Installation

I. Minimum Requirements

Operating System: Ubuntu 16.04 (or higher)

Python: Python 2.7.x

Installations: OpenCV, Cassandra

II. OpenCV and Required Libraries

(Note: OpenCV must be installed alongside OpenCV_contrib to use SIFT)

```
1. sudo apt-get update
2. sudo apt-get upgrade
3. sudo apt-get install build-essential
4. sudo apt-get install cmake git libgtk2.0-dev pkg-config
   libavcodec-dev libavformat-dev libswscale-dev
5. sudo apt-get install python-dev python-numpy libtbb2
   libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-
   dev libdc1394-22-dev
6. sudo apt-get install qt5-default
7. cd ~/path/to/install/directory
8. git clone https://github.com/opencv/opencv
9. git clone https://github.com/opencv/opencv_contrib
10.  mkdir opencv/build
11.  cd opencv/build
12.  cmake -D CMAKE_BUILD_TYPE=RELEASE -D
   CMAKE_INSTALL_PREFIX=/usr/local -D
   OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules -D
   WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON
   -D INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D
   BUILD_EXAMPLES=OFF -D WITH_QT=ON -D WITH_OPENGL=ON -D
   ENABLE_FAST_MATH=1 ..
```

III. Cassandra Database

```
1. sudo add-apt-repository ppa:webupd8team/java
2. sudo apt-get update
3. sudo apt-get -y install oracle-java8-installer
4. echo "deb-src http://www.apache.org/dist/cassandra/debian
   37x main" | sudo tee -a
   /etc/apt/sources.list.d/cassandra.sources.list
```

5. gpg --keyserver pgp.mit.edu --recv-keys F758CE318D77295D
6. gpg --export --armor F758CE318D77295D | sudo apt-key add –
7. gpg --keyserver pgp.mit.edu --recv-keys 2B5C1B00
8. gpg --export --armor 2B5C1B00 | sudo apt-key add –
9. gpg --keyserver pgp.mit.edu --recv-keys 0353B12C
10. gpg --export --armor 0353B12C | sudo apt-key add –
11. sudo apt-get install Cassandra

Code Summary

I. BOVW_Config.py

Used to set up the directories and customize options for running the code. In order to change configuration, each of the variables below can be changed.

1. *GEN_MODEL* – Set to TRUE when a dataset needs to be downloaded; FALSE otherwise.
2. *TEST_IMAGES* – Set to TRUE when downloading query images for testing; FALSE otherwise.
3. *QUIET* – When FALSE the code will run without displaying anything to the terminal window. When TRUE the code will print its current position in the terminal window.
4. *K* – The number of clusters to use in the k-means algorithm when clustering the dataset's image features.
5. *LIMIT* – The maximum number of relevant images that will be returned from a query image.
6. *FEATURES* – Accepts specific values for each implemented algorithm for image feature extraction.
    a. sift - uses the standard SIFT algorithm
    b. rootsift – uses the RootSIFT algorithm, an improvement on SIFT by applying the Hellinger kernel and L1-normalization.
7. *CLUSTERING_ALGORITHM* – Accepts specific values for each implemented algorithm to cluster image features.
    a. kmeans – Utilizes the k-means algorithm with the specified value of K.
8. *K_MEANS_ITERATION* – The number of times to run k-means before determining the final clusters.
9. *DISTANCE_LIMIT* – The maximum real-world distance in meters for a returned image to be considered relevant to a query image.
10. *FOV_LIMIT* – The maximum real world field-of-view angle for a returned image to be considered relevant (0-360).
11. *CORPUS_PATH* – The directory where the dataset GPX files are stored. These files will be used to download the dataset images.
12. *ROUTE_PATH* – The directory where the query GPX files are stored. These files will be used to download the query images.
13. *DATASET_PATH* – The directory in which to store the dataset images. It will be created if it does not already exist.
14. *QUERY_PATH* – The directory in which to store the query images. It will be created if it does not already exist.
15. *CODEBOOK_PATH* – The directory in which to store the codebook created by k-means clustering. A directory is created to test different values of K.

II. BOVW.py

Interprets the configuration set in BOVW_Config.py and makes the proper calls to BOVW_Modules.py. Specific lines are outlined below.

1. 17 – checks to see whether a dataset needs to be downloaded
2. 19-25 – prepares a database for storing image's GPS metadata
3. 27 – downloads the dataset images
4. 30,31 – prepares a database for storing the extracted features of the images
5. 35-54 – calculates and stores the features for each dataset image
6. 56,57 – performs the appropriate clustering algorithm and stores the codebook in a pickled file
7. 65,66 – prepares a database for storing image histograms which are used to compare features
8. 80-123 – performs the same functions for query images, but also calculates a ground truth table (which dataset images are truly relevant to each query image)
9. 124 – performs a search on each query image and returns dataset images based on histogram comparisons.
10. 127-157 – evaluates the results of the search based on the ground truth table and the real-world distance between the query images and the returned images.

III. BOVW_Modules.py

A collection of methods necessary to perform a search in BOVW.py

1. *CONNECT_MSACS_DB* – connects to the local Cassandra server. If necessary, or the first time running on a machine, creates a new keyspace.
    a. Returns the session which is used to perform all database operations.
2. *compiledataset* – uses Google's StreetView API to download images at the latitude and longitude values specified in the GPX files.
    a. No return value, images are stored on disk and details are stored in the database.
3. *calcfeatures* – uses the specified algorithm to extract image details. The following auxiliary methods are called:
    a. *sift* – uses the standard OpenCV implementation of SIFT and returns the keypoints and descriptors for each image.
    b. *rootsift* – improves the standard OpenCV implementation of SIFT by applying Hellinger kernel and L1-normalization. Returns keypoints and descriptors.
4. *calccodebook* – performs the appropriate clustering algorithm and creates a codebook (currently only k-means is implemented).
    a. Returns the codebook in array format.
5. *createhistograms* – compares the image features to the codebook, and creates a histogram for each image.
    a. Returns the histograms stored in a pickled file.
6. *computeGroundTruth* – determines the relevant images for each query image
    a. Returns 1 for relevant images, 0 for all others.
7. *relevantcalc* – determines the number of relevant images
8. *search* – performs the search on all query images.
    a. Returns a sorted list of the returned images and their calculated (not actual) distance from the queried image.

Running the Code

1. Start the Cassandra service
    a. Navigate to the install location of Cassandra
    b. `cd bin`
    c. `./Cassandra`
    d. If the message `JUMP STATE TO OK` is displayed, Cassandra has started
        i. `sudo service cassandra status`
        Can be used to check whether Cassandra has started
        ii. `sudo service cassandra start`
        Force start method if the server will not start
2. Configure variables in BOVW_Config.py
3. Download .gpx files for dataset and/or route
    a. Default GPX files are provided on github
    b. Custom routes can be created online, using sites such as: http://gpx.cgtk.co.uk/
4. Run BOVW.py
    a. Run directly from an IDE
    b. Run from terminal: `python ~/path/to/code/BOVW.py`