descriptor works best on colored images.

Even though a normalization step is optional, in some cases it can improve performance of the HOG descriptor; specifically square-root normalization has been demonstrated by Dalal and Triggs to improve accuracy. This is true in our use case, as shown by the two pictures below:

Square root normalization No square root normalization

With respect to resizing, as again demonstrated in the original paper1, the most common image size is 64×128 (width x height) pixels, although is possible to use any image size that has the ratio 1:2 (e.g. 128×256 or 256×512). To note this should be taken in consideration if the HOG algorithm will be used together with machine learning applications, since in the training step it is necessary that images are of the same size.

As for the canny edge algorithm, preprocessing steps will include converting the image to gray scale (ensures that there will be less noise during the edge detection process) and optionally applying additional blurring: while the Canny edge detector does apply blurring before edge detection, extra blurring prior to the edge detector can be used to further reduce pixel noise - remove or minimize unnecessary detail that could lead to undesirable edges, allowing to better find the objects in an image.

1. FEATURE DESCRIPTOR: HOG, CANNY EDGES

HOG

[The Histogram of Oriented Gradients (HOG)](https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients) feature descriptor counts events of gradient orientation in a specific portion of an image. or region of interest. While such descriptors are mainly used in conjunction to computer vision and machine learning for object detection, they can be as well employed for supporting image recognition by quantifying and representing both shape and texture of an image: histogram of oriented gradients can describe local object appearance and shape within an image by the distribution of intensity gradients or edge directions. The gradients( x and y derivatives of an image) are useful because the magnitude of gradients is large around edges and corners due to abrupt change in intensity, which pack in a lot of information about object shape than flat regions.

The implementation of HOG used is the following:

fd, hog\_img = hog(img, orientations=9, pixels\_per\_cell=(8, 8), cells\_per\_block= (2, 2), block\_norm= 'L2-Hys', transform\_sqrt= True, visualize= True, channel\_axis= -1)

To note: the feature vector is not really useful for the purpose of viewing the image, but rather for tasks like image recognition and object detection. The feature vector produced by these algorithms, when fed into an image classification algorithms like Support Vector Machine (SVM), can produce good results: the HOG feature extractor has been used to take histogram descriptors of both positive (images that contain objects) and negative (images that does not contain objects) samples and training the SVM model on that.

CANNY EDGES

Another feature extraction algorithm is the one known as canny edge detection. Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. The process of image detection involves detecting sharp edges in the image: tracking down edges in an image, should allow to find boundaries of objects in an image, boundaries of shadowing or lighting conditions in an image, boundaries of “parts” within an object.

Edge detection is essential in context of image recognition or [object localization/detection](https://en.wikipedia.org/wiki/Object_detection" \t "_blank), and has been widely applied in various [computer vision](https://en.wikipedia.org/wiki/Computer_vision" \o "Computer vision) systems.

The canny edge detection algorithm implementation used our case is the following:

edges = cv.Canny(img, 80, 200, L2gradient= True)

The resulted output for the sample images is the following:



A main drawbacks of the Canny edge detector is tuning the upper and lower thresholds for the hysteresis step: too wide threshold can lead to too many edges, too tight to edges not detected at all.

1. FILTERS COMBINATION

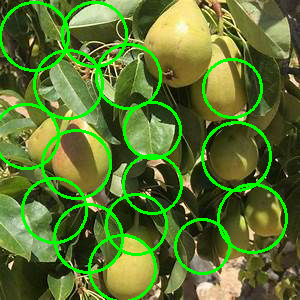
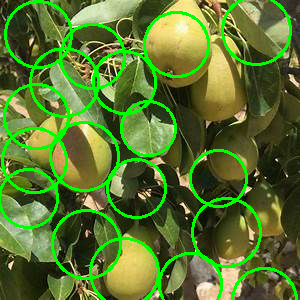
The resulting images from the feature detection step are then passed as input images to the hough circle transform method in order to detect circular shapes.

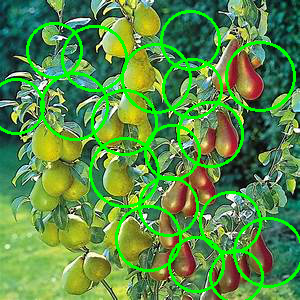
The implementation of such is the following, for which non trivial is the definition of the different parameters:

circles = cv.HoughCircles(gray, cv.HOUGH\_GRADIENT, 1, minDist= gray.shape[0]/8, param1=100, param2=20, minRadius=25, maxRadius=35)

The resulting output images are the one below, from which we can deduce that applying hough circles on canny edges provides for better results.

Hough circles on HOG image Hough circles on canny edges







Nonetheless, as the images show, the implementation of such method appears to be not particularly useful in detecting shapes that are not perfectly circular, and therefore might not be totally useful for the scope of the work. Despite this, an improved tuning of the parameters and an application of it in combination with other techniques, can successfully identifying non perfect circular object (still with some limitations in case of distant or occluded objects), as some studies demonstrated.[[2]](#footnote-2)

**WORKFLOW 2** (thresholding, contour detection)

1. PREPROCESSING

As a preprocessing step to contour detection, the image is converted in gray scale. In addition, adjustment of contrast and brightness and blurring of the image could be applied to reduce noise and minimize unnecessary details.

With respect to blurring, while convolutions such as blur/median blur often result in a loss of important edge information, since they blur out everything irrespective of it being noise or an edge, a bilateral filter is used for smoothening images and reducing noise, while preserving edges[[3]](#footnote-3) .

The implementations used are the following:

gray = cv.cvtColor(img, cv.COLOR\_RGB2GRAY)

image = cv.bilateralFilter(gray, 20, 60, 60)

1. THRESHOLDING METHOD AND CONTOURS

A contour is a closed curve joining all the continuous points having some color or intensity, they represent the shapes of objects found in an image. Contour detection is a useful technique for shape analysis, [object detection and recognition](https://www.thepythoncode.com/article/detect-faces-opencv-python" \t "_blank" \o " How to Detect Human Faces in Python using OpenCV), simple-image segmentation.

To successfully detect contours in an image, OpenCv requires to convert the image to a binary image through either thresholding or edge detection. The code to perform thresholding could be one of the following:

* \_, binary = cv.threshold(gray, 240, 255, cv.THRESH\_BINARY)
* retVal, binary = cv.threshold(gray,0,255,cv.THRESH\_BINARY+cv.THRESH\_OTSU)
* binary = cv.adaptiveThreshold(gray, 255, cv.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv.THRESH\_BINARY, 135, 2)

While the first two are global thresholding methods, meaning that the same value of *T* is used to test *all pixels* in the input image, the last one is referred to as adaptive (local) thresholding: considers small neighbors of pixels and then finds an optimal threshold value *T* for each neighbor. This implies that local regions of an image will have similar lighting, reducing therefore the effect that shadowing and variations in lighting conditions could produce: in situations where the lighting is non-uniform across the image, having only a single value of *T* can seriously hurt our thresholding performance.[[4]](#footnote-4) Being this our work case, we opt for applying the adaptive thresholding method .

Once the binary image is created, it can be used to find the contours:

contours, hierarchy = cv.findContours(binary, cv.RETR\_EXTERNAL, cv.CHAIN\_APPROX\_SIMPLE)

Next step would be the drawing of the identified contours:

cv.drawContours(img, contours, -1, (0, 0, 255), 2)

Drawing an approximation of the identified contours, seems to reduce insignificant noisy ones:

for cnt in contours:

approx = cv.approxPolyDP(cnt,0.01\*cv.arcLength(cnt,True),True)

if len(approx) > 12:

cv.drawContours(img,[cnt],0,(0,0,255),2)

all contours and resulting image approximated contours and resulting image

1. FILTERS COMBINATION

On the obtained contoured image, hough circle transform method can be used to detect circular shapes:

circles = cv.HoughCircles(gray, cv.HOUGH\_GRADIENT, 1, minDist= gray.shape[0]/8, param1=100, param2=20, minRadius=25, maxRadius=35)

The final output images obtained are the one below



The same considerations on the usage of the hough circle transform for circle detection made before apply also in this case.

BIBLIOGRAPHY

[Image Recognition using Histogram of Oriented Gradients (debuggercafe.com)](https://debuggercafe.com/image-recognition-using-histogram-of-oriented-gradients-hog-descriptor/)

[Histogram of Oriented Gradients — skimage 0.22.0rc0.dev0 documentation (scikit-image.org)](https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html#id3)

[Histogram of Oriented Gradients (and car logo recognition) – PyImageSearch](https://customers.pyimagesearch.com/lesson-sample-histogram-of-oriented-gradients-and-car-logo-recognition/)

[Histogram of Oriented Gradients explained using OpenCV (learnopencv.com)](https://learnopencv.com/histogram-of-oriented-gradients/)

<https://medium.com/@dnemutlu/hog-feature-descriptor-263313c3b40d>

<https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f#:~:text=Histogram%20of%20Oriented%20Gradients%2C%20also,the%20purpose%20of%20object%20detection>.

<https://learnopencv.com/edge-detection-using-opencv/>

<https://pyimagesearch.com/2021/05/12/opencv-edge-detection-cv2-canny/>

<https://towardsdatascience.com/easy-method-of-edge-detection-in-opencv-python-db26972deb2d>

<https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-2-adaptive-thresholding/>

<https://pyimagesearch.com/2021/05/12/adaptive-thresholding-with-opencv-cv2-adaptivethreshold/>

<https://medium.com/simply-dev/detecting-geometrical-shapes-in-an-image-using-opencv-bad67c40174f>

<https://www.authentise.com/post/detecting-circular-shapes-using-contours>

<https://pyimagesearch.com/2021/10/06/opencv-contour-approximation/>

<https://learnopencv.com/contour-detection-using-opencv-python-c/>

<https://hackthedeveloper.com/circle-detection-opencv-python/>

<https://www.geeksforgeeks.org/circle-detection-using-opencv-python/>

<https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html>

1. Dalal, N. and Triggs, B., “Histograms of Oriented Gradients for Human Detection,” IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, San Diego, CA, USA [↑](#footnote-ref-1)
2. <https://www.researchgate.net/publication/250801437_Object_Detection_using_Circular_Hough_Transform> [↑](#footnote-ref-2)
3. <https://www.geeksforgeeks.org/python-bilateral-filtering/> [↑](#footnote-ref-3)
4. <https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-2-adaptive-thresholding/> [↑](#footnote-ref-4)