The awesome IEEE-style report for CMSC733 Homework 0!

Anton Mitrokhin

***Abstract*—PinIt and PB Lite.**

This is a report for CMSC733 class Homework 0. Here you can find some details on color segmentation algorithms - 1D gaussian and 3D gaussian, as well as 'Probability of Boundary' algorithm for boundary detection. Some cool pictures are included.

I. Introduction

We use Python 2.7 on Ubuntu 16.04 witn numpy, scipy and OpenCV 3.1 (which comes with ROS Kinetic). Note, that the code will probably work with OpenCV 2.4, but the kmeans function has a slightly different API... Run the executebles (for Part 1 and Part 2) with -h option to get the list of options to tweak parameters!

January 29, 2017

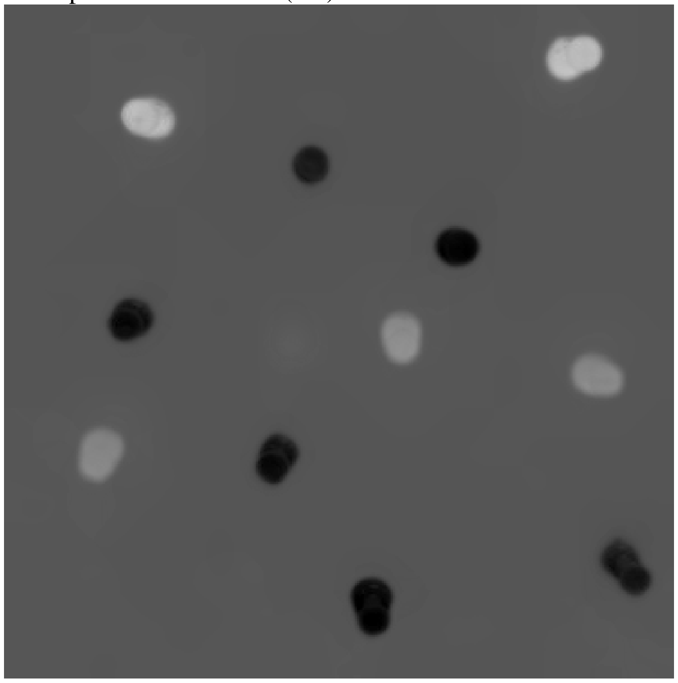
A. Pin It!

Primitive color segmentation

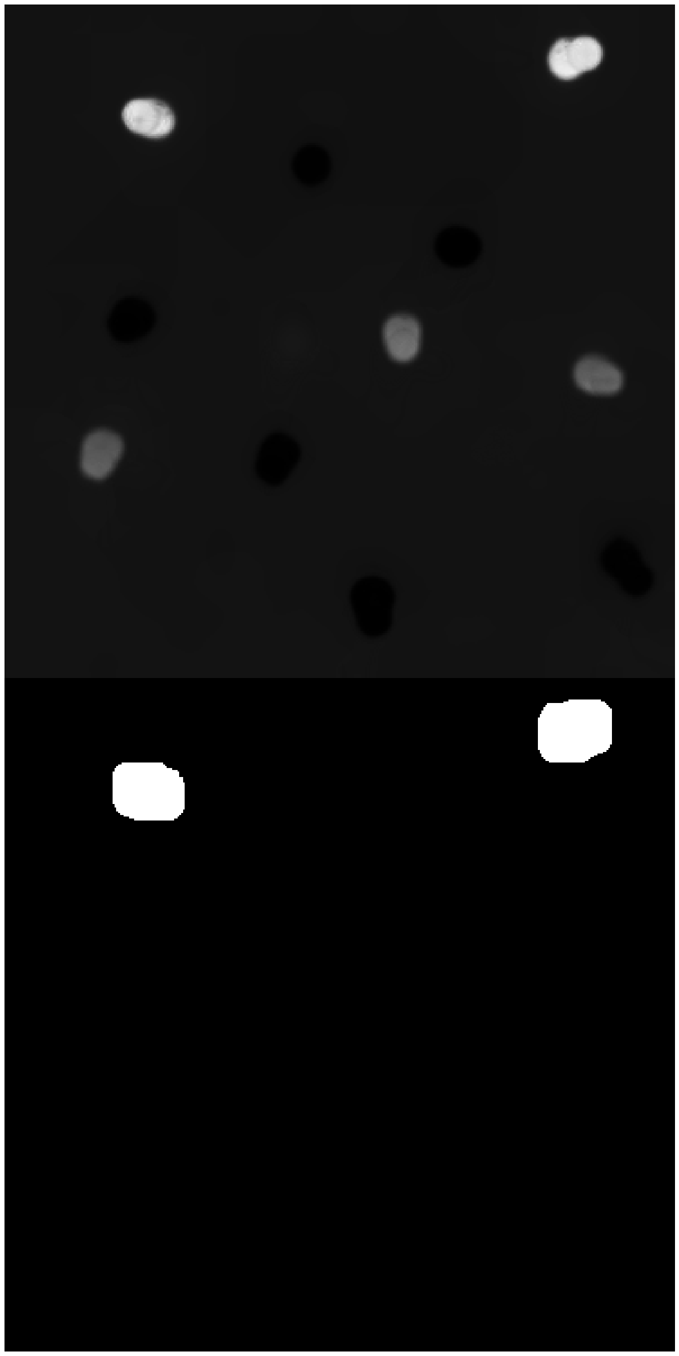
1) One-dimensional gaussian: Run

```
./pin_it.py --method=G1D
```

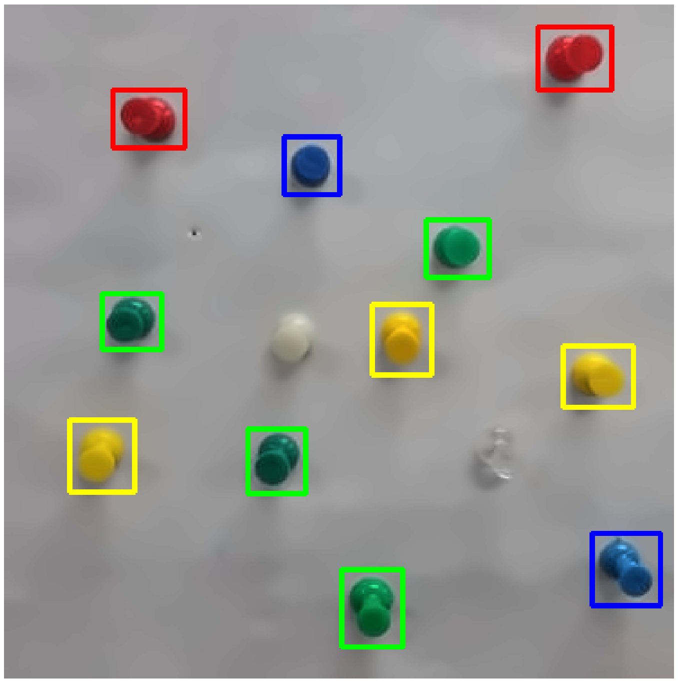
The input image was split into R, G and B channels, here is an example of one of them (red):



A number of 1 - dimensional gaussian functions were created, and median and sigma were adjusted for each pin color in order to detect each of them separately. The sample probability of outputs for each channel:



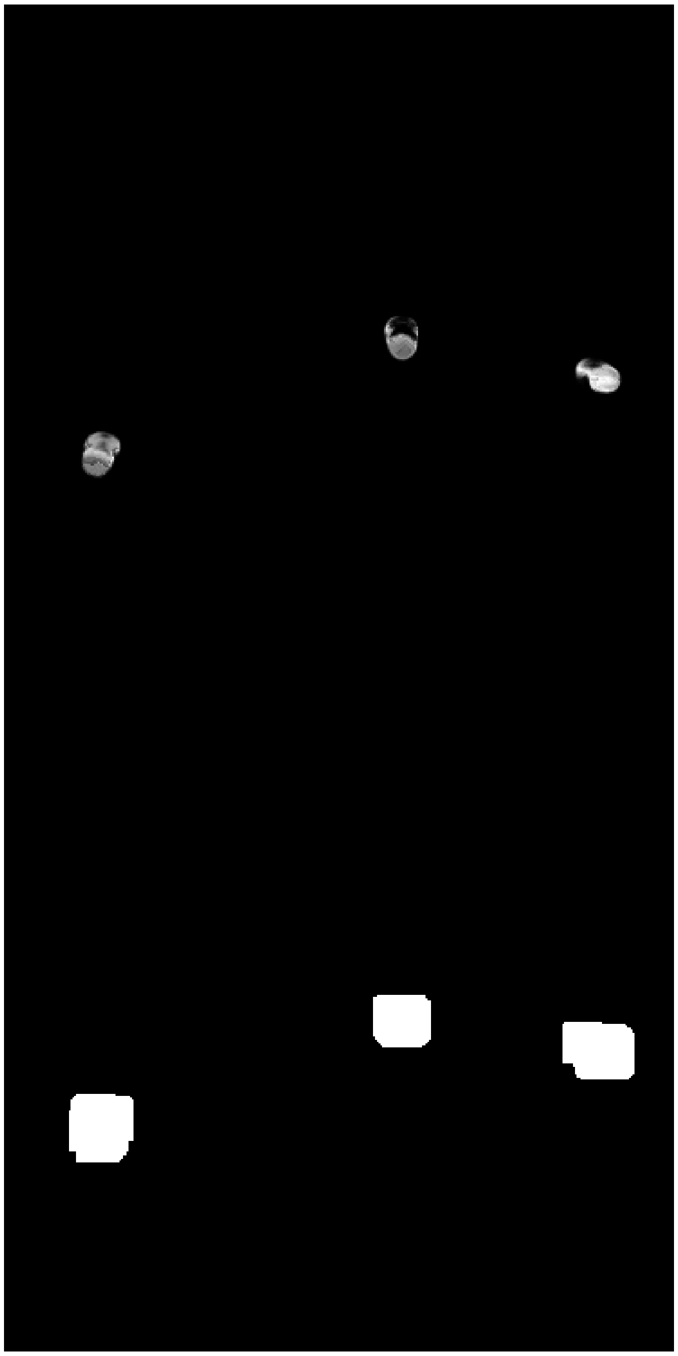
The top one is the output of gaussian, the second one is the threshold, eroded and dilated. The edges then are found on the threshold image and a bounding box is constructed for each kind of pin separately. The white and transparent pins can not be detected with this method. The result is on the following picture:



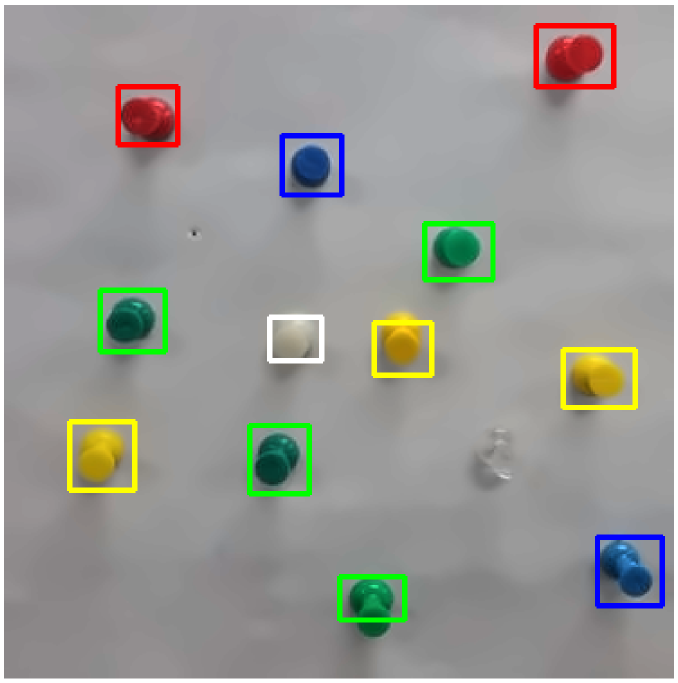
2) Three-dimensional gaussian: Run

```
./pin_it.py --method=G3D
```

The pins from the original image were manually cropped to create a color database (located in database folder) and the 'average.py' script was used to generate mean vector and covariance matrix for each pin type. Those were used in 3D gaussians (similarly to the 1D gaussians) to get probability maps for each pin type. The sample map is (for a yellow pin):

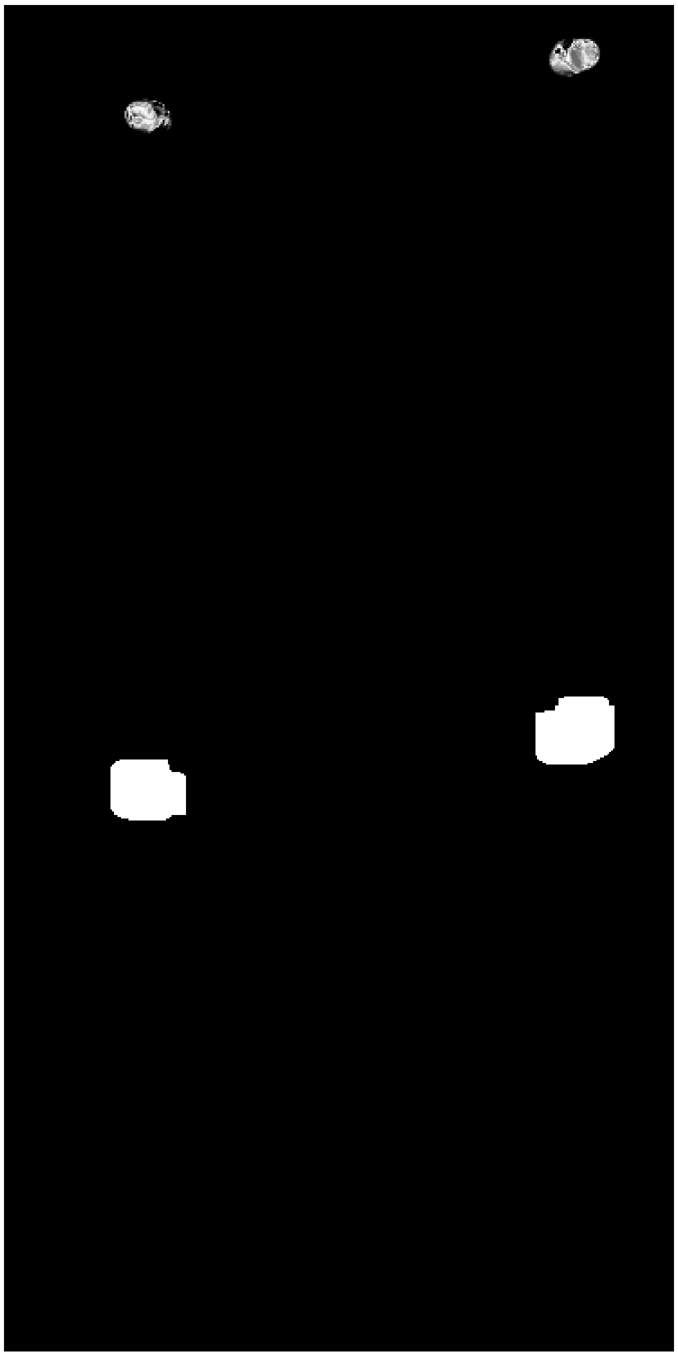


The final result is (with the white pin):

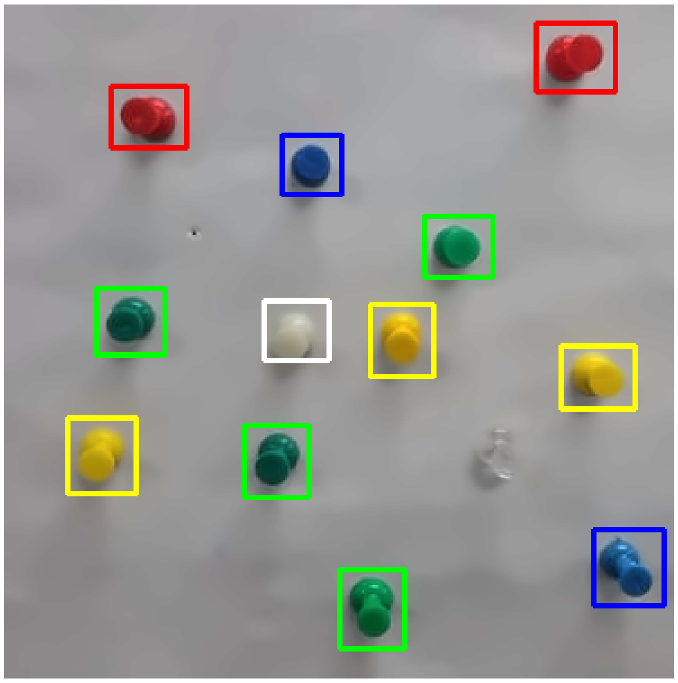


The database for the transparent pin results in a ininvertible covariance matrix...

3) Experimental k-means sort of GMM: For this part the averager.py was modified to utilize k-means clustering on the original color database images. 6 clusters were used and (consequently) 6 means and covariance matrices per pin produced. Each of the mean - covariance pair was then processed separately by 3D gaussian algorithm and the sum of the 6 outputs was treated as the resulting probability. This approach allowed to remove some outliers (especially when manual cropping is done poorly) and in theory increased the robustness of the algorithm. The probability map is:



The result was similar to 3D gaussian (although somewhat more precise):

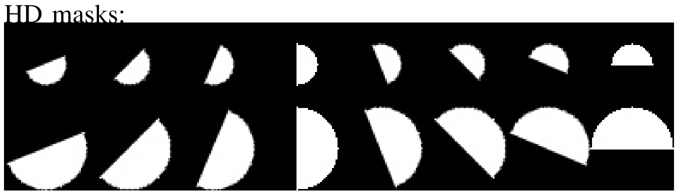


B. PB Lite

Probability of boundary: simplified version

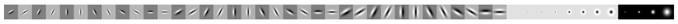
Half-disk filters, gaussians, LM, S and MR filters are generated. The default (and by far the best) values are: 49 pixels for filter size, 2 scales and 8 rotations. The filter images are:

HD masks:

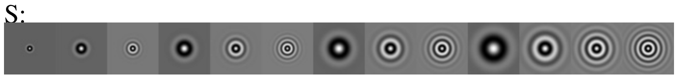


Gaussians:

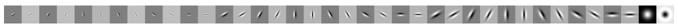
LM:



S:



MR:

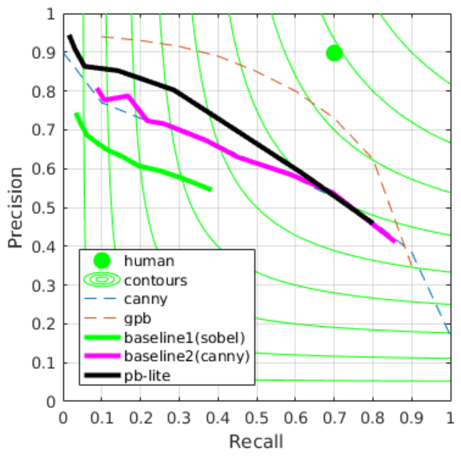


After that, Gaussians, LM and S are convolved with the a and b channels of the original image in the L*a*b space. MR is not used as it makes the result slightly worse.

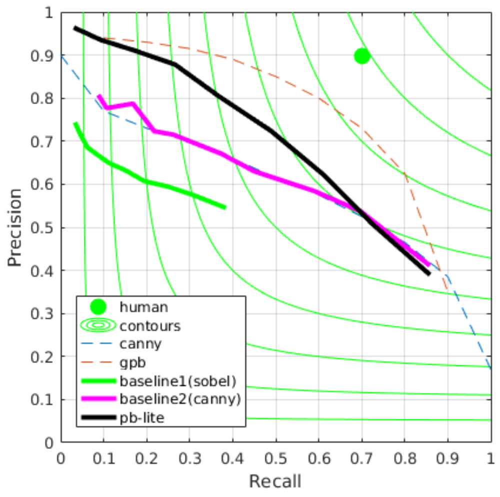
k-means with 16 bins is used to generate texton map.

L channel of the image is used to generate brigtness map (20 bins). A and B channels are used to generate color maps (20 bins). The Texton/Brightness/Color gradients are generated by means of chi-square measure. The resulting gradients are divided by their own mean value to adjust dynamically for the false positives. The canny and sobel are myltiplied by the sum of their pixels respectively to strengthen their responses. The sum of all gradiends is them multiplied by the sum of canny and sobel and the result is normalized and saved.

The benchmark with no dynamic weights:



The benchmark with the dynamic weights:



As can be seen, the dynamic weight adjusting gives far better performance.

II. Conclusion

We (I) did a great job here and deserve an extra credit for being awesome.

Acknowledgment

The authors would like to thank... nobody. As they had to do everything on their own!

References

[1] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. IEEE transactions on pattern analysis and machine intelligence, 33(5):898–916, 2011.