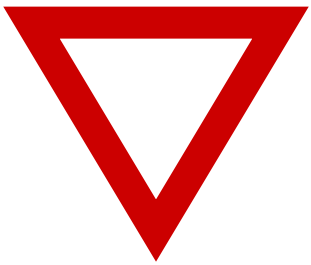


Yield Sign Detection

Felix Hamann

June 8, 2017

Contents

Chapter 1

Introduction

This document describes the implementation of a system to detect yield signs from images and videos. It is a proof of concept realization based on a course of machine vision at RheinMain University of Applied Sciences in 2017. The scope of the task is to implement some operations on images including but not restricted to color segmentation, morphological operations, hough transform for lines and highlighting found yield signs. This chapter describes the model, requirements and notation used in this document. Chapter 2 outlines a pipeline of operations and all utilized algorithms and heuristics for detecting yield signs. The following chapter 3 describes the implementation of a graphical system to interactively adjust free parameters to test detection and the transition of the pipeline for use in videos. Chapter 4 highlights what the current implementation is able to detect, describes cases where detection fails and discusses performance considerations. The final chapter contains a prospect for utilizing this system.

1.1 Model

Given that detecting yield signs with high accuracy is a hard problem as many factors complicate the detection process some constraints apply for working examples. The color values of traffic signs are standardized in Germany. Most signs color is faded due to their age however and when capturing pictures the daytime and weather also alter the red value of the sign. For this proof of concept yield signs have distinct closed red border and are prominently placed in the picture. The images are of high quality and may have various resolutions. Videos are constrained likewise and have a resolution of 480p.

1.2 Requirements

The system should be able to detect yield signs from the perspective of a car participating in traffic. This means that the point of view is approximately between one and two meters from the ground on the right lane[1] of a street. For testing purposes both small images with inferior quality and high resolution images should be classified. This is necessary to make a prediction whether the system is suitable for real-time use in videos. For videos a real time application is not needed but a thorough evaluation and prediction should be made whether it would be possible to do so.

1.3 Notation

Images are either two or three dimensional. The first dimension is the images' height, the second is the images' width. For three dimensional images, the third axis contains the three color components red, green and blue (in this order). Origin of each image is the top left corner and iteration is row-major. This is reflected when addressing single pixel values. Given some color image I, then $I(y, x, 2)$ returns the blue color components pixel value of y in the images' vertical and x in the images' horizontal direction. Color and gray scale pictures have a range of $\{0, 1, ..., 255\}$. Binary images are two dimensional and every value is in $\{0, 1\}$. Thus, boolean operations can be applied. Pixel values of 0 are called "black" or "false" and values of 1 are called "white" or "true". Each step in the pipeline that is described in chapter 2 has a $source$ image which is the result of the previous step of the pipeline and a $target$ image that is written to.

[1] Except for countries with left hand traffic that are not considered here.

Figure 1.1: Result of detecting a yield sign. The shape of the sign and an indicator is drawn to the result image.

Chapter 2

The pipeline

For yield sign detection a modular system of components is used. Each step of the pipeline takes at least the result of the previous step as input for processing. Simply put, color images are fed into the pipeline and a set of yield sign descriptions is returned. Each element of this set contains all three vertices and the center of the sign.

The pipeline itself consists of the following steps: First the image is segmented to extract areas which resemble the red color of a yield sign. A threshold is introduced to create a binary image masking all those areas. This is described in section 2.1. The next step applies morphological operations to enhance the quality of the extracted areas. A description of this can be found in section 2.2. The following section 2.3 describes how the amount of relevant red areas can be greatly reduced by filling the "background" of the image utilizing component labeling. As this proved quite useful but hindered detection completely for some corner cases, this step can be enabled optionally. The now remaining relevant areas are trans-

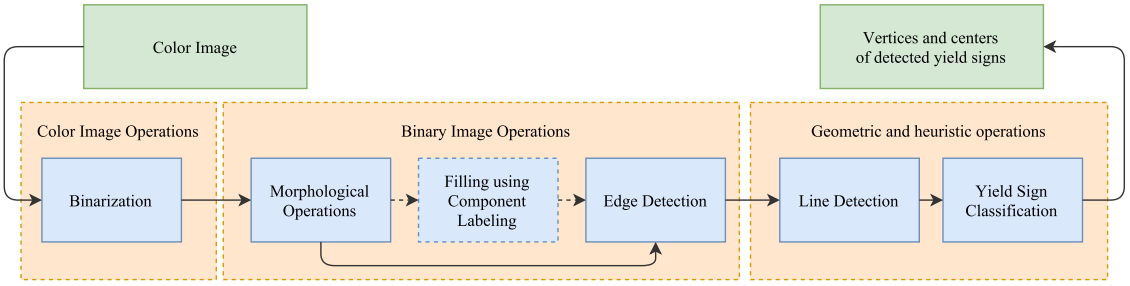


Figure 2.1: A high level overview of the processing pipeline. The various steps can be grouped by the domain they are working on. The first group handles colored images, the second group works with binary images and the third group with abstract geometrical representations.

Figure 2.2: Color segmentation to create a binary image with red areas marked true.
Used parameters are $v_{ref} = (180, 20, 20)$ and $\delta = 100$.

formed in such way that their shapes are defined by their edges. Section 2.4 describes this and the succeeding section 2.4.1 explains how this information can be used to detect and extract lines from the image. After all lines are extracted section 2.5 discusses how some heuristics are applied to determine all relevant triangles and classify whether those triangles could be yield signs. A graphical depiction of this process is given in 2.1.

2.1 Segmentation

The first step of the whole pipeline consists of extracting the red color from the colored 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. The method used in this application takes a predefined red color vector, calculates the distance between the currently considered pixel value and applies a threshold for binarization to the target image. Equation 2.1 describes this mapping formally. Let $\Gamma = \{0, 1, ..., 255\}$ be all possible pixel values, $v_{ref} \in \Gamma^3$ the reference color and $\delta \in \mathbb{N}$ the threshold, then the pixel value written to $target$ depends on whether the threshold undercuts the Frobenius norm.

$$
\begin{aligned}
norm &: \Gamma^3 \to \mathbb{R} \\
norm(v, w) &= \sqrt{\sum_{1 \leq i \leq 3} (v_i - w_i)^2} \\
target(y, x) &= \begin{cases} 1 & \text{if } norm(source(y, x), v_{ref}) < \delta \\ 0 & \text{else} \end{cases}
\end{aligned}
\tag{2.1}
$$

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.

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

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

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.

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

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. To close even these cuts, a new structuring element can be used which returns all eight neighboring pixels instead of just four.

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

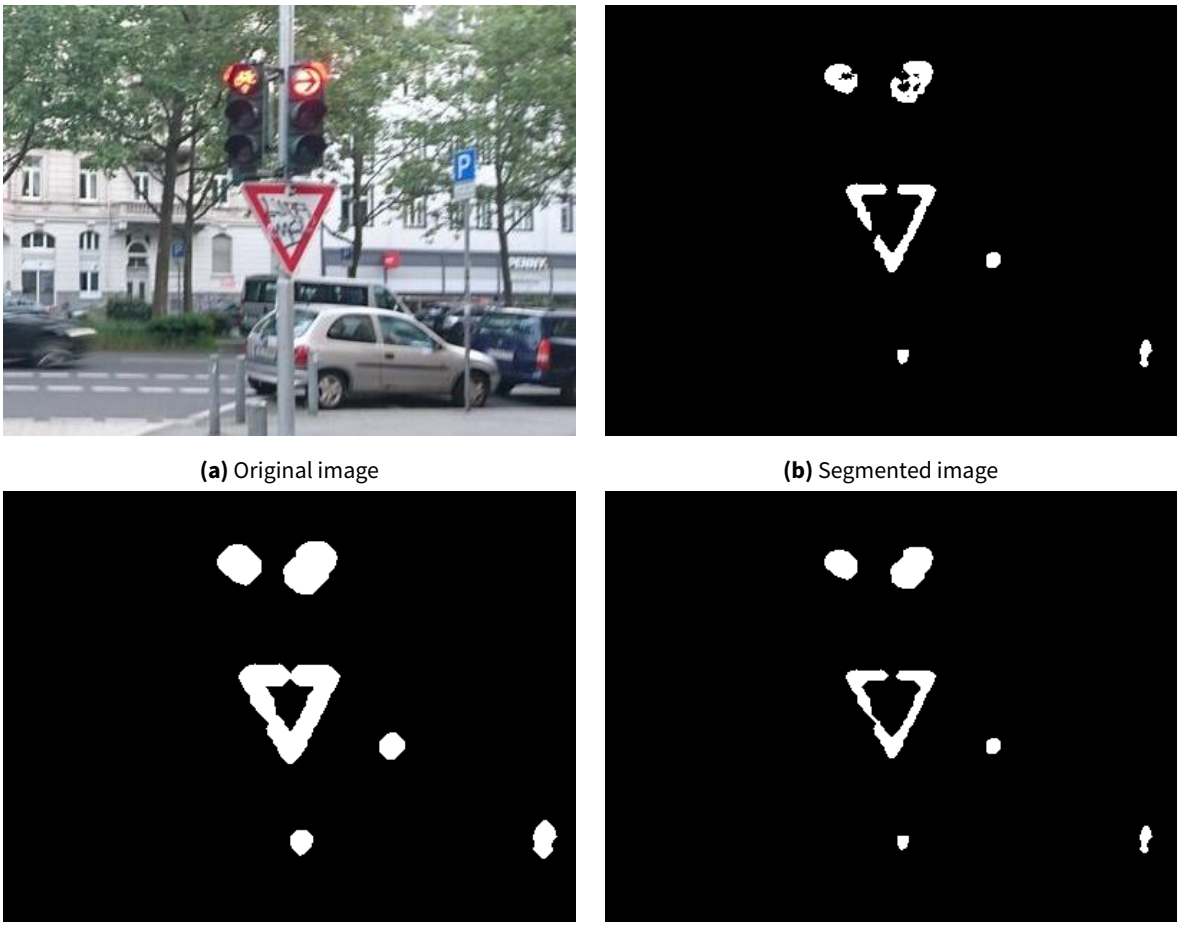
$$source = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad \mapsto \quad target = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1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

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. The number of iterations are free parameters of the system and may be adjusted. Introducing another erosion step to remove white noise mostly resulted in eroding the yield sign itself and is therefore no longer part of the pipeline.

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.



(a) Original image

(b) Segmented image

(c) After dilation

(d) After erosion

Figure 2.3: Closing applied to an image with a vandalized yield sign. Note how the form is closed in 2.3d on the left side. However, the number of iterations were not sufficient to close the form completely. All noise is removed from the red lights.

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.

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

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

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 optionally be enabled or disabled. An example where this step proves quite useful can be found in figure 2.4

2.4 Edge and Line Detection

The next step of the pipeline consists of detecting straight lines in the image, selecting triangles based on their points of intersection and applying heuristics to classify whether they might have emerged from a yield sign in the original image. The algorithm implemented in the application uses another dilation step for edge exposure and straight line hough transform for line detection. Note however that this is comparatively slow and thus another approach is described also. This can not be evaluated however as it was not implemented but should display much better performance.

2.4.1 Dilation and straight line hough transform

The goal of applying a hough transform to an image is to detect lines. Thus both for accuracy and performance reasons one pixel wide edges need to be extracted from the binary source image. A simple approach is to apply one iteration of dilation and XOR'ing the result with the source image. The inner white area is punched out that way. An example is given in figure 2.5.

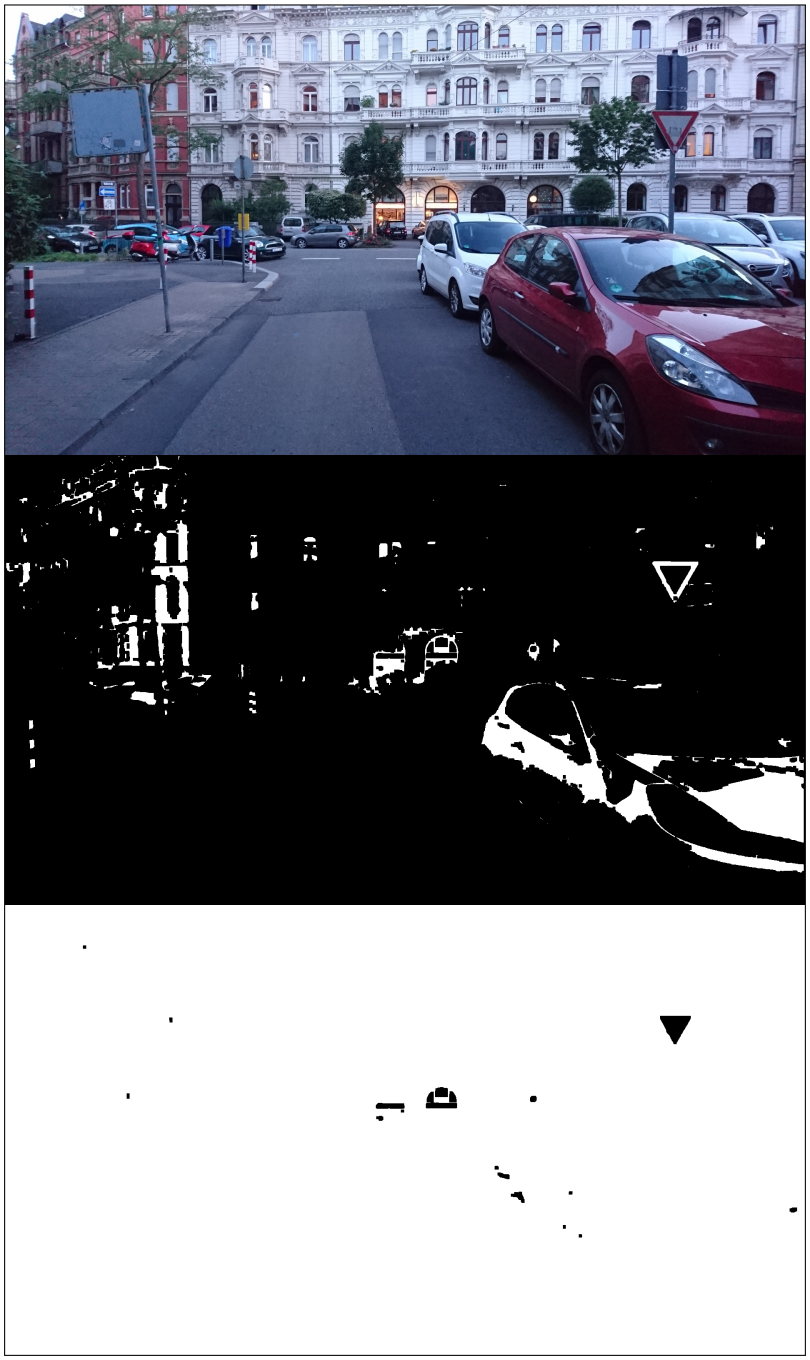


Figure 2.4: This image exposes multiple problems for detection as both a lot of more or less red objects can be found and it was captured at dawn where due to the low light intensity the color variance diminishes. Note however that due to the filling step utilizing component labeling most of the interfering objects vanish. The middle image exhibits the state after applying segmentation and closing.

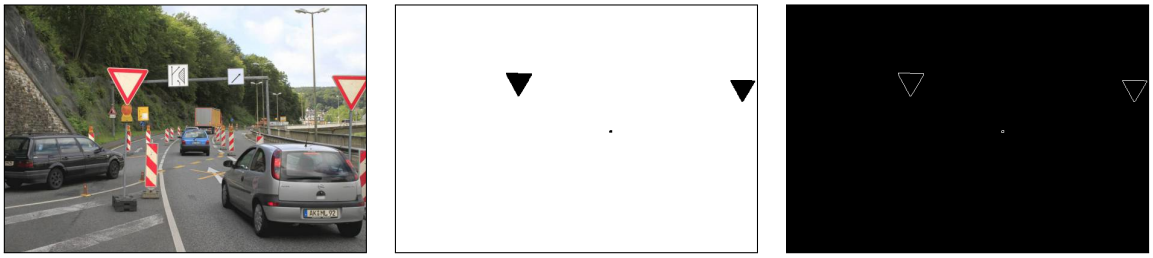


Figure 2.5: The middle image exhibits the state after the component labeling step and the right image only exposes one pixel wide edges of the found shapes using dilation and XOR.

Geometrically lines can be expressed by the Hesse normal form. In this notation, a line is expressed by its orthogonal normal vector and its distance from the coordinate systems origin. This normal vector is defined by its angle. An equation for the line is given in 2.9. For detecting lines with the hough transform a matrix is created that counts for each angle and distance the amount of white pixels found in the source image. In this application, angles range from -90 degrees to 90 degrees and the distance may be negative. Thus the hough matrix (given h for the images height and w for the images width) is of size $180 \times r$, where $r = 2 * \sqrt{h^2 + w^2}$. The first dimension contains all angles α and the second dimension all distances δ.

$$g : x * cos(\alpha) + y * sin(\alpha) = \delta \tag{2.9}$$

The basic idea of the hough transform for detecting lines is now to consider for every white pixel found in the source image the possibility that it belongs to one or more lines. Thus for each white pixel the hough matrix is incremented at every possible angle for that pixels distance from the origin. Peaks in the matrix indicate that many pixels with the same distance from the origin were found for the respective angle. Selecting these peaks lead to a set of possible lines, expressed by angle and distance. An example hough matrix can be seen in figure 2.6

2.4.2 Sobel and fast hough transform

The former approach is computationally expensive as for every white pixel found the whole range of angles needs to be incremented. The only reason for doing so stems from not knowing from that single value which angle corresponds to the possibly found line. When detecting edges in the image the Sobel filter can be applied. This filter calculates the direction depended gradient of the gray value difference of consecutive pixels. The direction is either horizontal or vertical. For getting the horizontal gradients of an image, a 2d convolution is applied using the source image and the matrices defined in 2.10.

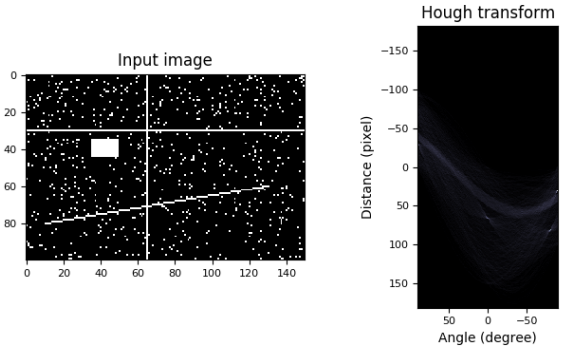


Figure 2.6: Hough matrix visualization with the distances in the first dimension (Image taken from the scikit-image project: scikit-image.org)

$$sobel_h := \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad sobel_v := sobel_h^T \tag{2.10}$$

Thus for getting the direction independent gradients, which is necessary to find the actual orientation of the edges, the convolution is applied both for the horizontal and vertical direction and the two resulting matrices $A_h = conv2d(source, sobel_h)$ and $A_v = conv2d(source, sobel_v)$ are combined.

$$gradients = \sqrt{A_h^2 + A_v^2} \tag{2.11}$$

The matrix $gradients$ now contains the length of each gradient at every pixel. For every gradient that exceeds some free parameter threshold the hough matrix is incremented with the following value of α.

$$\alpha_{y,x} = \frac{180}{\pi} * arctan2(A_{h,y,x}, A_{v,y,x}) \tag{2.12}$$

2.5 Yield Sign Detection

The result of the line detection in the previous section only returns all lines of the most prominent edges in the source image. There is no concept of closed shapes or even any classification for yield signs yet. How to detect shapes and discriminate yield signs from these shapes is described below.

2.5.1 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:

$$\begin{aligned} A &= (\alpha_1, \alpha_2, \ldots, \alpha_n) \\ \Delta &= (\delta_1, \delta_2, \ldots, \delta_n) \end{aligned} \tag{2.13}$$

$$v_i = \begin{pmatrix} sin(\alpha_i) * \delta_i \\ cos(\alpha_i) * \delta_i \\ 1 \end{pmatrix} \qquad w_i = \begin{pmatrix} cos(\alpha_i) \\ -sin(\alpha_i) \\ 0 \end{pmatrix} + v_i \qquad h_i = v_i \times w_i \tag{2.14}$$

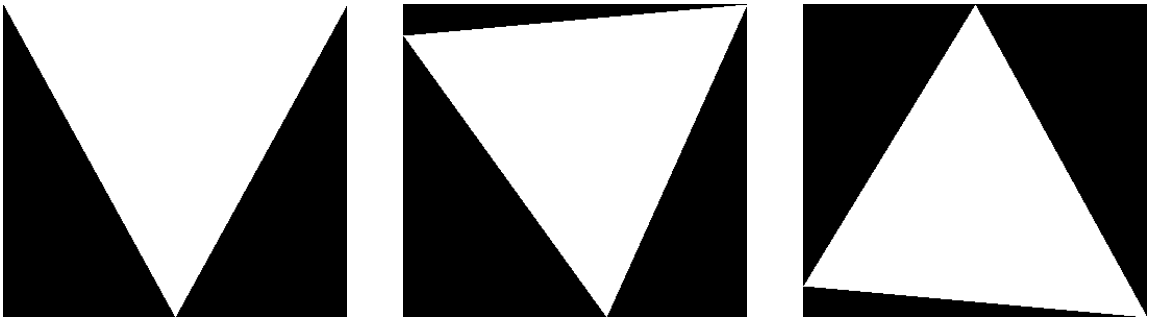
Now all 2-permutations $(i, j) \in \pi_2(n)$ are tried for calculating intersections $p_{i,j} = h_i \times h_j$. Thus the set of all intersections is $\{(y, x)_{i,j} | y = \frac{p_{i,j,0}}{p_{i,j,2}}, x = \frac{p_{i,j,1}}{p_{i,j,2}} \forall p_{i,j,2} \neq 0\}$. Also, to keep all further computation small 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. An example of why this a useful step is given in figure 2.7 where a triangle is falsely classified as a yield sign although there is no sign remotely close. 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 $T = min(i, j) \rightarrow max(i, j) \rightarrow (y, x)_{i,j}$.

2.5.2 Determining Triangles and Yield Signs

All the previously detected candidates are now iterated by checking whether any i, j, k exists where a combination of intersections is transitive, meaning $(i, j, p_0), (j, k, p_1), (i, k, p_2) \in T$. If the condition holds, than the three points p_0, p_1, p_2 are the vertices of a triangle and the tuple $(i, min(j, k), max(j, k))$ unambiguously identifies it. Now they are filtered further by discarding all triangles whose ratio does not fit (yield signs are nearly equilateral, even when taking perspective into account). Also signs can be discarded whose total size exceeds or undercuts some threshold. These thresholds can be calculated based on the images size.

The last step of detecting whether the found triangle actually corresponds to a yield sign consists of looking at the actual shape of the triangle itself. To do so, a reference triangle is created, mimicking the

Figure 2.7: An example where a triangle with the shape of a yield sign is found if no prior filtering of intersections based on detected red value is applied.



(a) The reference triangle

(b) Most likely a yield sign with $\Delta \approx 0.12$

(c) This could be a warning sign with $\Delta \approx 0.47$

Figure 2.8: Example results of equation 2.15

shape of a yield sign, and an equally sized image with a triangle having the shape of the detected one. Both images are binary images where the triangles are white. Let ref be the reference triangle image and tri the detected triangle, h the reference images' height and w the reference images' width, then the difference between the two is defined in 2.15. The resulting real number $\Delta \in [0, 1]$ expresses the similarity between the two and the larger that value gets the more different the triangles are. Whether the triangle is classified as a yield sign is controlled by a free parameter threshold. This threshold can be raised to detect yield signs that are distorted by perspective.

$$\Delta = \frac{1}{h * w} \left(\sum_{0 \leq y < h} \sum_{0 \leq x < w} ref(y, x) \oplus tri(y, x) \right) \tag{2.15}$$

Chapter 3

Implementation

A reference implementation of the previously described pipeline exists for both static images and video. For images a graphical interface is offered which allows for tuning free parameters and explore the capabilities of the system. Videos are processed by a script which, in its current form, does not allow real time processing. It offers performance metrics for evaluating whether it is suitable for real time use however.

3.1 The graphical user interface

The implementation sticks close to the definition of the pipeline. After loading some image which is displayed on a canvas where it can be panned and zoomed, the systems start to apply all steps of the pipeline and draws indicators for found yield signs if any were found. Different tabs can be clicked where steps of the pipeline are grouped and intermediate results displayed. Free parameters are easily adjustable by using the widgets on the left hand side. When altering any of those parameters, the pipeline is executed again and the results are updated. Figure 3.1 displays the steps where the image is segmented and morphological operations are applied. Figure 3.2 shows how the geometrical operations are visualized. The lines found by the hough transform are drawn in red and all considered points of intersection are marked with their shape indicating the area where some red was found.

3.2 Working with videos

When applying the system to videos, the pipeline can be used frame-wise for doing a full scan for yield signs. Computationally this is rather expensive. For speeding up the computation, knowledge extracted from prior frames can be used. If a yield sign is found, it is most likely to be found on the succeeding frames

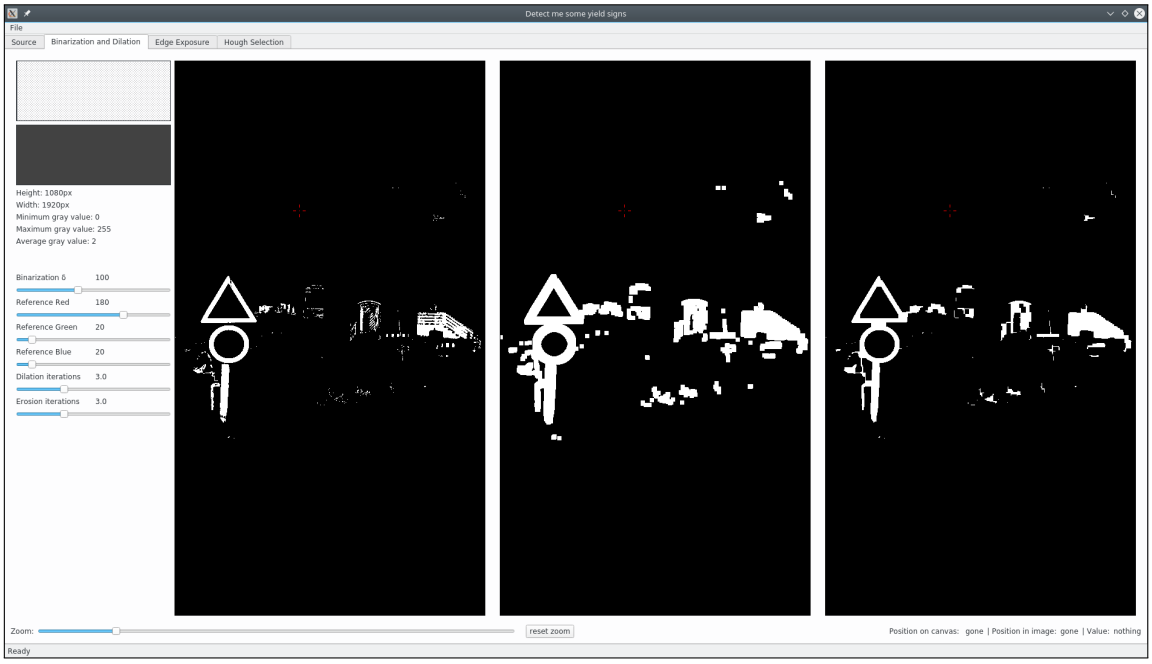


Figure 3.1: Selected tab of the GUI working with binary images. Note the sliders for adjusting the different free parameters used in those steps. The intermediate results are displayed on the respective canvases (e.g. the middle canvas where dilation was applied).

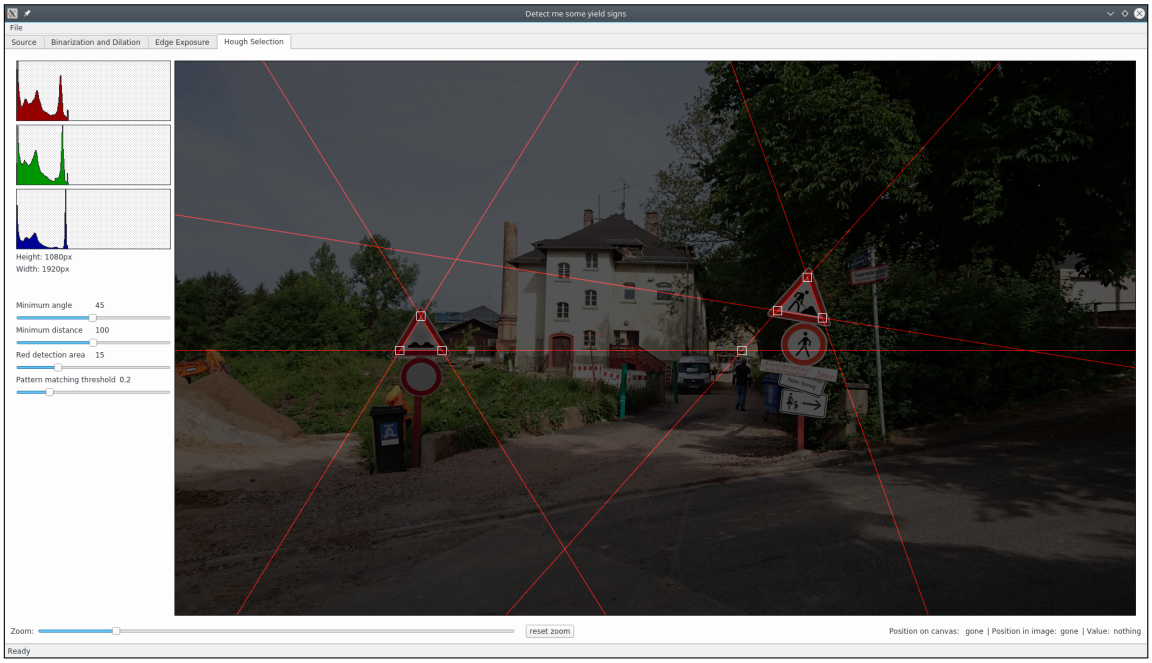


Figure 3.2: Visualization of the results of all geometrical operations. It can be seen that although some triangles are found, none classifies as a yield sign.

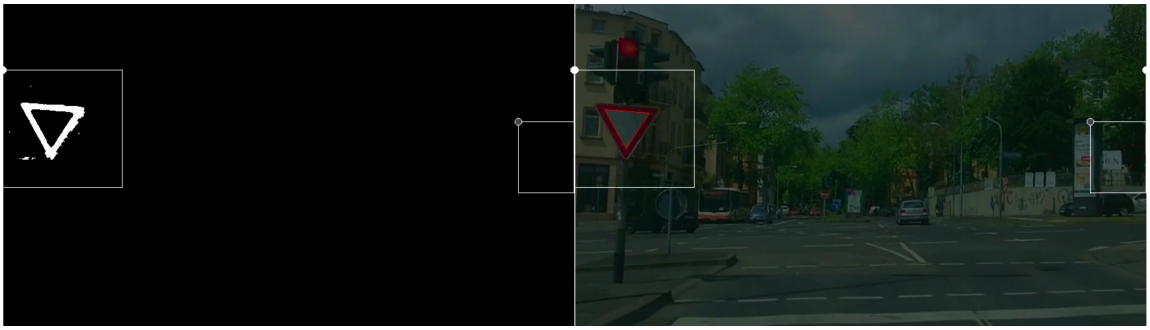
near its old position. Thus the region for checking for a sign can be reduced to this “region of interest” (ROI). To make the system stable and allow for detecting multiple ROIs at once the following algorithm was devised and implemented.

Every ROI is described by two points $p_0 = (y_0, x_0), p_1 = (y_1, x_1)$ which spans a rectangle with a prior detected yield sign. Also each ROI has an integer value > 1 assigned which is referred to as its health. Now for every frame - as long as no ROI was detected in the previous frame - a full scan is commenced. If no yield sign is detected, the next frame is read. If one or more yield signs were found a ROI is initialized for each yield signs. ROIs must not overlap as same traffic signs always have some physical distance. The succeeding frame only scans the areas defined by the ROIs currently held. Every so often a full scan is done nonetheless to detect additional yield signs. If no yield sign could be found inside a ROI it is not immediately discarded, but its health is decremented. This prevents premature deletion of ROIs due to temporary occlusion or other failure of the system. If the health of a ROI reaches zero it is considered dead and discarded. This happens mostly when the sign is no longer visible.

(a) No ROI was found so the full image is scanned for yield signs.

(b) The first of two yield signs was found. A ROI is established and only its area is scanned.

(c) After one of the sporadic full scans, the second yield sign was found and two ROIs exist.



(d) The right sign is no longer visible and its ROI loses health indicated by the circle which has a lower gray value.

(e) Both ROIs died and the systems resumes to do full scans.

Chapter 4

Evaluation

The system correctly identifies yield signs for nearly all examples that were given to it. It correctly discriminates yield signs from other warning signs such as construction work signs etc. Even for images which have many other red objects in it, the detection stays quite stable mainly due to the component labeling based filling. As this is only a proof of concept implementation no reliable misclassification rate can be supplied as even building the ground truth would go beyond the scope of this project.

Video processing - although technically not implemented for real time use - proved to perform promising. The current implementation is written in Python which has some overhead intrinsically. The used hough implementation is not the fast hough implementation described in 2.4.2. Half of the computation time is needed to do the hough transform and this would improve dramatically when switching implementations. Currently, video processing for full scans handles around 20 frames per second on one 3.3GHz processor. This means if only every second frame would be used (which does not decrease the quality of detection) a steady processing of 40 frames per second for 480p video material is possible. When working with ROIs, even the current implementation handles about 80 frames per second.

The system behaves reasonably well for easy detection problems but fails in bad light conditions or for obfuscated signs. The most prominent reason for failure can easily be pinpointed to the color segmentation step. It is crucial to detect the correct red to have a distinct form to work with. Nearly all real-world problems do not exhibit this property. The following chapter concludes with proposals to improve the current system.

Chapter 5

Conclusion

The results show that the pipeline is able to detect and classify yield signs in an ideal world. In its current form it is not able to cope with real world problems stable enough as it would be needed for full or semi autonomous driving vehicles or assistance systems.

As mentioned, color segmentation is crucial for a correctly working pipeline. The appearance of red changes drastically when captured under bad light conditions, with cheap camera sensors or simply when the sign itself is becoming old. It must be explored whether there is a more stable feature to rely the pipeline on.

To speed up computation for videos some areas of the video are not important. Given the camera is constantly in the same position pointing straight to the front of the car, only the top-left and top-right areas are necessary to analyze. Also, as mentioned before, an implementation in a language without interpreter overhead or utilizing a GPU accelerated system will speed up the computation.

Using heuristics to classify yield signs is a somewhat error prone approach. Additionally it does not scale for use with other types of road signs. For each sign a new method must be devised and this will lead to both classification errors and a system which is not easily maintainable. Recent advances in machine learning lead to highly effective topologies for convolutional neural networks trained to classify images or objects in images. They offer a more general approach for classification.

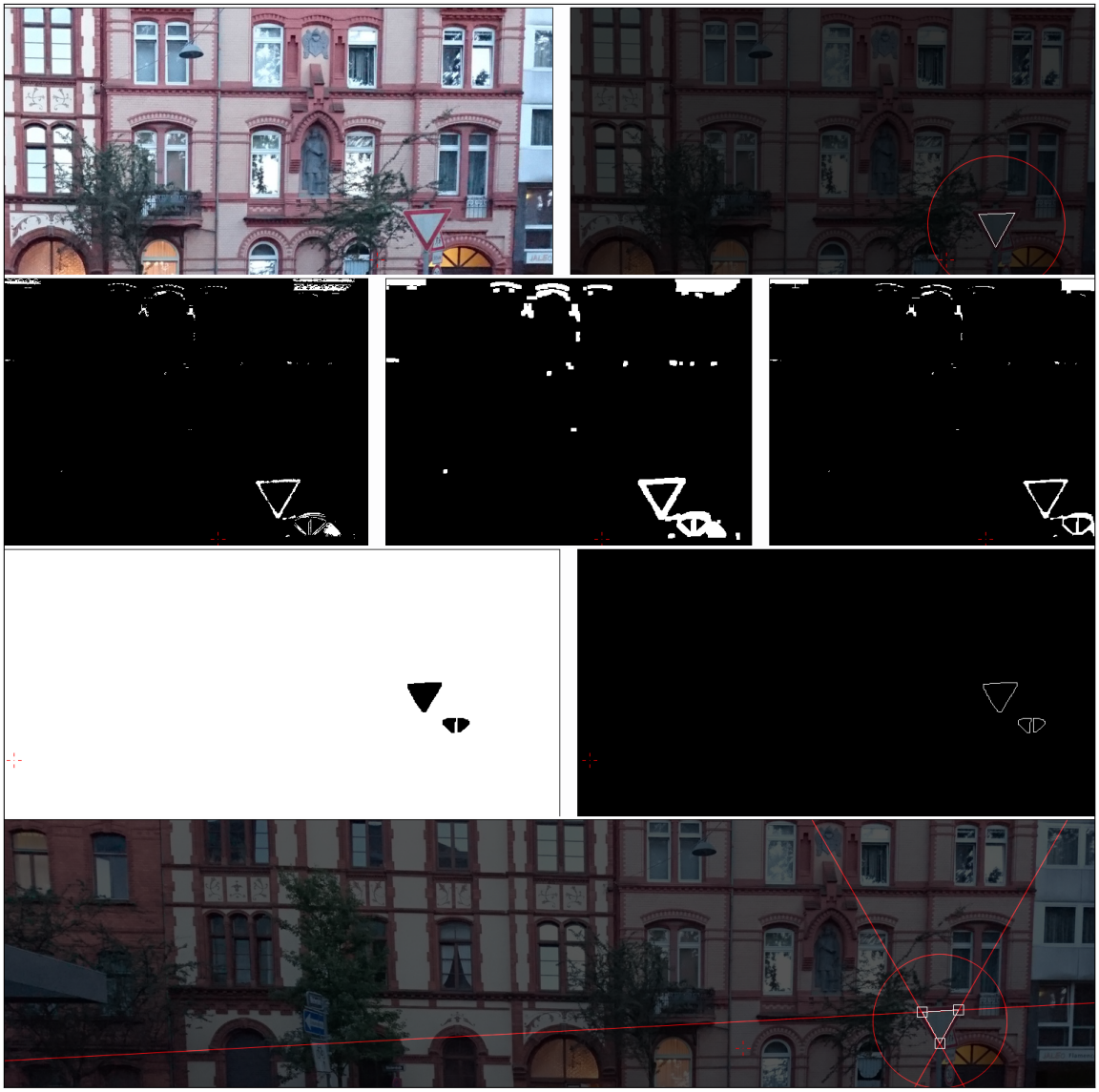


Figure 5.1: The whole pipeline in one picture