# Image processing miniproject – Rotation

## Algorithm

A rotation in image processing is performed by using the rotation matrix. This matrix is used to calculate the new x and y coordinates for a pixel in the image, also called x’ and y’. The rotation matrix for forward mapping is illustrated bellow.



The matrix uses an angle, cosine and sine to calculate the x’ and y’ points of the pixels in the image. This is because cosine and sine values generate a circle called the unit circle. Therefor cosine to an angle multiplied by x, will be two points that lies on the same circle arc. In order to only get one point, one must subtract y multiplied by sin to angle. The same rules apply when calculating a rotation on a y coordinate, though in this case it is x multiplied by sine of an angle plus y multiplied by cosine of an angle.

The rotation performed to an image when using forward mapping is done counter clockwise. Using forward mapping might also cause holes in the image. These holes can mostly be avoided by using backward mapping. The difference in the algorithm between backward and forward mapping, is that index 1,2 I changed from negative to positive and that index 2,1 is changed to negative. Using backward mapping to rotate an image will rotate the image clockwise. The rotation matrix for backward mapping is illustrated below.



## Code

import cv2

import math

import numpy as np

class rotator:

angle = 30.0

x = 330

y = 330

radians = float(angle\*(math.pi/180))

img = cv2.imread('lena.jpg',0)

width,height = img.shape

def showImg(name, self):

cv2.imshow(name, self.img)

self.img = np.pad(self.img, (self.height) ,'constant', constant\_values=0)

self.width,self.height = self.img.shape

def printWH(self):

print(self.width)

print(self.height)

def getImage(self):

return self.img

#Rotates an image using forward mapping

def forward(self, img):

empty = np.zeros((self.width,self.height),dtype="uint8")

for i in range(self.width):

for j in range(self.height):

#forward mapping

x = int((i-self.x)\*math.cos(self.radians)-(j-self.y)\*math.sin(self.radians))+self.x

y = int((i-self.x)\*math.sin(self.radians)+(j-self.y)\*math.cos(self.radians))+self.x

if x < self.width and y < self.height and x>0 and y > 0:

empty[i,j] = self.img[int(x),int(y)]

else:

pass

return empty

#Rotates an image using backward mapping

def backward(self, img):

empty = np.zeros((self.width,self.height),dtype="uint8")

for i in range(self.width):

for j in range(self.height):

#forward mapping

x = int((i-self.x)\*math.cos(self.radians)+(j-self.y)\*math.sin(self.radians))+self.x

y = int(-(i-self.x)\*math.sin(self.radians)+(j-self.y)\*math.cos(self.radians))+self.x

if x < self.width and y < self.height and x>0 and y > 0:

empty[i,j] = self.img[int(x),int(y)]

else:

pass

return empty

#Rotates an image using forward mapping and then rotates it back using backward mapping

def backwardForward(self, img):

empty = np.zeros((self.width,self.height),dtype="uint8")

for i in range(self.width):

for j in range(self.height):

#forward mapping

xO = int((i-self.x)\*math.cos(self.radians)-(j-self.y)\*math.sin(self.radians))+self.x

yO = int((i-self.x)\*math.sin(self.radians)+(j-self.y)\*math.cos(self.radians))+self.x

x = int((xO-self.x)\*math.cos(self.radians)+(yO-self.y)\*math.sin(self.radians))+self.x

y = int(-(xO-self.x)\*math.sin(self.radians)+(yO-self.y)\*math.cos(self.radians))+self.x

if x < self.width and y < self.height and x>0 and y > 0:

empty[i,j] = self.img[int(x),int(y)]

else:

pass

return empty

def main():

rotator.showImg('normal', rotator)

rotator.printWH(rotator)

cv2.imshow('forward', rotator.forward(rotator, rotator.getImage(rotator)))

cv2.imshow('backward', rotator.backward(rotator, rotator.getImage(rotator)))

cv2.imshow('foward image backward', rotator.backwardForward(rotator, rotator.getImage(rotator)))

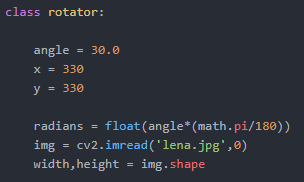
cv2.waitKey(0)

cv2.destroyAllWindows

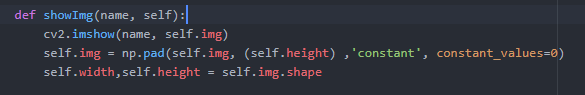
if \_\_name\_\_ == '\_\_main\_\_':

main()

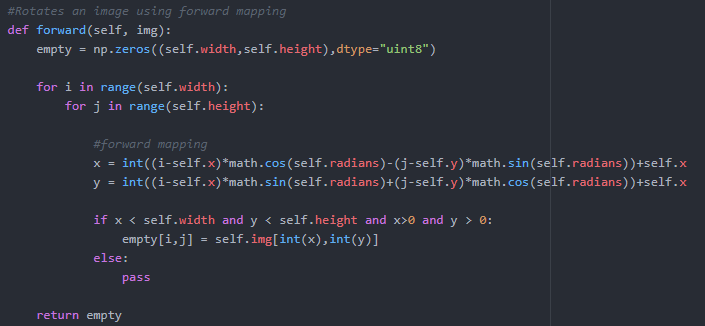
## Explanation of code

The program has one class, which is the rotator class. In this class there are seven attributes and 6 methods. Three of these attributes are angle, x and y. These three attributes defines what angle the image should rotate, and at what x and y coordinate. The other attributes are radians, img, width and height. The attributes are shown in the image to the right.

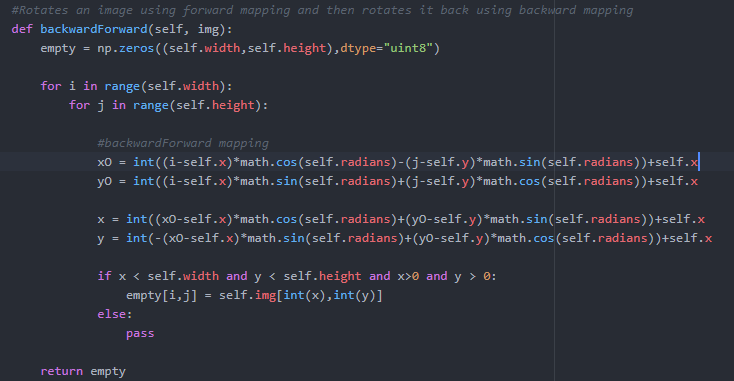
One of the methods in the rotator class is showImg(name, self). This method is used to show the normal unrotated image, to make padding around the image after it is shown and to assign values to the attributes width and height. The padding is used to make black bars around the image, so it can be rotated later, without some of the being cut of do to the increase in width and height. The method is shown below.



The actual rotation of the image happens in three different methods, forward(self, img), backward(self,img) and backwardForward(self,img). Each of these methods uses different algorithms to rotate the image. Though they all creates an empty image and uses a double for loop, to loop through all the pixels in the image. The empty image becomes the rotated image at the end of each method. In the double for loop, the new x and y coordinates for the given pixel is calculated using the algorithms which were explained early. When x and y has been calculated the method checks if the pixels x or y coordinate is between 0 and the attributes width and height, using an if statement. If this is true, then the pixel is inside the image and then the pixel is assigned a pixel value from the original image. When the for loop is finished the method returns empty, which now is the rotated image. The code for forward mapping is showed below.

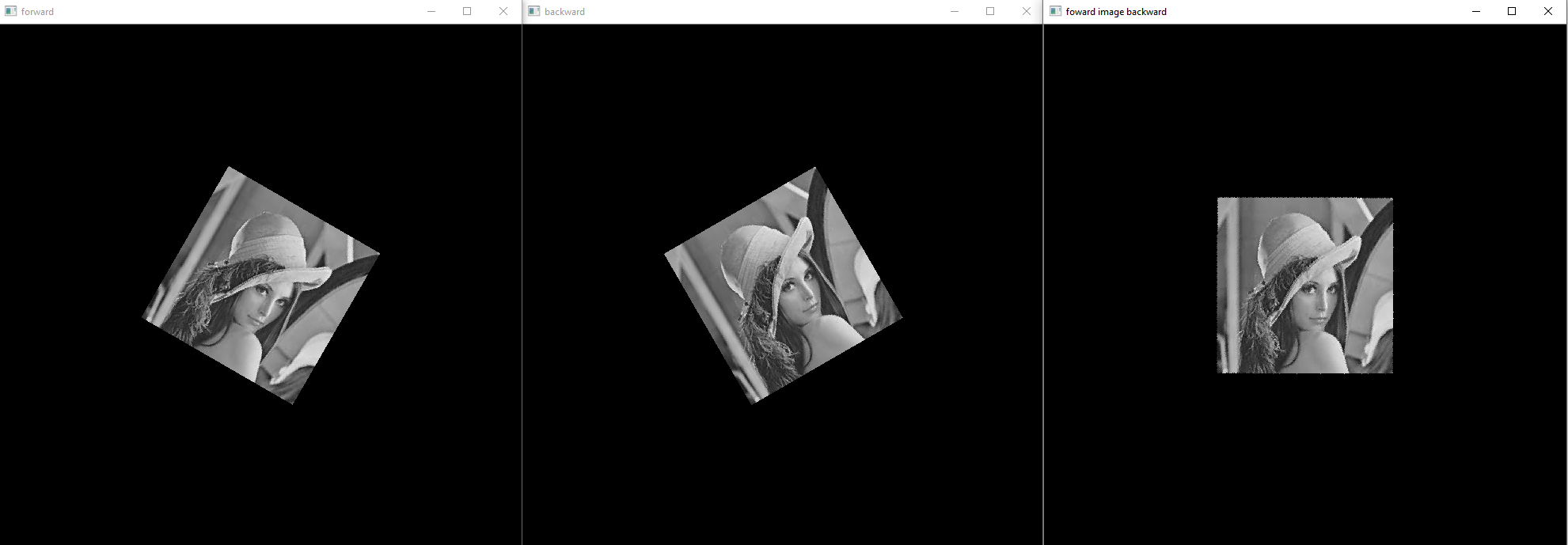


The backwardForward method is a little different than the others. This method first calculates the forward mapped coordinates, and uses backward mapped rotation on those coordinates. This means that there are two x and y values calculated. The once which will be assigned to the image backward mapped coordinates. The result of this is shown in the next section. The code for this method is shown below.



## Documentation of program

The output of the program with the values, angle = 30, x = 330 and y = 330, will be the image rotated around the center 30 degrees. In the image below the forward mapped picture is showed to the left the backward mapped in the middle and the backward forward mapped image to the right. The difference between the forward mapped and the backward mapped image, is the rotation direction and that the backward mapped is a tiny bit sharper, therefore if one had to choose between them they should choose backward mapping. The backward forward mapped image is the result of first using forward mapped rotation on an image and the applying backward mapped rotation. This means that the backward forward mapped image will have the same rotation as the original image, but it will be more pixelated, due to the loss of information when applying the rotations.



If the value of the x and y coordinates changed to 100 and 200, and the angle changed to 10, the output will look like the image below. The image will be rotated less than before, and the image would have rotated around a different point.

