CS 478: Hello Camera

Tony Hyun Kim

January 31, 2012

1 MyAutoFocus

My `MyAutoFocus` implementation is based on a very simple state machine whose public methods

- `MyAutoFocus.startSweep`
- `MyAutoFocus.update`

control the transitions as shown in Fig. 1. The autofocus "logic" (*i.e.* determining next focus, determining when to terminate, etc.) is implemented in the state `MYAUTOFOCUS_SWEEP`. I basically use a linear (exhaustive) search over the possible lens positions.

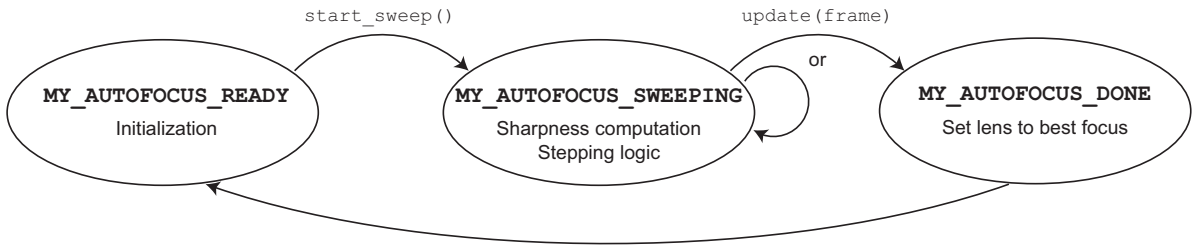

Figure 1: State diagram illustrating the behavior of MyAutoFocus.

The actual image processing to determine the sharpness of the viewfinder contents is very simple, and is contained in the private method `MyAutoFocus.computeSharpness`: (1) For each pixel I simply compute the up-down-left-right luminance gradients; (2) these gradients are summed over the entire image and normalized by the image intensity; (3) the result is my "sharpness" score.

2 Failure mode

Because my search algorithm is basically an exhaustive linear search, it is quite robust. However, I did add one "bell and whistle" that causes the occasional glitch. Namely, as `MyAutoFocus` is performing a lens sweep,

it keeps track of the best sharpness score obsered until that point. If the current frame's sharpness score is below 80% of the best score, the search will terminate. While this behavior gives better (faster) performance most of the time, it obviously introduces the possibility of premature termination.

3 Determination of parameters

As one of the first steps in this assignment, I made table dumps of (lens focus, sharpness) for various scenes, with the object at different distances from the tablet and under various lighting conditions. Fig. 2 shows few such results.

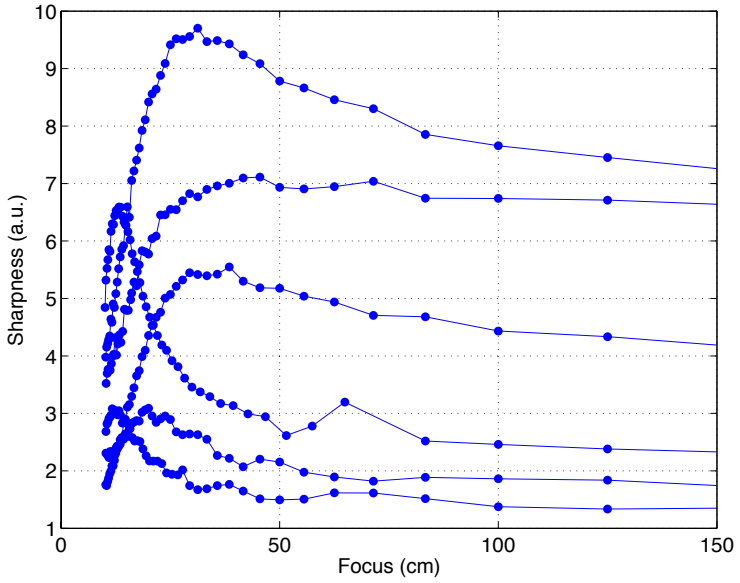

Figure 2: Sharpness (arbitrary units) as a function of lens focus (cm) for a variety of scenes. Between each curve, I moved the object's distance with respect to the tablet as well as illumination conditions.

From Fig. 2 I made a few conclusions:

- For most scenes, the sharpness curve shows noisy local maxima. For this reason, I pursued the linear search strategy (and tweaked `MyAutoFocus.computeSharpness` so that the scan runs adequately quickly) rather than other methods based on the local derivatives of the sharpness curve.
- I deduced the early-stopping criterion of 80% from these curves (and subsequently tested it out in practice, of course).

I also use some hard-coded values for my color-based autofocus algorithm. Here, I had FCamPro log the chrominance values of its pixels while I shoved my objects (a red binder and an orange children's book) in front of the tablet's camera. I then determined the color filter threshold (± 5 chrominance units from the target value) experimentally.

4 Color mode

I implemented color-based autofocus.

A natural UI for specifying the color is to have the user simply select the region of the viewfinder with the desired color. However, I thought that this was too similar to the interface for local touch-to-focus. Furthermore, I wanted to learn a bit more about using Android widgets. So, I have the application prompt the user for color selection in a dialog window, where the user selects the desired color in a drop-down menu (a color palette would be even better).

The color filter is implemented by checking each candidate pixel in the sharpness calculation whether its chrominance values are within a specific window. While this implementation is simple, it took me a while to determine the correct addressing for `U` and `V`! (Namely, that only 1/4 of the entries – when compared to `Y` – are stored in memory.)

In Fig. 3 are a series of (resized) images that illustrate the performance of the color autofocus routine. I have attached the full-size versions along with this report submission.

5 FCam API

The Frankencamera API seemed reasonable after a day of digging about. FCam seems to be extensible. I have no particular complaints.

6 Tegra development environment

I have no complaints against the Android development environment (Eclipse and ADB). The `Log` feature and `adb logcat` were sufficient for me to debug my application. I'd like to figure out how to get a transparent overlay on top of the Viewfinder view.

However, the Tegra 3 tablet itself is extremely unstable. You already know about my earlier complaints about the finnickness of the hardware. It seems, after seeing other students' comments on Piazza, that my tablet was not an isolated case. We definitely need a way to hard-boot the device without touch confirmation, since many of us have observed the tablet to lose touch sensitivity. This is a must. I can't imagine having to work with a device for the rest of the quarter that is liable to brick at random times. I have now come to regard the tablet as a very, very developmental device, which is really disappointing.

Figure 3: Demonstration of color-based autofocus. (a) Global autofocus; (b) Color-autofocus for orange; (c) Color-autofocus for red.