EL2310 Project 1 – MATLAB
SCR15
Martin Isaksson, misakss@kth.se 9/16/2015

Project 1

From Input Value to Picture

The first thing I do is to read in three pictures, where each picture is a 3D-matrix. After I have the pictures, I tell the user to pick one of the three pictures by taking the input value that the user gives me and re-define the input value to the picture that is already read in, with help of three if-statements. I have also made sure that if the user gives another input value than one of the three that I want, I give the user an error message and give the user another try to give me one of the right input values.

Making the Picture Grayscale

I make the 3D-matrix (the picture) to a 2D-matrix and also I make the picture grayscale. To do that I run a double for-loop where I create a 2D matrix by multiplying the colors R, G and B with given decimal to get a grayscale-picture. When I multiply R, G and B (that are the 3'rd dimension of the picture-matrix from beginning) I remove the 3'rd dimension of the matrix by make it "flat" with new values in the elements, so that I get a grayscale picture. To do that I multiply each element in the 2D-matrix with the new colors of R, G and B.

Convolution

With help of two filter masks, I convolve the grayscale picture with each filter mask to get the gradient in x-direction and y-direction, respectively. After I have each gradient direction, I get the total magnitude of the gradient by adding the gradients in x/y-direction in square, and then taking the square root of that.

To convolve the grayscale picture with the filter masks, I have created a double for-loop. I will explain how it works in the x-direction, because it works in the same way in the y-direction. So, the first thing I do is to create a matrix filled with zeros that has the same size as the grayscale picture. This is to save time, because then MATLAB doesn't have to create each element along as the for-loop runs.

Then I run a for-loop for as many times as there are rows in the grayscale picture (columns in the y-direction case). For each time the for-loop runs, a 3x3-matrix will be multiplied with the filter mask, elementwise. Then I take the sum of the multiplications and this sum (that is a value) will be put in an element in our new matrix, which represent the gradient in the x-direction. When the whole for-loop is done, I have created a new matrix that has the convolved information that just was calculated (see figure 1.a) and 1.b) on the next page).

Something that is important when convolving two matrices in this way is that because it is supposed to be an elementwise multiplication, the created 3x3-matrix (from the grayscale picture) has to be placed so that this works. I also have to consider that the image pixels that I need are 3x3 so that means that I have to start in element (2,2).

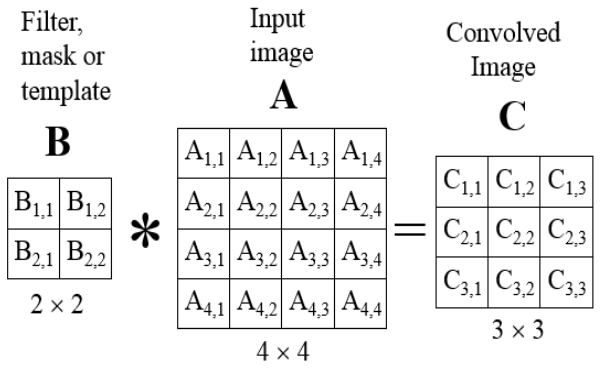

Figure 1.a) The process

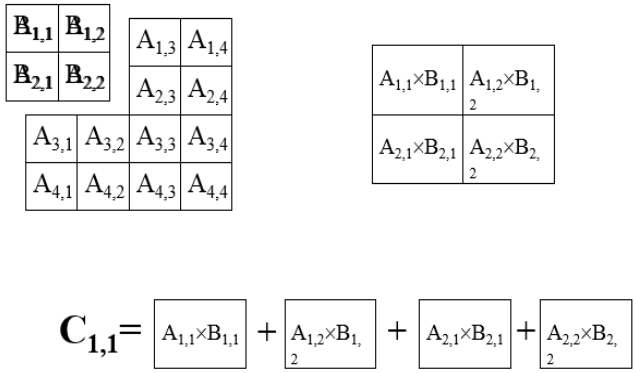

Figure 1.b) The elementwise multiplication

Binary Picture

Now I want to know where in the matrix, that is which elements, that represents a value where we have a circle. In the gradient, that will be the maximums, or more specific I set a threshold that is the limit of what counts as such a maximum and what doesn't.

With this threshold I then create a new matrix that only contains ones and zeros. Ones are the maximums and zeros aren't. This will be my binary picture.

Hough Transform

Here I want to create a 3D-matrix, an accumulator, which contains all the points of a circles center and the radius that correspond to that points. And each point in the 3D-space of the accumulator corresponds to a circle in the 2D-space.

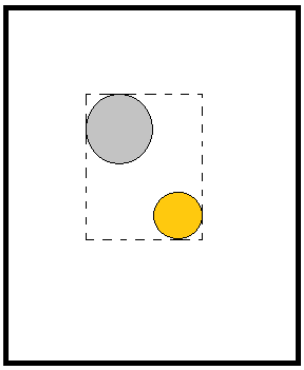

*Figure 2 The dashed square is the area where **a** and **b** are checked*

To make the process a little bit quicker I only run the for-loops for **a** and **b** (the points that correspond to a circles center) in the area of a square that has it edges on where the first one is in the binary picture to the last one in the binary picture (see figure 2, where the dashed square is the area where I run **a** and **b**).

But since I run a triple for-loop this still takes time. The third for-loop is needed because of the varying radius on the coins.

The next thing I do is to create an array that contains so many elements equal to the amount of times that the equation of a circle is fulfilled. There is a slight modification to the equation of a circle in this case, and that is that I have set a threshold. And for every time the equation is smaller or equal to this threshold, MATLAB should do what I say after the if-statement if the created array isn't empty.

Then I say that if that array is not empty MATLAB should add this size to the accumulator. Since we are still in the for-loop, this is for a certain point (**a**,**b**). So the accumulator gets a value in that point, which will be bigger or smaller depending on for how many **a** and **b** the equation is fulfilled.

After this is done, I want to check the accumulator and see for which points (**a**,**b**) I have gotten maximums. To do this I use MATLABs function ***imregionalmax***. Also here I have filtered the accumulator, since I have many maximums, I define what I mean with maximum and that is because if I don't I just get a lot of circles later, everywhere in the picture. That is because there are circles (points in the accumulator) in many places but I only want the circles where it has been repeated the most, that would be where the coins are (this is the principle of Hough transform).

Drawing Circles

Now I want to find the location in the filtered accumulator of the maximums, so I get the points of the circles that I want to draw on the original picture. I also want the radius corresponding to that point.

First I create the x-coordinates for where the circle-points should be placed in the original image, and then I create the y-coordinates, both in the same way.

To do this I have created a double for-loop. One that iterates so many time as the length of so many circles that will be drawn, that is as the number of points of the filtered accumulator. The other for-loop is that I let alpha go from zero to 2*pi, which is necessary if I want to create a circle with help of my points.

The new x-coordinates is created with some geometry (see figure 3). As inputs I have the points from the filtered accumulator and the corresponding radius. Then I do the same with the new y-coordinates and I put them in the original images as inputs. Also I give the circles a color of green. This I to almost as the same way in the beginning, just a little bit easier. Since R, G and B stands for Red, Green and Blue, I just have to set green to its maximum value, which is 255, and red and blue to zero.

To get cleaner circles, with that I mean fewer and just the one on the coin, I use a filter when I draw my pictures. If the coins are on top of each other (so that the center of the top coin is over the edge of the bottom coin) this filter can't be used. But I simply create an if-statement that MATLAB should run if the distance between two center points is larger than the radius. This takes care of the really close circles to that circle that I really want.

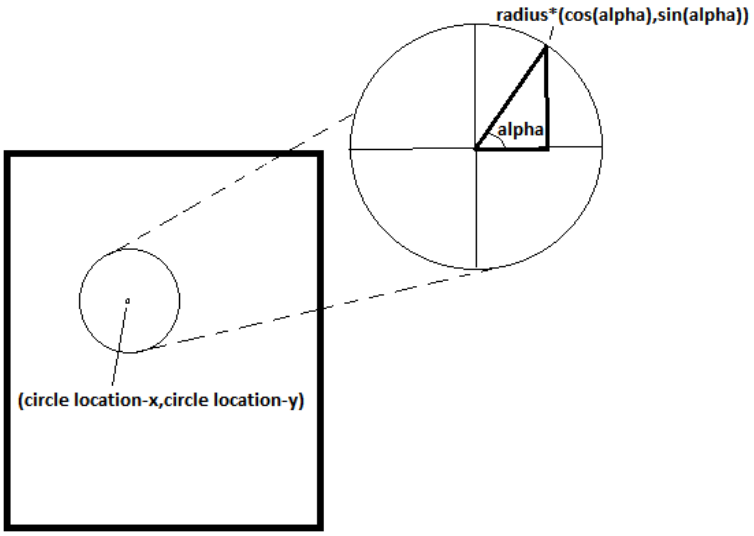

Figure 3 The geometry when drawing a circle on the original picture