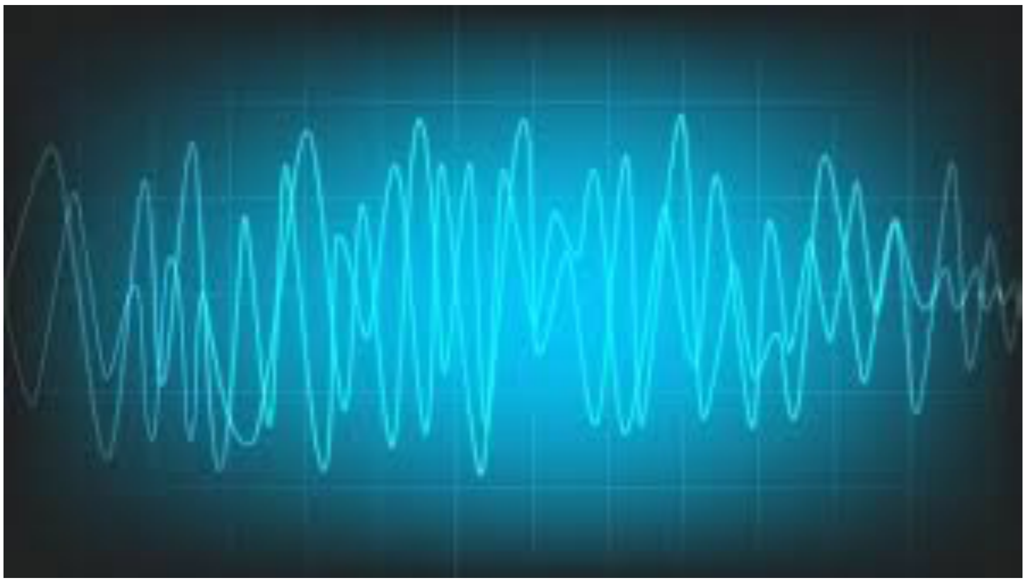
Automation of Slide Matching

DSAA Project Report



Sarthak Singhal 20171091

Abhinav Gupta 20171059

Report

The Question:

We are given a set of noisy slide images, and we need to map them correctly to the original slides.

The Idea:

Now of course, the first thing we thought of was doing a normalised cross correlation on the 2D images, as it is the most popular measure for similarity between two signals/objects. But this wouldn't be very accurate for a robust database with many noisy images, and also the clue in the question said we must try to this beyond normxcorr2.

One of us had worked for Five Fingers Solution as part of the SSAD project last semester, in which the objective was to use depth images to reconstruct a scene in 3D, using different image stitching and panorama techniques. So for this, image processing was needed, in which images were stitched based on the number of the 'keypoint' matches. (Stitching is combining multiple images with overlapping fields of view to produce high resolution images.) So we had to use Scale Invariant Feature Transform (SIFT), and that's where we got the idea from.

The Implementation:

We have two arrays: slides[] and images[] which in which we store the paths of the slides and frames respectively. We the load the SIFT algorithm using cv2:

```
sift = cv2.xfeatures2d.SIFT_create()
```

And then we run 2 loops: outer one for slides, and inner for the images/frames, matching each slide with the images. We read the images using cv2.imread and store them in I1 and I2 respectively. For each of these, we calculate the key points and the descriptors. Keypoints are like interest points, those points in the image that stand out and define what is interesting about it.

And what makes key points different between frameworks is the way we describe these key points - these are called descriptors. Every key point has an associated descriptor with it.

```
kp_1,desc_1=sift.detectAndCompute(I1, None)
kp_2,desc_2=sift.detectAndCompute(I2, None)
```

So now we have the key points of the images we want to compare, and now we want to match these. For this, we use FLANN, a library for performing fast approximate nearest neighbour searches in high dimensional spaces. Its complexity is O(nlogn), which is definitely better than doing brute force. We load the FlannBasedMatcher to find the matches between the descriptors of the two images.

```
index_params=dict(algorithm=0, trees=5)
search_params=dict()
flann=cv2.FlannBasedMatcher(index_params, search_params)
```

And then we find matches between the 2 images. These are all the possible matches, including the false ones.

```
matches=flann.knnMatch(desc_1, desc_2, k=2)
```

Now, we need to filter the good matches from the bad ones. For this, we use the ratio test. The quality of a match is define by the distance, which was caculated in 'matches'. The similarity of features is inversely proportional to the value of distance. That is, lower distance value implies higher quality.
Also, the value of the ratio determines the quality of the matches. If it's very low, we get high quality matches, but there will be very little.

```
for m,n in matches:
if m.distance < ratio*n.distance:
good_points=good_points+1
```

And if number of good points that matched were more in the current iteration, then we accordingly update gparr[] and imgarr[]

Hence, we get the best matches possible!

The Shortcomings:

~ This algorithm detects images when the they are quite different from one another and not very similar. In case of similar images the accuracy may decrease. But since we are given images of slides which are quite distinguishable, we can use this method.

~ This method requires slides to be given in complete form. For instance, if one slide has one line in it and then in the next slide, a new line is added. In such a scenario, this method would fail as it will match more when the content is more.

~ Apart from the above two cases, our algorithm would work well on the test cases. Also we have not used the OCR method as slides may contain images and animations too. The OCR method would fail in that case.