Yield Sign Detection

Felix Hamann

June 3, 2017

Contents

Chapter 1

Introduction

This document describes the implementation of a system to detect yield signs from images.

1. *Notation definitions*

    (a) *Always height x width*

    (b) *For binary images: 1 corresponds to white*

    (c) *For every step in the pipeline there is 'source' and 'target'*

2. *Image test set*

3. *Requirements*

    (a) *Changing light (night, day, dawn, dusk)*

    (b) *Image sizes*

    (c) *Performance*

Chapter 2

The filter pipeline

...

1. Binarization

   (a) Red Amplification

   (b) Binarization

2.1 Binarization

The first step of the whole pipeline consists of extracting the red color from the coloured input image *source*. Thus a mapping for each pixel from three dimensions to one dimension is needed. Simply returning the red color component does not suffice, as with larger values of the green and blue components the source color either shifts towards yellow, magenta or white. A very simple method that is used in this application takes the red component, optionally amplifies it by some factor and substracts the green and blue components values.

Let $\Gamma = \{0, 1, ..., 255\}$ be all possible pixel values, $\delta \in \Gamma$ the threshold and $\alpha \in \mathbb{R}, \alpha \geq 1$ the factor for red amplification, then the pixel written to the binarized image *target* is:

$$f : \Gamma \times \Gamma \times \Gamma \to \mathbb{Z}$$
$$f(r, g, b) = \alpha r - (g + b)$$
$$target(y, x) = \begin{cases} 1 & \text{if } f(source(y, x)) > \delta \\ 0 & \text{else} \end{cases}$$

(2.1)

Color	Red	Green	Blue	f(r, g, b)	target(y, x)
■	0	0	0	0	0
	255	255	255	-51	0
■	0	255	0	-255	0
■	0	0	255	-255	0
■	255	0	0	459	1
■	255	125	125	209	1
■	255	125	0	334	1
■	255	0	126	334	1

Table 2.1: Example results of function f in 2.1 with $\alpha = 1.8, \delta = 200$. Note that the result is equivalent for light red, orange and magenta.

This method has one drawback however. There is no distinction between less saturated or darker shades of red and orange or magenta. This is a result of not taking the distance between green and blue into account. An example is depicted in Table 2.1. More elaborate variations of this calculation were tested (considering the ratio of red to all colors or weighting by green-blue distance) but this did not increase the overall quality of red detection. Actually, without constantly white balancing the camera used for capturing, narrowing the range of red would perform worse as the ambient light differs greatly over the course of a day. The drawback of this approach are more falsely marked areas of the image that have to be considered in all further steps.

Performance, Pictures

2.2 Morphological operations

At this point of the pipeline, the *source* image is a binary image with all areas considered red marked true. The quality of the image and found red areas depends greatly on a variety of factors. For one the camera itself might introduce pixel errors to the picture, resulting in falsely colored pixels (when the sensor is of inferior quality or getting old) or blacked out areas (e.g. when the lens is dirty). On the other hand the to-be-detected yield sign could be dirty, altered by vandalism or simply reflect light which would obscure its shape. As it is essential to expose the signs form for best detection, some morphological operations may be applied for noise removal.

The system implements two morphological operations that are separately adjustable. These operations are called dilation and erosion. This is called "closing" when combined. The purpose and functioning of these algorithms are described below.

2.2.1 Erosion and Dilation

Main purpose of erosion is to remove white noise from the binary image. For every pixel of the *source* image, some range around the pixel is considered. This range - as used the current implementation - can be seen in 2.2 and is commonly called "structuring element" *S*. Now, for every pixel in the *source* image, the mask is applied and the target pixel is set based on 2.3. Commonly speaking, the target pixel is only set if all surrounding pixels masked by *S* are also set.

$$S = \begin{bmatrix} - & source(y-1, x) & - \\ source(y, x-1) & source(y, x) & source(y, x+1) \\ - & source(y+1, x) & - \end{bmatrix} \tag{2.2}$$

$$target(y, x) = 1 \iff \bigwedge_{s \in S} s \neq 0 \tag{2.3}$$

The functioning of the algorithm is depicted in 2.4. Closed white components shrink by the factor determined by the size of *S*. Spurious grains of white are eliminated completely. Thus the erosion is used to remove white noise on black backgrounds such as the grain on the bottom right of the example.

$$source = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \mapsto target = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.4}$$

The dilation operation alters the image in such way that closed white forms expand by the factor determined by *S*. Grains of black are removed from white backgrounds. The algorithm itself works analogous to the erosion algorithm but pixels in the target image are set if any of the masked source pixels is white. The equation changes accordingly and is defined in 2.5. An example is given in 2.6. It can be observed that the white form grows and the black grains inside that form vanish. Note that the form is not smoothed and the cut propagates to the target.

$$target(y, x) = 1 \iff \bigvee_{s \in S} s \neq 0 \tag{2.5}$$

$$source = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \mapsto target = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad (2.6)$$

2.2.2 Opening and closing

When erosion and dilation are combined they form an operation called opening. This is due to the fact that if any forms are connected by thinner areas this connection vanishes through erosion and the original forms size is restored by dilating again afterwards. The opposite effect occurs when first dilating and then eroding the image. Here, thin connections are strengthened and connections between forms may be established. This proves useful if a closed yield sign shapes are required even if somebody took a pencil and drew over the red area or light reflections break the red surface.

2.2.3 Application to the system

As stated before, closing is embedded to the system. Erosion nearly never proved useful however. Most of the time the already small red areas spanned by the yield signs are lost. This leads to never detecting any yield signs that are far away. The number of iterations are free parameters of the system and may be adjusted. Using one or two iterations of dilation and one or zero iterations of erosion work fairly well.

Picture examples

2.3 Component labeling

This optional step tries to greatly reduce the impact of red image areas that are not yield signs. The basic idea is that yield signs enclose a white triangle and thus there are black areas enclosed by closed white borders in the binary image. If all black areas are labeled, the label with the most associated pixels must be the background - assuming the yield sign is not the most prominent object captured. All forms that do not enclose any area vanish that way, greatly reducing the amount of considered objects. Because the algorithm is defined for labeling white areas, the source image needs to be inverted first.

The algorithm works by iterating the image from top-left, selecting all white pixels and checking whether a new label needs to be introduced or - if the neighbors are already labeled adopt that label for the current pixel. Formally, given a mask M and a set $L = \{1, 2, ...\}$ of unassigned labels, the target pixels value

is given by formula 2.8. Because multiple values can be returned by *M*, another set *E* is needed to store all equivalents. After the first labeling step, the equivalents are resolved by assigning the smallest label of possible candidates. This algorithm is called two-pass algorithm and the form of *L* checks for 8-connectivity as all eight neighboring pixels are taken into account.

$$M = \begin{bmatrix} source(y-1, x-1) & source(y-1, x) & source(y-1, x+1) \\ source(y, x-1) & source(y, x) & - \end{bmatrix} \tag{2.7}$$

$$target(y, x) = \begin{cases} min(L) & L = L \setminus l & \text{if } \forall m \in M : m = 0 \\ any(M) & E = E \cup \{(m, n)\} & m, n \in M & \text{else} \end{cases} \tag{2.8}$$

After labeling all distinct components of the image, the component with the largest amount of associated pixels is set to white and the image is again reversed. Although this step proves to be very effective it has one big drawback: As soon as the detected yield sign shape is not closed the whole detection system fails immediately. This does not happen otherwise as even without a closed form, the other edges are prominent enough to allow detection. Thus this step can be optionally enabled or disabled.

Picture examples

2.4 Edge Detection

To be written

2.5 Line Detection

To be written

2.6 Yield Sign Detection

To be written

2.6.1 Finding points of intersection

The vectors returned by the hough transform contain the respective values describing each found line by their normal vector and distance from the images' origin. A simple method for detecting points of intersections is to create homogeneous vectors and use their cross product. If the third component is

zero, no point of intersection is found. Let A be a vector containing all radii and Δ contain all distances for n found lines. Thus for any $i \in 1, \ldots, n$ the homogeneous vector h_I is defined as:

$$A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$$
$$\Delta = (\delta_1, \delta_2, \ldots, \delta_n)$$

(2.9)

$$v_i = \begin{pmatrix} sin(\alpha_i) * \delta_i \\ cos(\alpha_i) * \delta_i \\ 1 \end{pmatrix} \qquad w_i = \begin{pmatrix} cos(\alpha_i) \\ -sin(\alpha_i) \\ 0 \end{pmatrix} + v_i \qquad h_i = v_i \times w_i \qquad (2.10)$$

Now all 2-permutations $(i, j) \in \pi_2(n)$ are tried for calculating intersections $p_{i,j} = h_i \times h_j$. Thus the set of all intersections is $\{(y, x) | y = \frac{p_{i,j,0}}{p_{i,j,2}}, x = \frac{p_{i,j,1}}{p_{i,j,2}} \forall p_{i,j,2} \neq 0\}$. Also, to keep all further computation small even if a lot of lines are found, the points of intersection are filtered by whether there is some red to be found nearby. To do so, the result of the morphological computations (section 2.2) is consulted and the point of intersection is kept if any white pixel is found there. The points are further filtered by whether they are inside the image plus some boundary. Even for truncated signs the relevant point of intersection is near the image boundary. The parameters for controlling the size of both the lookup area and the image boundary are free parameters. The intersections are saved as an unambiguous mapping of $min(i, j) \rightarrow max(i, j) \rightarrow (y, x)$

2.6.2   Determining Triangles

2.6.3   Filtering Triangles for Yield Signs

To be written

Chapter 3

Evaluation and Conclusion