Day 2

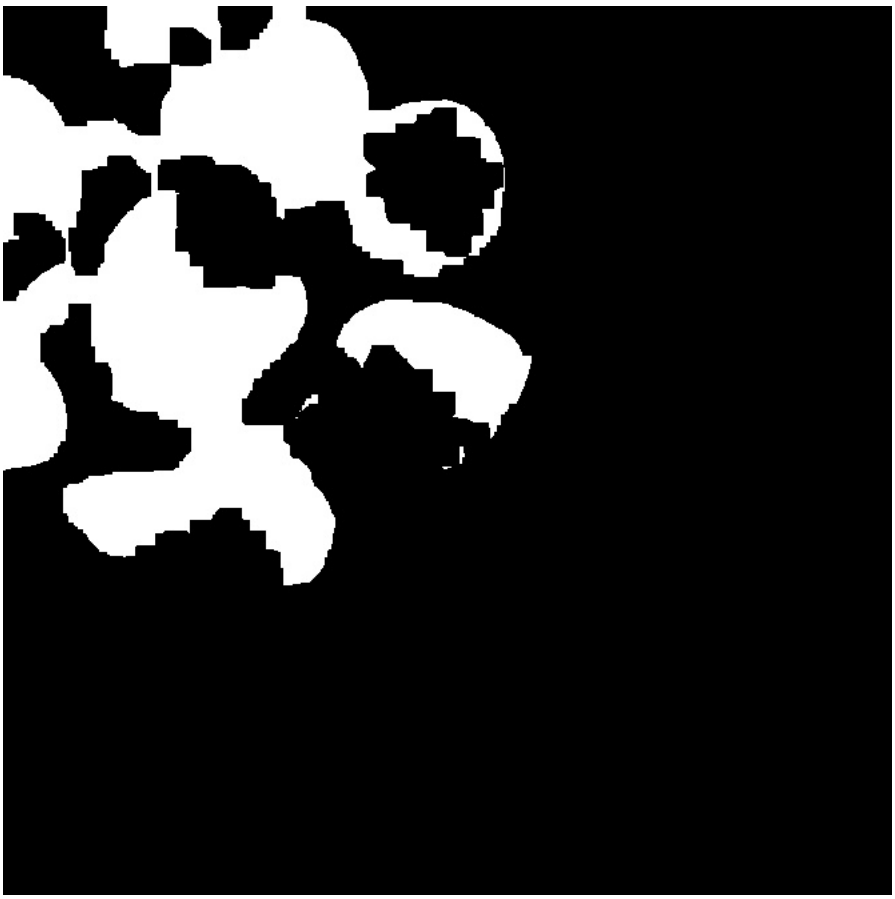
Teemu Sarapisto

May 15, 2018

1 Hands-on day 2

Hola world

1.1 Use OpenCV's morphological dilation and closing with different structuring element shapes and sizes to fill in the "holes" in the strawberries. Try to also clean away the noisy stray pixels with mild erosion or opening



1.2 What seems to be the best operations, structuring element shapes and their sizes? In which order were they performed?

- Closing operation with a rectangular kernel

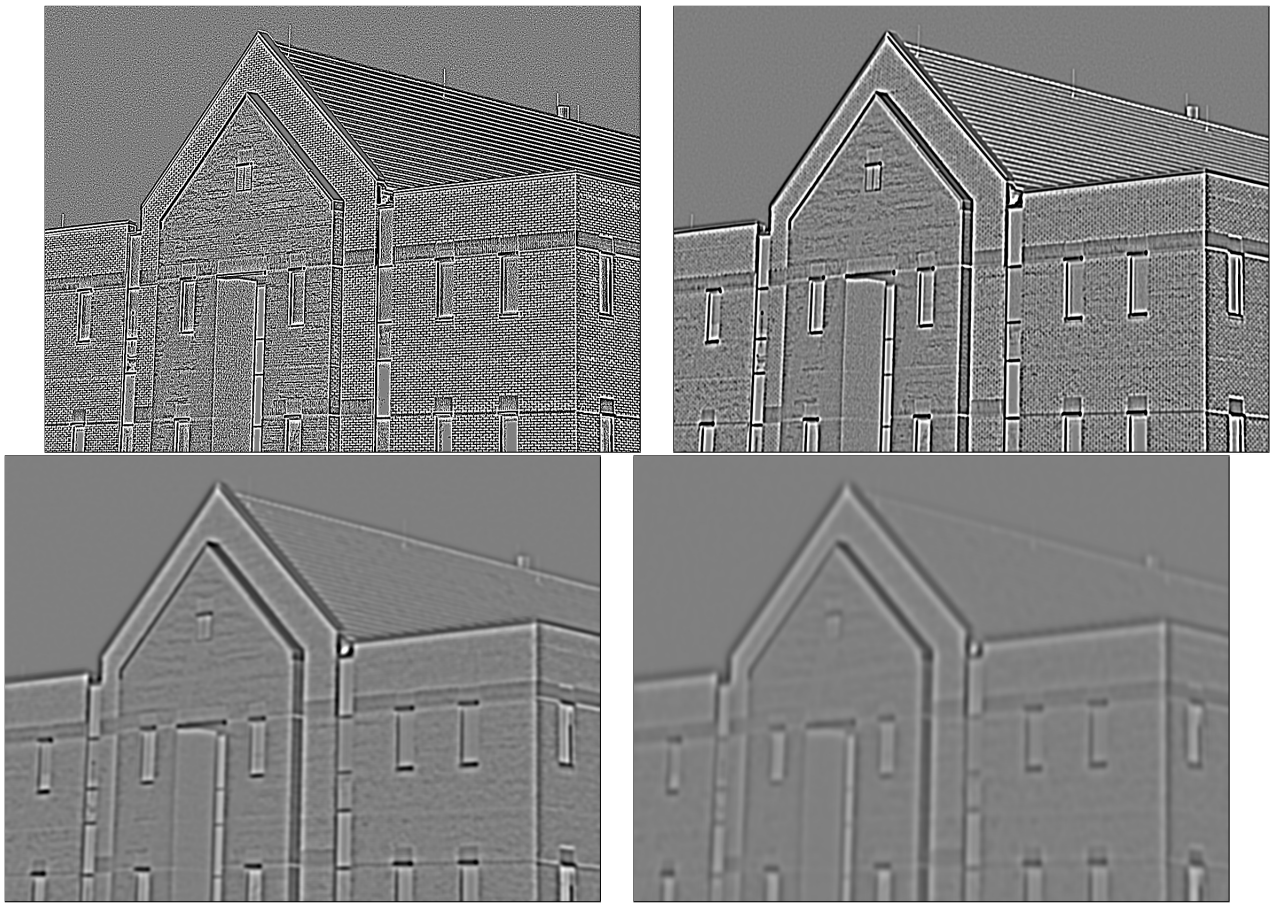
- Dilation operation with a rectangular kernel
- Closing operation 2 with a rectangular kernel
- Erosion operation with a rectangular kernel

1.3 Apply a series of OpenCV's Gaussian blurring operations with increasing kernel sizes to get smoothed versions of the image

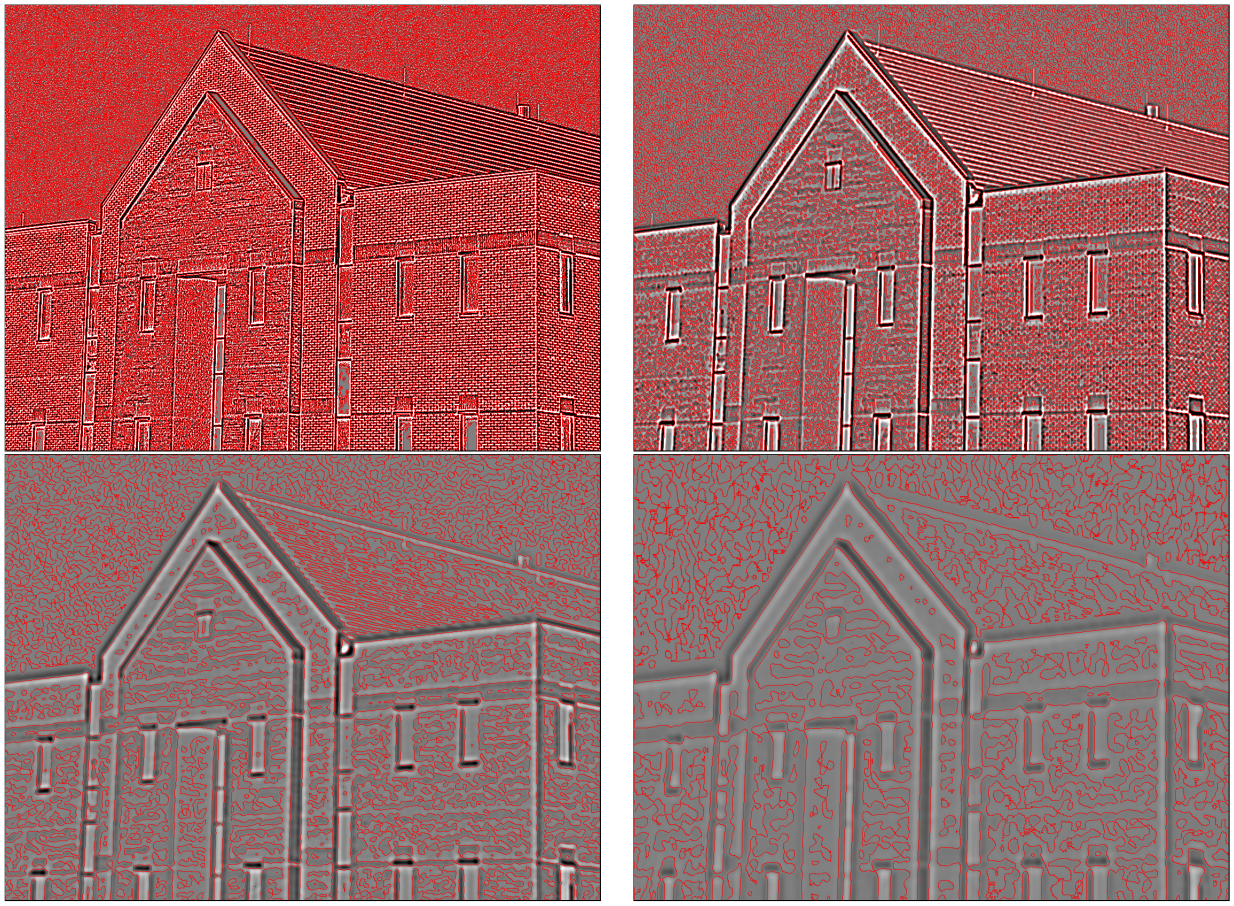
Kernel sizes: 3/5 11/15 27/27 37/47

1.4 Apply OpenCV's Laplace filtering to the blurred images

oispa kaljaa



1.5 Analyze how the size of the detected details increases with the increasing blur size



With the increasing blur size the size of the detected details increases.:

1.6 how long it took

Around 4 hours

1.7 Hands-on code

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from scipy import ndimage as nd
from scipy.spatial import distance
```

```
# We will use the image, strawberries-binary.pbm that is a binary mask or segmentation res

strawberries_red = cv2.imread('strawberries-binary.pbm')

#Read the image in and scale it to have \white" values 1 for the strawberry pixels and \bl
binary_strawberries = strawberries_red / 255

#Use OpenCV's morphological dilation and closing with different structuring element shapes
#Try to also clean away the noisy stray pixels with mild erosion or opening
rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
elliptical_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
cross_kernel = cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))

closing = cv2.morphologyEx(binary_strawberries, cv2.MORPH_CLOSE, rect_kernel)

dilation = cv2.dilate(closing, rect_kernel, iterations = 2)

closing2 = cv2.morphologyEx(dilation, cv2.MORPH_CLOSE, rect_kernel)

kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(closing2, kernel, iterations = 3)

#cv2.imshow("homma", erosion * 255)
cv2.imwrite("morph.jpg", erosion * 255)
#cv2.waitKey()


###Next we use building.tiff to study scale space filtering, Laplacian of Gaussian filteri
# Ookoo kuulostaapa hyveltä

###Read the image in and scale it as a single-channel (greyscale) image in range [0,1]
building = cv2.imread('building.tiff', cv2.IMREAD_GRAYSCALE) / 255

###Apply a series of OpenCV's Gaussian blurring operations with increasing kernel sizes to

#cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])  dst
gaussed1 = cv2.GaussianBlur(building, (3,5), 0)
gaussed2 = cv2.GaussianBlur(building, (11,15), 0)
gaussed3 = cv2.GaussianBlur(building, (27,27), 0)
gaussed4 = cv2.GaussianBlur(building, (37,47), 0)

cv2.imwrite('gaussed3-5.jpg', gaussed1 * 255)
cv2.imwrite('gaussed11-15.jpg', gaussed2 * 255)
cv2.imwrite('gaussed27-27.jpg', gaussed3 * 255)
cv2.imwrite('gaussed37-47.jpg', gaussed4 * 255)
```

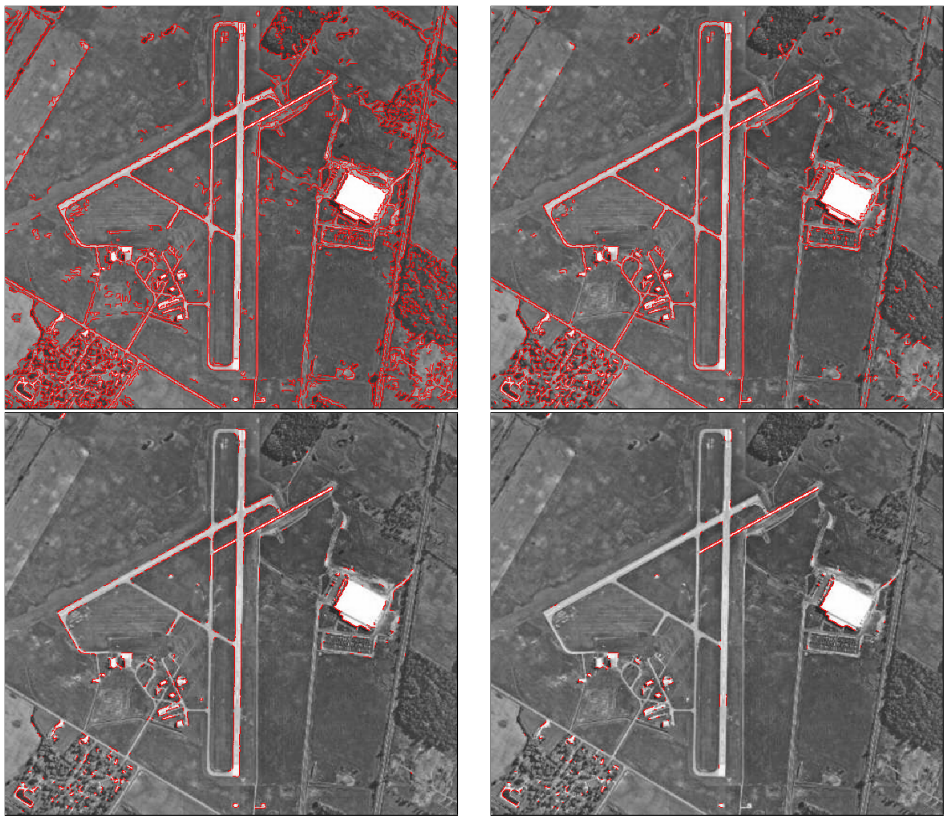
```
#Apply OpenCV's Laplace filtering to the blurred images
#When saving the Laplace images for the report, multiply them with a proper constant, add
def clip(i):
    return np.clip(i, 0, 1)

kernel_size = 3
scale = 1
delta = 0
laplace1 = cv2.Laplacian(gaussed1, cv2.CV_64F)
laplace2 = cv2.Laplacian(gaussed2, cv2.CV_64F)
laplace3 = cv2.Laplacian(gaussed3, cv2.CV_64F)
laplace4 = cv2.Laplacian(gaussed4, cv2.CV_64F)

cv2.imwrite('laplace1.jpg', clip(laplace1 * 40 + 0.5) * 255)
cv2.imwrite('laplace2.jpg', clip(laplace2 * 40 + 0.5) * 255)
cv2.imwrite('laplace3.jpg', clip(laplace3 * 40 + 0.5) * 255)
cv2.imwrite('laplace4.jpg', clip(laplace4 * 40 + 0.5) * 255)

#Write your own function to detect zero crossings (aka sign changes) in the Laplace filter
###Compare each pixel value's sign (-1, 0 or +1) to that of the neighboring pixel on the l

def zero_cross(A):
    new = np.zeros(A.shape)
    for x in range(A.shape[1]):
        for y in range(A.shape[0]):

            if x == 0 and y == 0:
                new[y][x] = 0
            elif x == 0:
                if np.sign(A[y][x]) != np.sign(A[y][x-1]):
                    new[y][x] = 1
                else:
                    new[y][x] = 0
            elif y == 0:
                if np.sign(A[y][x]) != np.sign(A[y-1][x]):
                    new[y][x] = 1
                else:
                    new[y][x] = 0
            else:
                if np.sign(A[y][x]) != np.sign(A[y-1][x]) or np.sign(A[y][x]) != np.sign(A
                    new[y][x] = 1
                else:
                    new[y][x] = 0
    return new
```

```
zbuilding = zero_cross(building)
#print(zbuilding[23])

#cv2.imwrite('zero_building.jpg', zbuilding)

#Mark the zero crossing pixels with red color in the images
def color_zero(zero_cross, original):
    original_uint8 = np.array(original, dtype=np.uint8)
    new = cv2.cvtColor(original_uint8,cv2.COLOR_GRAY2BGR)
    for x in range(zero_cross.shape[0]):
        for y in range(zero_cross.shape[1]):
            if zero_cross[x,y] == 1:
                new[x,y] = [0,0,255]
            else:
                new[x,y] = original[x,y]

    return new

#cv2.imshow('haloo', color_zero(zbuilding, building))
#cv2.waitKey()


#Analyze how the size of the detected details increases with the increasing blur size
laplace1_cross = zero_cross(laplace1)
laplace2_cross = zero_cross(laplace2)
laplace3_cross = zero_cross(laplace3)
laplace4_cross = zero_cross(laplace4)
cv2.imwrite('laplace1_zcross.png', color_zero(laplace1_cross, clip(laplace1 * 40 + 0.5) *
cv2.imwrite('laplace2_zcross.png', color_zero(laplace2_cross, clip(laplace2 * 40 + 0.5) *
cv2.imwrite('laplace3_zcross.png', color_zero(laplace3_cross, clip(laplace3 * 40 + 0.5) *
cv2.imwrite('laplace4_zcross.png', color_zero(laplace4_cross, clip(laplace4 * 40 + 0.5) *
#cv2.waitKey()
#Report again how long it take to complete these assignments
```

2 Homework

2.1 Maximum Strawberry Finder Professional Edition

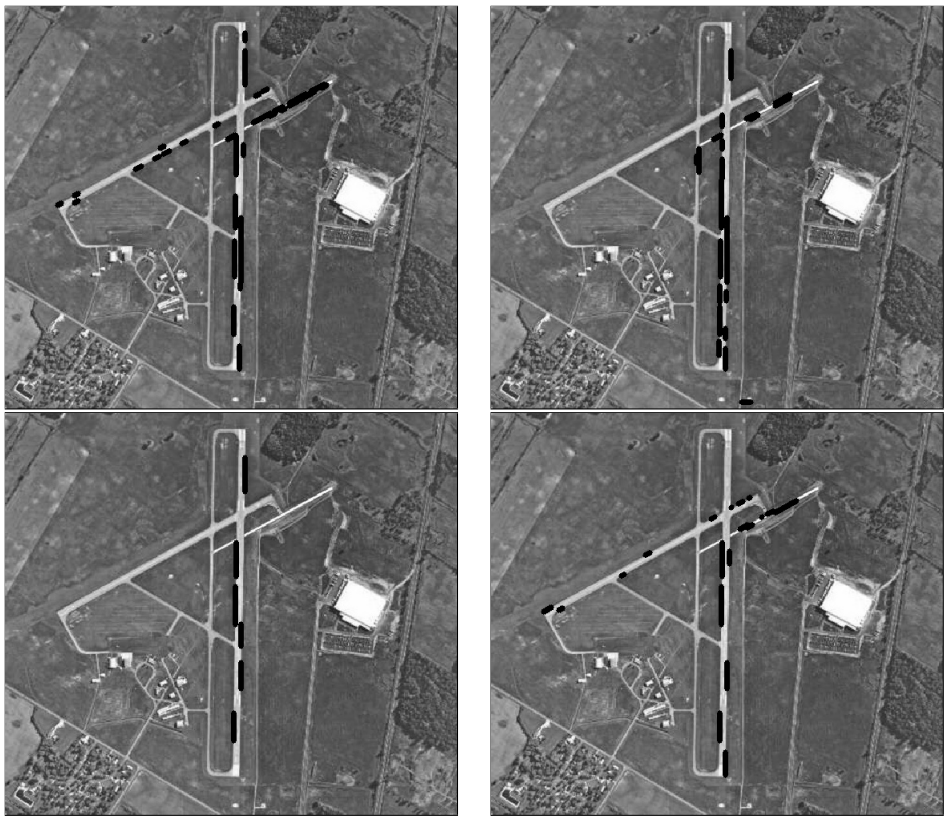
2.2 We try to find airport runways in image marion_airport.tiff. Use OpenCV Canny edge detector function to find edges in the image



second row first image, canny3, was best, it was created with params 400, 500 below:

```
canny1 = cv2.Canny(airport, 100, 200)
canny2 = cv2.Canny(airport, 200, 300)
canny3 = cv2.Canny(airport, 400, 500)
canny4 = cv2.Canny(airport, 500, 600)
```

2.3 Use OpenCV's Hough line transform function to detect the runway pixels experiment with the parameter values to get a good result



2.4 Explain the meaning of the parameters of both the Canny and Hough implementations in OpenCV

Canny parameters are low and high thresholds for determining whether the edge at a pixel is a 'weak' or 'strong' one, by comparing whether the gradient at that pixel is under/over the low/high thresholds respectively.

rho – Distance resolution of the accumulator in pixels.

```
rho - Distance resolution of the accumulator in pixels.
```

Maybe how far the cells are looked for in the accumulator? Not sure.

```
theta - Angle resolution of the accumulator in radians.
```

I'm guessing this is for setting how circle'y curves are searched for

```
threshold - Accumulator threshold parameter. Only those lines are returned that get enough
```

Sets how many grid cells must a curve match with the image for it to count as a found curve.

I'm not sure what the other are

2.5 Would it be possible to make the parameter value selection automatic or is human inspection necessary?

For Canny there is Otsu's Method for selecting the thresholds.

For Hough at least some of the parameters have to be locked while searching for the others, for example fixing the size of a circle that is being searched for.

2.6 Time used

Around 4 hours

2.7 Homework code

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from scipy import ndimage as nd
from scipy.spatial import distance

print(cv2.__version__)


# Implement integral image iif(x,y) and apply it to strawberries-binary.pbm after scaling

def summed_area_table(A):
    new = np.full((A.shape[0] + 1, A.shape[1] + 1), 0)
    print(new.shape)
    for x in range(A.shape[0]):
        if x == 0: continue
        for y in range(A.shape[1]):
            if y == 0: continue
            new[x,y] = new[x,y-1] + new[x-1,y] + A[x,y] - new[x-1,y-1]
    return new

def iif(x0, y0, x1, y1, A):
    return A[x1,y1] - A[x1, y0 - 1] - A[x0 - 1, y1] + A[x0 - 1, y0 - 1]

strawberries_red = cv2.imread('strawberries-binary.pbm', cv2.IMREAD_GRAYSCALE)

#Read the image in and scale it to have \white" values 1 for the strawberry pixels and \bl
binary_strawberries = strawberries_red / 255
```

```
#print(iif(0,1, binary_strawberries))
def get_max(A):
    table = summed_area_table(A)
    #result = np.zeroes((A.shape[0] - 100, A.shape[1] - 100))
    max = 0
    max_pos = (0,0)
    for x in range(A.shape[1] - 100):
        for y in range(A.shape[0] - 100):
            area_sum = iif(x, y, x + 100, y + 100, table)
            #result[x,y] = area_sum
            if area_sum > max:
                max = area_sum
                max_pos = (x,y)

    return max_pos

max_pos = get_max(strawberries_red)

strawberries_color = cv2.imread('strawberries_color.jpg')
rectangled = cv2.rectangle(strawberries_color, (max_pos[1], max_pos[0]), (max_pos[1]+100,

cv2.imwrite('max_mansikkuus.jpg', rectangled)
cv2.waitKey()



#We try to find airport runways in image marion_airport.tiff
#Use OpenCV's Canny edge detector function to find edges in the image
#edges relating to the runways should be found as much as possible and the others as littl
#experiment with the parameter values to get a good result
#include in the report some different outcomes

def add_edges(edges, airport_rgb):
    new = np.copy(edges)
    for x in range(edges.shape[0]):
        for y in range(edges.shape[1]):
            if np.all(edges[x,y] != 0):
                edges[x,y] = [0,255,0]
                new[x,y] = [0, 0, 255]
            else:
                new[x,y] = airport_rgb[x,y]
    return new

airport = cv2.imread('marion_airport.tiff', cv2.IMREAD_GRAYSCALE)
airport = np.uint8(airport)
```

```
canny1 = cv2.Canny(airport, 100, 200)
edges1 = cv2.cvtColor(canny1, cv2.COLOR_GRAY2RGB)
canny2 = cv2.Canny(airport, 200, 300)
edges2 = cv2.cvtColor(canny2, cv2.COLOR_GRAY2RGB)
canny3 = cv2.Canny(airport, 400, 500)
edges3 = cv2.cvtColor(canny3, cv2.COLOR_GRAY2RGB)
canny4 = cv2.Canny(airport, 500, 600)
edges4 = cv2.cvtColor(canny4, cv2.COLOR_GRAY2RGB)

airport_rgb = cv2.cvtColor(airport, cv2.COLOR_GRAY2RGB)
canny1out = add_edges(edges1, airport_rgb)
canny2out = add_edges(edges2, airport_rgb)
canny3out = add_edges(edges3, airport_rgb)
canny4out = add_edges(edges4, airport_rgb)

cv2.imwrite('canny1.png', canny1out)
cv2.imwrite('canny2.png', canny2out)
cv2.imwrite('canny3.png', canny3out)
cv2.imwrite('canny4.png', canny4out)


#Use OpenCV's Hough line transform function to detect the runway pixels
#experiment with the parameter values to get a good result
#include in the report some different outcomes

def houghify(minLineLength, maxLineGap, threshold, canny, A):
    image = np.copy(A)
    lines = cv2.HoughLinesP(canny, 1, np.pi/180, threshold, minLineLength, maxLineGap)
    for line in lines:
        for x1,y1,x2,y2 in line:
            cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 5)
    return image

#minLineLength = 1000
#maxLineGap = 1
#threshold = 40

cv2.imwrite('hough-100-5-50.jpg', houghify(100, 5, 50, canny3, airport))
cv2.imwrite('hough-10-10-10.jpg', houghify(10, 10, 10, canny3, airport))
cv2.imwrite('hough-1000-27-100.jpg', houghify(1000, 27, 100, canny3, airport))
cv2.imwrite('hough-1000-1-100.jpg', houghify(1000, 1, 100, canny3, airport))

#cv2.imwrite('hough.jpg', airport)
```