Assignment 1: Mean Shift Segmentation

Daniel Sawyer
University of South Florida
danielsawyer@mail.usf.edu

Abstract

Using Harris detector for feature detector by finding Eigen vectors from correlating an image with itself. Then taking the top K feature points as x,y values of pixel location. We then use a Kalman filter with a constant velocity in order to get the minimum value for the covariance patch by correlating with the difference squared of the intensities from the original image and current image summed up within a set window area. We then update the state every optimal amount of frames f(x) based off some function dependent on the domain, velocity, etc factors.

1. Algorithm

The algorithm is broken into four parts. The convolution to get dx and dy gradients, feature detection, Kalman filter, and model update function.

2. Related Work

I used the Gaussian kernel Dr. Sarkar used in class along with the basic CUDA tutorial for Numba in python[1]. Besides that I did not use any code from any other sources and you can see my python code implementation in smcImgCuda.py located in the same directory as this report file.

3. Examples

I initially ran the algorithm with Hr=8, Hs=7, and M=40 along with the number of standard deviations for Hr, called SDR, and Hs, called SDS of 3. The following images are comparisons between the diffent parameters set for Hr, Hs, and M. fig:baboon

4. Lessons Learned

I learned quite a few lessons with this project. The major lesson is how you can use just intensity/color values combined with spatial location in order to segment an image. It is quite amazing how you can use a Gaussian kernel in order to make the pixels cluster and shift to a relative mean that converges everytime. I was quite suprised on how well it converges. In figure 2 and 3, the only difference is that figure 2 is 50 steps and 3 is 1000 steps. You can tell that it basically converged at 50 steps and by 1000 it had completely converged with very minimal differences between the two. It was the same for the MM images. Figure 5 was 100 steps where figure 6 was 2000. I dont think there is hardly any difference between the two, so after a certain amount of steps, there is no noticable difference in intensity

Figure 1. Baboon Original

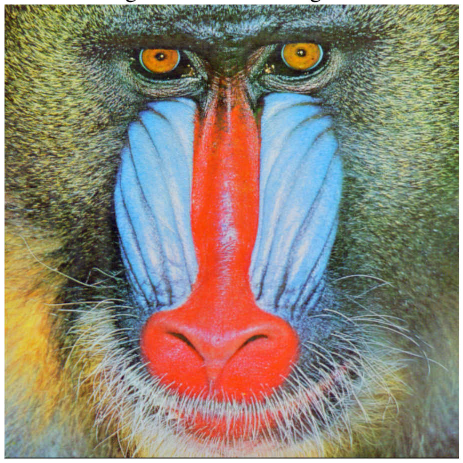


Figure 2. Baboon(8, 7, 40) 50 Steps

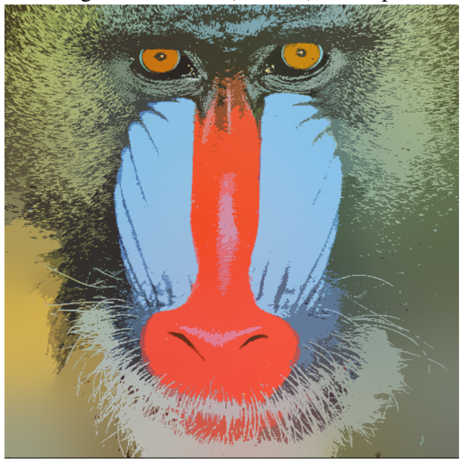


Figure 3. Baboon(8, 7, 40) 1000 Steps

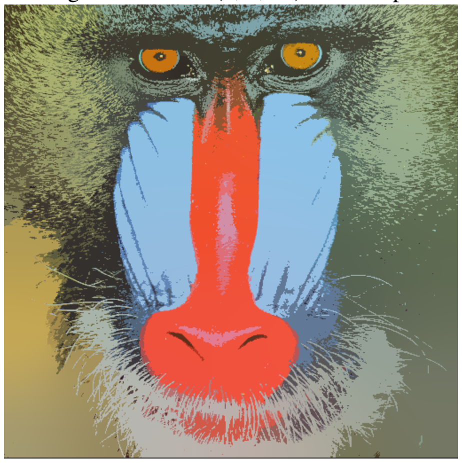


Figure 4. MM Original

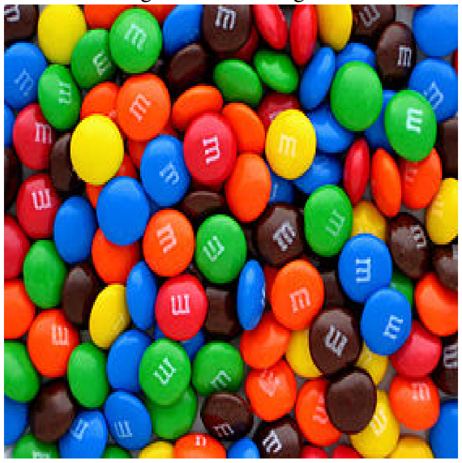
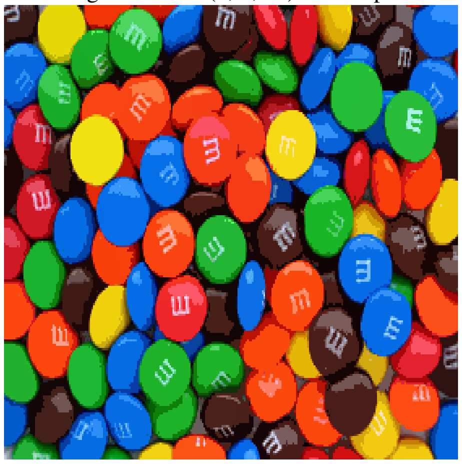
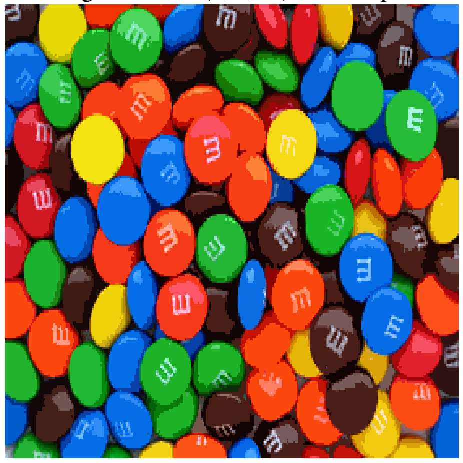


Figure 5. MM (8, 7, 40) 100 Steps



values, hence the image has converged. I also learned how valuable parallel processing is, I would have never been able to complete this project on time if it was not for the CUDA implementation of the algorithm since the naive CPU version is very, very slow.

Figure 6. MM (8, 7, 40) 2000 Steps



References

[1] Numba cuda programming. 2019.