HW0: Pin It!

Gudjon Einar Magnusson

September 6, 2016

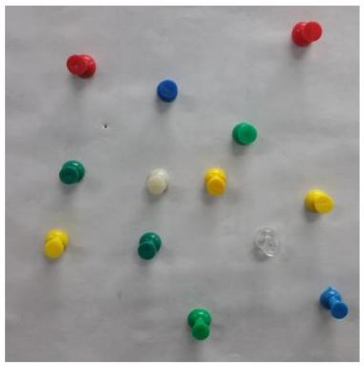
1 Denoise

Filtering out the noise is not strictly speaking needed to get the job done but it produces a slightly cleaner result. To filter the noise I used the *medfilt2* function, mainly because its very simple. *medfilt2* operates on a 2D matrix a therefore I apply it to each channel of the RGB image separately and then recombine the filtered channels for the final result. Figure 1 shows the image before and after the noise filter.

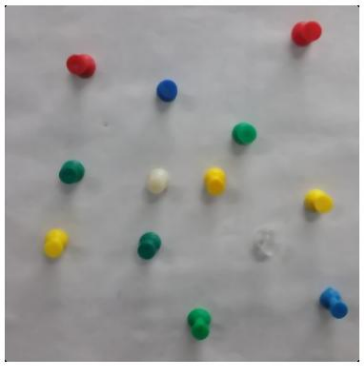
2 Find the pins

2.1 Color Normalization

To start with I want to find all pins in the image, of any color. To do this I came up with a very simple normalization function. For each pixel of the image I divided the minimum value of any channel with mean value of all the channels. This is a very effective way to separate colored pixels from white pixels. White and gray pixels have a high value close the the average in each color channels but bright colors have more variable values in each channel. This produces a gray-scale image where bright colors appear very dark and white and gray appears very bright. Figure 2 shows the image before and after normalization. Unfortunately, as you can see, this normalization function does not do a good job of finding the white and transparent pins.

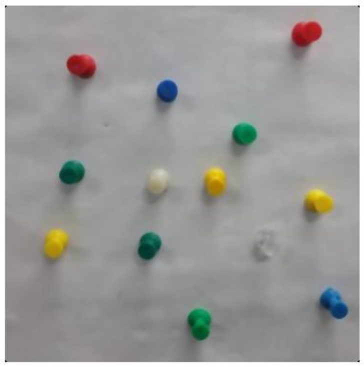


(a) Original pins image

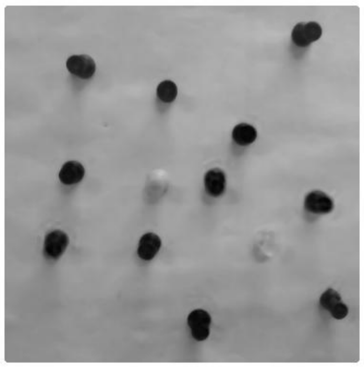


(b) Pins image after noise filter

Figure 1: This shows the subtle effect of the noise filter

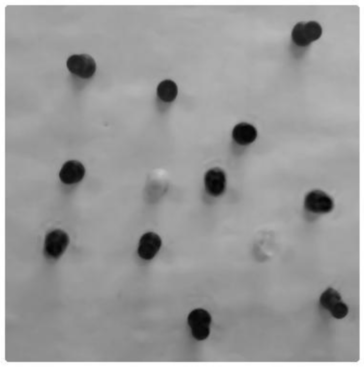


(a) Color image



(b) Normalized image

Figure 2: This shows the effect of the normalization function


(a) Normalized image

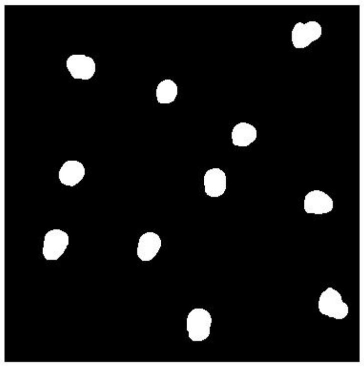

(b) Binary image

Figure 3: The effect of applying a 0.4 threshold to the normalized image.

2.2 Separate the pins

To separate pins from the background and localize them, I produce a binary image where 1 in pin and 0 is background. I do this by using the *im2bw* function. *im2bw* take a 2D matrix and threshold value. Any pixel with a value greater than the threshold becomes a 1 and any value below becomes a 0. With my normalization function a threshold of 0.4 works well. With a slightly higher threshold I can just start to make out the white and clear pins, but I don't think thats a reliable result. Since the normalization function shows the pins as dark and the background as bight but I want the pins to be classified with a value of 1 I invert the resulting binary image using the logical NOT operator. Figure 3 shows the image before and after applying the *im2bw* function.

2.3 Localize the pins

To count an localize the pins in binary image I use the *regionprops* function. *regionprops* returns information about connected regions in a binary image, such a bounding box. With the bounding box I can draw a box around the pins and I'll do that after I figure out their color.

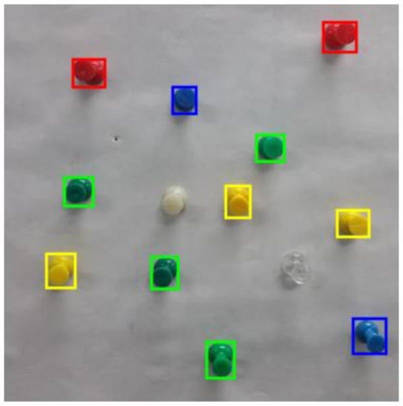


Figure 4: Pins labeled with their classified color

3 Classify the pins

To classify the color of the pins I look at each pin separately. I produce a small sub-image using the bounding box returned by the *regionprops* function and calculate its mean color value. This gives me a single RGB vector for each pin. Next I calculate the euclidean distance between that vector and each of my 4 reference colors, red, green, blue and yellow. Which ever color has the smallest distance is the color of the pin. Now I can draw the bounding box using the *rectangle* function.