* How to run
  + The application is designed to run each of the three matching methods at once. To execute it on a specific set of images modify the variables specified in the “Modifiable variables” section. Then simply execute the program and the results will be presented to the console.
* Implementation
  + Overview
    1. Loads the query image from the file system
    2. Loads all of the database images into an array
    3. Preforms template matching on each image in the database
       - Uses TM\_CCOEFF\_NORMED by default as the difference algorithm
       - Computes the best matching location and scores it between 0 and 100%
    4. Preforms Color Histogram comparison on each image in the database
       - Converts the query image to HSV and calculates its color histogram
       - Loops through each element of the database and converts them to HSV
       - Calculates the histogram for each converted element in the database
       - Calculates the percent difference between the query histogram and the database histogram.
    5. Preforms SIFT detection on each image in the database
       - Preforms SIFT on the query image to find the keypoints
       - Runs SIFT on the database images then utilizes FLANN to calculate the matches between the query image keypoints and the database images
       - Each keypoint returned is then filtered based upon the total distance of their feature vector
       - Based upon the number of valid keypoints a percentage is calculated and returned for each image
    6. Processes each matching algorithm to determine the scores per algorithm and the overall score between all three of the matching algorithm.
  + Database loading

In order for my project to load the database images I utilized a method of reading in all files that follow a specific standard. I accomplished this using the python glob function. This function allows the user to specify a format such as “ukbench\*.jpg” where “\*” allows the function to fill in data such as numbers. This will result in a list of all files that are of type “.jpg” and begin with “ukbench” in alphabetical order. The user can enter this pattern in the “TEST\_FILE\_PATERN” variable. Along with allowing the user to specify the pattern for the image files the user is required to enter the query image in the “QUERY\_IMG\_FILE\_NAME” and correct matching images into the “CORECT\_MATCHES” variable. Finally I allowed the user to specify the storage location of the database in the “DATABASE\_LOC” variable. Once all of this is read in the images to test against are stored into an array of pairs that stores the file name and the image data.

* + Detection Algorithms
    1. Template Matching

The template matching algorithm utilizes the “cv2.TM\_CCOEFF\_NORMED” by default to compare all of the images in the database against the query image. Although this is the default matching method the user can change the method used by modify the “TEMPLATE\_MATCHING\_METHOD” variable to use any of the opencv normalized matching algorithms. The score returned by this is then multiplied by 100 to convert it to a value between 0 and 100.

* + 1. Color Histogram

The color histogram algorithm uses standard histogram comparison to compute the percent match between the query image and each image in the image list. To accomplish this each image is converted from BGR format to HSV format. The color histogram of each image is then generated and the “cv2.compareHist” function is used to get the difference between the two images. The retuned value is then converted to a 0 to 100 scale.

* + 1. SIFT

The SIFT algorithm utilizes the built in opencv SIFT algorithm to compute the keypoints in the query image and in each image from the database. These matching keypoints are then filtered using the standard Lowes cost ratio which compares the radius of the matches to see if they are valid or not. The good matches are then saved off and used to calculate a percent match on a scale from 0 to 100.

* + Scoring
* Performance
  + Template Matching

The template matching algorithm was constantly the worst preforming method of matching images in the database to the query image. It was consistent with finding the query image in the database but rarely found any of the other correct images. This is due to its simplistic method of shifting the query image around the database images and computer a score based upon the best match it found.

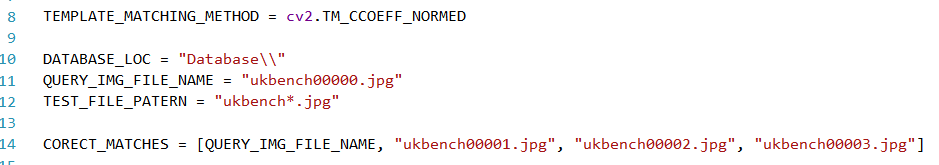
* + Color Histogram

The color histogram algorithm in general was the fastest method to find matches in terms of processing time. It was also able to consistently find one or more images in the database that matched the query image. This is only true however for images where the background does not change significantly and the general lighting of the object is the same. In addition the objects color needs to be consistent between images.

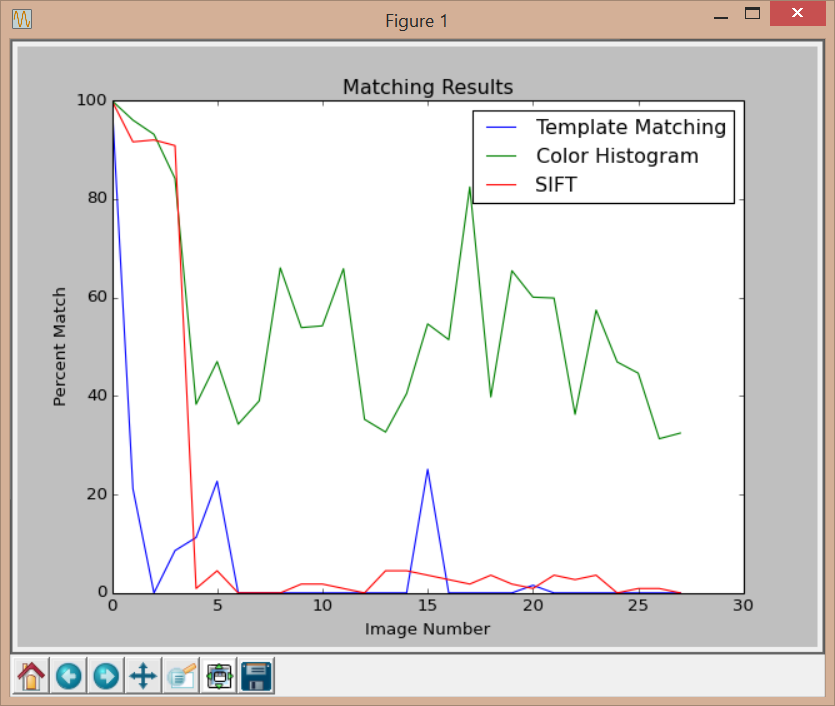
* + SIFT

The SIFT algorithm was the slowest method to perform object recognition, often taking several minutes to complete on a medium database. Although it was the slowest it almost always returned each of the 4 correct images. This is because it utilizes key points on the object to perform matching. As long as the background is not cluttered with other objects SIFT will consistently return correct images.

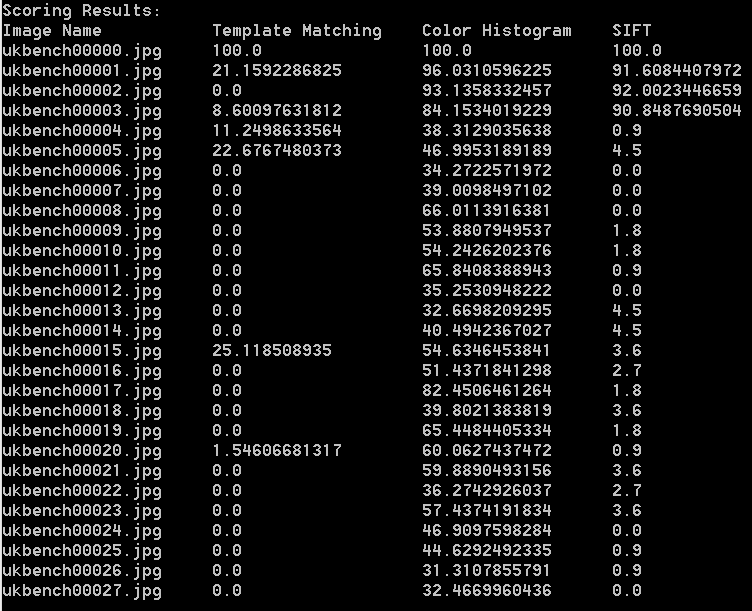
* Modifiable variables



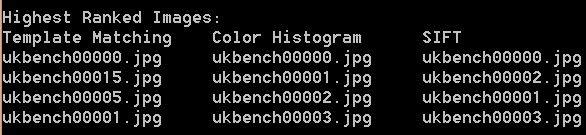
* + TEMPLATE\_MATCHING\_METHOD
    - The method used in the template matching section. This should only be one of the normalized template matching methods
  + DATABASE\_LOC
    - The path to the database folder
  + QUERY\_IMG\_FILE\_NAME
    - The file name of the image to use for comparisons
  + TEST\_FILE\_PATERN
    - The pattern that the input files in the database follow
* Output
  + Line plot
    - Displays the match percentage results of each image for each matching method.
    - The X-axis is the image number and the Y-axis is the match percentage
  + Command line
    - Scoring Results
      * Displays the match percentage for each image and each matching method.
    - Highest Ranked Images
      * Displays the 4 highest ranked images for each matching method.
    - Final Scores Individual
      * Displays the individual scores for each matching method.
    - Final overall scores
      * Returns the 4 highest average results of each method combined and the final result score.
* Sample Results
  + Uses images “ukbench00000.jpg” to “ukbench00027.jpg” with “ukbench00000.jpg” as the query image.
  + Output
    - The percentage of each images match to the query image



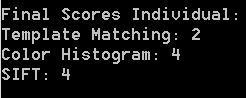
* + - The raw score of each image with each of the 3 algorithms



* The four highest rated images for each of the 3 matching methods



* The calculated score for each of the matching methods



* The top 4 images based upon the average of the 3 algorithms and the average score

