1. Panorama Stitching

#Example 1:

Input:

Image 1

Image 2

Output:

Panorama Stitched Image

#Example 2:

Input:

Image 1

Image 2

Output:

Panorama Stitched Image

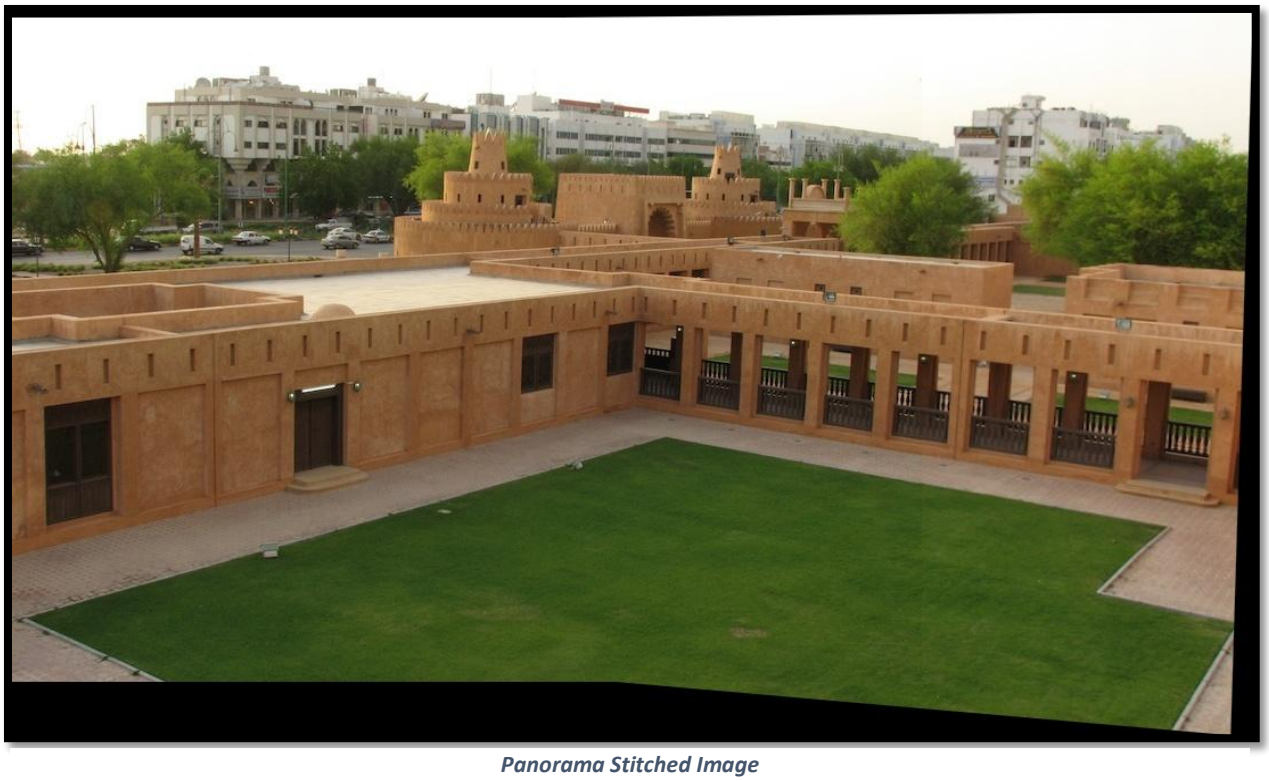
#Test 01:

Input:

Image 1

Image 2

Output:



Panorama Stitched Image

#Test 02:

Input:

Image 1

Image 2

Output:

Panorama Stitched Image

#Test 03:

Input:

Image 1

Image 2

Output:

Panorama Stitched Image

Differences between SURF and SIFT

SIFT	SURF
SIFT takes more time for computation.	SURF takes quite less time for computation.
SIFT is more accurate.	SURF is less accurate.
In sift we use Difference of Gaussian to build the image pyramid.	In surf we use an integer approximation to the determinant of Hessian blob detector.
In sift pyramids we use different scales of image.	In surf pyramid, we use different scales of Gaussian masks, while the scale of the image is always unaltered.
In sift we use an orientation histogram to find the main orientation of the feature descriptor.	In surf, we use the sum of the haar wavelet response around the point of interest.

Principles of Flann Matcher:

- The problem of finding the nearest neighbour is one of the major importance in many of applications. However solving this problem in high dimensional spaces become very difficult, and there is no algorithm that performs significantly better than the brute force search.
- This has led to an increasing interest in a class of algorithms that perform approximate nearest neighbour searches, which have proven to be a good-enough approximation in most practical applications and FLANN in one of them.
- FLANN is a library for performing fast approximate nearest neighbour searches in high dimensional spaces. It contains a collection of algorithms which work best for nearest neighbour search and a system for automatically choosing the best algorithm and optimum parameters depending on the dataset.
- FLANN based matcher will perform a quick and efficient matching by using the clustering and search in multi-Dimensional Space modules.
- This matcher trains a model on a train descriptor collection and calls its nearest search methods to find the best matches.

Implement Bike vs. Horse Classification

(Use Bag-of-visual words approach (SIFT/SURF + Kmeans + KNN))

Procedure (KNN):

1. Get the path of images in the training set.
2. Extract SIFT features from each and every image in the set.
3. Compute K-Means over the entire set of SIFT features, extracted from the training set.
4. Compute the histogram of features.
5. Train the KNearest classifier with the features (samples) and their corresponding class names (responses).
6. Get the path of image in the test set.
7. Implement step 2 to step 6 for the image in the test set.
8. Now give the Test feature vector and the K value (Number of neighbours to be considered for classification) to the trained classifier (KNearest).
9. Get the prediction.
10. Print the prediction on to the image in the test data set.

Procedure (SVM): The procedure is same except the change in classifier

1. Get the path of images in the training set.
2. Extract SIFT features from each and every image in the set.
3. Compute K-Means over the entire set of SIFT features, extracted from the training set.
4. Compute the histogram of features.
5. Train the LinearSVM classifier with the features (samples) and their corresponding class names (responses).
6. Get the path of image in the test set.
7. Implement step 2 to step 6 for the image in the test set.
8. Now give the Test feature vector to the trained classifier (LinearSVM).
9. Get the prediction.
10. Print the prediction on to the image in the test data set.

Approach (KNN & SVM):

The code was written in two parts both for KNN and SVM as well:

- Finding Features(Features.py)
- Classifier(class.py)---Training & Testing

Steps followed in finding the features are :--(Features.py)

1. Importing the necessary libraries.
2. Fetching path of training dataset.
3. Get the training classes names and store them in a list.
4. Get all the path to the images and save them in a list (image_paths) and the corresponding label in another list (image_classes).
5. Create feature extraction and key point detector objects.
6. Stack all the descriptors vertically in a numpy array i.e. convert a list into a vertical numpy array.
7. Perform the K-means clustering over the descriptors.
8. Calculate the histogram of features.
9. Now give more weightage to the clusters which occur more frequently in the data set and scale the visual words (This is used to increase the efficiency of classifier).
10. Now save the entire copy of visual vocabulary, feature (samples), Image_classes (responses) on to the disk. (This will reduce the complexity and confusion).—For KNN
11. Now save the entire copy of visual vocabulary, feature (samples), Image_classes (responses) as a dictionary file (.pkl) on to the disk. (This will reduce the complexity and confusion).—For SVM

Steps followed to get class of test image are (Training & Testing) :--(class.py)

1. Load the classifier, class names, number, vocabulary in to corresponding data frames.
2. Train the classifier (K Nearest Neighbourhood (KNN) or SVM classifier) with the training feature vector and the corresponding responses (class names).
3. Get the path of the test image or the path of the test set (multiple test images).
4. Create feature extraction and key point detector object for the test image.
5. Stack all the descriptor(s) of test set vertically in a numpy array i.e. converting a list in to a numpy array.
6. Compute the histogram of features of the test image(s).
7. Perform If-Idf vectorization on the testing image feature set and scale those features. (This is done for better efficiency).

8. Compute the prediction by using the trained classifier (kNN) and with test features and K value (i.e. number of neighbours to be considered for prediction) as inputs to classifier.---For KNN
9. Compute the prediction by using the trained classifier (SVM)---For SVM
10. Print the predicted output on to the corresponding test image.

Output comparison between kNN and SVM (linear):

- **The wrongly predicted images are boarder with red colour**

Image classified as Horse (kNN)

Image classified as horse (svm)

Image Classified as Horse (kNN)

Image classified as Horse (svm)

Image classified as Horse (kNN)

Image classified as Horse (svm)

Image classified as Horse (kNN)

Image classified as Bike (svm)

Image classified as Horse (kNN)

Image classified as Horse (svm)

Image classified as Bike (kNN)

Image classified as Bike (svm))

Image Classified as Horse (kNN)

Image classified as bike (svm)

Image classified as Horse (kNN)

Image classified as Bike (svm)

Image classified as Horse (kNN)

Image classified as Bike (svm)

Image classified as Horse (kNN)

Image classified as Horse (svm)

Observations:

- I observed that this classifier is less efficient in predicting the class.
- SIFT feature extraction is not efficient in case of blur.
- The efficiency is based on the number of clusters i.e. K value which should be in the range of 500-800 for around 50 % efficiency.
- The K value is directly proportional to the time complexity of training the classifier i.e. more the K value more is the time to compute the cluster. It took around an hour to get the vocabulary from all the training images with k value of 500 (Machine specs: 2GB RAM, Pentium Processor).
- The time complexity is reduced if SURF feature extractor is used in place of SIFT
- The prediction also depends on the number of neighbours that I am taking for predicting the class of the test image.

- For predicting a complex feature set(such as horse with a grassy background), we need more training examples when compared to the number of training examples required to predict a relatively simpler feature set (bike).
- From the above shown results the kNN is predicting efficiency is very low, in specific kNN was not able to classify the bikes correct. It is also shown the linear support vector machine was very efficient in classifying the images in to the corresponding correct classes.
- I also suspect that if we increase the number of training examples for bike, then even the kNN can also predict efficiently.