CVML final project

**SC3653\_PWN2107**

**‘snapFox’**

**Introduction**

The last few years marked a tremendous rise in the capabilities of mobile phones. One of the features that saw consistent and steady improvement in mobile phones has been the mobile camera. As the computation power of mobile phones has increased tremendously, it is now possible to use the mobile cameras to capture images, apply cool vision algorithms and get really interesting results.

The hot market for mobile camera apps, coupled with the enthusiasm to implement some really cool algorithms that we learnt in the Computer vision course, we decided to come up with a mobile camera app – **snapFox.** The name desig

**snapFox** allows a user to capture multiple snaps of a given scene, taken at different focal points. The captured images are run through two vision algorithms that stabilize the images and generate a ‘focal-index map’. The ‘focal-index map’ maps each pixel in the image to the index number of image in the stack for which it has the best focus. This focal-index map is used to enable a user to refocus images when he/she taps on any part of the image

**snapFox** has been developed in iOS and uses openCV for implementing the vision algorithm that enable this effect.

**Implementation specifics**

The core functionality of snapFox is based on the following two techniques:

1) Image stabilization

2) Focal-map generation

Image Stabilization

The heart of image stabilization is in template matching. We send across our stack of images to an openCV function which does template matching for all the images and generates a common bounding area for the whole set. The common bounding area is used to generate a new set of images from each of the images in stack, such that they all depict the same scene image. In case there is too much of shake in the camera, the bounding region returned is really small. For a bounding region below a certain area, the user is asked to retake the photo. This part is still in testing phase and might not be used in the app as of now. However, the inclusion of this feature would make the stack of snaps robust. For now, the user would be advised to use a tripod/gorillapod to take the snaps.

Focal-index map generation

Once the image-stabilization has been done, the focal-index map is generated. In order to do this, we do the following for each image in the stack:

* Get the 2nd order derivatives of each image in the stack in the x as well as the y direction.
* Sum up the absolute values of each of the two 2nd-order direction derivative images.
* Boost the intensity value of the resultant image by applying a local spatial filter throughout the image.
* Use a spatial box-filter to smooth the entire image. The size of the smoothing kernel is selected, based on the size of the image.
* The resultant matrix gives us the focal-measure for each of the pixel in the given stack image.

Similarly, the focal-measure matrices are calculated for all the images in the stack. Now, for each scene point, the maximum focal-measure is located by traversing all the focal-measure matrices in the 3rd dimension. A new matrix is generated which contains the location of the stack frame for which the given pixel has maximum focal-measure. This newly generated matrix is called the focal-index map.

Once the focal-index map has been generated, we display the first image from the stack and allow the user to tap on the screen. When the user taps, the value corresponding to the tap point coordinates is determined from the focal-index map. This number identifies the frame that should be displayed for the tapped point. The image smoothly transitions to that frame now.

**UI Design specifics**

snapFox has the following UI elements:

**>>Home screen:**

1) ‘Example’ – It shows the refocusing example of a focus stack composed of 25 images and the results of refocusing done on it.

2) ‘’Snap” – Takes the user to a custom camera interface, where he can click a scene and ‘snapFox’ it.

**>>Example page:**

Shows the example image and a link to return to the Home screen

**>> Snap page:**

Shows the custom camera-interface and allows the user to snapFox a scene. It leads to the preview page

**>>preview page**

This page displays the image from the top of the image stack and allows the user to refocus it. It also gives the option of returning back to the home screen.

**Learning**

The project work was divided into two broad categories: Implementing algorithms in OpenCV, and tackling the objective-C Goliath. Our constraint (that of only one person having a MAC and iPhone), rather than us, made the decision for the division of work for us. Here is how we divided the work and what we learnt from our modules:

**Siddhartha** – My major learning from the project was getting a grasp of the AVFoundation framework for mobile camera operations. This framework allowed me to control the internal settings of the mobile camera and allow it to take shots focused at scene points specified by me. There is a more generalized and easier UIimage picker class that simplifies the ‘photo-capturing’ process. However, it does not allow controlling the camera functionalities, so we had to stick to AVFoundation framework.

As there were parts of the code that were time consuming and blocked the main thread, I also implemented an additional thread for handling events in parallel. Never having implemented a multi-threaded program, this app gave me a chance to appreciate the importance of having multiple threads in a program. The other tricky parts were about getting a seamless flow in the app, writing and reading files to the photo-album, integrating the openCV part to the main code and figuring out those small, pesky bugs that inadvertently stopped progress and threw me into bouts of pessimism about the successful creation of snapFox.

**Peter-**

**Results**

**Challenges faced**

**Siddhartha –** I thought that implementation would be an easy task if the algorithm and approach to solving a problem is known. I was so wrong! It is not the obscure parts in the algorithm, but the pesky, small bugs that need to be feared. Working on and solving the language specific bugs in objective-c challenged me at multiple places and took a majority of my time. Another issue was testing the code was a fairly time-consuming process as it took several minutes to do one round of test. As iOS does not allow access to the features of the camera so easily, it took a lot of efforts to figure out how to get what I was looking for. Online forums proved to be of a lot of help at many places, but I did not get much help when I was stuck at the issue of taking multiple pictures from the camera.

**Limitations of snapFox**

* Currently, it takes a lot of time to process the images and generate the focal stack. Due to this the app has a delay as the flow passes from one view to another.
* As we did the testing on the lab’s device(iPhone4), we were not able to test the feature of exposure locking. (iphone 4 does not support it). So snapFox does not support exposure locking while taking snaps
* snapFox only focuses at 9 points to take the shot. As the camera needs adequate time to focus and capture a shot, we decided to cap the shots to 9. Due to this the number of focal planes generated are limited. This results in more abrupt transitions as compared to the smooth transitions observed in the example image.
* It does not give good results for an image stack that has a lot of variations. This can partly be overcome by integrating the module for image stabilization. For now, one must use a tripod to get the best results (unless ofcourse if you have the steady hands)

**Future enhancements**

* The view gallery would be improved in terms of displaying thumbnails of snapFoxed images that are already present. This feature is not supported currently.
* Image stabilization would be introduced.
* Currently snapFox is an isolated app. In order to make it more social, we would like to integrate it with the facebook api. The challenge here would be in displaying the snapFoxed image based on user click.

**Conclusion**

**REFERENCES**

1. Image courtesy – Professor Shree Nayar