3

LISA Traffic Sign Dataset

When doing computer vision projects, training data and test data is essential. If algorithms that train themselves based on images are used, the need of training data is obvious. However, also simpler systems need training data that the algorithms can run on, to asses and adjust their performance. A test set is necessary for evaluating system performance. It is important that the system is not tested on the same data as it is trained on, since that cannot tell if the system generalizes to other data.

This chapter is about the traffic sign dataset that has been created in this project. The survey in chapter 2 showed that no dataset with American signs existed, an given that the ultimate goal of this project was to investigate detection of US traffic signs, creating such a database was necessary.

3.1 Introduction

When training and testing a system like this, it is generally preferable to use data from existing public databases as opposed to collecting data for the particular project. Not only is it saving time and effort, it is also convenient for comparing results to previous works in the area.

In more established fields, such as detection of people, a number of public databases exist. Notable examples are the MIT CBCL Pedestrian Database and the INRIA Person Dataset (Dalal and Triggs, 2005). In the realm of TSR, a few databases has recently emerged, as described in chapter 2, but none of them include US signs.

The recent emergence of these datasets is a very welcome addition to the field of TSR, as the performances of methods in previous papers have been very hard to compare without actually implementing them. Considerations on how the dataset created for this project has been assembled are presented in the following sections.

3.2 Methods

This section describes the methods used to obtain the LISA Traffic Sign Dataset. It describes both considerations about the actual data content, and the technical means with which is was collected. During this section there will be no distinction between training and test data. In the end, splitting the dataset into separate training and test pools is discussed.

3.2.1 Dataset content and structure

The design and content of the real-world dataset has been inspired by the other available datasets presented earlier. The more similarly packaged the different datasets are, the easier they are to use, so some effort has gone into this.

All significant existing datasets contain a number of images annotated with the type and position of signs. That is the bare minimum required for any traffic sign dataset. The German Traffic Sign Recognition Benchmark (GTSRB) dataset has significantly more classes and pictures than the STS dataset. The KUL set is also large and contains the most classes of all. However, the STS and KUL datasets one significant advantage: They include full frames, making them useful for both detection and classification systems. This is critical, especially since this project is on detection. Furthermore, the STS dataset includes additional annotation data: Information on whether a sign is visible, blurred or occluded, and whether it belongs to the current road or a side road. As described in chapter 2, the task of determining whether a recognized sign actually belongs to the current road is not yet well explored, and including this information in datasets lays the groundwork for this effort. All datasets save their annotations in comma separated text files (csv-files) in slightly different formats.

The dataset from this report - named the LISA dataset from the name of the Laboratory for Intelligent and Safe Automobiles at UCSD - tries to take the best from each dataset while also adding even more data. None of the existing databases include video to any large extent (the KUL dataset has 4 video tracks). They all have a large collection of annotated single images, but this means that they cannot be used to test detectors relying on temporal information. Many systems already use various tracking schemes to minimize the number of false positives, and it is quite likely that in the future, detections using temporal data even more will emerge. Therefore, the LISA dataset includes video as well as stand alone frames. The organization of the dataset can be seen in fig. 3.1 and is described further below.

In order to actually create the dataset, two tools were developed: Video Annotator and Frame Annotator, each responsible for one part of the annotation process (see below). Care was taken to create the annotation tools in a way so the can be used for general annotation purposes, not only for annotating traffic signs. This way, the programs may also be used for future unrelated projects which require video annotation.

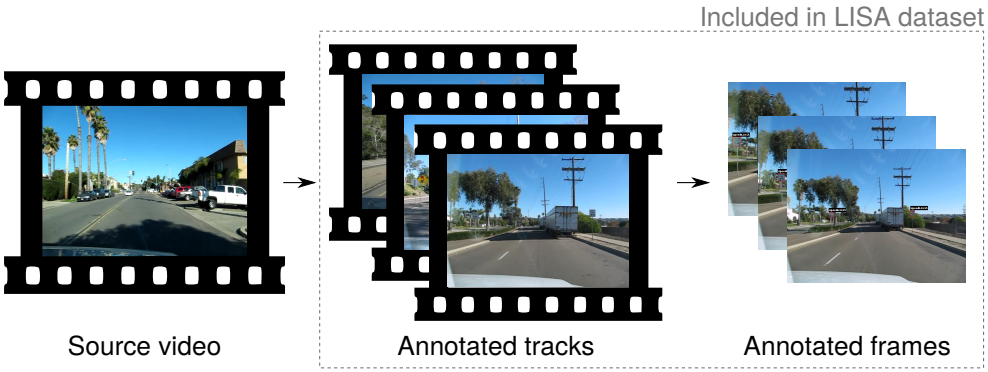

Figure 3.1: An overview of the dataset structure. Each source video is split into a number of tracks. The tracks are annotated with the sign type they contain, but may include other signs. From each track, up to 30 frames are extracted and all signs in these framed are tagged with position, type, and some additional meta data.

Tracks

The source videos are obtained from a driving vehicle. Several different cameras and vehicles have been used to ensure a dataset that is not favored towards a specific setup. Some of the source data is in color, while some of it is grayscale video. Each source video is split into smaller videos which contain signs. These smaller videos are called *tracks*, just like in the GTSRB dataset. Each track has an annotation of the sign that is present in it. In urban driving, it is very common to have several other signs present in one track. This means that the only guarantee about a track is that it contains the sign described by the annotation, but it may still contain other signs as well.

Track annotations do not contain any information on the pixel-wise position of the annotated sign, simply that the sign is present. If pixel positions are desired, they can be inferred from the frame annotations discussed below. Along with the sign type, the annotation of each track includes information on which source file the track comes from and exactly which frames it consists of. This information ensures the traceability of the track to its source.

The technical details and the exact format of the tracks and their annotations are covered below in section 3.2.2.

Frames

A number of frames are extracted from each track and annotated in detailed fashion. Each extracted frame is annotated with the exact position of all signs present in the image. This means that even if a track is annotated as a speed limit sign-track, all other signs in the frames will be annotated too. Inspired by the STS dataset, each sign annotation contain this information:

Tag: The type of sign as a string without spaces, e.g. speedLimit or pedestrianCrossing.

Position: The rectangular bounding box of the sign, given as the upper left and lower right corner.

Occluded: True if the sign is partially occluded.

On side road: True if the sign does not belong to the road currently being driven on, but instead to a side road.

Similar to the track annotations, additional traceability information is included so every annotated frame can be traced back to both its origin track and origin source video. Contrary to the STS dataset, the frame annotations in the LISA database does not contain information on whether a particular sign is blurred. This is omitted since it seems to be a subjective measure and of limited use.

For the LISA dataset, it was decided not to annotate all frames in tracks. In each track a maximum of 30 equidistant frames were annotated, an the annotated frames were required to be at least 5 frames apart. This ensured that long tracks are not overrepresented in the dataset and that there is some variation from sign to sign, since two adjacent frames are likely to produce very similar signs. It also lessens the work load of the annotator.

The technical details for the frame annotations are covered in section 3.2.3.

3.2.2 Video annotation tool

The program Video Annotator was implemented in C++ using OpenCV for image and video I/O and drawing, and Qt for the user interface. An OpenGL based custom Qt widget was used for connecting Qt and OpenCV. A screenshot can be seen in fig. 3.2. A short guide to the usage of Video Annotator can be found in appendix D, so the actual usage and interface of the program will not be discussed here. It should be mentioned, though, that it was created to have a minimum of visual clutter and with keyboard shortcuts to make the usage as efficient as possible.

The output from Video Annotator is the most interesting thing to discuss here. The main output format is a plain text file with annotations. This format was chosen for several reasons:

- A text-based format is standard and used in all other available databases.
- It is human readable, making manual lookups possible for debugging of tools.
- Unix/Linux contains a very comprehensive suite of text-file handling tools, which helps when handling thousands of annotations in many files.
- Developing tools to handle text-based formats is easy and does not depend on the ability to decode and encode obscure binary formats.

When a video is annotated, a semi-colon separated text file with the annotations is created with the name of the video file as a base. If the video is named `vid_cmp2.avi`, the

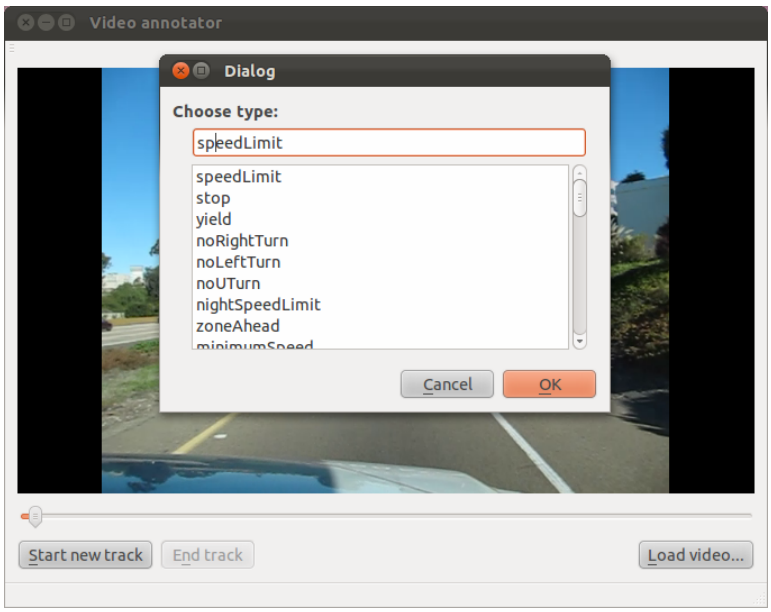

Figure 3.2: Screenshot of Video Annotator

Listing 3.1: The header and two sample annotations in a video annotation file

```
1 Filename;Track type;Origin file;First frame;Length
2 stop_1323802788.avi;vid0/vid_cmp2.avi;10243;145
3 yield_1323802820.avi;yield;vid0/vid_cmp2.avi;10963;30
```

annotation file is name `vid_cmp2.avi_annotations.csv`. Each line contains an annotation of a track in the format listed in listing 3.1.

The first column, `Filename` contains the name of the other output Video Annotator produces: An XviD compressed video file containing the track. The track video is named after the sign in the track, and a creation timestamp is created. The timestamp acts as a unique key for that particular track, ensuring no naming conflicts between tracks with the same sign appear, even in tracks across different source videos. Another option would be to use a video-content based hash of some sort. That might be necessary if the annotation program should work in an environment where multiple videos are annotated in parallel (so to tracks could theoretically be created in the same second), but a timestamp was deemed secure enough for these purposes and chosen due to its simplicity. `Track type` is the sign type the video contains. As mentioned earlier, there is no guarantees that no other signs are present in the track. To ensure traceability, the remaining three columns contain a reference to the original video file and frame numbers so the video of the track can be recreated as long as the original file is retained.

The output file from Video Annotator can be loaded into Frame Annotator, or used stand-alone, for example to extract all tracks which contain a specific sign. This could be relevant a as test pool for a detector.

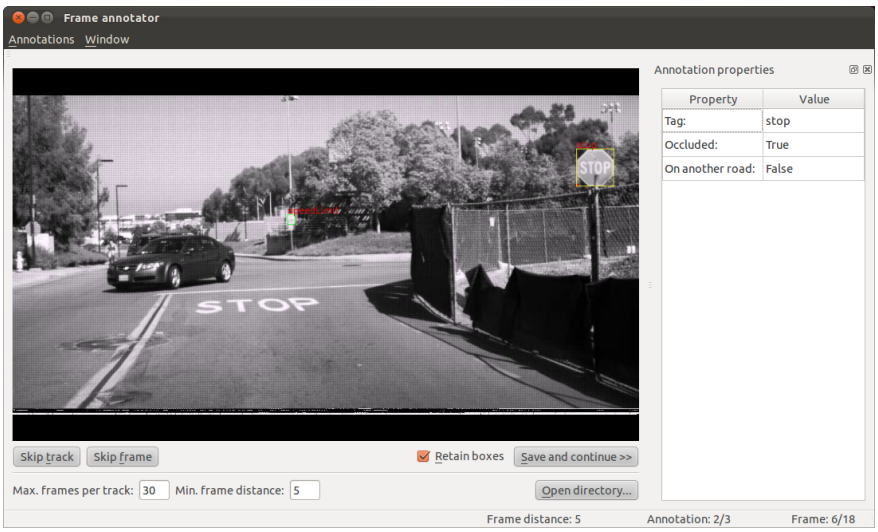

Figure 3.3: Screenshot of Frame Annotator

3.2.3 Frame annotation tool

Frame Annotator was created using the same tools as video annotator: C++ with OpenCV for image and video I/O and drawing, and Qt for the user interface. An OpenGL based custom Qt widget was used for connecting Qt and OpenCV. A short user guide for Frame Annotator is included in appendix E. A screenshot is in fig. 3.3.

Frame Annotator is a more complex program, both implementation- and interface-wise. It is built to work on top of Video Annotator, but as described in the user guide, it can be used with stand alone videos if necessary. The philosophy of Video Annotator is carried over: The output consists of a text-file with annotations and it also saves all annotated frames as PNG image files. The basic idea is simple: The program allows the user to draw rectangular annotations around objects and attach a tag to them - in this case the sign type. The user can annotate multiple signs in the same frame and also save various meta data for each annotation. The program can be set up to annotate all frames in a track, or a maximum number of frames with a minimum internal distance as discussed earlier.

The main output file is called `frameAnnotations.csv` and a short example can be seen in listing 3.2. Each line is an annotation, so if a frame contains several signs, it will have several lines in the annotation file. The two first columns contain the filename for the image of the frame and the tag (sign type) associated with the image. Then four columns describe the location of the annotation. The next column is dynamic: It contains a comma-separated list of binary meta data for the annotation. Because Frame Annotator is flexible and can be used with any number of meta-data fields, but tools should be able to expect at fixed number of columns, all meta data are put in the same column. Finally, four columns provide traceability, both back to the origin track, but also back to the origin video. This means that any annotation made in Frame Annotator can be carried all the way back to the original video and used to test the

Listing 3.2: The header and 5 sample annotations in a frame annotation file. Extra spaces have been added for readability.

```
1 Filename;Annotation tag;Upper left corner X;Upper left corner Y;Lower ¶
    right corner X;Lower right corner Y;Occluded,On another road;Origin¶
     file;Origin frame number;Origin track;Origin track frame number
2
3 yield_1323802820.avi_image0.png;yield;651;38;684;66;0,0;vid0/vid_cmp2.¶
    avi;10965;yield_1323802820.avi;2
4
5 yield_1323802820.avi_image1.png;yield;660;42;693;70;0,0;vid0/vid_cmp2.¶
    avi;10970;yield_1323802820.avi;7
6
7 yield_1323802820.avi_image2.png;yield;668;36;701;65;0,0;vid0/vid_cmp2.¶
    avi;10975;yield_1323802820.avi;12
8
9 keepRight_1323802829.avi_image1.png;keepRight;229;45;250;68;0,0;vid0/¶
    vid_cmp2.avi;11134;keepRight_1323802829.avi;46
10
11 noLeftTurn_1323803031.avi_image0.png;noLeftTurn;62;29;86;51;0,0;vid0/¶
    vid_cmp2.avi;16018;noLeftTurn_1323803031.avi;2
```

performance of a detector which takes advantage of tracking or needs video for other reasons.

3.2.4 Annotation handling tools

Apart from the to tools to do annotations, a set of tools to handle the annotation files was created. The set consists of 4 Python scripts accessed through a command line interface:

- `mergeAnnotationFiles.py`
- `splitAnnotationFiles.py`
- `extractAnnotations.py`
- `evaluateDetections.py`

The tools have been tested with Python 2.7.3, but care was taken to make the code Python 3 compatible, so provided all dependencies exist for Python 3, they should be easily portable. Instead of a user guide, each tool has a comprehensive help-page which explains the command line parameters. It can be accessed using the `-h` parameter. Example: `python extractAnnotations.py -h`.

`mergeAnnotationFiles.py` can combine multiple annotation files into one. When using Frame Annotator, one `frameAnnotations.csv`-file is created for each source video and put in a subfolder. `mergeAnnotationFiles.py` crawls a directory structure at combines all the annotation files into one, while changing the file paths to be relative to the new, large annotation file. It works with both video annotation files and frame annotation files.

It is used to create an easy-to-use index of all annotations in the same format as the individual annotation files.

`splitAnnotationFiles.py` is used to randomly split annotation files into two. It is most commonly used on the combined annotation file to create random sets of training and test data. It allows for the user to specify the split percentage (for example 80% to one file and 20% to the other) and is it also possible to create a split only for a specific sign type. Creating an 80/20 split of all stop signs can for example be done with the command `python splitAnnotationFiles.py -f stopSign 80 allAnnotations.csv`.

`extractAnnotations.py` is the most powerful of the annotation handling tools. It also operates on annotation files, most commonly the combined one. It output is always a set of images in a folder called `annotations`. It has several use cases, which can all be used with a type-filter:

Copy : This mode simply copies all the full images to a different directory. It can be used to easily extract all images containing a stop sign, for example.

Mark : Marks the annotations on the full frames and saves them to the output folder. Good for manually evaluating annotations.

Black-out : Puts a black box over all annotations. This can be used to black out signs that should not be present in test-images.

Crop : Crops the images at the annotation. A margin can be set. Used for extracting training images from the dataset.

As mentioned, all functions can be used with a tag-filter, but they can also be used with a category-filter. Speed limit signs, for example, span several tags: speedLimit15, speedLimit25, etc., which could each be extracted with the filter switch: `-f speedLimit15¶` . If all speed limit signs, regardless of speed, should be extracted, the category switch can be used instead: `-c speedLimit`. Categories are defined in the file `categories.txt` which must be present in the current working directory.

`evaluateDetections.py` is used to evaluate the performance of a detector. It compares the annotation file (ground truth) with the output of any detector. The detections are given in a csv file. It displays various detection statistics and can be set up to display or save false detections for further manual evaluation. Currently, the script tests if all four corners of the detection are within +/- 10 pixels of the ground truth in both dimensions. In the future, the Pascal detection measure should be used instead. It has been implemented, but too late to be used for the bulk of this project (it was implemented in conjunction with the pedestrian detection project, see chapter 6). Appendix F contains commands which demonstrate the use of these tools along with come convenient Linux commands for handling the dataset.

Table 3.1: The content of the LISA Traffic Sign Dataset broken down by sign type.

294	addedLane	34	slow
37	curveLeft	11	speedLimit15
50	curveRight	349	speedLimit25
35	dip	140	speedLimit30
23	doNotEnter	538	speedLimit35
9	doNotPass	73	speedLimit40
2	intersection	141	speedLimit45
331	keepRight	48	speedLimit50
210	laneEnds	2	speedLimit55
266	merge	74	speedLimit65
47	noLeftTurn	132	speedLimitUrdbl
26	noRightTurn	1821	stop
1085	pedestrianCrossing	168	stopAhead
11	rampSpeedAdvisory20	5	thruMergeLeft
5	rampSpeedAdvisory35	7	thruMergeRight
3	rampSpeedAdvisory40	19	thruTrafficMergeLeft
29	rampSpeedAdvisory45	60	truckSpeedLimit55
16	rampSpeedAdvisory50	32	turnLeft
3	rampSpeedAdvisoryUrdbl	92	turnRight
77	rightLaneMustTurn	236	yield
53	roundabout	57	yieldAhead
133	school	21	zoneAhead25
105	schoolSpeedLimit25	20	zoneAhead45
925	signalAhead		

In total: 7855 sign annotations

3.3 Results

Using the tools described above, a database with traffic sign images and videos was collected. The dataset consists of 7855 annotations on 6610 images. It contains 47 classes, listed in table 3.1 along with the number of instances for each type. Speed limit signs has been broken into types after their denominations. Those which are named Urdbl had an unreadable speed limit, but could still be determined to be speed limit signs. Fig. 3.4 shows some annotated sample images from the dataset. The dataset was assembled from footage from drives around California, mostly in San Diego, with several different vehicles and cameras. The resolution of the full captured frames vary from 640x480 to 1024x522 pixels and the annotations vary from 6x6 to 167x168 pixels. Some images are in color and some are grayscale. A comparison of key stats between this dataset and others is shown on table 2.2 in chapter 2.

(a) (b)

(c) (d)

Figure 3.4: Randomly chosen examples of annotated signs from the LISA dataset.

3.4 Discussion

From the survey, it was evident that there was a need for a set of US traffic signs and a need for datasets which include full video tracks to allow for development of tracking-based detection systems. As a part of this project such a dataset has been assembled. There has been great emphasis on enabling complete traceability for all annotations throughout the annotation tool chain. Two questions must be asked when evaluating such a database: Does it contain enough signs? Does it cover the sign types necessary?

The size of the dataset is in league with the two similar European datasets, the STS dataset and the KUL dataset. However, it spans many sign types, and not all sign types are well represented. Depending on the learning algorithm used for a TSR system, thousands of images might be needed. pedestrian crossing, stop, and signal ahead are the only signs with numbers anywhere near that. If the larger sign categories are evaluated, however, speed limit signs and warning signs are both well represented. A larger dataset would not hurt, but in many situations, especially if training a general purpose warning or speed limit sign detector, the size of the dataset is sufficient.

The dataset does not cover the full MUTCD, which is nearly impossible, but it does cover a good selection of signs. Given that the dataset has been collected from hours of real driving through urban environments, it should represent the real-world class distribution well.

One potential problem with the dataset is that a major part of it is in grayscale.

This makes it useless for developing color-dependent detectors. While this sounds like a major drawback, it also reflects reality: The grayscale images originate from the newest vehicle in the laboratory, an Audi A8 outfitted with sensors directly from Audi. Due to cost considerations when mounting cameras in mass produced cars, it is common to use grayscale cameras only for other purposes, such as lane detection. Thus, the dataset reflects reality, and if it forces researchers to focus on shape-based detectors, that is only good, because those are the detectors that can be implemented in today's cars.