Nenad Gvozdenov, BSc

Test environment for camera-based systems of autonomous vehicles

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Daniel Watzenig

Institute of Automation and Control
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Martin Horn

Graz, October 2020

This document is set in Palatino, compiled with pdfLaTeX2e and Biber.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present masters thesis.

Date	Signature

Acknowledgments

I am deeply grateful to my parents and older sister for supporting me during the whole study period.

I must mention Stefan Obergröbner who allowed me to use his framework of PathPlanner application which was completed and adapted by me for the purposes of this project. Also, lots of thanks go to Emina Hadrovic for providing the implementation of the lane-detection algorithm which was used in the evaluation phase of this thesis.

Special thanks go to Amar Civgin for the preliminary review of the chapters of my thesis.

The deepest appreciation from my side goes to Prof. Watzenig and my supervisor Christian Schwarzl for always having enough time in their busy schedules to support and guide me through the whole process of my master thesis.

Abstract

This master thesis defines an approach for testing camera-based autonomous vehicle functions on proving grounds. The goal of the work is to provide a test environment consisting of a variety of software tools for simulation and evaluation of advanced driving assistance systems (ADAS). The evaluation of the ADAS is done as black-box testing procedure where we just provide inputs and observe the behavior of a system in the simulated environment by checking the produced output.

The test environment uses projectors to create projected images as input to the system under test (SUT). One or multiple projectors are used to project the videos or images of the testing scenarios in front of the vehicle. This master thesis takes challenges into account caused by projectors. The so-called Keystone effect occurs when the projector is placed at a certain angle towards the projecting surface. The projected images or video frames get deformed under those conditions. An image calibrator tool was developed to eliminate the distortion by applying the image transformation on the projected image.

Two additional tools have been developed which are the path-planner and the image-splitter video player. The path-planner is used to manually specify driving trajectories and the image-splitter video player to create videos for the test environment. The video is used to investigate ADAS traffic line detection with variations in shapes and colors in the simulated environment.

The basis for video creation is an image of drawn street markings. The path-planning tool uses a road map and enables the user to draw a path on the map. The user can then exports a list of waypoints along the defined path to a file. Out of this map and exported path, the image-splitter tool creates a video animation. The video is constructed by cropping rectangular

parts from the road map along the path. Those extracted parts are then stacked into the animation timeline. The tool is capable of defining multiple image parts at the time, depending on the number of used video projectors in the simulated environment. In this way, the tool splits the video into multiple areas, where each area is displayed by one projector. This allows for a much bigger coverage of the projected area in the test environment.

This master thesis also investigates the problem of detection of depthless objects. We have used images of different objects like traffic signs, people or some other obstacles as input to the image calibrator tool. The tool changes the perspective of the image so that its ground projection creates an optical illusion from a certain point of view. The 2D projected image is then perceived from the ADAS systems as a real 3D object. We have evaluated the vehicles in the simulated environment with different ADAS systems such as Mobileye 630, VW Sign Assist and VW Lane Assist, and have been able to stimulate them with our projection approach.

Keywords: ADAS, test environment, keystone effect, black-box testing procedure, optical illusion, perspective transformation, SOTIF, video-animation

Contents

List of Figures

1 Introduction

The near future brings us an increasing usage of autonomous vehicles on the roads. Autonomous driving will provide many advantages and benefits to our daily lifestyle. However, this way of transporting people and goods brings certain challenges with it. According to a survey conducted in the US [1], as shown in Figure 1.1, buyers of new cars named car safety as the most important characteristic. Based on this information, it is to be expected that the future users of autonomous vehicles will be most interested in the level of safety of their vehicles.

The challenge with autonomous vehicles is that the responsibilities are shifted from the driver to the vehicle. Thus, vehicle safety is linked to autonomous decision making of advanced assistance systems. In order to achieve a certain level of safety and thus the trust of users, it is necessary to perform a huge amount of testing of the vehicles advanced assistance systems during the development phase. In other words, the functional safety of such systems must be ensured, and has to be shown using testing. The reason for the huge number of required tests lies in the increasing system complexity and the increasing number of safety-critical functions which need to be evaluated in vehicles.

This master thesis defines an approach for testing camera-based autonomous vehicle functions on proving grounds. The goal of the work is to create a test environment consisting of a variety of software tools for simulation and evaluation of advanced driving assistance system (ADAS). The evaluation of ADAS is done as black-box testing procedure, where we just provide inputs and observe the behavior of a system in the simulated environment by checking the produced output. The test environment uses video projectors to display images as input to the system under test (SUT).

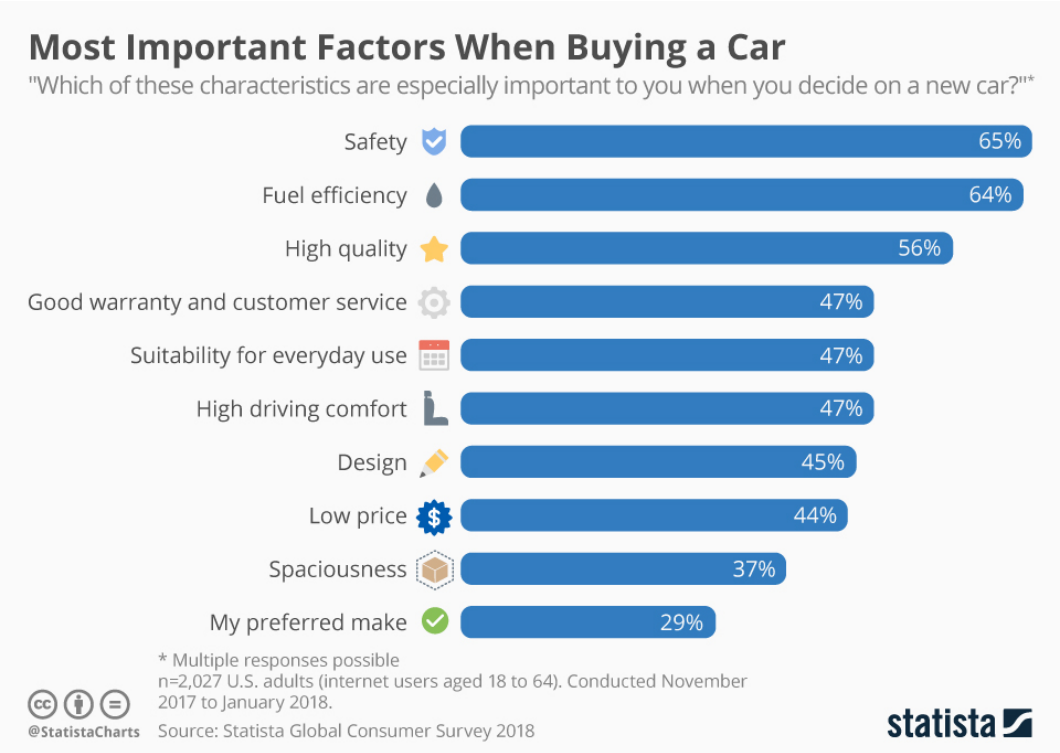

Figure 1.1: "Most important factors when buying a car" survey is taken from the following website [1]

Here is the short outlook of our master thesis structure. In this chapter, we will continue by describing the motivation for creating the test environment and the problem statement of this project. The project specification is presented in Chapter 2. First, we will introduce tools and libraries which are used for software implementation. Further on, we will list all devices used during test environment construction, as well as different ADASs used in the evaluation process. Chapter 3 will explain the implementation details of the tools used to provide input scenarios for a test environment. Chapter 4 of this thesis defines the testing procedure, possible test samples which could be produced with our approach and the evaluation results of ADASs will be presented.

1.1 Motivation

According to the article [2], some of the governments are discussing the regulations which would require the usage of advanced driving assistance systems (ADAS) in every vehicle by the year of 2022. As the present number of ADAS in vehicles will increase soon, it is necessary to find an efficient way to evaluate them. The evaluation process of such systems requires a huge amount of testing hours and a lot of mileage on autonomous test-vehicles.

Therefore, the main motivation for this project has been to enable thorough testing of some safety-critical functions in a simulated environment. We have decided to create a test environment where we could evaluate some of the camera-based autonomous vehicle functions.

A test environment executes specific driving scenarios which are created to investigate the behavior of a system under test in simulated conditions. It makes the testing procedure for some critical situations much faster and easier than the testing via test drives on public roads. And also because of the high number of tests required, in terms of ensuring and verifying function safety of a system, we need some kind of test automation. In our particular case, the decision has been made to use the black-box testing procedure. For this type of procedure, no further changes on the system under test are required. This is the reason why the unit under test is considered as black-box. Furthermore, for this type of testing, we just provide different scenarios to the test environment as input and observe which output the system under test will provide.

The Figure 1.2 shows three examples of inputs for the test environment. Videos or images of the testing scenario are displayed in front of a standing vehicle. The first image from Figure 1.2 represents a testing scenario of moving street markings. This type of test will be used to evaluate the traffic line-detection feature of an ADAS. Instead of working with fixed street markings projections, we wanted to have high flexibility while testing the driving scenarios. The motivation is to make test environment capable of executing test scenarios of moving street lines in a variety of different shapes

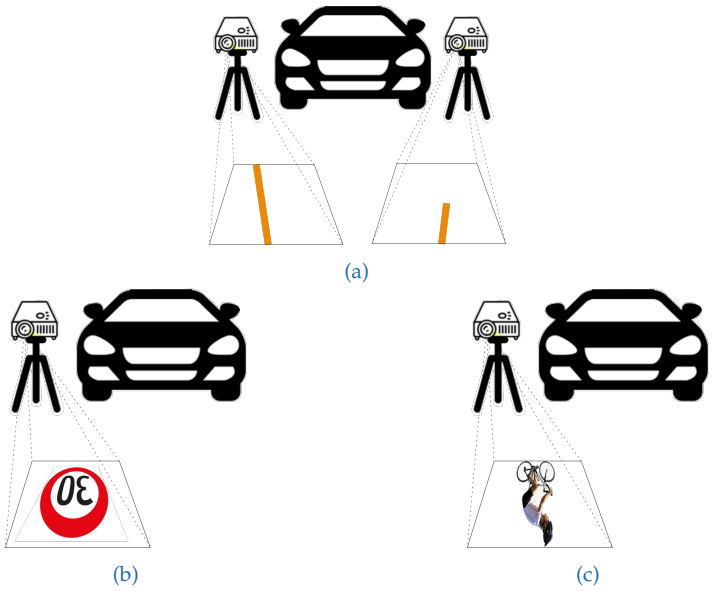

Figure 1.2: The different types of inputs created for the test environment

and colors. And also we wanted to project those test scenarios over multiple projectors, which enables a much bigger coverage on the projecting surface and makes the simulated environment better representing the vehicle surroundings.

Figure 1.2 (b) and (c) represent test scenarios of static image projections. As input for the test environment, images of different objects are used, like traffic signs, people, or some other obstacles. Those images are then transformed so that their ground projection creates an optical illusion from a defined point of view. The motivation for creating such tests is to investigate the problem of detection of depthless objects. In other words, we wanted to create a tool which could provide our test environment with such input images and make our test environment capable of checking if next-generation ADAS or some other available system like an automatic emergency brake (AEB) will detect these 2D projected images as real 3D objects. The potential false detection of such projections could have a larger impact on vehicle safety in real-world situations. The importance of dealing with such unintended behavior of current and future autonomous vehicle

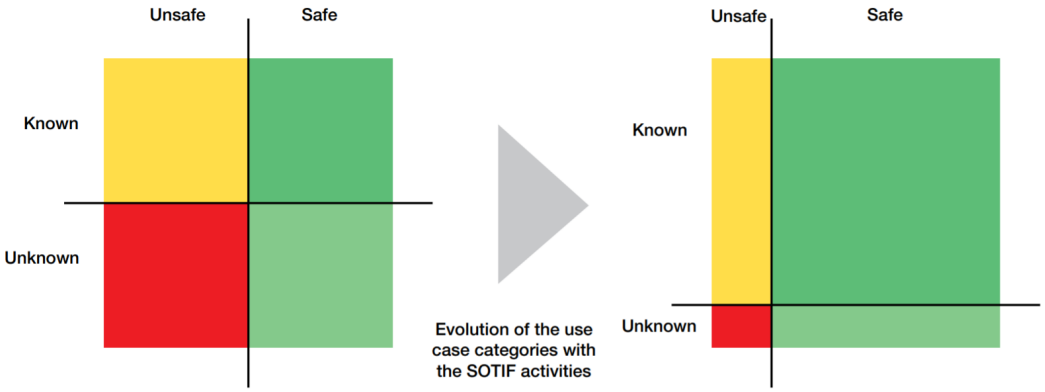

Figure 1.3: Influence of SOTIF activities on the evolution of the use case categories [3]

systems over time is confirmed by the SOTIF standard, whereof a short explanation is given below.

"Safety of the intended functionality or short SOTIF focusses on the prevention of hazardous situations caused by technical shortcomings or misuse of the E/E system. The main challenge in the SOTIF activities is, that not all technical shortcomings are known during development and in the worst case might be revealed during operation, after the vehicle has been brought onto the market. For this reason, the SOTIF activities explicitly includes tracking the vehicle performance in the field to identify unsafe scenarios not known during development. This evolution of the use case categories is shown in Figure 1.3, where the number of unknown and unsafe scenarios is minimized over time. [3]"

1.2 Challenges

In this section some of the challenges and thoughts which have been considered during the development of a test environment are presented. The first issue is related to problems caused by hardware devices, the projectors in particular. And the other challenges are more related to the implementation decisions and solutions made during the development of the required tools.

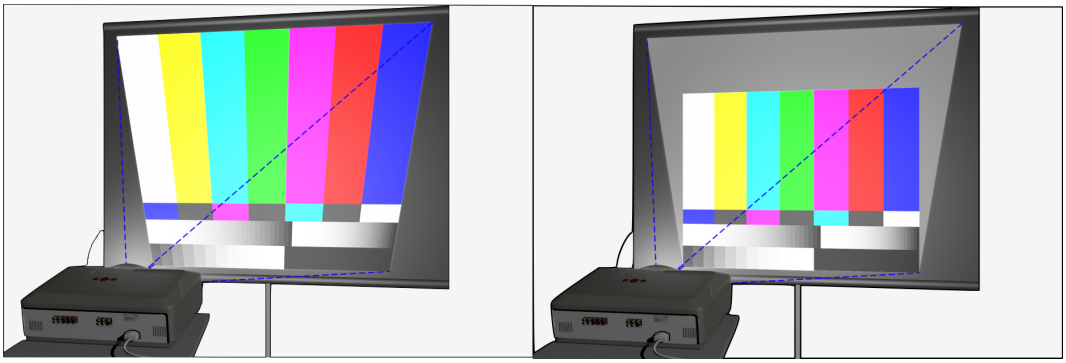

Figure 1.4: Example of the keystone effect (left), The correction of a projected image (right)

1.2.1 Keystone Effect

The usage of projectors for test environment purposes could cause some distortions of the projected images. The distortion occurs when a projector is mounted at a certain angle towards the projecting surface. In that case, the projection receives the shape of a trapeze instead of a rectangle. This type of distortion is also known as the keystone effect [4]. In Figure 1.4 an example of a keystone effect is shown.

The keystone effect will appear in our case as well since the projectors in our setup will be mounted on a certain height tilted towards the ground. The parts of distorted image projections which are further away from a projector will have significantly lower contrast and brightness than the rest of the projection on the projecting surface. These low contrast and brightness changes, lead to the loss of pixel information of the projected image.

There are several possible solutions for keystone correction. One of the solutions is manual keystone correction which considers physical adjustment to the lens of the projector. It works well for situations when the distortion of projection is not so obvious. This type of solution is not suitable in our case, since the level of distorted projection images is much higher, so other solutions are required. The other solution is digital keystone correction where the projected image is transformed digitally on the computer before it reaches the projector lens. So we decided to implement a tool

which is capable of calibrating the projector over an image transformation. The tool displays the checkerboard calibration pattern on the projecting surface. If the projected lines of a checkerboard pattern are not parallel, the user can adjust the transformation values using the tool until all the lines become parallel. The idea is to find appropriate values for image correction with the image calibration tool and to apply the same transformation to each video frame in the testing scenarios. Otherwise, the projected video will be distorted and unsuitable for testing purposes.

1.2.2 Optical Illusion and Depth Perception

The next challenge in our master thesis is related to the creation of testing scenarios which will be used to investigate the perceptual challenge of camera-based functions. Figure 1.5 shows the example of such optical illusions where the painting on the ground on the left image does not have a clear meaning, but the same painting on the right when observed from a certain point of view looks similar to a 3D representation of planet earth.

In order to generate such image projections for our test environment, we used the calibration tool which we use to eliminate the keystone effect of the projector. This tool is already capable of applying the image transformation by changing the perspective of a projected image. With this tool, we can create an optical illusion to the perception algorithm of different obstacles by transforming the 2D images of for example: traffic signs, peoples, or cars.

Figure 1.5: Optical illusion, taken from following website [5]

1.2.3 Video creation

The Virtual Vehicle Research GmbH has collected sensor data of different ADAS from the real world. The visualized data can be seen in Figure 1.6. Those two frames are showing exactly the critical point in time where the Mobileye 630 ADAS has an issue of detecting the lane in front of the vehicle.

The time difference between those two captured images is just one second, and this has been exactly the critical point where the Mobileye 630 ADAS has an issue of detecting the lane in front of the vehicle. The vehicle has at that moment approached a road construction site and the color of the street marking has changed to orange.

Hence, we wanted to reproduce some of the critical situations in the simulated environment, like for example changing the color of lines by creating a video of moving street markings. The challenge has been to find a procedure for producing needed video projections of moving street markings that appear as real road markings from the camera-based system point of view.

The two tools developed for that purpose are the path-planner and the image-splitter video player. The path-planner is used to manually specify driving trajectories and the image-splitter video player to create videos for the test environment. The basis for video creation is an image of drawn

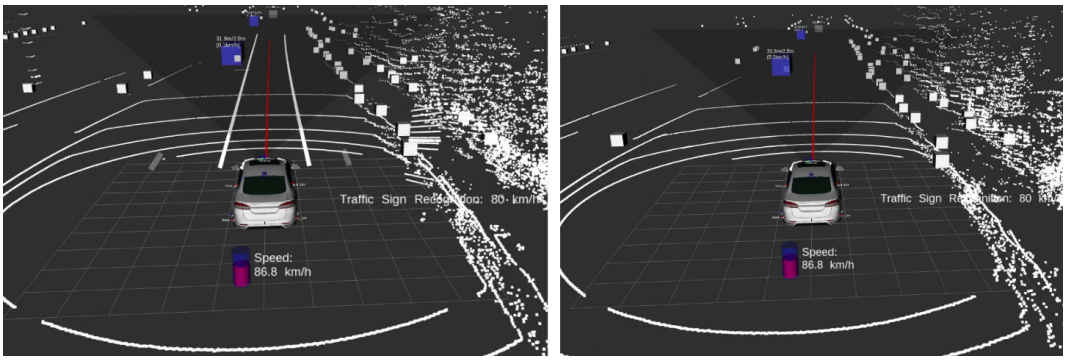

Figure 1.6: Visualized sensor data

street markings on a map. The different road map images come in a variety of line sizes, colors, and shapes and are generated with the Inkscape tool. Different maps have been used to produce different testing scenarios. The path-planning tool uses a road map and enables the user to draw a path on the map. The user can then export a list of waypoints along the defined path to a file. With this map and the exported path, the image-splitter tool creates a video animation. The video is constructed by cropping rectangular parts from the road map along the path. Those extracted parts are then stacked into the animation timeline.

The other challenge has been to produce video simulation that could be displayed over multiple projectors. Image-splitter video player tool has been implemented in such a way, that it is capable of defining multiple image parts at the defined time instance, depending on the number of used projector devices in the simulated environment. This tool splits the video into multiple areas, where each area is displayed by a certain projector. This allows for a much bigger coverage of the projected area in the test environment.

1.3 State of the art

The basic idea for this master thesis comes from a research paper with the title "Phantom of the ADAS: phantom attacks on driver-assistance systems [6]". This paper investigates a perceptual challenge that causes the ADASs and autopilots of semi/fully autonomous vehicles to consider depthless objects (phantoms) as real. The attacker in their case uses a drone with a small mounted projector to project the "phantom" in front of the moving vehicle. Where the term "phantoms" indicates the image projections of traffic lines, people and traffic signs. In their showcases, a car ADAS or an autopilot has detected phantoms as real objects, which caused the system to trigger the brake, or to steer into the lane of oncoming traffic lines, and it also caused the issue of detecting fakeroad signs. From this paper, we get the information that camera-based systems can detect ground projection as a real object. Based on that, we wanted to develop our test environment capable of projecting the different testing scenarios to evaluate ADASs.

There are already research works which consider the evaluation of ADASs within a simulated environment, for example: "Testing ADAS though simulated driving situations analysis: environment configuration [7]" and "A Method for Testing Camera Based Advanced Driving Assistance Systems [8]". However, for testing purposes they are using the commercial simulator systems, which are capable of reproducing the driving scenarios from the real world. There are also semi-virtual approaches used for ADAS evaluation, which are combining real test drive captured data and the simulations data represented in the following paper "Semivirtual Simulations for the Evaluation of Vision-based ADAS [9]".

Since our test environment uses projector devices, which suffer from the keystone effect, we had to handle this problem with an approach described in the following papers [10], [11], [12]. In those papers, the researchers have solved the keystone effect by using an additional camera to calculate geometric relations between the projector and the projecting surface. They are computing two homographies, first between the camera and projector, and the second between the projecting surface and the camera. From those two homographies, they found the image transformation which produces

the pre-warped image which could be projected without distortion. In our solution, we have proceeded without an additional camera. We have created the tool with which the user is able to manually adjust the values required for the pre-warp image transformation.

We have also considered the paper "Real time detection of lane markers in urban streets [13]" to evaluate the quality of projections in our test environment. In our work, we have used a dataset of image frames collected from our test environment with the webcam. The algorithm introduced in the mentioned paper is evaluated on the collected dataset, to verify the quality of testing scenarios and test environment itself.

2 Tools and Libraries

In this chapter applications and frameworks used in software development are presented. JavaFX is used as the main programing language, which is a Java library suitable for the development of applications with a graphical user interface. And we also use the OpenCV library to be able to implement some of the image processing functions utilized in our tools.

In this chapter, all of the hardware devices and other components used for the construction of our test environment are listed. At the end of this chapter, some advanced driver assistance systems (ADAS) and their features which could eventually be evaluated in our test environment are also introduced.

2.1 JavaFX and Scene Builder

For this master thesis, three tools which are used to generate and display different testing scenarios for our test environment are developed. For the development of these tools, we use the JavaFX[1] library. "JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms. [14]" The creation of graphical user interfaces is pretty easy with the provided Scene Builder tool.

Figure 2.1 shows Scene Builder during the design of one of our user tools. The left panel in the Scene Builder [14] application contains a set of sections with different scene elements like panes, buttons, text fields, labels, progress

[1] https://openjfx.io

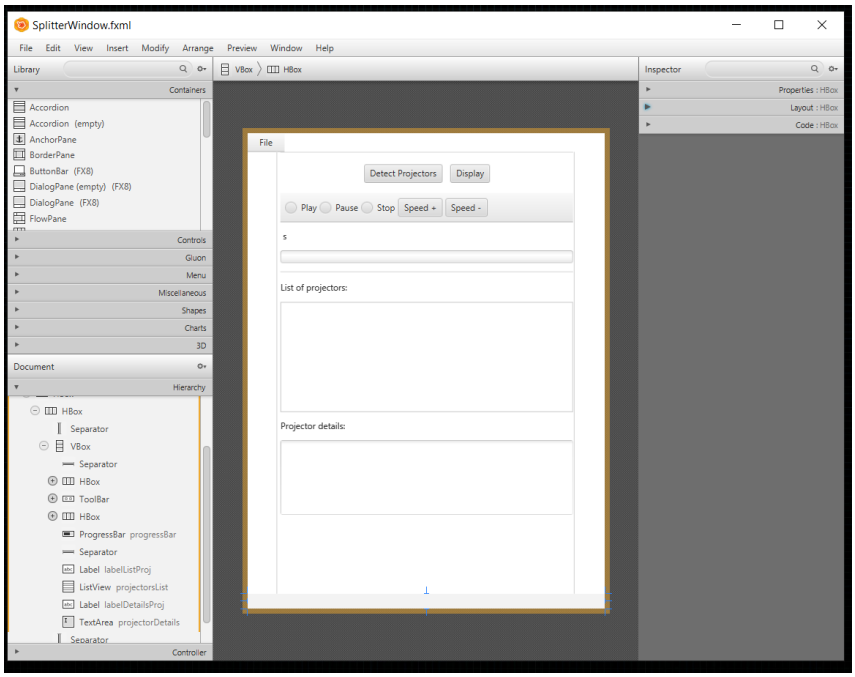

Figure 2.1: Scene Builder Application

bars, and others. All those elements could be dragged over to the hierarchy tree on the bottom left side of Scene Builder. This hierarchical tree of nodes defines the layout of the visual elements of the tools user interface. This is a good starting point for constructing the application. Each node from a hierarchical tree has an ID, style class, and bounding volume. These node properties could be defined in the right section of the Scene Builder. For each node event handlers for mouse, keys or input methods can be specified. When a hierarchical tree is completed and the properties of each node are defined, Scene Builder will generate an FXML file of the application. FXML is an XML-based user interface markup language for defining the interface of an application. When the FXML file is generated the developer can continue with implementing the business logic in the IDE. The details of our tools implementation are presented in Chapter 3.

JavaFX also supports the OpenCV library. OpenCV [2] is an open-source com-

[2] https://opencv.org

puter vision library. The decision to include this library in the application has been made because of its computational efficiency when dealing with some image transformations. The methods from OpenCV are applied in two of our applications:

- Image Calibrator application, used for calculating perspective transformation of the output image.
- Image-Splitter Player application, used for the creation and transformation of video frames.

2.2 Inkscape

Path-Planner and Image-Splitter Video Player tools require a raster image file as a tool input. The image defines the road map where only the traffic lines are shown. In order to generate such images, we use Inkscape. "Inkscape is a free and open-source vector graphics editor. It uses the standardized SVG file format as its main format. [3]" Inkscape is capable of exporting to other file formats that are required for our tools such as the portable network graphics (png) format. Figure 2.2 shows some of the drawn road sections. These sections have been used to construct the road network by connecting them to each other. Depending on the testing scenarios which were planned to be evaluated in our test environment, we drew street markings of different shapes, sizes, and colors.

[3] https://inkscape.org/about/

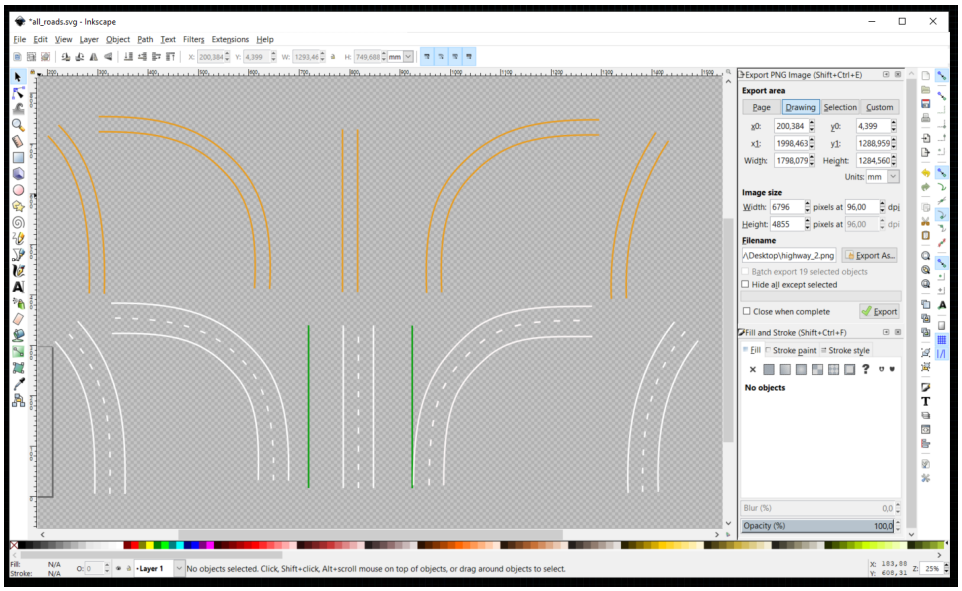

Figure 2.2: Inkscape Application

2.3 Test Environment Setup

In Figure 2.3 an example of a test environment setup is shown. This particular setup uses the following devices and parts:

- Two projectors (“Nec LT245” and “Nec LT30”)
- Three tripods
- Five meters SVGA cable
- One VGA to DisplayPort adapter
- One laptop docking station
- Logitech C920 HD Pro Webcam

As shown in Figure 2.3, the two projectors are placed at a height of two-

Figure 2.3: Test environment setup

meters, slightly tilted towards the ground, so that they could project and simulate the left and right side of the moving lane. The Webcam is placed in the middle to capture the video projections so that we could evaluate their quality. The evaluation part and the different setup of the test environment are explained in more details in Chapter 4.

We use the docking station in this setup, because it allows the connection of two projectors to the computer. The docking station has one VGA and one display port, and if we want the test environment to use more than two projectors, then a splitter hub with multiple ports with additional SVGA cables is required.

The tripod used is the "Bosch professional BT 250" built for construction sites, which can lift the projector to the maximal of two and a half meters height. The mounted projector could also be rotated in each direction, which enables easier positioning of the projectors within the test environment.

2.4 Advanced Driver Assistance System (ADAS)

In this section, advanced driver assistance systems (ADAS), which could be evaluated in our test environment are presented. "Advanced driver assistance systems (ADAS) are defined as vehicle-based intelligent safety systems which could improve road safety in terms of crash avoidance, crash severity mitigation and protection and post-crash phases. ADAS can, indeed, be defined as integrated in-vehicle or infrastructure based systems which contribute to more than one of these crash-phases. [15]" We will focus only on the camera-based integrated in-vehicle ADASs. Those systems are currently designed to reduce collisions by providing technologies that alert the driver over the combination of audible and visual warning signals. The usage of such systems increases general vehicle safety and enables better and comfortable driving. In the following sections, several ADAS with their features are presented.

2.4.1 Mobileye

The Mobileye 630[4] [16] is an ADAS produced by an Israeli subsidiary of the Intel corporation. This system consists of a camera-system for detection which is mounted on the vehicles front windshield. As shown in Figure 2.4 b), the small indicator is installed on the driver's left side to inform the driver of the potential critical situations and to give him the opportunity to manually configure some of the product features. The Mobileye has the following features:

- Forward collision warning, including urban forward-collision warning
- Lane departure warning
- Headway monitoring and warning

[4] https://www.mobileye.com

(a) Volkswagen Sign Assist [17] (b) Mobileye 630 [16]

Figure 2.4: Advanced Driver Assistance System (ADAS)

- Pedestrian and cyclist detection and collision warning
- Speed limit indicator
- Intelligent high beam control

2.4.2 Volkswagen Sign Assist

This driver-assist system is the product of the Volkswagen company [17], whose main feature is to detect road signs such as speed limits and no-overtaking-signs. The detected sing is then shown on the small multifunctional display near the speedometer as illustrated in Figure 2.4 a). This system is mainly included in cars for driver safety purposes, to reduce the probability of the driver missing the signs on the roadside while driving the car.

2.4.3 Volkswagen Lane Assist

This is another drive-assist system from the Volkswagen company [18], which uses a camera to monitor the road markings. This system prevents the car from an unintended deviation from the road, in case of the turn signals not being activated. The increase in driver's safety is done by the

system providing them a combination of visual and audible warnings. This system is automatically activated if the car accelerates over 65km/h.

3 Software Implementation

In this chapter the implementation details and their main purposes in the creation of testing scenarios for a test environment are presented. This chapter is divided into three sections corresponding to the developed tools, namely Image Calibrator, Path-Planner and Image Splitter Video Player.

3.1 Image Calibrator

The test environment is based on the usage of projectors. Such an environment setup allows displaying images or videos to stimulate the system under test (SUT). In order to generate a valid projection for the environment, we have developed this tool to eliminate the keystone effect explained in Section 1.2.1. The keystone correction is done by applying the required image transformation on the projecting image. This calibration tool gives the user a possibility to find the appropriate image transformation for any kind of projection distortions. The image transformation applied in this tool is an affine perspective transformation. The OpenCV methods required to calculate this transformation will be explained in Section 3.1.2.

The second purpose of this tool is the creation of transformed images, which could be used to evaluate the p erceptual problem, explained in Section 1.2.2, of camera-based systems. In the following sections, we will describe the JavaFX implementation of the user interface functions.

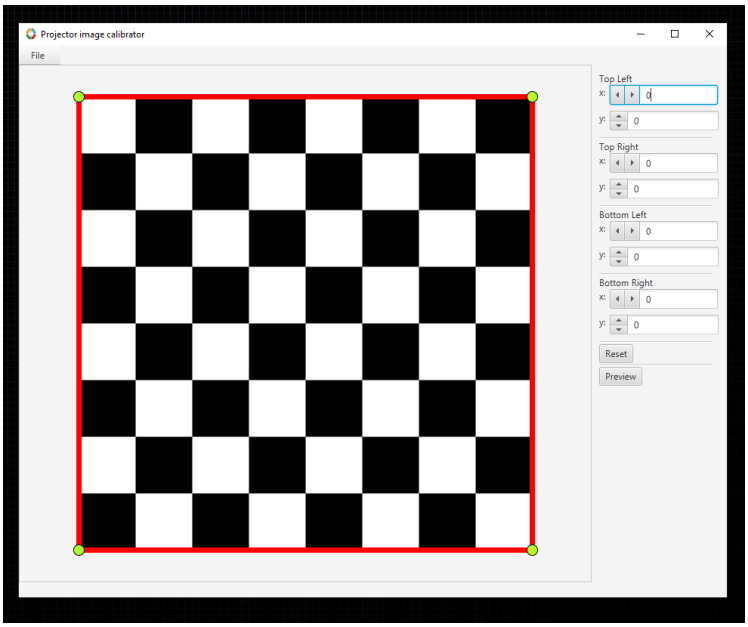

Figure 3.1: Image-Calibrator tool

3.1.1 The User Interface

The image calibrator tool interface is divided into three sections which are the menu bar, control fields, and the image viewer. The menu bar offers only a "File" menu element within which three options are contained: "Load", "Export" and "Exit".

If the user selects the "Load" option, the file chooser window appears and then the user can select an image file. This tool allows only certain file formats to be loaded. Namely, it supports JPG and PNG formats. When the user selects the wanted image file with the file-chooser, an image will be displayed in the middle of the tool where the image viewer section is defined. The example of the calibration tool with a loaded checkerboard image is shown in Figure 3.1, where four green circles in each of the image corners overlaying the loaded image. The circles are connected with red lines and displayed the affine image transformation.

The corner circle elements can be moved in the x- y- directions, either by dragging the mouse over them or by using the appropriate spinner field positioned on the right side of the tool. When a new image is loaded, each of the corner circles has a zero distance in both directions to the image corners. By dragging one of the circles at the time the tool calculates the new x and y values. These values define the displacement distance, in pixels, of a circle relative to its starting corner position.

In the control fields section on the right side of the tool, spinner fields are placed. Each of these spinners can manipulate the position of a specific circle either by clicking on the spinners up/down arrows or by entering a value into the spinner field. The range of the values which can be entered in the spinner fields is limited to +/- half of the image width for the x-direction and the image height for the y-direction. For example, if the loaded image is 600 pixel wide, the limit values for the x field are set in the range of -300 to 300. If the user enters the value which is over the limit, the tool will notify the user with a popup warning message that Drag value is out of the transformation range and the value would be set to the maximal or the minimal one, respectively. Below the spinner fields, two additional buttons are located. The “Reset” button, which sets the values of all spinners to zero and consequently the circles back to their starting position. The “Preview” is a toggle button and, depending on its state, a new window appears with the resulting transformed image. The calculation of the transformation is described in Section 3.1.2. An observer-pattern has been implemented to update the transformed image in the preview window each time when the user changes the position of a circle in the main tool window.

This preview window is undecorated and it can be moved around the screen or to the display of the plugged-in projector by dragging it with a mouse. If the user double-clicks the left mouse button on the preview window, it will be set to full-screen mode. With this preview window, we are able to manually calibrate the projector device in order to eliminate a potential keystone effect.

The procedure of projector calibration is explained in the evaluation part of

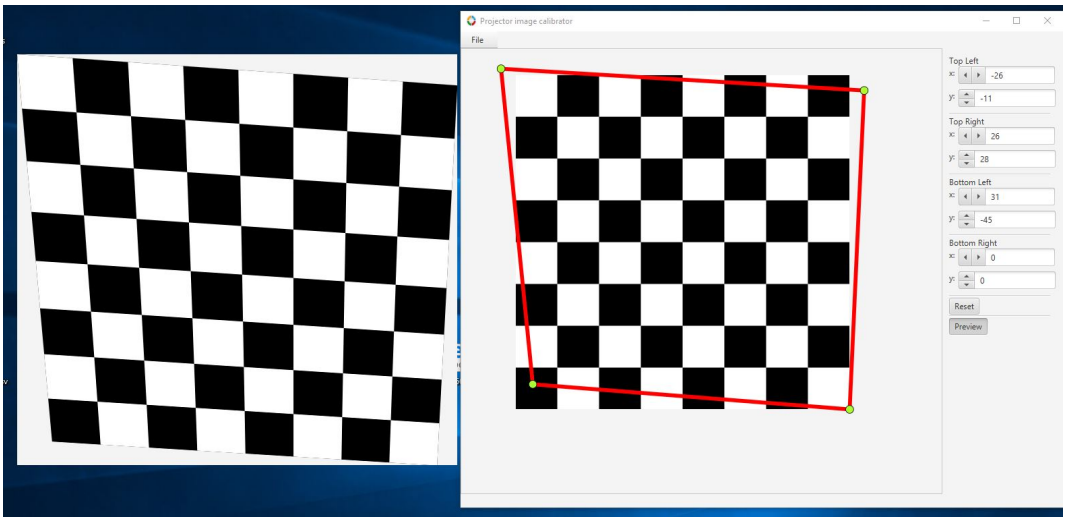

Figure 3.2: Image-Calibrator tool with opened preview window

this master thesis. When the procedure of projector calibration is done, we could save the projector configuration by exporting a file.

The exported file uses the extensible markup language (XML) format. This file is later included in the configuration file of the image-splitter video player tool as a configuration file for the specific projector. The exporting of this file is done with the "Export" option. When the user selects the "Export" option the file chooser window pop-up, where the user is able to select one of the existing XML files or to give the name for a new file and save it. When the file is saved, the user receives the confirmation alert message, with the information that the file is successfully saved and the path of the exported file.

The IOHandler is developed to construct the structure of the exported file. The exported XML file, shown bellow, has a structure with one root element <transformationPoints> and four child elements. Each child element defines one of the corner points with x- and y- values generated from the calibration tool.

Listing 3.1: Export File

```
<?xml version="1.0" encoding="UTF-8"?>
<transformationPoints>
  <topLeft y="11" x="26"/>
  <topRight y="-28" x="-26"/>
  <bottomRight y="0" x="0"/>
  <bottomLeft y="45" x="-31"/>
</transformationPoints>
```

The "Exit" option simply terminates the application. The launcher thread will then shutdown. With that, we have described all features implemented with the JavaFX for the image-calibration tool. The next section explains the mathematical background and OpenCV methods used in the development of this application.

3.1.2 Perspective Transformation

We have applied several methods from the OpenCV library [19] to obtain the pre-warped images that could be displayed without distortion. The first used method is "getPerspectiveTransformation". This method calculates the perspective transformation matrix between the coordinates of vertices of a quadrilateral in the source image and the corresponding coordinates of vertices in the destination image, shown in the code snippet 3.2. The coordinates of vertices of a quadrilateral in the source image are defined as the coordinates of the input image corners plus the relative displacement distance of the moved circles provided by the tool interface. This is the first parameter in the "getPerspectiveTrasformation" method. The second parameter is a matrix of the coordinates of the destination image corner points. For this function, the order of points put into the matrix is also important, namely, the first position is the top-left point, second the top-right, third the bottom-right and fourth is the bottom-left.

Listing 3.2: Usage of the "getPerspectiveTransformation" method

```
Point topLeft = new Point(xTopLeft, yTopLeft);
Point topRight = new Point(input.cols() + xTopRight, yTopRight);
Point bottomRight = new Point(input.cols() + xBotomRight,
    input.rows() + yBotomRight);
Point bottomLeft = new Point(xBotomLeft, input.rows() + yBotomLeft);
MatOfPoint2f inputImagePoints = new MatOfPoint2f(topLeft, topRight,
    bottomRight, bottomLeft);

Point outputTopLeft = new Point(ZERO, ZERO);
Point outputTopRight = new Point(input.cols() - ONE, ZERO);
Point outputBottomRight = new Point(input.cols() - ONE, input.rows()
    - ONE);
Point outputBottomLeft = new Point(ZERO, input.rows() - ONE);

MatOfPoint2f outputImagePoints = new MatOfPoint2f(outputTopLeft,
    outputTopRight, outputBottomRight, outputBottomLeft);

Mat perspectiveMat = new Mat(3, 3, CvType.CV_64F);
perspectiveMat = Imgproc.getPerspectiveTransform(inputImagePoints,
    outputImagePoints);
```

The method "getPerspectiveTransformation" returns the 3 x 3 perspective transformation matrix so that the following equation holds:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

In this equation x_1 and x_2 are the x, y coordinates of a pixel on the input image and x'_1 and x'_2 are the x, y coordinates of the resulting pre-warped image.

Since the user can move the corners in each direction, it can cause the situation where the perspective transformed image has a bigger size than the original one. If bottom-left or top-left corners are moved into the positive x-direction, or the top-right or top-left into the positive y-direction, it requires that the resulting matrix is modified by multiplying it with a

translation matrix. This is defined by the following formula:

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} 1 & 0 & x' \\ 0 & 1 & y' \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} h'_{11} & h'_{12} & h'_{13} \\ h'_{21} & h'_{22} & h'_{23} \\ h'_{31} & h'_{32} & h'_{33} \end{bmatrix}$$

The translation matrix is formed from the additional height and width values.
To find the size of the perspective transformed image the following function has been implemented 3.3. It takes the positions of four-corner points of the input image and calculates their position in the new perspective transformed image. Then the bounding box is constructed around these resulting positions, the size of a bounding box is the size of the transformed image that will be calculated.

Finally, to process the whole perspectively transformed image the OpenCV "warpPerspective" method is called. It takes four-parameter: input image, output image, transformation matrix and size, where the output image is the transformed image with a changed perspective.

Listing 3.3: Function that calculates the size of the transformed image

```
private Size findSize(MatOfPoint2f cornerPoints, Mat
    transformationMat) {

        MatOfPoint2f cornersMoved = new MatOfPoint2f();
        Core.perspectiveTransform(cornerPoints, cornersMoved,
            transformationMat);
        MatOfPoint points = new MatOfPoint(cornersMoved.toArray());

        Rect rect = Imgproc.boundingRect(points);
        Size size = rect.size();
        return size;
    }
```

3.2 Path Planner

The Path-Planner tool is used to manually specify driving trajectories on the map. The map is an image file of drawn street markings. We use the Inkscape application to create different maps where the road lanes with different shapes and colors heve been drawn. The Path-Planner tool loads a road map and enables the user to draw a path on the map. The user can then export a list of waypoints along the defined path to a file. Out of this map and the exported path, the Image-Splitter tool creates a video animation of moving street markings.

3.2.1 The User Interface

In this section, all implemented features of this a Path-Planner tool are presented. The application is divided into four layout sections which are the menubar, tree-view, image-view, and the controlling fields. In the menubar section the "File", "Edit", "Zoom", and "Help" elements are contained. Under the "File" element four options: "Load", "Save", "Export", and "Exit" are listed. When the user selects the load option, then the file-chooser window appears and the user can select one of the predefined project configuration files. The content of the configuration file is explained in detail in Section 3.2.3. When the configuration file is loaded, it is processed with the IO-Handler. If the IO-Handler parses the whole configuration file without any errors, the map image will be displayed in the image-view part of the application. The tool is then ready to use and the user can then start defining the driving trajectory on the map. The path trajectory is constructed as a collection of Bezier curves explained in Section 3.2.2. Each curve is defined with three points: the start-, control- and the endpoint. When the wanted path is drawn on the map, the user can then export the path waypoints to a file by selecting the Export option. The exported file is described in Section 3.2.4.

In the "Edit" menubar element all methods required for the drawing of the path such as "Planning-path", "Freehand-path", "Create-scene" and "Reset" are implemented. The "Create-scene" option creates a new driving

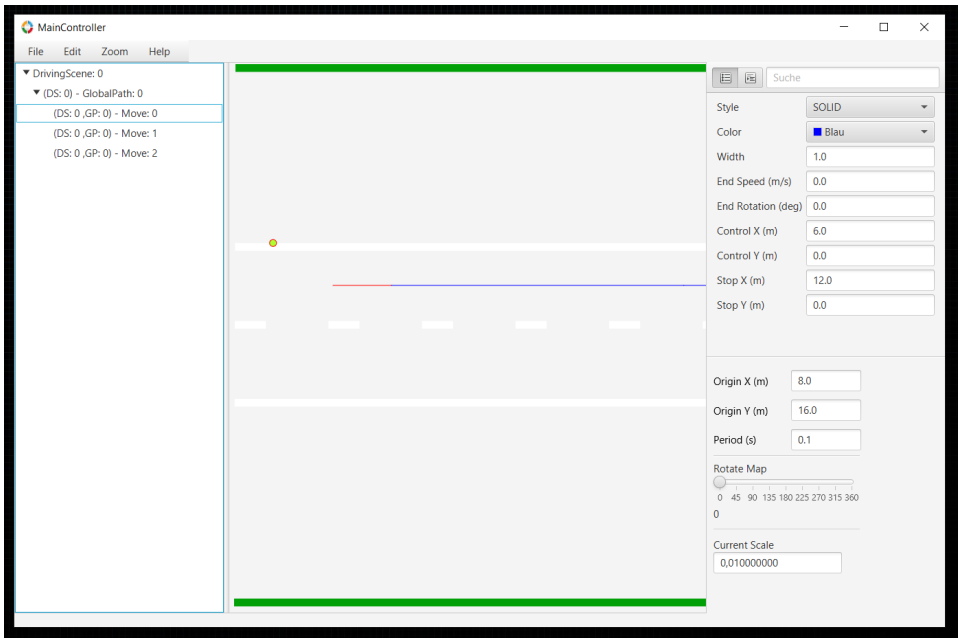

Figure 3.3: Path-Planner tool

scene in the driving scenario. If the driving scene is not pre-loaded from a configuration file, then the user needs to create one to be able to draw a path within it. Each driving scene can contain multiple paths, and each path consists of one or multiple moves.

The user can start drawing the path only when the “Planning-path” option is activated. When this option is activated, one small check symbol will be displayed beside this option in the “Edit” section. Then the user can start with placing the points on the road map. The placing of the points is implemented with a mouse-event, where the JavaFX circle object is drawn exactly on the pixel-position of an image where the user has pressed the left mouse button. Since the move is defined with a Bezier curve, the user places one initial point at the beginning of a new path, the second point is control point of the move and the third point is the stop point. If the path consists of multiple moves, then a stop point is also the starting point of the next move within the same path.
After the user has finished with the positioning of all path points on the

map, the "Planning-path" option can be deactivated by toggling this option under the "Edit" section. The finalization of a path is done with the "Freehand-path" option whereupon all points are connected in the same order that they have been created by the user, forming a driving trajectory.

When the path is constructed, on the left side of path-planner tool the whole hierarchy of the driving scene is displayed in the tree-view form. We have implemented the drag-and-drop functionalities for the tree-view structure, so that the user is able to change the order of the moves inside the same path or to drag one move over to another path inside the same driving scene. The user can also select one of the moves from the tree-view which is then highlighted with the red color on the map, see Figure 3.3.

When the move has been selected, the controlling fields are displayed on the right side of the application. With those controlling fields, the user can manipulate the different move attributes. One of these is the moves graphical representation on the map such as its style, color and width.

Similarly, on the right-hand side view with the control and stop fields, the user can change the position of the selected move points for both axes. The loaded image map is scaled so that a certain number of image pixels has an appropriate value in meters. Accordingly, the values for distances entered in fields are also defined in meters and represent the relative distance of the position of control- or endpoint to the selected moves start point.

The "End Speed (m/s)" field enables the user to input a certain value for the moves speed attribute. This value defines the speed in meters per second, which the vehicle will have at the end of a selected move. This attribute is important for the calculation of the number of samples of waypoints for the export file. One more field from the controlling fields is important for the same calculation which is the "Period (s)" field. This field defines a period, in seconds, which will also be used to calculate the number of waypoint samples for the exporting file which is explained in Section 3.2.4. The last option from the "Edit" element is the "Reset" option. It deletes all project resources and sets the application ready for the loading of the new project.

Under the "Zoom" menubar element three methods for zooming on the map are defined, "Zoom-reset", "Zoom-In" and "Zoom-Out". "Zoom-reset" resets the zooming scale factor of the map to the initial value. "Zoom-In" and "Zoom-Out" options have the same implementation as the scrolling with the scroll button on the mouse over the map image, which can also be used in this application.

3.2.2 Bezier Curve

A Bezier curve [20] is a parametric curve that uses the Bernstein polynomials as a basis. The curve is defined with a finite set of points. If the curve is defined with just two points (start- and endpoint), we would then get the linear Bezier curve. If we add one additional control point between the start-point and the end-point then we will obtain a quadratic Bezier curve. Two control points define the cubic Bezier curve and so on. For our purposes, only the quadratic form is considered.

Figure 3.4 a) shows a linear Bezier curve defined as a straight line between two points. While in Figure 3.4 b) two quadratic curves are shown where we can observe how the slope of the Bezier curves differs based on the position of its control point. The function that forms a curvature between given points of a quadratic Bezier curve, where P_0 start-, P_1 contol- and P_2 end-point are, is defined as follows:

$$B(t) = (1-t)((1-t)P_0 + tP_1) + t((1-t)P_1 + tP_2),\ 0 \leq t \leq 1 \tag{3.1}$$

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2,\ 0 \leq t \leq 1 \tag{3.2}$$

Path-Planner tool uses the formula of quadratic Bezier curves to interpolate the certain number of positions of a generated move. The number of samples that could be considered along the Bezier curve is defined with the factor t.

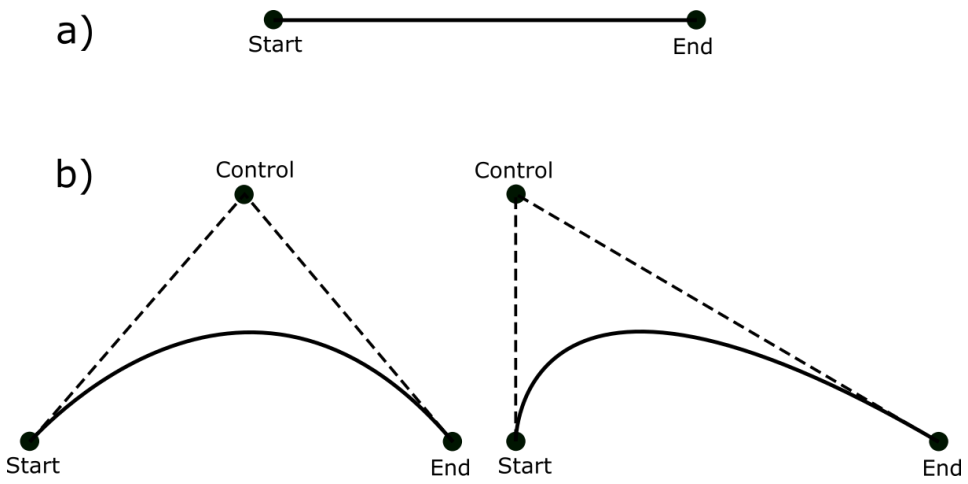

Figure 3.4: Examples of Bezier curves

3.2.3 Configuration File

The project configuration file for the Path-Planner Application is a single eXtensible Markup Language (XML) file. An example of the configuration file structure is shown in code snippet 3.4 which consists of the following elements:

- <Project> is the root element that encloses the whole XML-file. <Project> element has an id attribute, generated as (UUID) Universally Unique Identifier, a 128-bit long value that is unique for all practical purposes.

- <map> is a child element of the <Project> element. Only two attributes of this element are relevant for our thesis, which are image and dimension.

- <image> element has a resource attribute, where the URL of the image map is defined.

- <dimension> element has two attributes width and height. The values defined under those attributes are used as scaling factors while calculating from pixels to meters.

The following elements of the configuration file define the predefined driving scene. The path-planner application can read a file with multiple driving scenes, where each scene can have multiple paths. But for our testing scenarios, we just require one driving scene with one path.

- <DrivingScene> element has UUID attribute. This element could have multiple <GlobalPath> child nodes depending on how many paths have been predefined.

- <GlobapPath> element besides UUID has x- and y-attributes, which define an initial path position on the map. The values represent the distance in meters from the upper left corner of the map. The <GlobapPath> element can have one or more <QuadMove> elements.

- <QuadMove> is the parent element of <StopPoint> and <ControlPoint> elements. Based on the explanation in the section bezier curve, each QuadMove is defined with control and stop points. The x- and y-attributes are the distance in meters relative to the start point. If the path consists of several moves, then control and stop point distances are relative to the stop point of the previous move within the same path.

Listing 3.4: Example of the Path-Planner's configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<Project id="d33d2ad9-...">
  <map>
    <gnss lon="0.0" lat="0.0"/>
    <origin y="1.0" x="1.0"/>
    <image resource="pathplanningui/resources/..."/>
    <dimension width="20.0" height="20.0"/>
    <rotation angle="0.0"/>
  </map>
  <DrivingScene id="950da27b-...">
    <GlobalPath id="60c8b6dc-..." y="2.0" x="2.0" speed="0.0"
        rotation="0.0">
      <QuadMove id="0000016b-...">
        <StopPoint y="12.0" x="-3.0" speed="0.0" rotation="0.0"/>
        <ControlPoint y="4.0" x="-3.0"/>
      </QuadMove>
    </GlobalPath>
  </DrivingScene>
</Project>
```

3.2.4 Export File

The Path Planner tool generates a collection of waypoint samples along the path trajectory. These samples are exported in a comma-separated values (CSV) file format. First and second column in the file defines the x- and y-pixel coordinates of a waypoint sample. The top left corner of the image map refers to the image origins zero-zero pixel position. From that origin position, all waypoints towards the right direction of the image have positive x values and towards the bottom, positive y values. The waypoint pixel positions are extracted from the map as floating values.

The number of samples for each move from a path is calculated with a given end speed attribute and a time-period value, which is entered into the application "Period (s)" field.
The duration of each move is calculated by dividing the length of a move

with its given end speed. The move duration is then divided with a time-period value, which is entered using the application "Period (s)" field. The result of this division is the number of waypoint samples, which should be exported per move. For each waypoint sample in the third column in the file, a timestamp is defined. The timestamp for the first sample starts with zero and for each consecutive one its value is increased by the time-period.

The last line in the file shows the final timestamp, which is the duration time of the whole video simulation. The moves with a higher-end speed will have less number of waypoints. It follows that these parts of the path will get less frames per second in the video which creates the effect of acceleration in the video. The format of the exported file of waypoints is crucial for creating video simulations with the Image splitter tool, which is described in the following sections.

The example of an export file is shown below:

Listing 3.5: Export file with just three waypoints samples

```
x(pix);y(pix);timestemp(sec)
589.0000;550.9314;0.0000
584.3252;556.5514;0.1000
582.1279;553.1534;0.2000
...
...
```

3.3 Image-Splitter Video Player

This video player tool is developed to create and display videos for the simulation environment. The videos represent the testing scenarios for a test environment that are made to stimulate the system under test (SUT). The video displays the moving street markings such as road lines of different shapes and colors. This tool generates such videos from a map image of drawn street markings and the exported files of previously described tools. Depending on the number of projectors used in the simulated environment, the tool is developed in such a way that it can split the video animation for each projector. This allows for a much bigger coverage of the projected area in the test environment. In the following sections, the user interface implementations and the structure of configuration files is presented. At the end of this chapter the implementation of a timeline animation and OpenCV functions which were used are explained.

3.3.1 The User Interface

This tool is designed as a simple video player, see Figure 3.5. It consists of a menu bar, video control buttons and an information section. The menu bar offers only a "File" menu element to the user. Under the "File" element are three options, namely "Load", "Reset" and "Exit". The "Load" option opens the file chooser window and enables the user to select a project configuration file. The structure of the configuration file is presented in the following sections. When the user selects a configuration file, the XML parser checks and creates all project objects required for the video creation. If some nodes or their content of the XML configuration file are not correctly defined, the error window with an appropriate exception will pop up. Otherwise, the loading window with a progress bar will pop up. The progress bar shows the creation progress of the simulation in percentages. The creation of a simulation video is executed within a separate thread so that the user can stop the loading process by closing the loading window without closing the main application thread. If some exception occurs during the creation of the simulation video, the user will be informed with an error window. The

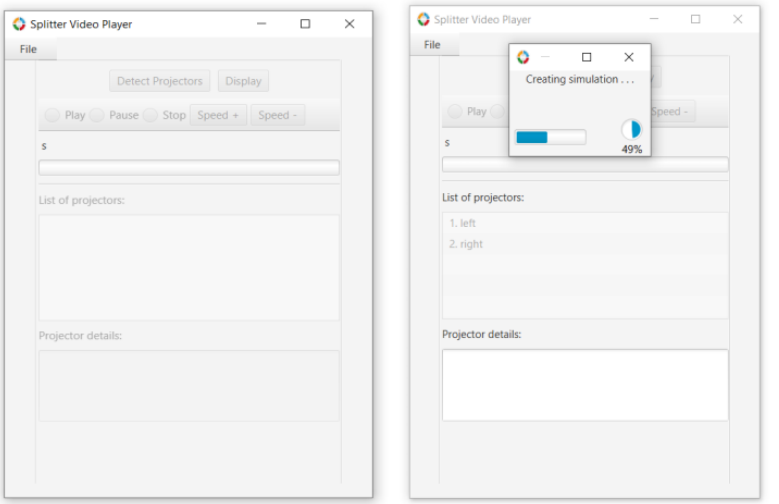

Figure 3.5: Image-Splitter Video Player tool

"Reset" option deletes all actual project objects and makes the tool ready to load some other configuration files. The "Exit" option terminates the application thread and closes the main tool window.

Below the menu bar the collection of buttons is shown. All buttons are enabled for use when the project loading process is finished. When the videos are generated, the user can open additional windows for displaying the video animation for all projectors by selecting the "Display" button, as it is shown in Figure 3.6. When "Display" is activated a certain number of undecorated preview windows will be displayed, depending on the number of projectors specified in the configuration file. The user can move them around by dragging them with the mouse. The preview windows can also be resized by holding the left mouse button over the windows edge and moving it in a certain direction. We have implemented additional functionality to set the preview window to fullscreen mode to its predefined projector device display by simply pressing a "Detect Projectors" button. Other buttons inside this collection are implemented to manipulate the flow of the displayed video. The user can select one of the possible radio buttons

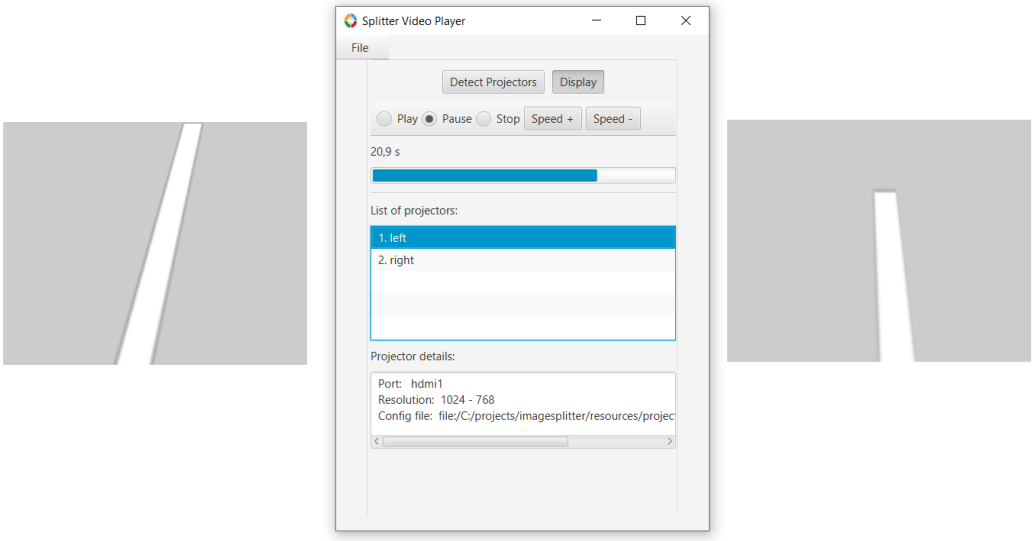

Figure 3.6: Image-Splitter tool while displaying the video animation

to play, stop or pause the video projection. “Speed +” and “Speed -” buttons change the timelines flow rate of the video either to speed up or to speed down. The tool has a progress bar with a corresponding label to show the videos current time in seconds. Below the progress bar is a list view and a text area. When the configuration file is loaded, a list view is filled up with the projector IDs. If the user selects one of the projectors from the view list than the projector details will be displayed in the text area below the list. The displayed projector details are computer port, device resolution and the URL of a file of transformation values generated with the calibration tool used to eliminate potential projector distortions.

3.3.2 Configuration File

The configuration file of this tool is defined with the XML format. One example of a file is shown in code snippet 3.6. The structure of the configuration file defines the following elements.

- <Splitter> is a parent element and consists of two child elements:<Map> and <Projectors>.

- <Map> element has two child elements: <Image> and <GlobalPath>.

- <Image> element has a resource attribute. This attribute defines the URL of an image map file.

- <globalPath> element has a resource attribute. This attribute defines the URL of the global path file, which has been generated with the path-planner tool.

- <Projectors> element has a num attribute. This attribute defines the number of projectors contained in the test environment. This element contains a specific number of <Projector> child elements depending on a predefined number of projectors.

- <Projector> element has a name attribute. The name of the projector is given depending on the projector current position from the testing environment. It is considered as a projector ID. The <Projector> element contains five child elements. Those elements specify projector details.

- <transformConfig> element has a resource attribute. This attribute defines the URL of the projector transformation file which has been generated with an image-calibration tool.

- <Viewport> element has four child elements. This element defines the viewport specifications. <viewportHeight> and <viewportWidth> elements values define the height and width in number of pixels of a viewport respectively. <viewportPosX> and <viewportPosY> elements values define the distance position of start cropping pixel relative to the path waypoint sample.

- <resolutionWidth> and <resolutionHeight> element values define the resolution of a projector device.

- <outputDevice> element has a port attribute. This attribute defines the port of the computer to which the projector device is connected.

The XML parser has been implemented to parse the whole configuration file structure and to construct all relevant application objects. Besides this main parser, two additional parsers have been implemented to handle the resource files from the configuration project file. The first one is related to <transformConfig> resources, which are the files created with an image-calibrator tool for each of the projectors from the testing environment. The parser here passes through each file and sets the transformation values to the relevant projector objects. The second one handles the comma-separated values (CSV) file of global path data, which has been generated with the path-planner tool. The waypoints from the global path and all other objects defined with the configuration file enable the construction of a timeline animation which is explained in the following section.
The example of a configuration file is shown on the next page:

Listing 3.6: Configuration file for the test environment with two projectors

```
<?xml version="1.0" encoding="UTF-8"?>
<splitter>
  <map>
    <image resource="resources/inputimages/project_highway.png"/>
    <globalPath resource="resources/globalpath/path_highway.csv"/>
  </map>
  <projectors num="2">
    <projector name="left">
      <transformConfig
          resource="resources/projectorsconfig/leftTestingDay.xml"/>
      <viewport>
        <viewportHeight>80</viewportHeight>
        <viewportWidth>100</viewportWidth>
        <viewportPosX>-100</viewportPosX>
        <viewportPosY>-80</viewportPosY>
      </viewport>
      <resolutionWidth>1024</resolutionWidth>
      <resolutionHeight>768</resolutionHeight>
      <outputDevice port="hdmi1"/>
    </projector>
    <projector name="right">
      <transformConfig
          resource="resources/projectorsconfig/rightTestingDay.xml"/>
      <viewport>
        <viewportHeight>80</viewportHeight>
        <viewportWidth>100</viewportWidth>
        <viewportPosX>0</viewportPosX>
        <viewportPosY>-80</viewportPosY>
      </viewport>
      <resolutionWidth>1024</resolutionWidth>
      <resolutionHeight>768</resolutionHeight>
      <outputDevice port="hdmi2"/>
    </projector>
  </projectors>
</splitter>
```

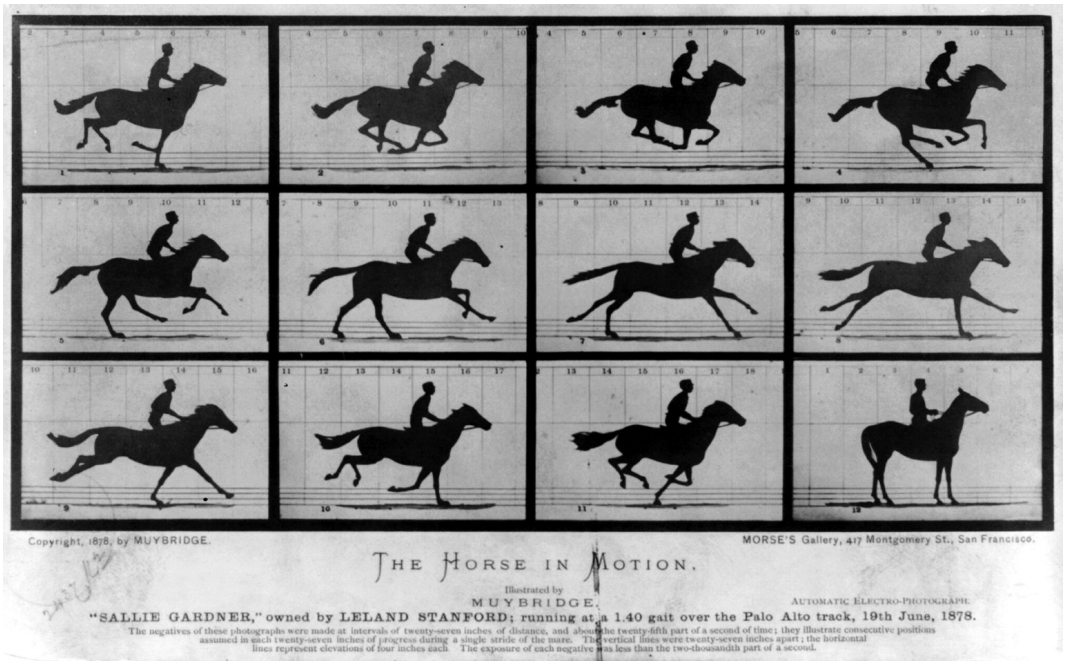

Figure 3.7: "The horse in motion [21]"

3.3.3 Timeline Animation

In this section, it is presented how the video animation timeline is constructed. The basic principle behind the implementation is based on stacking up the images sequentially on the animation timeline. There is an example of the first primitive animation based on a sequence of images called "The horse in motion [21]" shown in Figure 3.7. At that time it was called chronophotography, that is a photographic technique, where the multiple phases of a movement are captured. This was the first step in the development of motion pictures.

In the same manner, we want to capture the motion of street markings from the image map to imitate the movement of a vehicle. What was the photographs in "The horse in motion" example, in our case the so-called Viewport used to capture the motion. The viewport is in other words the

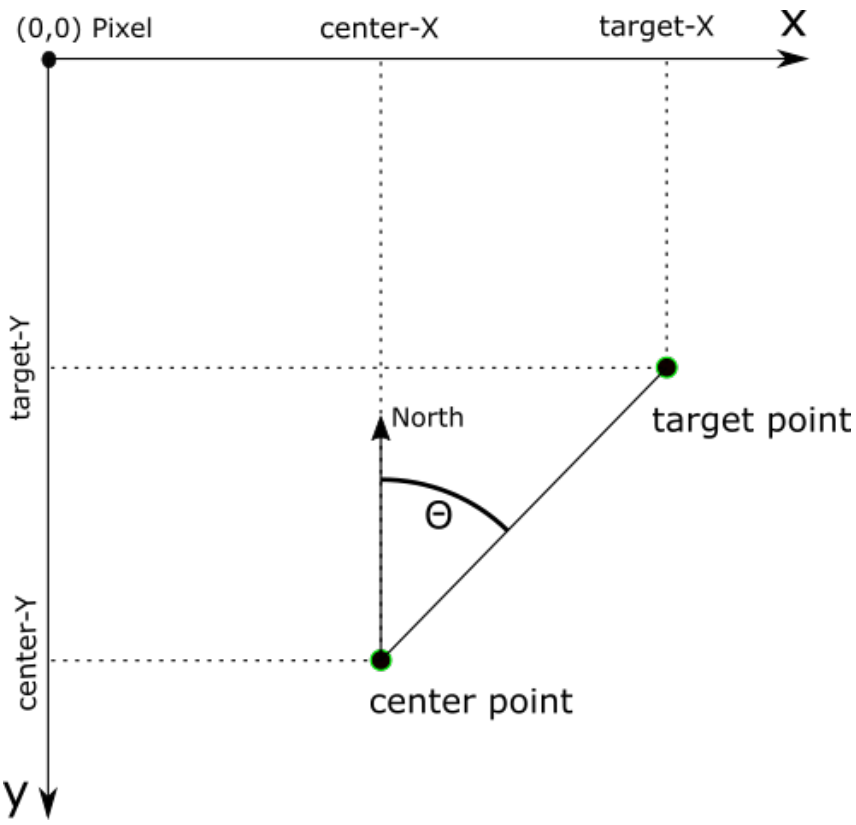

Figure 3.8: The calculation of an angle between two points on the image map

image clipping area which is used to extract the pixel-region from the image map along the predefined path. One extracted image part defines one frame in the video for one waypoint from a global path. The movement of the viewport will continue sequentially through all waypoint samples until the last waypoint from the path has been reached. Each extracted image frame is stacked up into the timeline of the video animation. The tool is capable of defining multiple viewports at the same time, depending on the number of projectors used in the simulated environment. Each viewport defines the extracted image area as the frame for a video intended for a certain projector.

In our implementation, we have first calculated a direction angle between the waypoints in the global path. The calculation is done for each neighboring pair. For example, a first waypoint is considered as the center point or

reference point, and the second waypoint from the path is the target point. We calculate the angle from a reference to a target point in degrees, see Figure 3.8. The calculated angle is set for the reference point. It means that the viewport in that specific waypoint should have the same orientation. The range between two waypoints is between 0 and 360 degrees, where the 0 and 360 values represent north orientation on the image map, 90 degrees is east, 180 is south and 270 is west.

When all orientation between waypoints over the whole global path are calculated, the next step in the implementation procedure is the positioning of the viewport on the image map. In the configuration file are the values for viewport attributes defined, such as viewport width, height and starting cropping position. The starting cropping position defines the pixel distance from a waypoint coordinate. This position defines the top left corner of the viewport. Figure 3.9 a) shows one part of the road map, where the two orange lines represent the street lanes and the red circle is the current waypoint from a global path, which is placed in the middle of the road lane. We know the pixel coordinate of the waypoint on the map from the global path file. By adding the $viewportPosX$ and $viewportPosY$ values from the configuration file to the waypoint position, we will obtain a top-left corner of the viewport. Now, when a top-left position is known, the other three corner points are computed by adding the viewport width and height to the top-left coordinate of the viewport. In this particular example the viewport is placed on top of the left street lane marking.
Figure 3.9 b) shows the next waypoint (blue circle) and the previously computed angle between these two waypoints. A viewport must also be rotated by the same angle in order to follow the trajectorys slope. Hence, all four corner points are rotated around the red waypoint which forms a rotated viewport. The bounding rectangular around a rotated viewport defines the pixel region. Instead of working with the whole image matrix, we limit the region of interest to be around the rotated viewport, as shown in Figure 3.9 c). In this way, we have optimized the creation of video simulation by limiting the image processing on a smaller pixel region instead of working with a whole map image. The following code snippet defines the creation of viewport and some of the OpenCV methods, which have been used to extract the pixel region of interest of the viewport from the map image.

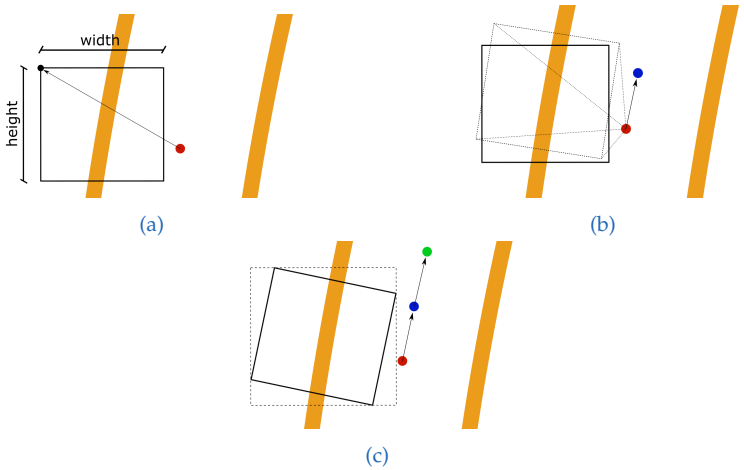

Figure 3.9: Explanation of the viewport positioning and finding the optimal cropping area on the map

```
MatOfPoint2f points = new MatOfPoint2f(topLRotated, topRRotated,
    bottomRRotated, bottomLRotated);
RotatedRect rrect = Imgproc.minAreaRect(points);
Rect bbox = rrect.boundingRect();
Mat cropedRect = mapMat.submat(bbox);
Mat rotatedViewport = rotateViewport(point.getRotation(),
    cropedRect);
Point center = new Point((rotatedViewport.cols()-1)/2,
    (rotatedViewport.rows()1)/2);
int vpTopLeftX = (int) (center.x - (viewportWidth / 2));
int vpTopLeftY = (int) (center.y - (viewportHeight / 2));
Rect vpRect = new Rect(vpTopLeftX, vpTopLeftY, viewportWidth,
    viewportHeight);
Mat viewPortMat = rotatedViewport.submat(vpRect);
projector.setRotatedImage(viewPortMat);
resizeAndTransformImage(projector);
```

The cropped viewport region is not appropriate for the animation frame, because of its small resolution which is not suitable for displaying it as

a frame with the projector. Because of that we resize the viewport image by factor of four and apply inter-cubic interpolation to increase the image quality. Then we apply image transformation on the newly resized image as specified by the configuration file, to avoid the distortion of the projection in the test environment. This image transformation is the same as used in the calibration tool for a given projector. The whole procedure: positioning, rotating, cropping, resizing, and transforming is applied for all viewports over one waypoint, and it is repeated through the whole global path positions. If the starting cropping pixel position of a viewport is outside the image map, an exception is thrown and the creation of a simulation is interupted.

After all viewport frames have been created, the function 3.7 is called to create video animations for our test environment. In JavaFX, an animation is driven by its associated properties. In our case, the "ImageView" property defined within each preview window of the tool is animated with the help of the Timeline class. The Timeline class provides the capability to update the property values with the progression of time. The timeline is filled by iterating over global path waypoints. Each waypoint in the loop represents one "KeyFrame", where a "KeyFrame" represent the target values at a specified point in time in the timeline. If multiple projectors are defined, the function then iterates through the list of projectors and sets the "ImageView" property value for each projector with an appropriate image. The keyframes for all projectors are placed at the same specified time on the timeline. When all the keyframes for a single time instance are processed, the keyframe time is increased and the function continues with the next waypoint from the global path list.

Listing 3.7: The function where the whole timeline video animation is constructed

```
Task<Void> task = new Task<Void>() {
  @Override
  public Void call() {
    int i = 0;
    double keyFrameTime = 0.0f;
    for (GlobalPathPoint wayPoint : gp.getGlobalPathPoints()) {
      for (Projector projector : splitter.getProjectors()) {
        vpt.createViewport(wayPoint, projector);
        ImageView iv =
            (ImageView)projector...getChildrenUnmodifiable().get(0);
        Image frameImg = projector.getTransformedImage();
        KeyFrame kf = new KeyFrame(Duration.seconds(keyFrameTime),
            new KeyValue(iv.imageProperty(), frameImg));
        timeline.getKeyFrames().add(kf);
      }
      keyFrameTime = keyFrameTime + wayPoint.getDuration();
      updateProgress(i, gp.getGlobalPathPoints().size());
      i++;
    }
    updateProgress(gp.getGlobalPathPoints().size(),
        gp.getGlobalPathPoints().size());
    return null;
  }
};
```

In this chapter, we have presented our used three tools. In the next chapter, we will evaluate the testing scenarios, which have been created with the help of these three tools, with our test environment.

4 Evaluation

The evaluation part of this master thesis presents how the test environment is set up, and how the input testing scenarios of a test environment have been created. The process of obtaining the experimental results during the testing session is also introduced in this chapter. The data is collected at two different conditions, namely indoor and outdoor. The evaluation process has been done always after the sunset, because the projector devices require dark conditions for better visibility of the projections.

4.1 Test environment and testing scenarios creation

The first step is to connect all required parts for our test environment that are listed up in Chapter 2.3. During our tests a setup that contains two projectors has been used, as shown in Figure 2.3, for displaying the moving street markings. When the tripods with mounted projectors are placed at the desired position, the next step is the projector calibration. That is done with an image calibration tool presented in the previous chapter.

The procedure of projector calibration is done as follows:

1. Plug in a projector to a computer where the image-calibrator tool is running

2. Position the projector in such a way that its projection is displayed on the planned spot of the test environment projecting surface

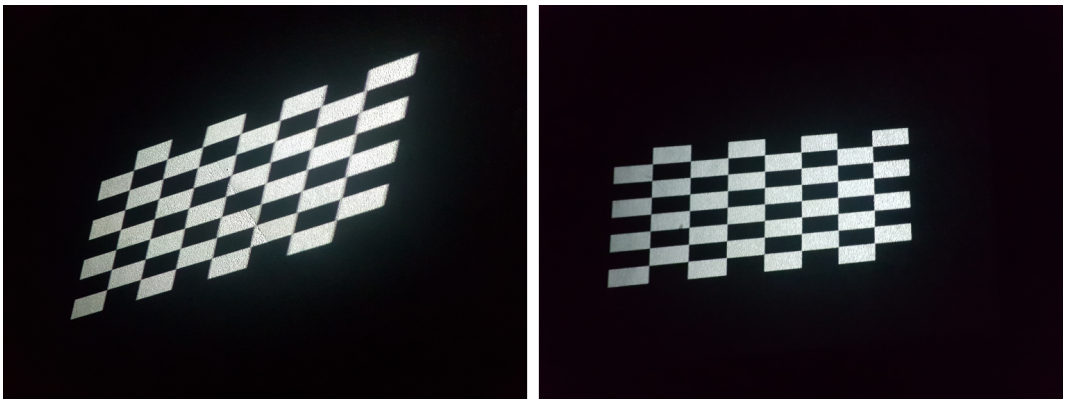

Figure 4.1: Ground projections of the checkerboard image, before and after the calibration of a projector

3. Load the checkerboard image to the calibration tool
4. Open the tool's preview window, as shown in Figure 3.2
5. Drag over the preview window to the display of the projector
6. Set the preview window to full-screen mode
7. If the lines of the checkerboard image on the projecting surface are not parallel, then a correction is needed
8. Drag the corner circles on calibrator tool until the lines of transformed checkerboard image on the projecting surface become all parallel
9. The values generated from circle displacement, which are required for image transformation, are exported from the tool to the XML file

The Figure 4.1 left) shows the ground projection of the checkerboard before the calibration of one projector, and the 4.1 right) represents a corrected checkerboard on the same spot of the projecting surface. By comparing those two figures, we could see that on the right image all squares of the checkerboard image are all with equal size and that all lines are parallel

compared to the left figure. It does not look like a perfect chessboard, where all sides of the squares are equal, because of the position of the camera used to take this picture.

The same procedure of projector calibration is applied to both projectors. When the calibration files are exported, their file paths for each projector will always be imported into every image splitter project file for a certain projector under the <transformConfig resource= "/..."> node attribute. In this way, during the video creation, the same transformation is applied to each video frame. The constructed video will be displayed on the projecting surface without the distortion, which is present under the same circumstances when the calibration procedure has been done.

When the calibration of all projector devices is completed, the creation of inputs for our test environment follows. For testing scenarios of moving street markings, the road maps have to be drawn first using the Inkscape application. We have created images, which contain drawn parallel lines representing the road lanes. Figure 4.2 shows an example of four different drawn roads used during the creation of map images. The following traffic marking combinations of a road lane have been considered in our map-making process:

- A common road (solid lines on the sides, an dashed line in the middle)
- Road with changing line types (dashed and solid line are frequently changing after a certain distance on each side of the road lane)
- Road with changing line sizes (sizes of dashed lines is varying along the road)
- Road widening (the lines are moving away from each other and create a road widening effect)
- Road with lines which signalize the construction site on the road (an dashed orange line and a solid one, as well as in combination with white line)

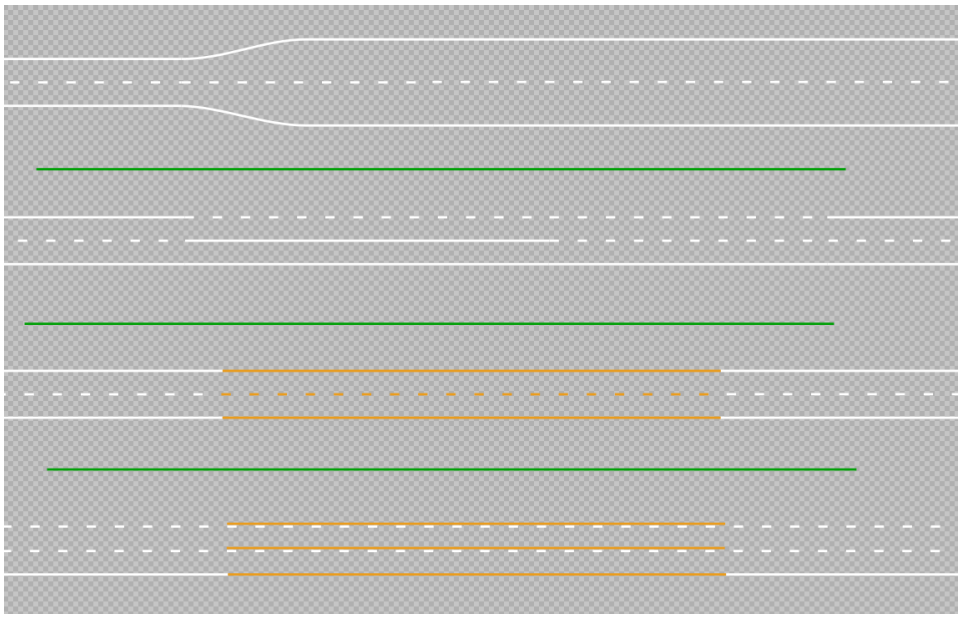

Figure 4.2: Different roads drawn within Inkscape

From each one of these listed cases, a separate road map is drawn. The created maps are then loaded and scaled with a Path-planner tool. The dimension scaling values in the configuration file of each pathplaner project have been adjusted for each map, until the length of a single dashed line is equal to the length of 6 meters, which is the standardized length on highway in Austria according to their legal regulation[1].

On the scaled map the path in the middle of the road lane is then drawn, to simulate the trajectory of a moving vehicle, see Figure 3.3. For each move inside the path the end-speed values in meter per second are defined, as well as the value inside the time period field of the path-planner tool, which defines the framerate for the video creation. When all path and tool properties are configured, the waypoint samples of the path are exported to a CSV file. The URL location of that file is ready to be included in the configuration files of the Image-splitter video player tool. In the end, for each testing scenario, a separate configuration file with a certain image map is created,

[1] https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10012574

with an appropriate CSV path file to that map, and projector calibration files. Each file is then ready to be loaded with an image-splitter tool that will construct and display the video projections for the test environment.

There is still one issue with memory space causing an Image-splitter tool to fail while creating the video. The problem is caused due to Java heap memory consumption, therefore our tool is capable to create the video with a lower framerate. For our videos, we have set only 10 frames per second, which has been good enough quality for our circumstances.

However, if the user requires the video of a higher framerate or creates the video animation for a test environment that consists of many projectors, then the heap memory must be increased by allocating the additional memory for the Java program. That is done via command-line arguments -Xms and -Xmx, that specifies the initial heap size and the maximum heap size, respectively.

4.2 Experimental Results

In the evaluation process of different ADAS with the proposed test environment, the following issues have been addressed. While trying to evaluate the Mobileye 630 and VW Lane assist system, by displaying the projection in front of the standing vehicles, it has been noticed that those systems cannot detect any of the provided test environments inputs. The reason for that is that the feature of ADAS's is only activated if the vehicle is driving above a certain speed. For example, the Mobileye lane departure warning feature is activated when the vehicle speed is above $55km/h$. Similarly, the lane assistant of the VW car is activated only when the vehicle speed is above $65km/h$. The possible solution to this issue could be the positioning of the vehicles on top of the rolling test stands and to project the moving street markings in front of the vehicle. So that the vehicle could reach a certain speed while staying in place. Instead of using these two ADAS's in the evaluation of our testing environment, it has been proceed with a different approach.

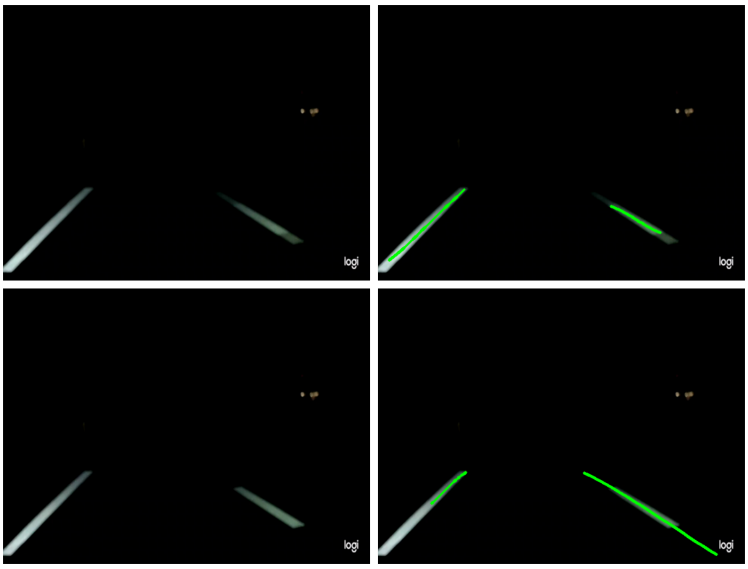

Figure 4.3: Testing scenario of normal road lane projections, successfully detected lines (left line full, right line dashed)

4.2.1 Moving street marking

Since the evaluation of ADAS over a planned black-box testing procedure has not been possible, the testing on individual perception algorithms has been applied instead. Firstly, the video projections of our test environment were captured with a web-camera, which has been positioned at the same height and orientation as some of the ADAS camera systems. The quality of the captured projections is then evaluated with the lane-detection algorithm [13]. The green splines on the video frames represent the position of the line detected by the algorithm.

In this section, the detection rate of different moving street makings is examined. Figures 4.3, 4.4 and 4.5 show the performance of collected data from our test environment which is measured with a lane detection algorithm, under outdoor condition.

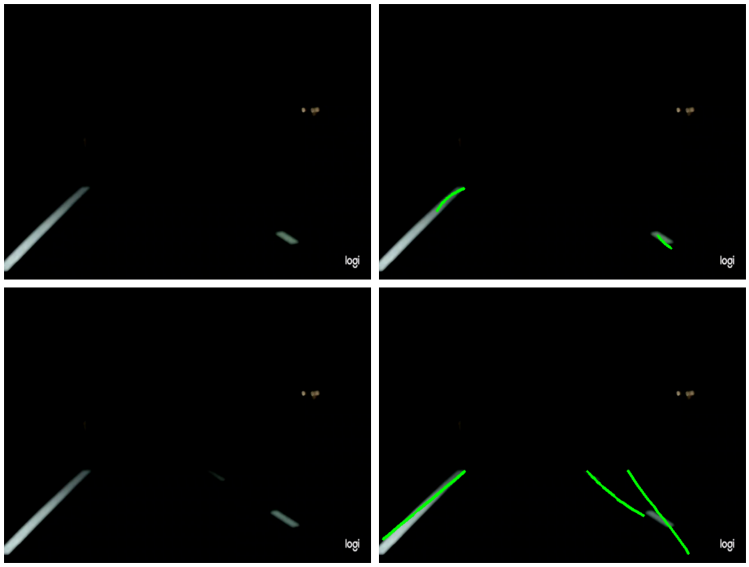

Figure 4.4: Testing scenario of dashed lines with different sizes, the algorithm sometimes failed to detect short lines

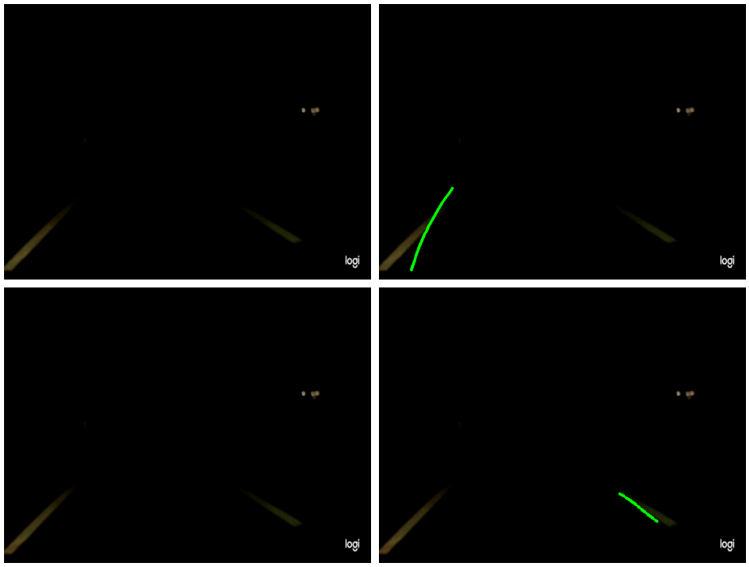

Figure 4.5: Testing scenatio with orange lines. Due to the low contrast of the projected orange lines, the algorithm is not able to detect one side of the lane

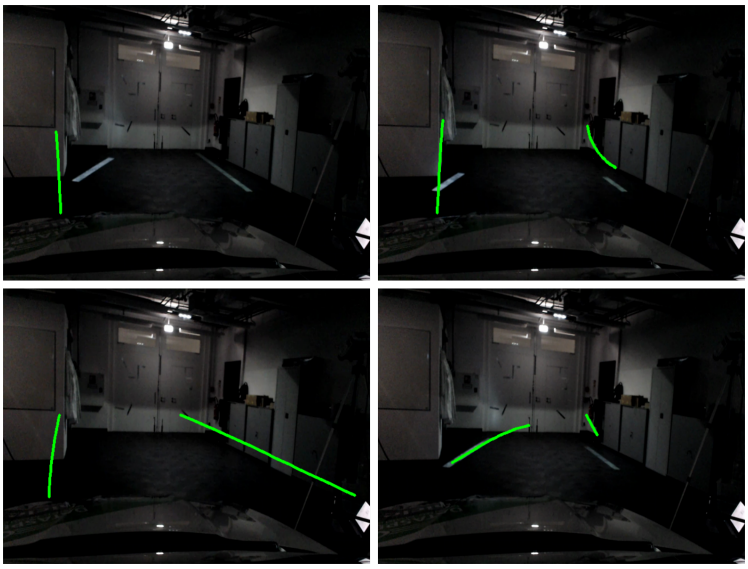

Figure 4.6: The video frame samples showing the challenges caused by the indoor environment

Figure 4.6 shows the main issue of detecting the projected lines, when the indoor test environment is used. The detection algorithm takes also the room corners and other room furniture edges into consideration. After comparing the quality of provided data between the indoor and outdoor conditions, indoor conditions produced data, which had a significantly lower detection rate of projected lines than the captured projection from the outside.

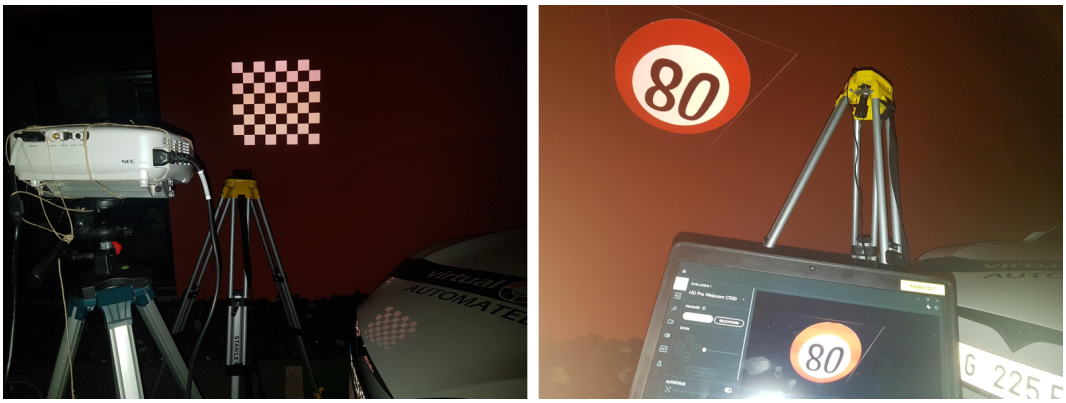

Figure 4.7: The test environment with one projector and a web-camera

4.2.2 Perception challenge

In this section some of the produced images, which could be used as input to a test environment, are presented. The goal of such input is to investigate a perceptual challenge of camera-based functions, explained in the Section 1.2.2.

In Figure 4.7 the test environment setup of one projector is shown, which has been used to display a single transformed image of different signs and obstacles. This type of input testing scenarios do not require saving the projector calibration, but instead, the same calibration tool is used to transform and show the images. In Figure 4.8 several examples of the transformed images are shown. Those images have been projected on the wall and captured by a web-camera, which has been positioned right in front of the projector, see Figure 4.7. The wall projections are created while the projector lens was pointing normal towards the projecting surface.

The first approach is to project such images on the ground as it is already explained in the motivation part of this master thesis. However, the problem occurs while trying to achieve the same shape of image ground projection as it is on the wall. The issue is caused by the projector's keystone effect since the projector is tilted towards the ground surface. Such an effect has an influence on its projection which is the reason why it is not possible

to obtain the similar image shape. The second issue is the maximal height of our tripod on which the projector can be positioned. If we even want to mount the projector to point normal towards the ground, the height of two and a half meters is too low to make the size of ground projection big enough to create valid testing scenarios, which could be detected by a camera-based system. One possible solution to those issues could be to use a small drone with a mounted projector to create bigger ground projections as the researchers in paper [6] have done. That enables them the projection of a slightly bigger transformed image in front of the moving vehicle that could be perceived from an ADAS as a real 3D-Object.

As the input to the calibration tool PNG images are used, downloaded from the following webpage[2].

[2] https://favpng.com/

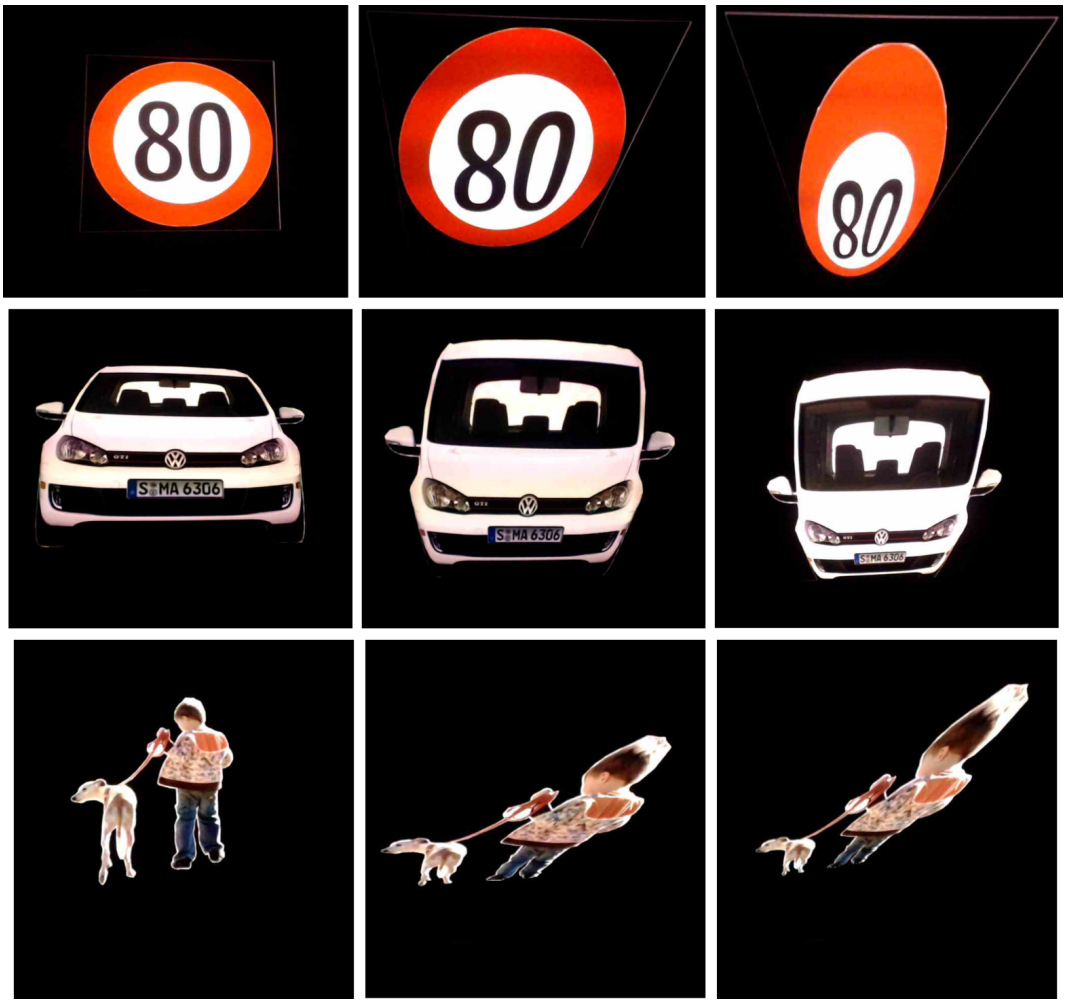

Figure 4.8: Captured wall projections of transformed images

One of the successfully conducted testing scenarios is shown in Figure 4.9. The projected images of road signs are recognized from the ADAS. The system under test in this case is a VW Sign Assist. This test case has been conducted in the narrow passage where the testing vehicle approaches the metal door on whose surface the image of 80km/h speed-limit sign has been projected. When the vehicle has passed by this metal door, the sign-assist system has detected the image which is confirmed by providing the 80km/h sign notification on the car's multimedia display on the driver's right side, as well as with the same notification on the multifunctional display near the speedometer, see Figure 4.9.

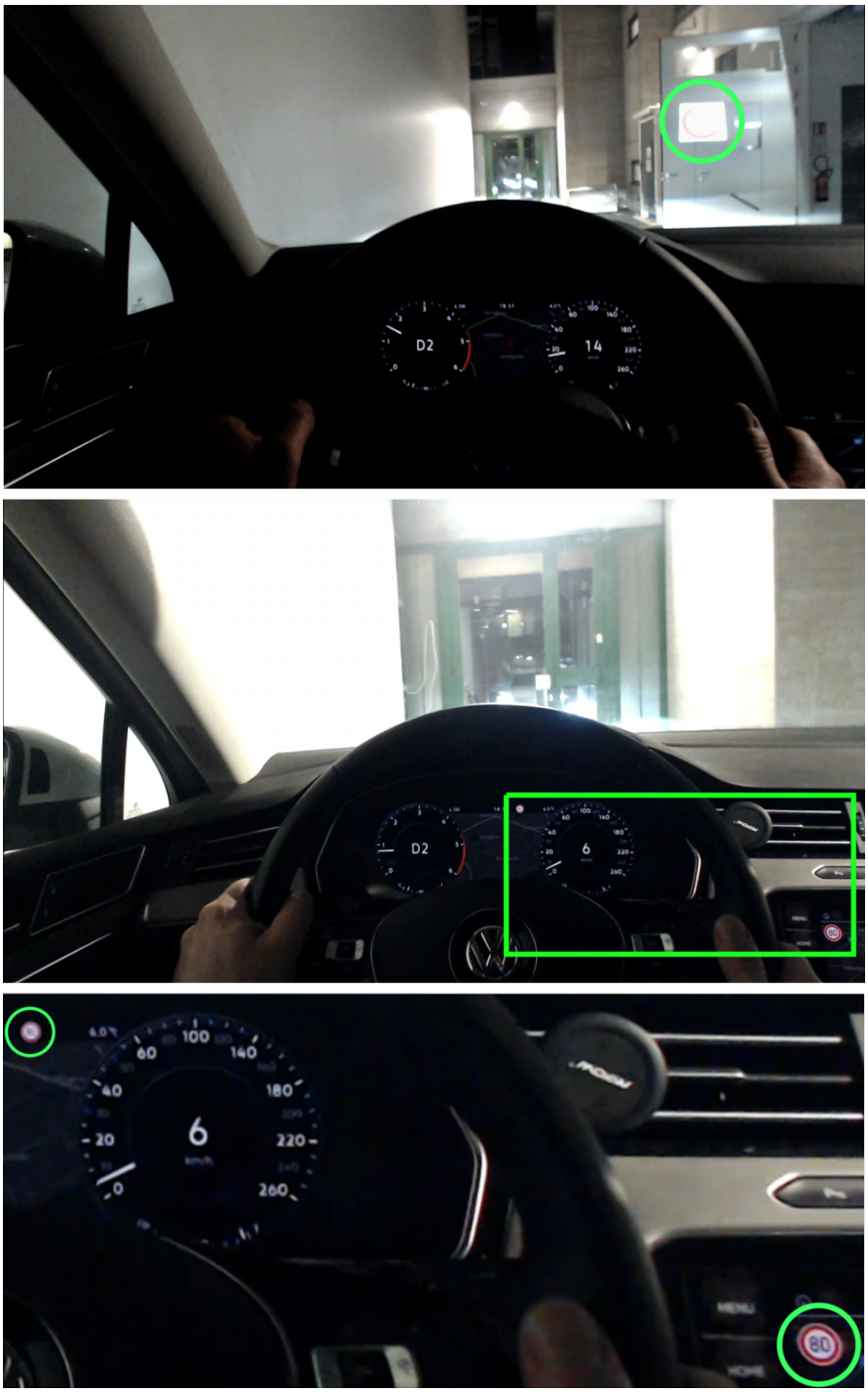

Figure 4.9: Testing scenario where the VW Sign Assist system detects the projected speed-limit sign

5 Conclusion

This master thesis aims to find an approach for testing the camera-based autonomous vehicle functions on a proving ground. Based on our developed test environment and their experimental results, it can be concluded that the advanced driving assistance systems (ADAS) features listed in Section 2.4 could be evaluated within simulated conditions. During the evaluation phase of this thesis, we have noticed that with additional adjustments to our implementation even better results in terms of the evaluation process of ADAS could be delivered. Hence, our tools and test environment are a good basis for further development.

We have taken this approach of developing a test environment because firstly, to make the execution of a black-box testing procedure and some kind of testing automation for easier and efficient evaluation possible. And secondly, to be able to reproduce some of the critical situations from the real world, or potentially to discover some new unknown and unsafe situations that could cause the ADAS functions to fail. The tools that have been developed to provide such simulated inputs for a test environment have completely satisfied our expectations, and also solved the challenges that are presented in the first chapter of this thesis.

As a short overview of our project, we have developed three applications: Image calibration, Path-planner, and Image splitter video player tool. An Image calibration has been developed to eliminate the projector keystone effect and to provide transformed images for the test environment. The path-planner is used to manually specify driving trajectories on the map, while the image-splitter video player is required to create videos of moving street markings along the predefined trajectories for a test environment. The full test environment consists of the projectors whose purpose is to stimulate the system under test by projecting the created testing scenarios

using the aforementioned tools.

However, users of these tools, and approaches, should be aware that the created moving street markings for our test environment could not stimulate the standing test vehicle with an integrated ADAS to deliver any results, because ADAS features for lane detection are activated only when the vehicle is moving above a certain speed. Also for the testing scenarios that are created with a calibration tool with the goal to investigate a perceptual challenge of camera-based ADAS function, it should be considered different approaches than in the examples shown in Figure 1.2 (b) and (c). To better understand the implications of these problems, future studies could consider these issues to provide an advanced solution which will then produce better experimental results.

A potential future work is related to further development of these tools. Since we have collected information about newly formed challenges during the evaluation phase, we could make our test environment easier and more comfortable to use, by e.g. adding the following feature to our tools. The potential upgrade could take the intrinsic and extrinsic parameters of projectors, to be able to detect projection overlaps on the projecting surface, which will make our test environment more accurate.

Bibliography

[1] M Armstrong. Most important factors when buying a car. https://www.statista.com/chart/13075/most-important-factors-when-buying-a-car/, last accessed on 22/09/20.

[2] D Dutta. Advanced driver aids might be mandatory by 2022 says Nitin Gadkari: Will make indian roads safer! https://www.financialexpress.com/auto/car-news/advanced-driver-aids-might-be-mandatory-by-2022-says-nitin-gadkari-will-make-indian-roads-safer/1312218/ , last accessed on 14/09/20.

[3] T Doms, B Rauch, B Schrammel, C Schwald, E Spahovic, and C Schwarzl. Highly automated driving-the new challenges for functional safety and cyber security. *TÜV Austria Holding AG and VIRTUAL VEHICLE, Vienna, Austria, Tech. Rep.*, 2018.

[4] D Yadav and S Agrawal. Keystone error correction method in camera-projector system to correct the projected image on planar surface and tilted projector. *International Journal of Computer Science & Engineering Technology*, 4(2):142–146, 2013.

[5] S Henderson. Cool optical illusions. https://coolopticalillusions.com/standing-on-top-of-the-world-optical-illusion-chalk-art/, last accessed on 21/09/20.

[6] B Nassi, D Nassi, R Ben-Netanel, Y Mirsky, O Drokin, and Y Elovici. Phantom of the adas: Phantom attacks on driver-assistance systems. *IACR Cryptol. ePrint Arch.*, 2020:85, 2020.

[7] O Sipele, V M Zamora, A Ledezma Espino, and A Sanchis de Miguel. Testing adas though simulated driving situations analysis: environment configuration. 11 2016.

[8] Yin Tan and B Hassan. A method for testing camera based advanced driving assistance systems. In *2013 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, pages 151–154, 2013.

[9] M R Zofka, R Kohlhaas, T Schamm, and J M Zöllner. Semivirtual simulations for the evaluation of vision-based adas. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 121–126, 2014.

[10] Jungho K, Youngbae H, and Byeongho C. Automatic keystone correction using a single camera. In *2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 576–577, 2015.

[11] Qian L and Shengtong C. Perspective correction of distorted projectors with an uncalibrated camera.

[12] Z Li, K Wong, Y Gong, and M Chang. An effective method for movable projector keystone correction. *IEEE Transactions on Multimedia*, 13(1):155–160, 2011.

[13] M Aly. Real time detection of lane markers in urban streets. In *2008 IEEE Intelligent Vehicles Symposium*, pages 7–12, 2008.

[14] J Gordon C Castillo J Potts, N Hildebrandt. *JavaFX, Getting Started with JavaFX*. Oracle Corporation, Release 8, E50607-02, 2014. https://docs.oracle.com/javase/8/javafx/JFXST.pdf, last accessed on 22/09/20.

[15] Advanced driver assistance systems 2016. https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/ersosynthesis2016-adas15_en.pdf , last accessed on 22/09/20, 2016.

[16] Mobileye Technologies Ltd. *Product Sheet Mobileye 630 System*.

[17] Volkswagen. Sign assist. https://www.volkswagen.co.uk/technology/driver-assist/sign-assist, last accessed on 21/09/20.

[18] Volkswagen. Sign assist. https://www.volkswagen.co.uk/technology/driver-assist/proximity-sensing, last accessed on 21/09/20.

[19] G Bradski and A Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[20] G E Farin and G Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann, 2002.

[21] E Muybridge. The horse in motion. https://www.loc.gov/item/97502309/, last accessed on 13/09/20.