cs512 Assignment 2: Report

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

October 21, 2015

Abstract:
The problem or the task is to detect corners in an image, using our own implementation of the Harris corner detection algorithm, alongside using other OpenCV library functions as necessary. The aim is to understand the algorithm and implement it in software. Many of the crucial parameters for the algorithm and their effect on detection are also to be studied, documented and analyzed.

1 Problem statement

To find corners in an image using own implementation of Harris corner detection algorithm. If the file names (with path, if the images are in some other folder) are specified as arguments while starting the program from command line, the files should be read and corners need to be detected. If not, the program should capture images using the computer's camera continuously and keep on detecting corners. User can control several parameters of the Harris corner detection using the track bars in the program GUI. Also, by using keyboard keys, identical corner matching and drawing shapes around corners can be achieved. Solving this problem, helps to understand the Harris corner detection concept and familiarizes us with the software implementation details of the same.

2 Proposed solution

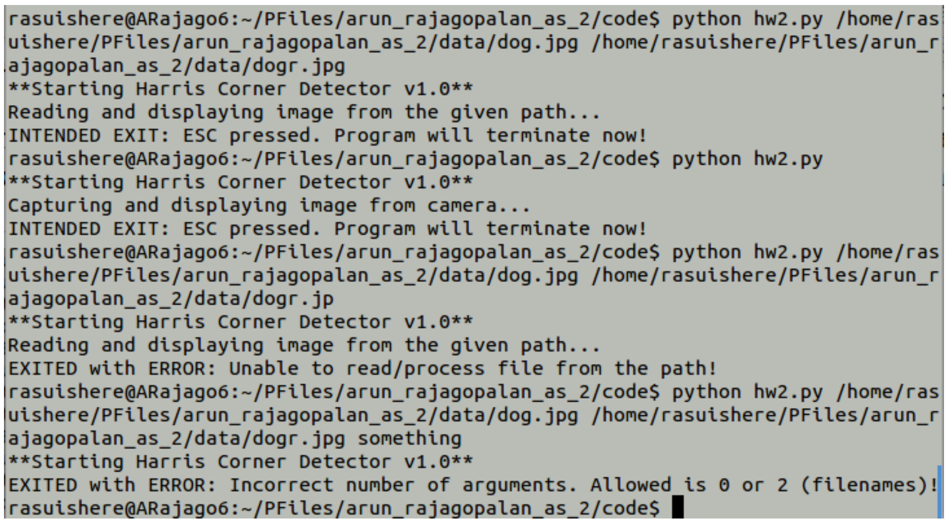
The solution starts with a check on the number of parameters being passed to the program while starting. Based on parameter count, either images are read from disk or several images are captured and processed for corners continuously. The solution should listen to the keyboard always and if a valid input key is pressed (say H or M), appropriate corner detection and processing should be done. The program should never terminate unless the user intends and instructs the program to do so. While providing for corner detection, the program should also allow users to experiment with parameters, match identical corners and draw shapes around corners in the given images. 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 (hw2.py) is placed in the 'code' folder under the parent directory 'arun_rajagopalan_as_2'. 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 three, images on the given path(s) are 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 images are 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 as the flag is set to a respective key, say 'H', which is to detect and highlight every corner using Harris algorithm, the program would continue detecting and marking corners in all the read/captured image frames, till the reception of another valid keyboard interrupt.



```
rasuishere@ARajago6:~/PFiles/arun_rajagopalan_as_2/code$ python hw2.py /home/ras
uishere/PFiles/arun_rajagopalan_as_2/data/dog.jpg /home/rasuishere/PFiles/arun_r
ajagopalan_as_2/data/dogr.jpg
**Starting Harris Corner Detector v1.0**
Reading and displaying image from the given path...
INTENDED EXIT: ESC pressed. Program will terminate now!
rasuishere@ARajago6:~/PFiles/arun_rajagopalan_as_2/code$ python hw2.py
**Starting Harris Corner Detector v1.0**
Capturing and displaying image from camera...
INTENDED EXIT: ESC pressed. Program will terminate now!
rasuishere@ARajago6:~/PFiles/arun_rajagopalan_as_2/code$ python hw2.py /home/ras
uishere/PFiles/arun_rajagopalan_as_2/data/dog.jpg /home/rasuishere/PFiles/arun_r
ajagopalan_as_2/data/dogr.jp
**Starting Harris Corner Detector v1.0**
Reading and displaying image from the given path...
EXITED with ERROR: Unable to read/process file from the path!
rasuishere@ARajago6:~/PFiles/arun_rajagopalan_as_2/code$ python hw2.py /home/ras
uishere/PFiles/arun_rajagopalan_as_2/data/dog.jpg /home/rasuishere/PFiles/arun_r
ajagopalan_as_2/data/dogr.jpg something
**Starting Harris Corner Detector v1.0**
EXITED with ERROR: Incorrect number of arguments. Allowed is 0 or 2 (filenames)!
rasuishere@ARajago6:~/PFiles/arun_rajagopalan_as_2/code$
```

Illustration 1: Startup Cases

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 (when images are read from the path(s)) 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 detecting corners in 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 'dog.jpg' and 'dogr.jpg' available in the 'data' folder and the 'h' key output are as below in Illustrations 2 and 3 respectively.

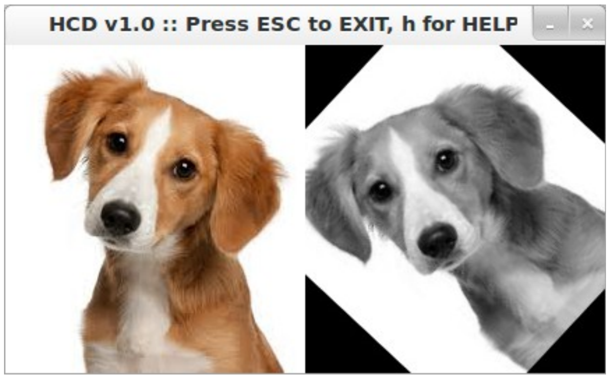
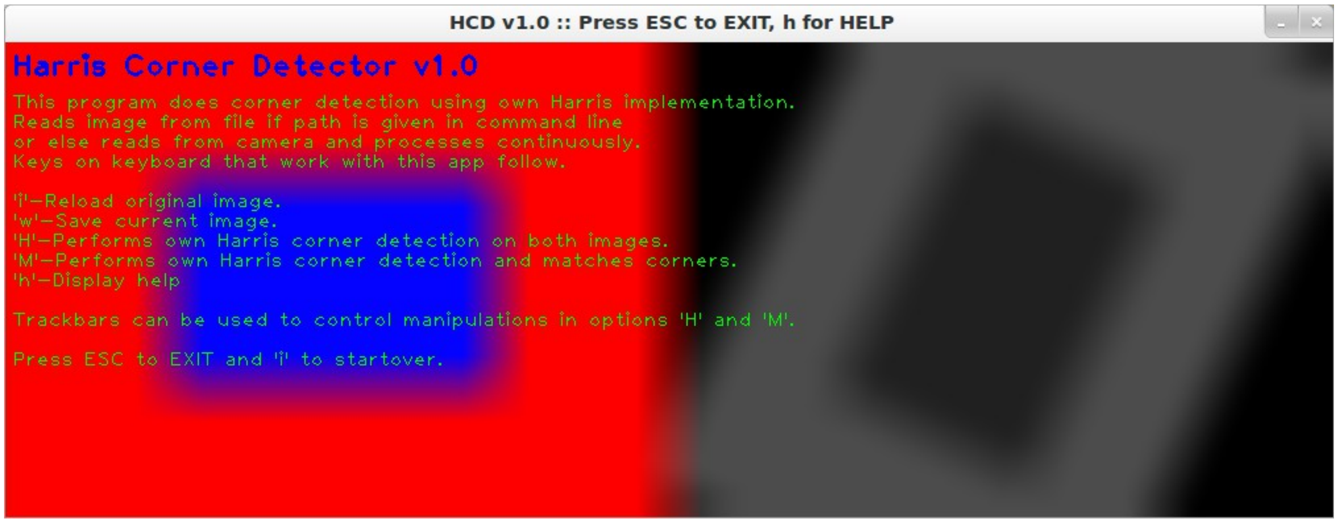


Illustration 2: GUI on startup

Corner detection, matching and drawing options

On start, program displays the read/captured images and pressing 'i' reloads this kind of unprocessed original image, cancelling the changes applied previously. 'w' key stores the current image in the current folder as 'out.jpg'.



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

Harris Corner Detector v1.0
This program does corner detection using own Harris implementation.
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.
'H'-Performs own Harris corner detection on both images.
'M'-Performs own Harris corner detection and matches corners.
'h'-Display help

Trackbars can be used to control manipulations in options 'H' and 'M'.

Press ESC to EXIT and 'i' to startover.
```

Illustration 3: Output of h key

'H' is the main key, which instructs the program to perform corner detection with own implementation of Harris corner algorithm and mark the detected corners. A brief description of the concept of Harris corner detection is as follows.

To find corners, Harris algorithm calculates the change in intensity for a displacement of (u,v) in all directions. The approach of Harris algorithm can be expressed mathematically as below.

$$E(u,v) = \Sigma_{x,y}\, w(x,y)\, [I(x+u,y+v)-I(x,y)]^2$$

Here, w is the windowing function (simplest case being w=1), I(x+u, y+v) is the shifted intensity and I(x,y) is the original intensity. Window function can just be a rectangular window or a Gaussian window that imparts some weights to the pixels lying beneath.

The difference between shifted intensity and original intensity will be almost 0 for nearly constant patches/windows of an image. For windows with corners, this value will be large. So, we find the windows where E(u,v) is large.

For maximizing E(u,v), we use the first order approximation of Taylor series, rewritten in matrix form and the final E(u,v) equation, after simplification, would look as below.

$$E(u,v) \cong [u,v]_{1X2}\, C\, [[u],[v]]_{2X1}$$

C is 2X2 matrix calculated from image derivatives in x and y directions (Ix is the derivative in x direction and Iy is the derivative in y direction).

$$C = \Sigma_{x,y}\, w(x,y)\, [[Ix^2,IxIy],[IxIy,Iy^2]]$$

Calculating the eigen values (λ_1 and λ_2) of this correlation matrix (C) is compute intensive. Instead, determinant and trace of the C matrix is used in Harris algorithm to find cornerness measure (Mc).

$$\text{Cornerness measure (Mc)} = \text{determinant}(C) + (-K*\text{trace}^2(C))$$

Here, determinant(C) is equivalent to the product and trace(C) is equivalent to the sum of the eigen values of C matrix.

K is the Harris detector free parameter. If K = 0, Harris algorithm acts as a corner detector and when K is 0.5, it acts an edge detector. Harris suggested the use of 0.04 for corner detection, which is the default value in this own implementation.

When the measure Mc is less than 0, the window is edge and when it is small, the window is flat. Corners will be present in windows where Mc is large.

So, wherever Mc is greater than a threshold, we can conclude that atleast one corner is present in that window. Localizing or pinpointing the corner in the detected window is given by the below formula, where the minimum P in a window is the corner.

$$P = C^{-1} \Sigma_i \ [Ixi,Iyi]*[Ixi,Iyi]^T [x,y]$$

Implementation detail of the own implementation is same as the above.

1. First the image gradients are calculated.

2. Correlation matrix and in turn, cornerness measure (Mc) are then calculated for every window of user-specified size. (Gaussian window is used in own implementation).

3. In the windows where Mc is large, corner localization is done. In own implementation, localization is done using an averaging filter and cv2.erode function for better performance.

4. Finally the corners are marked in colour over original grayscale image.

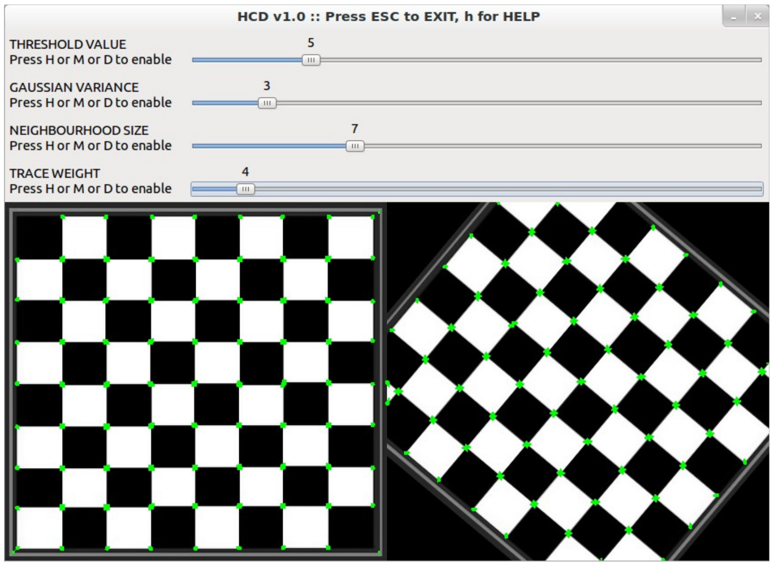
Thus, the critical parameters for the Harris algorithm are Threshold value for Mc, K value, neighbourhood size and variance of the Gaussian. Using the respective trackbars in the GUI, users can observe the effect of changes in these parameters on corner detection.

'M' key performs corner detection using own implementation and then matches corners in the two given images. Identical corners in two images are marked by a line drawn between them. Different coloured lines are used to differentiate between several identical corner pairs.
'D' key draws shapes around the detected corners in Harris algorithm output for easy visualization and validation of the output. Both 'M' and 'D' keys make use of the ord library functions available in OpenCV.

4 Results and discussion

Corner detection:

The outputs of 'H' key for different combinations of parameters are as follows. The images used are chess.jpg and chessr.jpg, available in the data folder.



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

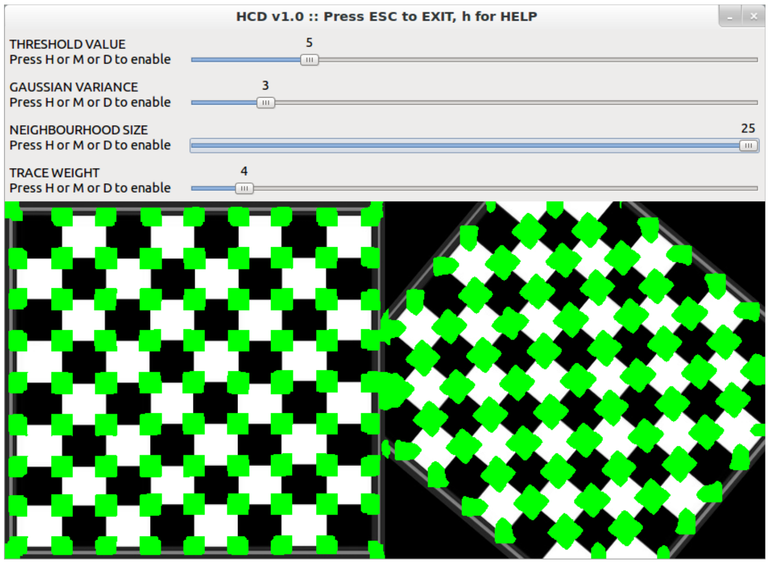
THRESHOLD VALUE 5
Press H or M or D to enable

GAUSSIAN VARIANCE 3
Press H or M or D to enable

NEIGHBOURHOOD SIZE 7
Press H or M or D to enable

TRACE WEIGHT 4
Press H or M or D to enable

Illustration 4: Output of H key - Desirable values



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

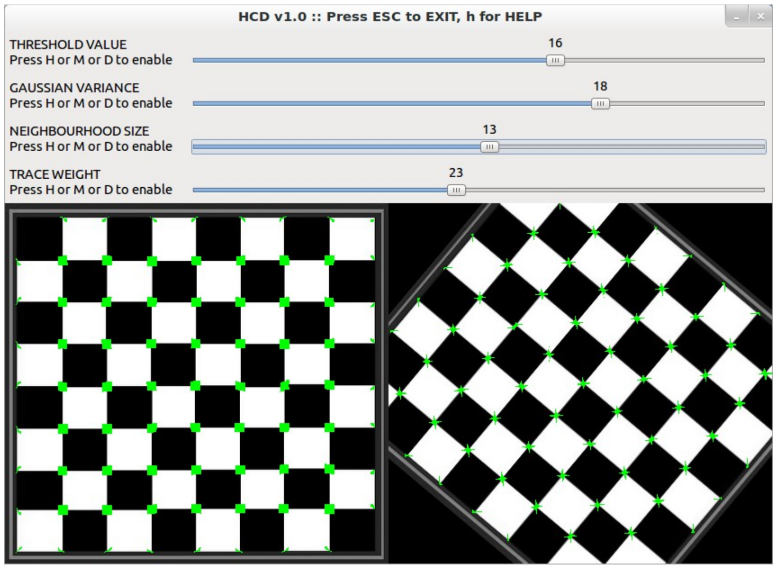
THRESHOLD VALUE 5
Press H or M or D to enable

GAUSSIAN VARIANCE 3
Press H or M or D to enable

NEIGHBOURHOOD SIZE 25
Press H or M or D to enable

TRACE WEIGHT 4
Press H or M or D to enable

Illustration 5: Output of H key - Large neighbourhood value



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

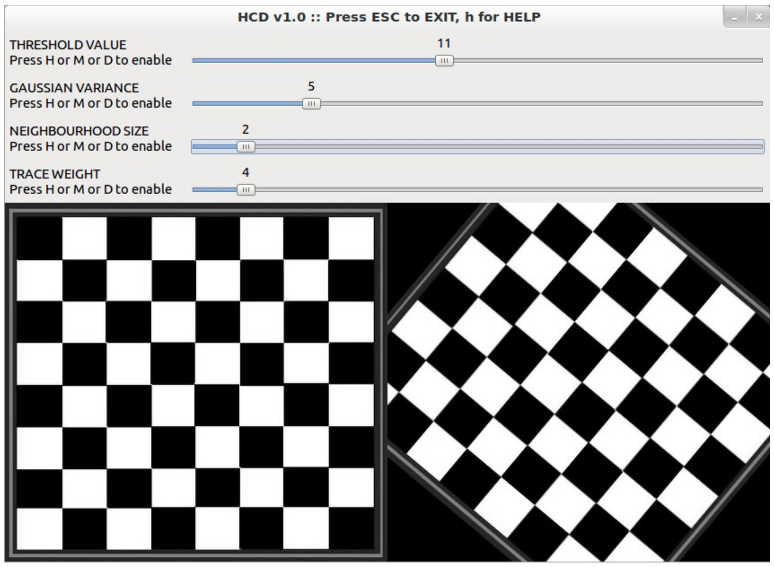
THRESHOLD VALUE 16
Press H or M or D to enable

GAUSSIAN VARIANCE 18
Press H or M or D to enable

NEIGHBOURHOOD SIZE 13
Press H or M or D to enable

TRACE WEIGHT 23
Press H or M or D to enable

Illustration 6: Output of H key - High variance, threshold & trace wt


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

THRESHOLD VALUE 11
Press H or M or D to enable

GAUSSIAN VARIANCE 5
Press H or M or D to enable

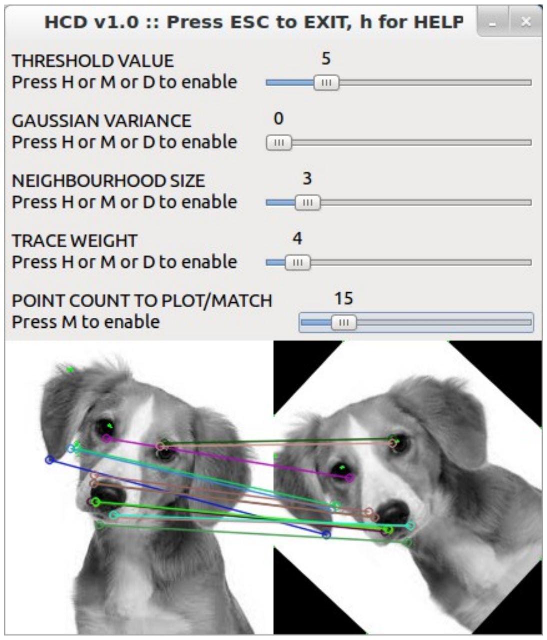
NEIGHBOURHOOD SIZE 2
Press H or M or D to enable

TRACE WEIGHT 4
Press H or M or D to enable

Illustration 7: Output of H key - Very low neighbourhood size

Thus, a large neighbourhood value gives rise to spread corner windows and our corner detection accuracy becomes less. When the neighbourhood value is too low, no corner gets detected. Also when the threshold for cornerness measure and variance of the Gaussian window are increased, number of detected corners becomes less. When trace value is increased, nearer to its given maximum value in the GUI, edges are also getting detected. Thus we desire optimum values for all of the given parameters for efficient corner detection. The default value in this own implementation for parameter k is 0.04, threshold is 0.01*maximum of cornerness response matrix, variance is 0 and neighbourhood size is 3.

Corner mapping and marking:


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

THRESHOLD VALUE 5
Press H or M or D to enable

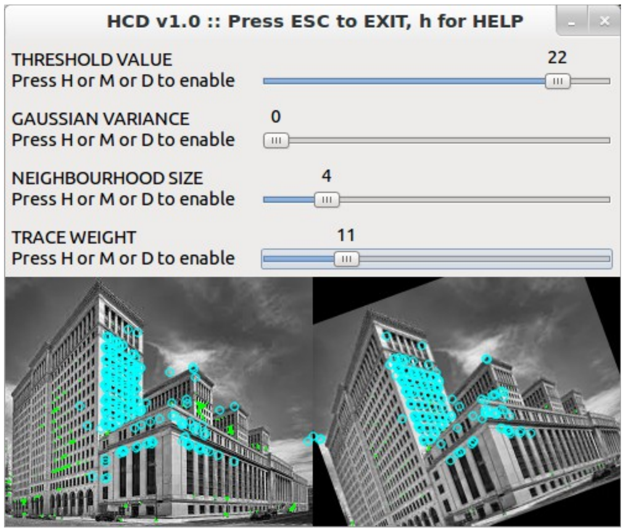
GAUSSIAN VARIANCE 0
Press H or M or D to enable

NEIGHBOURHOOD SIZE 3
Press H or M or D to enable

TRACE WEIGHT 4
Press H or M or D to enable

POINT COUNT TO PLOT/MATCH 15
Press M to enable

Illustration 8: Output of M key - Matching corners


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

THRESHOLD VALUE 22
Press H or M or D to enable

GAUSSIAN VARIANCE 0
Press H or M or D to enable

NEIGHBOURHOOD SIZE 4
Press H or M or D to enable

TRACE WEIGHT 11
Press H or M or D to enable

Illustration 9: Output of D key - Drawing shapes around corners

As seen above, M key draws matches between identical corners and D key draws shapes around prominent corners. M key also gives one more parameter to vary, the number of corners/points that need to be matched. Strong corners are given higher preference if the point count to plot is less than the total number of corners. User can change Harris detection parameters while using M and D keys as well. Observations made on change of parameters while using D and M keys are same as those made while using H key. The images used for the above testing are dog.jpg, dogr.jpg, build,jpg and buildr.jpg in that order.

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/
http://stackoverflow.com/questions/tagged/python-2.7
http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html
http://www.cse.psu.edu/~rtc12/CSE486/lecture06.pdf
http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html