ANPR minimum width

nathan.george **Jul 26**

I figured this would be easier to find in the future, and easier formatting to read than a private message:

In **6.5: Scissoring the license plate characters**, this part made no sense to me:

```
# loop over the contours
for c in cnts:
        # compute the bounding box for the contour while maintaining the minimum
        (boxX, boxY, boxW, boxH) = cv2.boundingRect(c)
        dX = min(self.minCharW, self.minCharW - boxW) / 2
        boxX -= dX
        boxW += (dX * 2)
```

We're not maintaining the minimum width, we're ensuring a uniform width.

The result of
`min(self.minCharW, self.minCharW - boxW)`
will always be
`self.minCharW - boxW`

I threw in some debugging to see what was going on:

```
# loop over the contours
for c in cnts:
        # compute the bounding box for the contour while maintaining the minimum
        (boxX, boxY, boxW, boxH) = cv2.boundingRect(c)
        dX = min(self.minCharW, self.minCharW - boxW) / 2
        print 'minCharW:', self.minCharW
        print 'boxW:', boxW
        print 'min result:', min(self.minCharW, self.minCharW - boxW)
        print 'old boxX, W:', boxX, boxW
        boxX -= dX
        boxW += (dX * 2)
        print 'new boxX, W:', boxX, boxW
```

and found that sometimes the boxW is actually being set to 1-minCharW, I think due to rounding:

```
'min result:', 3
```

```
'old boxX, W:', 345, 37
'new boxX, W:', 344, 39
'minCharW:', 40
'boxW:', 41
'min result:', -1
'old boxX, W:', 206, 41
'new boxX, W:', 207, 39
'minCharW:', 40
'boxW:', 34
'min result:', 6
'old boxX, W:', 127, 34
'new boxX, W:', 124, 40
```

I think this should be how the code is implemented:

```
# loop over the contours
for c in cnts:
        # compute the bounding box for the contour and ensure a uniform width
        (boxX, boxY, boxW, boxH) = cv2.boundingRect(c)
        dX = (self.minCharW - boxW) / 2
        boxX -= dX
        boxW = self.minCharW
```

and minCharW should be charW.

Now, after going through **6.6: Our first try at recognizing license plate characters**, I've found that we are going to extract the contour of the number/letter anyway, and rescale it:

```
@staticmethod
def preprocessChar(char):
        # find the largest contour in the character, grab its bounding box, and c
        (cnts, _) = cv2.findContours(char.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_AP
        c = max(cnts, key=cv2.contourArea)
        (x, y, w, h) = cv2.boundingRect(c)
        char = char[y:y + h, x:x + w]

        # return the processed character
        return char

def describe(self, image):
        # resize the image to the target size and initialize the feature vector
        image = cv2.resize(image, (self.targetSize[1], self.targetSize[0]))
        features = []
[funct clipped]...
```

So the whole ensuring a uniform width thing from 6.5 doesn't matter, right? We could just ignore that part of the process?

Cheers,
Nate

Adrian Chief PyImageSearcher **Jul 27**

Thanks for sharing the code and debugging information **@nathan.george** .

And yes, you are correct, we are ensuring a uniform width. I'll have to change the comment in the code.

However, it *does* matter that we maintain a uniform width. Consider for instance the digit "1". If we extracted the standard bounding box, the entire ROI would look be (essentially) white since a "1" is just a vertical line. This makes it harder for our block-binary pixel sum + machine learning algorithm to recognize the difference in characters. By ensuring a uniform width for each digit, we also ensure each digit is described with respect to the original image. If you remove this code, you'll actually notice that the classification accuracy of the characters decreases.

---

nathan.george **Jul 27**

Ah, right, I wasn't thinking about the BBPS algo 🍻