Template Matching. What do we want to achieve? We want to describe how successfully a match is found for a template when matching it with an image. Main Theory: Use filtering methods.

Correlation. All the implementations of template matching (that we use) utilizes variations of correlation, to compare templates with image segments. The technique is the same as filtering with various predetermined kernels, like gaussian, sobel, box-filter etc. , to gain various filter effects, but instead we use an actual image template as a kernel and measure the difference between this and a section of the image we want to look through.

Normal correlation (un-normalized) is mathematically described by:

h[m,n]=g[k,l] f[m+k,n+l]

Using this method for matching, is prone to a lot of "false" positives, as the result is dependent of the actual pixel values, not determined by the degree of match between the filter and the image segment. The sum of the multiplied pixel values, will be large in bright areas of the image and vice versa in the dark areas. To make sure the correlation also considers the overall brightness of the segment, a normalized version of the formula is introduced.

If you step out one level of abstraction and view the template H and image segment F as two vectors (achieved by joining each row or column after each other), the correlation can be viewed as finding the dot product between the two vectors. H dot F. (Just take a second and realize that the dot product is found the same way as we calculate correlation). As we know a vector can be normalized by dividing it by its length, we can also normalize the dot product by dividing it by the multiplication of the length of the two vectors. the relationship is described in the formula:

H dot F = ||H|| * ||F|| * cos V
cos V = (H dot F) / (||H|| * ||F||) , where V is the angle between the two vectors.

The normalised expression will always give a result between -1 and 1 as cos V [-1,1], and since the image vectors only contains positive numbers the result will be betweem zero and one.

The length of each the two vectors are found with the formula: (here with a vector in 3 dimensions)

Translating back from this abstraction level and looking at correlation again the formula is translated into:

Normalized Cross Correlation (x,y) = g[k,l] f[m+k,n+l]

This can be refined further by subtracting the means for both template and image patch. This gives us the zero mean normalized cross-correlation, AKA correlation coefficient:

This gives a very accurate result, but is pretty slow aswell.

Another implementations using correlation:

Sum of squared difference: This compares each corresponding pixels describing the difference between them with a calculated positive number. (because of the squared) The closer to zero the final result is, the better the match. The image viewed (where white equals good results) are calculated with: image (x,y) = 1 - sqrt(result of above expression).

Our Implementation: def GetEyeCorners(img, leftTemplate, rightTemplate,pupilPosition=None): #The method parameters are: the image, templates and optional pupil position. sliderVals = getSliderVals() #Enable adjustment from slidervalues for match threshold. matchLeft = cv2.matchTemplate(img,leftTemplate,cv2.TM_CCOEFF_NORMED) matchRight = cv2.matchTemplate(img,rightTemplate,cv2.TM_CCOEFF_NORMED) #The openCV library allows use of different matching methods. Here we use COOEFF_NORMED = normalized cross corellation. #Matching both for left template and right template. if (pupilPosition != None): #If the pupil position is set, slice the image in two halfs at the pupil position. pupX,pupY = pupilPosition matchRight = matchRight[:,pupX:] matchLeft = matchLeft[:,:pupX] matchListRight = np.nonzero(matchRight > (sliderVals['templateThr']*0.01)) matchListLeft = np.nonzero(matchLeft > (sliderVals['templateThr']*0.01)) #Sort out the results with values below the match threshold. matchList = (matchListLeft,matchListRight) return matchList

Our Results: (Also see video) We found that, when using the normalize cross corellation, a good starting threshold is 0.85. That doesn't draw to many detections from the start and, provides a good starting point for further adjustments. All the tests runs utilizes the pupil position, allowing matches for the left template only in the left side and opposite for the right template matches. Test 1 - Unknown Subject.

As we see here, with at threshold of 0.85, things work out fine, when there are not too much movement in the image. In controlled light, controlled subjects, the correlations methods works like a charm. As soon as the subject moves too much, the threshold has to be lowered to find any results. It quickly becomes a problem having a shared threshold for both template matchings (left & right), as the circumstances around the two places in the image are quite different. Lowering the threshold even more, allows the subject to move more freely and still finding matches in both sides. Again we see a big difference in which sides that struggle and which that finds loads of matches. Test 2 - Young Master Ghurt - recorded tuesday 12/03/13

Starting out with a hig threshold of 0.92 seems to work quite well. When the template contains enough contrasts/diversity, and the subject remains still, the errors are few.

It doesn't take a lot of change before the threshold has to be lowered to find a result. Again there is a huge difference on the sides, and we can conclude that a shared threshold wouldn't be suitable in an industrial/finished solution.

As we continue the path downwards lowering the threshold to find matches at

the sides, a lot of false results now appear, as the match no longer have to fit perfectly.

Concluding on the results:

Improving the method: