cs512 Assignment 1: Report

Arun Rajagopalan
Graduate student, A20360689
Department of Computer Science
Illinois Institute of Technology

October 4, 2015

Abstract:

The problem or the task is to apply various manipulations to an image, using OpenCV library and our own implementations mimicking some OpenCV library functions. The aim is to get used to processing pictures with OpenCV and also learn how they are actually implemented along the way. While there are no specific algorithms to fiddle with in this assignment, several image manipulation functions and concepts were tried, with their results and performance being documented and analysed.

1 Problem statement

To process an image as per the user input. If a file name (with path, if the image is in some other folder) is specified as an argument while starting the program from command line, the file should be read and manipulated. If not, the program should capture images using the computer's camera continuously and keep on processing it. User can control the manipulation using keyboard keys and the track bars in the program GUI. Solving this problem, while familiarizing the solver with OpenCV, also induces understanding of core image processing concepts and to think on their software implementation details.

2 Proposed solution

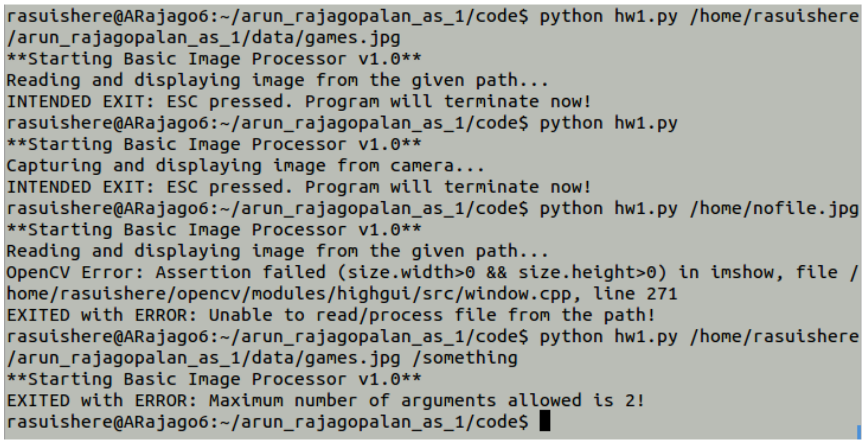
The solution starts with a check on the number of parameters being passed to the program while starting. Based on parameter count, either an image is read or several images are captured and processed continuously. The solution should listen to the keyboard always and if a valid input key is pressed, appropriate processing should be done. The program should never terminate unless the user intends and instructs the program to so. This is the basic outline of the solution. Deeper implementation details follow in the next paragraph.

3 Implementation details

To implement the solution, python 2.7.3 and OpenCV 3.0.0 was used. Main packages used include 'cv2' (for core image processing methods), 'numpy' (for image storing and array operation methods) and 'sys' (to get the arguments from command line). Source code (hw1.py) is placed in the 'code' folder under the parent directory 'arun_rajagopalan_as_1'. Images used to test the code are placed in 'data' folder under the same parent directory.

At first, as discussed above, the program checks for the number of arguments passed on startup. If it is 1, the program gains control of camera and starts capturing and displaying images for processing. If it is two, image on the given path is read and displayed. In case of invalid path or invalid number of arguments, an error message is thrown and program exits gracefully. A screen shot of the actual program behaviour on startup is shown in Illustration 1.

After the image is obtained, the program starts listening for keyboard inputs. On valid keyboard entries, a global flag variable for operation, 'op_flag' is set equal to the respective input. Main reason for using a flag is the constraint that this program should capture and process images continuously, when using camera. As long the flag is set to a respective key, say 'g', which is to convert image to grayscale, for example, the program would capture image, convert it to grayscale and show the black and white image



```
rasuishere@ARajago6:~/arun_rajagopalan_as_1/code$ python hw1.py /home/rasuishere
/arun_rajagopalan_as_1/data/games.jpg
**Starting Basic Image Processor v1.0**
Reading and displaying image from the given path...
INTENDED EXIT: ESC pressed. Program will terminate now!
rasuishere@ARajago6:~/arun_rajagopalan_as_1/code$ python hw1.py
**Starting Basic Image Processor v1.0**
Capturing and displaying image from camera...
INTENDED EXIT: ESC pressed. Program will terminate now!
rasuishere@ARajago6:~/arun_rajagopalan_as_1/code$ python hw1.py /home/nofile.jpg
**Starting Basic Image Processor v1.0**
Reading and displaying image from the given path...
OpenCV Error: Assertion failed (size.width>0 && size.height>0) in imshow, file /
home/rasuishere/opencv/modules/highgui/src/window.cpp, line 271
EXITED with ERROR: Unable to read/process file from the path!
rasuishere@ARajago6:~/arun_rajagopalan_as_1/code$ python hw1.py /home/rasuishere
/arun_rajagopalan_as_1/data/games.jpg /something
**Starting Basic Image Processor v1.0**
EXITED with ERROR: Maximum number of arguments allowed is 2!
rasuishere@ARajago6:~/arun_rajagopalan_as_1/code$
```

Illustration 1: Startup cases

to the user. This real time and sustained processing demands usage of a global flag. To keep the size of code base minimum and to re-use the available code, still image processing (the one that is read from the path) also uses the flag mechanism. Pressing 'h' with the program window active, will show the valid keyboard entries with their functions. User can end the program only by pressing ESC key. The close button on the GUI might close the program window for that instant, but since we are processing images continuously, the program will render the image for the next instant in another window of same name. This behaviour is also attributed to the video support constraint placed by the problem and the decision to re-use the existing code. Look of the GUI on startup with 'games.jpg' available in the 'data' folder and the 'h' key output are as below in Illustrations 2 and 3 respectively.

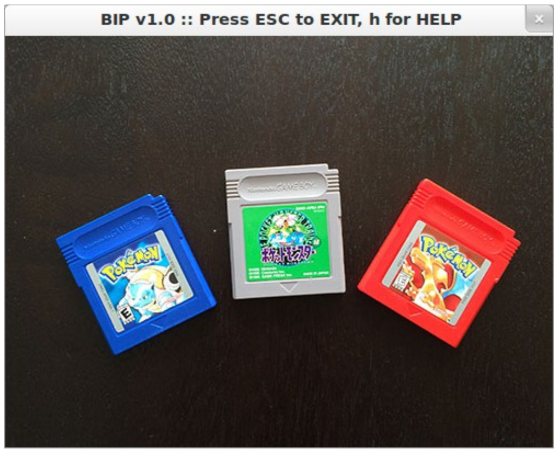
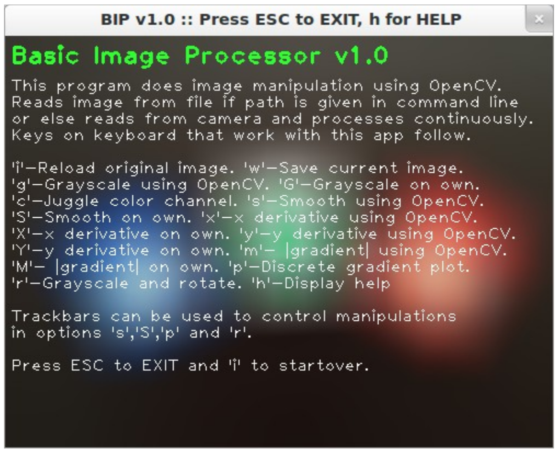


Illustration 2: GUI on startup



BIP v1.0 :: Press ESC to EXIT, h for HELP

Basic Image Processor v1.0

This program does image manipulation using OpenCV.
Reads image from file if path is given in command line
or else reads from camera and processes continuously.
Keys on keyboard that work with this app follow.

'i'–Reload original image. 'w'–Save current image.
'g'–Grayscale using OpenCV. 'G'–Grayscale on own.
'c'–Juggle color channel. 's'–Smooth using OpenCV.
'S'–Smooth on own. 'x'–x derivative using OpenCV.
'X'–x derivative on own. 'y'–y derivative using OpenCV.
'Y'–y derivative on own. 'm'– |gradient| using OpenCV.
'M'– |gradient| on own. 'p'–Discrete gradient plot.
'r'–Grayscale and rotate. 'h'–Display help

Trackbars can be used to control manipulations
in options 's','S','p' and 'r'.

Press ESC to EXIT and 'i' to startover.

Illustration 3: Output of the 'h' key

4 Results and discussion

Display, save and grayscale options

On start, program shows the captured image and pressing 'i' reloads this kind of unprocessed original image, cancelling the effects applied previously. 'w' key stores the current image in the current folder as 'out.jpg'. 'g' uses the OpenCV function to convert the image to grayscale while 'G' uses custom implementation for grayscaling, which reads every pixel of every color channel in the original image, adds them with weights as specified in the NTSC system to one channel and displays the resultant single channel image. The weights given to red, green and blue channel pixels are 0.299, 0.587 and 0.114 respectively. Green is given a higher weight because it is perceived more by humans, in any image. For every key press, terminal prints a corresponding message, so that it would be easy to keep track of what is happening currently and what has happened since the start of the program. Illustrations 4 and 5 show 'g'/'G' output and terminal output respectively.

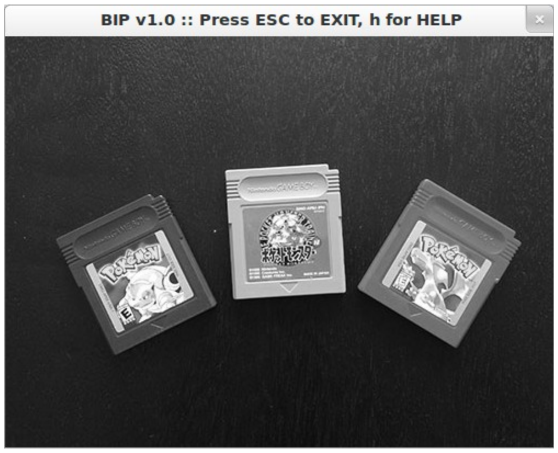
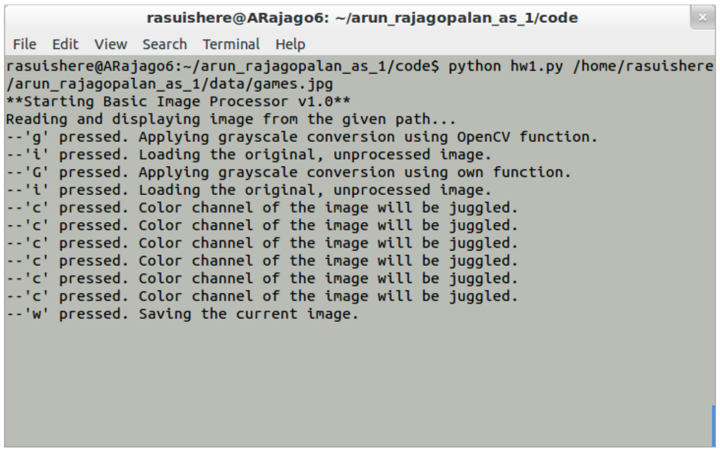


Illustration 4: Ouput of 'g' and 'G' keys


rasuishere@ARajago6: ~/arun_rajagopalan_as_1/code
File Edit View Search Terminal Help

```
rasuishere@ARajago6:~/arun_rajagopalan_as_1/code$ python hw1.py /home/rasuishere
/arun_rajagopalan_as_1/data/games.jpg
**Starting Basic Image Processor v1.0**
Reading and displaying image from the given path...
--'g' pressed. Applying grayscale conversion using OpenCV function.
--'i' pressed. Loading the original, unprocessed image.
--'G' pressed. Applying grayscale conversion using own function.
--'i' pressed. Loading the original, unprocessed image.
--'c' pressed. Color channel of the image will be juggled.
--'c' pressed. Color channel of the image will be juggled.
--'c' pressed. Color channel of the image will be juggled.
--'c' pressed. Color channel of the image will be juggled.
--'c' pressed. Color channel of the image will be juggled.
--'c' pressed. Color channel of the image will be juggled.
--'w' pressed. Saving the current image.
```

Illustration 5: Terminal keeps track of all keystrokes

Changing color channels

When 'c' is pressed, only one channel of a color image is shown. This is done slicing the image array and setting 0 for all channels except any one at a time. A function called 'juggle_color' takes care of this and makes sure to display alternate channel on every issue of 'c' key.

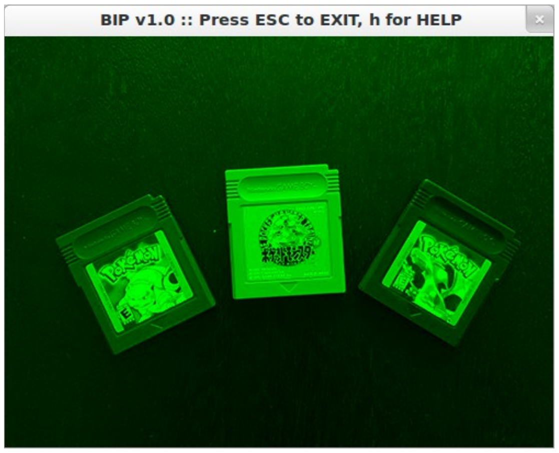


Illustration 6: Green channel of games.jpg

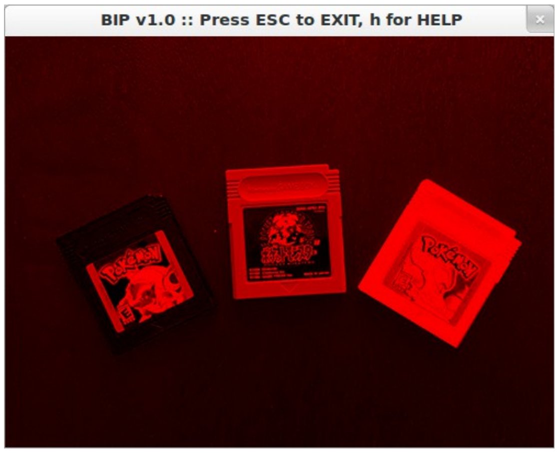
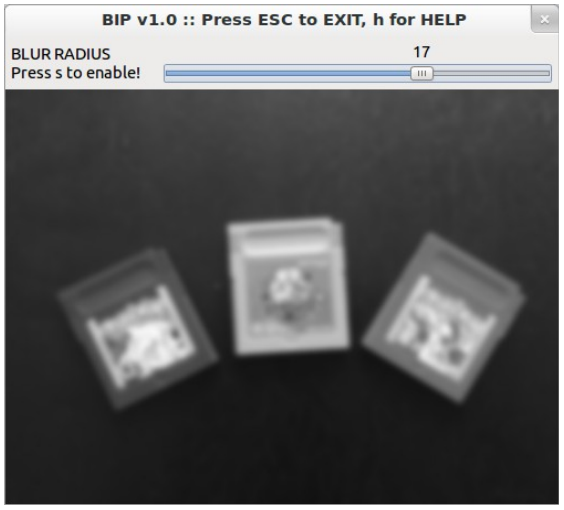


Illustration 7: Red channel of games.jpg

Smoothing using OpenCV and own implementations

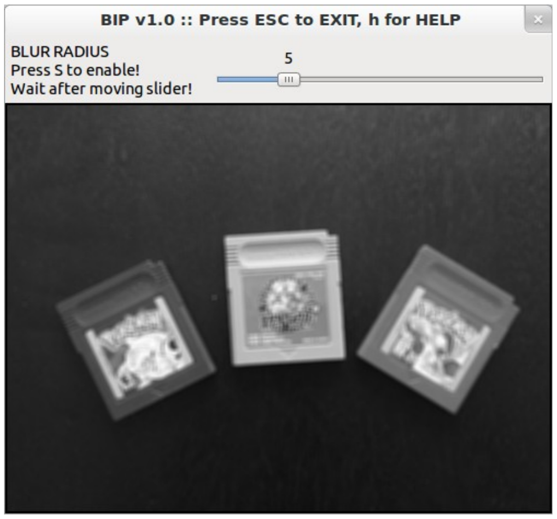
's' smooths the image using 'GaussianBlur' function available in OpenCV. The kernel size is obtained from the trackbar position. Since Gaussian kernel size needs to be positive and odd, the trackbar values are positive and if the trackbar position is an even number, 1 is subtracted from it. Hence, the trackbar positions of both 3 and 4 would produce a Gaussian kernel of size 3X3 only.

'S' key smooths the image using own function 'avg_filter' which applies an averaging filter to the image. Filter size is obtained from trackbar in a fashion similar to 's'. Edge pixels are not averaged and are ignored to get better performance and less code complexity. Outputs of 's' and 'S' are as below.


BIP v1.0 :: Press ESC to EXIT, h for HELP

BLUR RADIUS 17
Press s to enable!

Illustration 8: Output of 's' key


BIP v1.0 :: Press ESC to EXIT, h for HELP

BLUR RADIUS 5
Press S to enable!
Wait after moving slider!

Illustration 9: Output of 'S' key

Derivatives and magnitude using OpenCV and own implementations
'x' and 'y' keys display the x and y derivatives of the image, which are nothing but the edges of image in x and y directions, calculated using 'Sobel' OpenCV function. To normalize the output, OpenCV function 'convertScaleAbs' function is used. From the x and y derivatives, magnitude of the gradient is found out as follows.

Magnitude of gradient = √(x derivative^2 + y derivative^2)

We square the derivatives since they have both positive and negative values, add them and take square root to calculate magnitude. This magnitude, which is the combination of edges in x and y directions is calculated and displayed when 'm' is pressed.
'X', 'Y' and 'M' do the same operation except that they use own implementation (der_conv(axis)) and are quite slow. First the image is convolved with Sobel filters Sx and Sy which are given below and the resultant images are normalized.

Sx = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]] and Sy = [[-1, -2, -1], [0, 0, 0], [1, 2, 1]]

When M is pressed, the x derivative and y derivative calculated using own function are combined as per the above magnitude formula. The outputs of these keys are shown in the Illustrations numbered through 10 to 13.

Plot gradients
'p' key plots the directions of gradient vectors. First, magnitude of gradient is achieved like we did for 'm' key. OpenCV 'Sobel' function is particularly used here so that the implementation will be fast and it would be easy and quick to visualize vector directions, even in videos. Small circles are drawn on the magnitude image for every N pixels where N is specified by trackbar position. When the underlying pixel is not zero, a line will be drawn from every circle where line length is proportional to the magnitude of the pixel beneath and the angle is the arctangent of y derivative over x derivative.

angle = tan^(-1)(y derivative/x derivative)

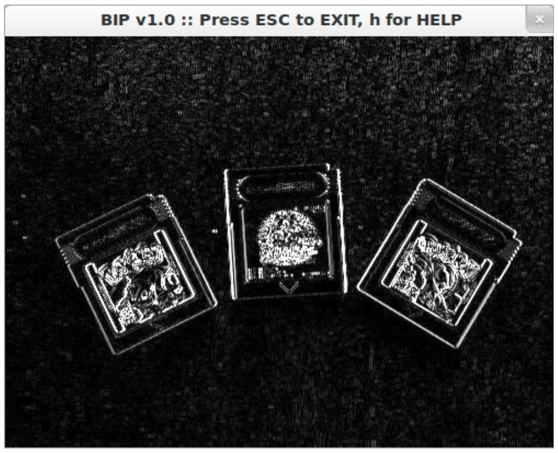


Illustration 10: Output of 'X' key using own function

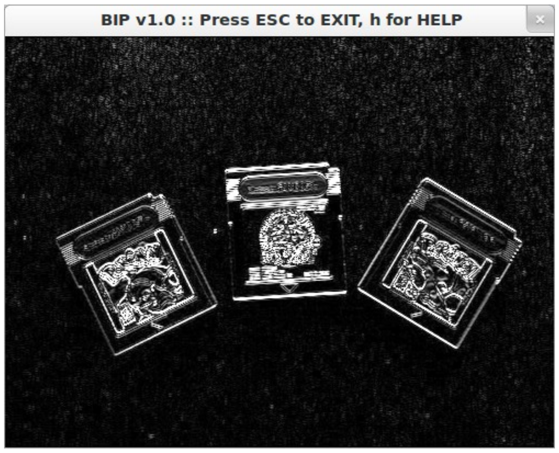


Illustration 11: Output of 'Y' key using own function

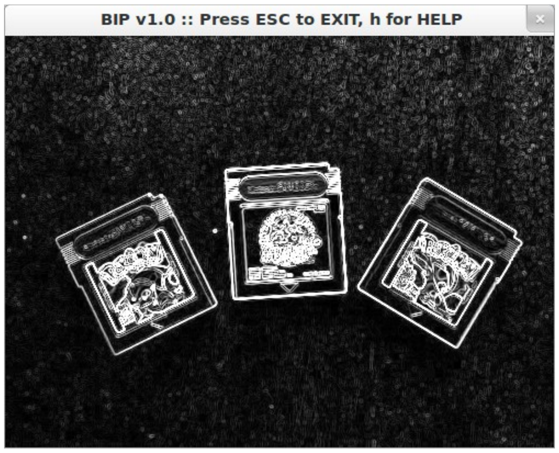


Illustration 12: Output of 'M' using own function

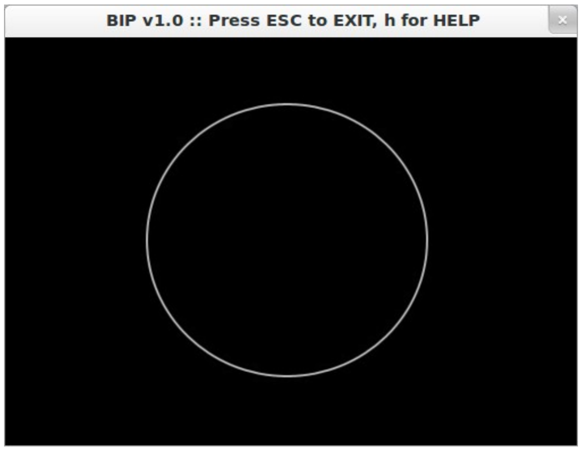


Illustration 13: Output of 'M' key on circle.jpg

Using angle, line length and centre of the circle, ending points of line vectors are calculated as below.

x co-ordinate of ending point = x co-ordinate of centre + line length * cos (angle+pi/180))
y co-ordinate of ending point = y co-ordinate of centre + line length * sin (angle+pi/180))

Thus, the line vector drawn from the centre of circle to the ending point displays the direction and strength of vector at that point. Illustrations 14 and 15 show the vector plot for circle.jpg and rectangle.jpg where the vectors are correctly plotted normal to the surface.

Rotating the image
'r' key rotates the image through an angle specified by the track bar position. First a rotation matrix to rotate the image about its centre through an angle Θ is achieved and using the rotation matrix, the current image undergoes an affine transformation resulting in an image rotated through angle Θ. Illustration 16 shows the output of image rotation function.

General rotation matrix, R = [[cos Θ, -sin Θ],[sin Θ, cos Θ]]

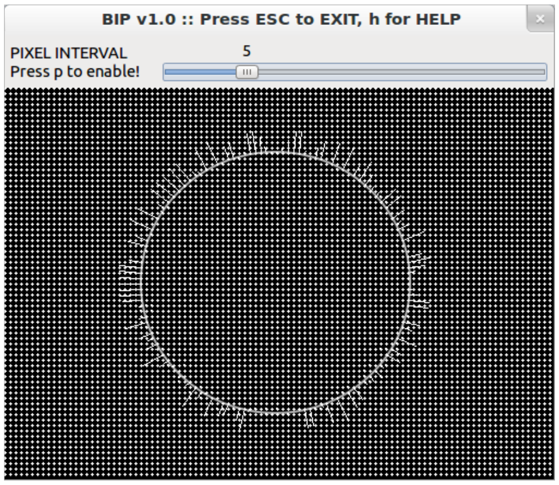


Illustration 14: Output of 'p' on circle.jpg showing normal vectors

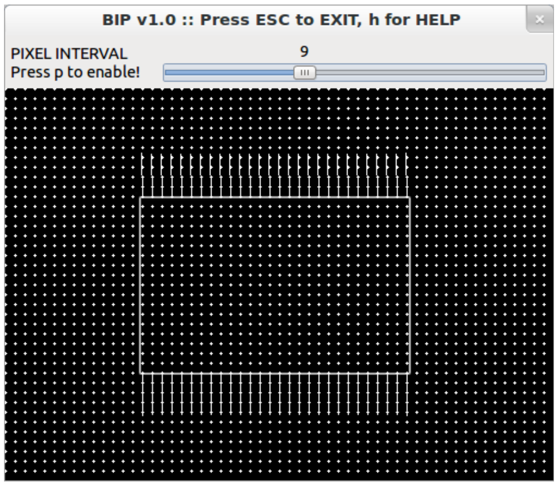


Illustration 15: Output of 'p' on rectangle.jpg showing normal vectors

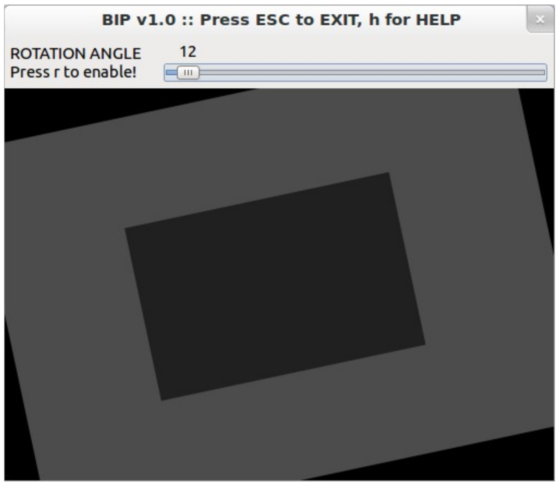


Illustration 16: Output of 'r' key

Displaying help

As discussed above, pressing 'h' key displays a help note with the keys supported and their functional descriptions. Illustration 3 shows the 'h' output screen.

5 References

https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_tutorials.html
http://docs.opencv.org/
https://docs.python.org/3/tutorial/
https://docs.python.org/2/
http://stackoverflow.com/questions/tagged/opencv
http://stackoverflow.com/questions/tagged/python-2.7