Functions:

------------------------------------------------------------------------------------------------------------------------

**SIFT\_Rect**

* **Command**: SIFT\_Rect(img1, img2, qimg, thresh)
* **Overview**:

This function finds out a way to relate two images,one image containing(*query image*) that has to be detected and the scene where this object is object is present(*train image*). To accomplish this this algo divides the two images into interesting points and tries to match all the interesting points of query image to train image. Once this match is done, it works to extract the image from the scene by using mathematical methods.

* **Prerequisites**:

|  |  |
| --- | --- |
| Img1 - | query image |
| Img2 - | train image |
| Qimg - | name of train image (string) |
| Thresh - | [0 , 1] {good range - [0.45, 0.80 ] } { as thresh reduces, accuracy of matching  interesting point increases } |

* **Returns** :

Saves <name\_of\_train\_image>\_result.jpg in the folder containing main.py. It returns warped, which is grayscale image of detected object in the scene

------------------------------------------------------------------------------------------------------------------------

**SIFT\_Rect\_W**

* **Command:** SIFT\_Rect\_W(img1, img2, qimg, thresh)
* **Overview**:

This functions works as same as above mentioned function. Reason for writing it separately is that this returns coordinates of matches points corresponding to the query image.

* **Prerequisites**:

|  |  |
| --- | --- |
| Img1 - | query image |
| Img2 - | train image |
| Qimg - | name of train image (string) |
| Thresh - | [0 , 1] {good range - [0.45, 0.80 ] } { as thresh reduces, accuracy of matching  interesting point increases } |

* **Returns : good\_pts**

coordinates of matches points corresponding to the query image.

------------------------------------------------------------------------------------------------------------------------

**Gauss\_grad**

* **Command:** gauss\_grad (image, name)
* **Overview:**

This function checks the quality of feature point, whether it is a good feature point or a bad feature point

* **Prerequisites**:

|  |  |
| --- | --- |
| image | Image which is checked |
| name | Name of image |

* **Returns :**

An image, which is grayscale, and only those coordinates are highlighted which have good feature points location

------------------------------------------------------------------------------------------------------------------------

**Normalize\_img**

* **Command :** normalize\_img(img, drange=255):
* **Overview**: This function normalises the value of the image in the mentioned range(o, *drange*) where drange is the upper limit. Lower limit is fixed to 0.

For example if the lowest value in an image is 5 and highest is 100, this will map 5 to 0 and 100 to 255 and all the intensities value in between accordingly

* **Prerequisites**:

|  |  |
| --- | --- |
| img | Image to be normalised |
| drange | Range in (0 to drange ) default is 255 |

* **Returns :** normalised image whose intensity varies from o to drange

------------------------------------------------------------------------------------------------------------------------

**Four\_point\_transform**

* **Command :** def four\_point\_transform(image, pts)
* **Overview** :

This code blocks extract the quadrilateral from an image and transform into a rectangle and returns the coordinates of the rectangle. Order points takes input the *pts* passed to it and its work is to just set a particular order which the *four\_point\_transform* needs to extract image from the scene

* **Prerequisites**:

|  |  |
| --- | --- |
| image | The image from which you want to extract rectangle |
| pts | An array of 4 points in the format (x,y) |

* **Returns :**

A target image (typically called as warped image) extracted from the wild scene

------------------------------------------------------------------------------------------------------------------------

**Find\_defect**

* **Command :** find\_defect(img1 , img2, img3, qimg, thresh)
* **Overview :**

This code block runs into four steps:

\* Step 1: It uses Sift\_rect\_w to extract all the good\_points corresponding to good wild image

\* Step 2: It uses Sift\_rect\_w to extract all the good\_points corresponding to being tested wild image

\* Step 3: It now looks that how many good points from step 2 is missing which are present in step 1.

\* Step 4: Finally using the coordinates of those points it draws rectangle across their boundary.

* **Prerequisites**:

|  |  |
| --- | --- |
| img1 | Digital reference image (query image) |
| img2 | Testing wild image (train image 1) |
| img3 | Good wild image (train image 2) |
| qimg | Name of testing image(train image 1) |
| thresh | [0 , 1] {good range - [0.45, 0.80 ] } { as thresh reduces, accuracy of matching  interesting point increases } |

* **Returns:** *Nothing to return*

Saves image with defects marked in form of a rectangle in the folder as same as main.py with <image\_name>\_defects.jpg