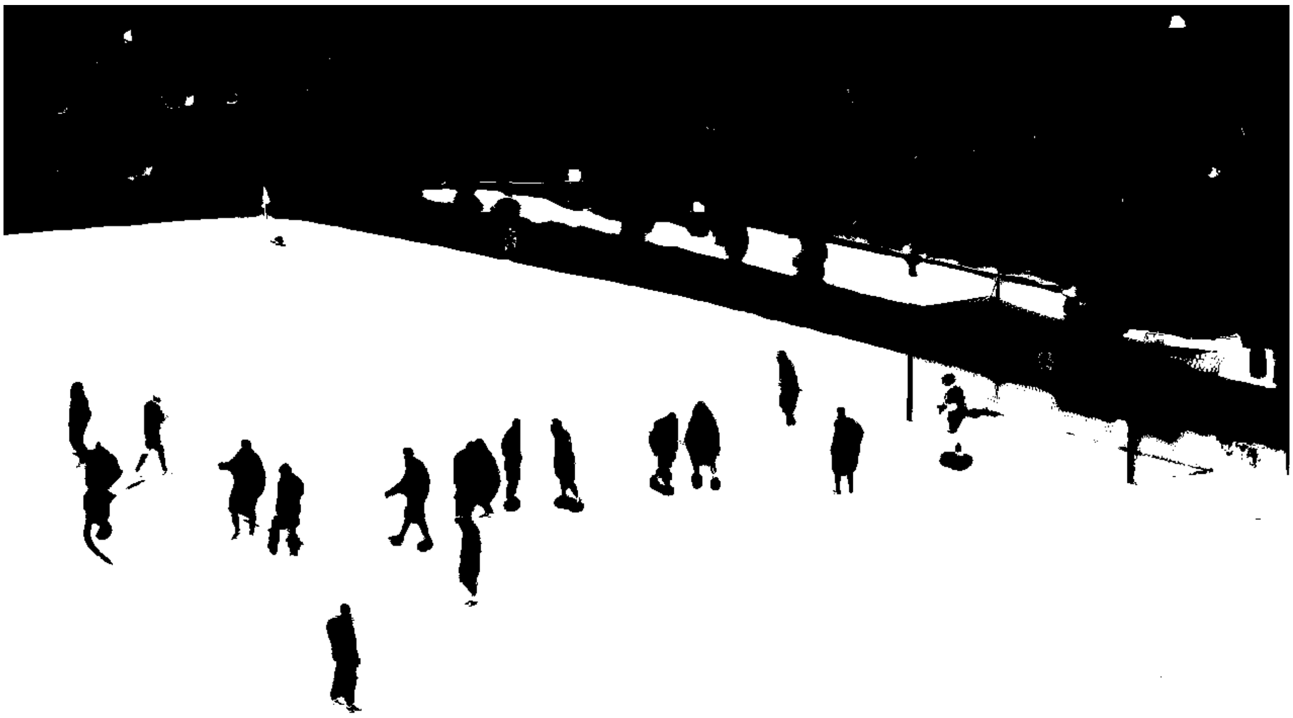
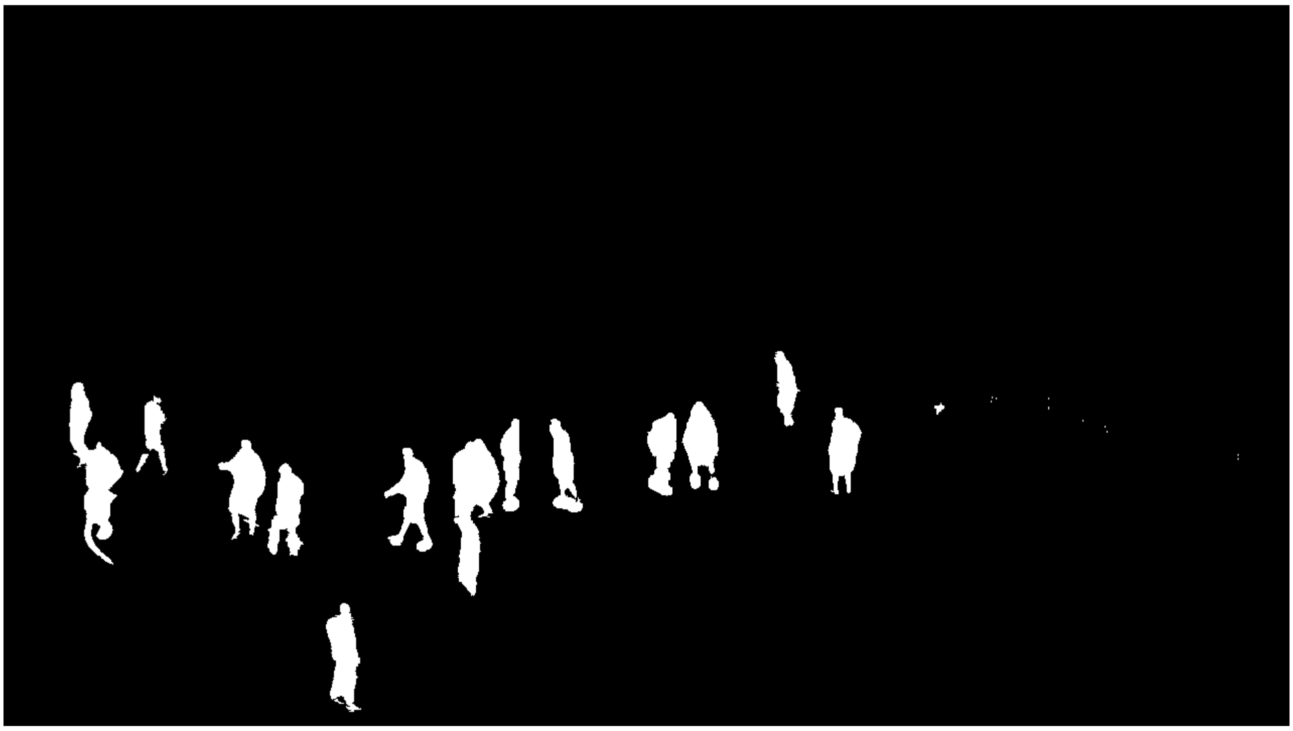
Digital Image Processing: Assignment 3

1. Write a code to segment out the players in the image 'soccer 1.png'. The output should be a new binary image, taking the value 1 for the players (pixels occupiedby the players) and 0 elsewhere. Now try to make your code generic, so that it works for other nine soccer images without any changes in the code. Upload the obtained results as well as the codes.

The main advantage of having a restricted setting to pick up the players from, is that we can narrow down our initial area to search for. We can do this simply by focusing on the area of the field. The field is green, so our first step is to extract the field. Looking at all the pictures, we realize that due to variations in the intensity/saturation across all images, the HSV space would be well suited for extraction.



After appropriate thresholding, we have a mask of the rough field. Obviously the players will be represented as holes in the field. Notice now that both the players and the audience are marked by white, due to our mask, we would like to treat them seperately. Here we have to distinguish between the crowd and players. So we first invert the mask obtained. Now, the players, and the audience both are marked with white. Obviously, nothing within the field will be connected to the crowd, since the field demarkates the region between the players and the crowd. We make use of this fact, and simply find the largest connected component in our given mask and eliminate it(This component for obvious reasons will be the region of the crowd). Now with both the field, and audience marked in black, we find that we will only have the players marked in white, along with anything else in the field like markings and the football. We need to come up with a way to eliminate these. We perform a simple closing operation to eliminate the noise from the field. Below is the image obtained by performing this method on soccer_1.

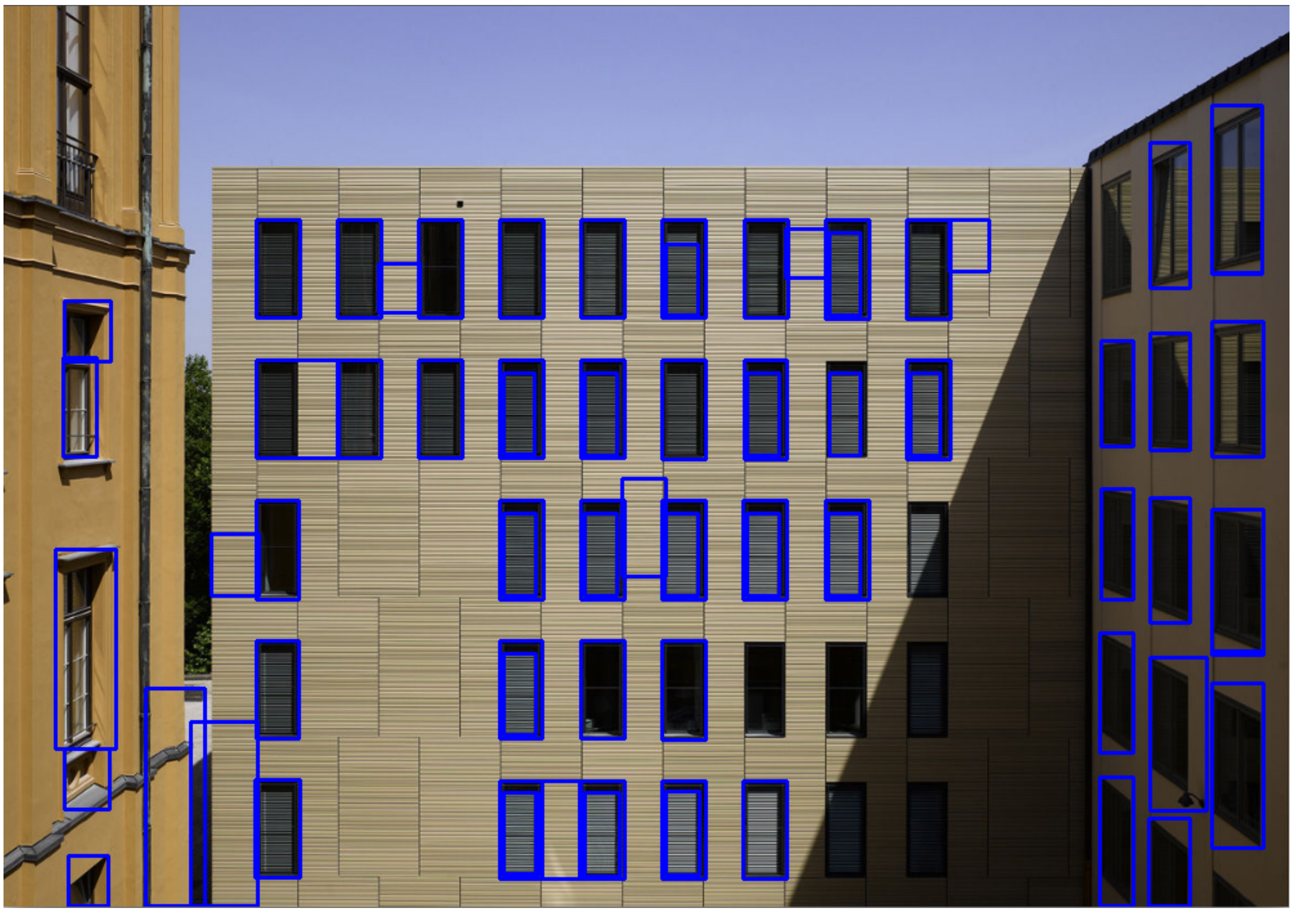


This method obviously has it’s drawbacks, such as the players near the top edge of the field,and players taking throw-ins get cropped off.

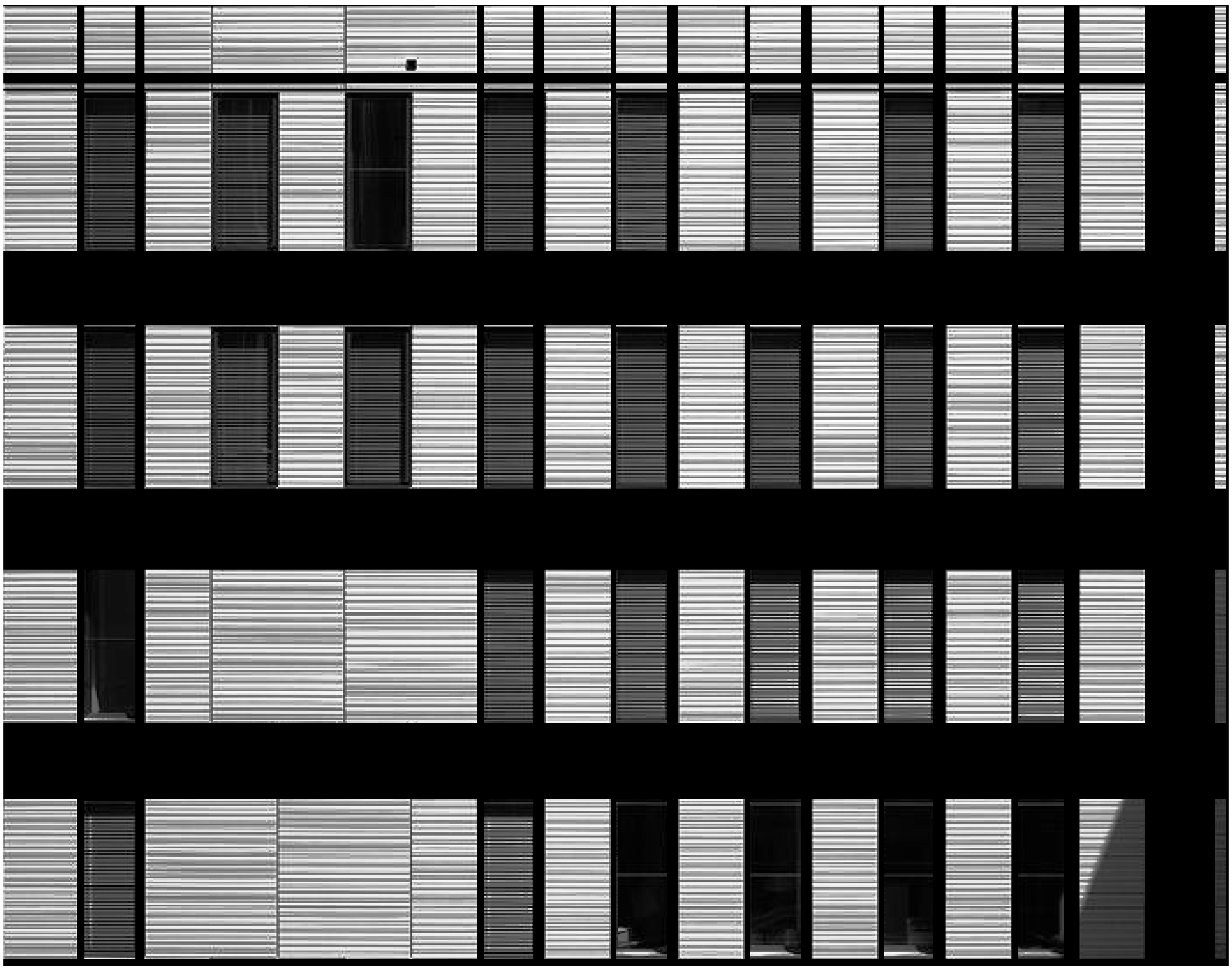
2. Write a code to detect windows in a given image. Try your code on images ‘building 1.jpg’ to image ‘building 5.jpg’. Upload the obtained results as well as the codes.

Looking at the image, we get an intuition to deal with the illumination variation in the image. We first load the image in grayscale and perform a histogram equalization to ensure the image has equally distributed color. We recognize that each building has texture, and that may produce noise while trying to detect the windows, so we selectively blur the image using a bilateral filter to preserve the edges in the image. Now, we can estimate the edges of the image using Canny’s edge detection algorithm. We now have a very noisy version of our original image. For each object in the image, we can iterate over each contour produced by the edges, and show a bounding box if that object’s bounding box matches the template of a window(Simple conditions like Area is non zero and height of the window is greater than the width).

The result is shown below.



Originally I tried segmenting the image based on directional histograms, but this method ended up giving me a lot of variation and putting the image back together was problematic. In general, this method could be used directly. The results for which seem promising, but they miss out the edges on the left and right fascades.



Combining the above two methods, maybe to eliminate extraneous boxes would help.