

# Wireless 360 Degree Surround View Camera System for Leisure Vehicles

Hashim Karim, Dennis Landman & Stevan Pavlović

# Wireless 360 Degree Surround View Camera System for Leisure Vehicles

by

Hashim Karim	4811518
Dennis Landman	4911202
Stevan Pavlović	5641217

to obtain the degree of Bachelor of Science

at the Delft University of Technology,

Faculty of Electrical Engineering, Mathematics and Computer Science.

Project duration:	April 5, 2024 – July 14, 2024
Project proposer:	E-Trailer B.V.
Project supervisor:	Ing. J. Bastemeijer, TU Delft

Cover: Caravan on a mountain road. Image by Manolis Soilentakis from Pixabay

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.



# Preface

This thesis is written for the Bachelor Graduation Project, based on a project proposal by the company E-trailer. The goal of this project was to implement a 360° surround-view camera system, adapted for the use in the leisure vehicle industry. This proposal was modified to include both a sensing [1] and camera system, the latter of which is described in this thesis.

We would like to thank our supervisor Ing. Jeroen Bastemeijer for his guidance during this project. Furthermore, we would like to express our gratitude to everyone involved at E-trailer for their availability and willingness to help us during these past two months. Lastly, we would like to thank our colleagues Yash Dwarkasing and Batuhan Yildiz for their hard work and productive discussions.

*Hashim Karim, Dennis Landman & Stevan Pavlović  
Delft, June 2024*

# Abstract

The leisure vehicle industry has long trailed the main automotive sector in regards to driver assistance technologies, despite one-sided accidents at low speed being a common occurrence in the industry. This thesis describes the process of designing a wireless 360° camera sensing system in an attempt to catch up this leisure vehicle market to present automotive technology. Such systems can be found in many modern vehicles, however these systems are all wired implementations with there being no need for a wireless solution. The design emphasizes quick and straightforward calibration to facilitate usage as an aftermarket product. A prototype is built using Raspberry Pi's to demonstrate the feasibility of the design. This paper also presents research on possible further improvements of the designed system, and it presents a concept of a composite system through integration with a proximity sensing system.



# Contents

<b>Preface</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Programme of Requirements</b>	<b>3</b>
<b>3 System Outline</b>	<b>4</b>
<b>4 Image Processing</b>	<b>5</b>
4.1 Pre-processing . . . . .	5
4.1.1 Lens Distortion Correction . . . . .	6
4.1.2 Intrinsic calibration . . . . .	9
4.1.3 Top-down View Projection . . . . .	10
4.1.4 Flipping . . . . .	15
4.2 Bird's-eye view . . . . .	16
4.2.1 Stitching . . . . .	16
4.2.2 Brightness Correction . . . . .	17
<b>5 Transmission</b>	<b>19</b>
5.1 Wireless Communication . . . . .	19
5.1.1 GStreamer . . . . .	20
5.1.2 Network Realization . . . . .	23
5.2 Wired Connections . . . . .	24
5.2.1 USB Video Class . . . . .	24
5.2.2 Serializer / Deserializer . . . . .	24
<b>6 Human Machine Interaction</b>	<b>26</b>
6.1 User Experience . . . . .	26
6.1.1 Initial Setup . . . . .	26
6.1.2 Regular Operation . . . . .	27
6.2 Graphical User Interface . . . . .	27
6.2.1 Boot Wizard . . . . .	27
6.2.2 Camsense View . . . . .	27
6.2.3 Rear Cam View . . . . .	27
<b>7 Prototype Implementation</b>	<b>28</b>
7.1 Hardware Selection . . . . .	28
7.2 Proof of concept setup . . . . .	29
7.3 Validation Results . . . . .	29
<b>8 Conclusion and Recommendations</b>	<b>30</b>
<b>Glossary</b>	<b>31</b>
<b>References</b>	<b>32</b>
<b>A Additional Figures</b>	<b>34</b>

# 1

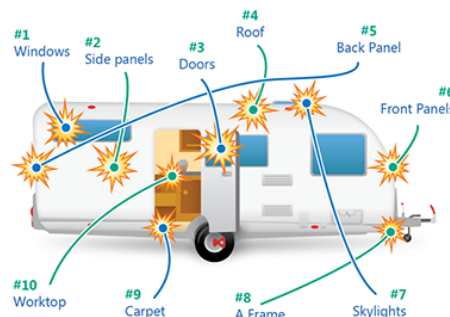
## Introduction

In the Netherlands, leisure vehicles such as caravans and campers are a popular option for vacations. As of 2023, there were over 400,000 registered caravans and over 170,000 registered campers in the country [2]. As people are generally not used to driving such big vehicles and often only drive them during vacation time, it is not unusual for damage to occur to the caravan or camper. The challenges drivers encounter are evident from the insurance data regarding the most common causes of caravan damage [3]:

1. One-sided accidents (parking damage)
2. Storm
3. Hail
4. Collisions with a fixed object (fence, wall, lamppost)
5. Collisions in traffic

In 2023, 35% of damages were caused by one-sided accidents and collisions with fixed objects, such as dent damage suffered during parking or collisions with structures like posts and fences. This statistic seems to imply that many drivers struggle with handling the larger leisure vehicles, especially when they have to navigate their caravan or camper at low speeds. It shows maneuvering through tight spaces can pose challenges for individuals accustomed to smaller, everyday vehicles. Consequently, it is not uncommon for mishaps to occur during such driving scenarios. Almost 16% of all claims occurred while driving the caravan in traffic, again indicating the difficulties drivers face when maneuvering these vehicles outside of their usual driving habits [3].

In order to gain further insight into the challenges drivers encounter while driving their leisure vehicles, data regarding the most accident-prone components of a caravan can also be examined. According to Caravan Guard [4], the parts of the caravan most susceptible to damage are depicted in Figure 1.1.



**Figure 1.1:** Most accident prone parts of a caravan [4]



That same article also provides the common causes of damage to the specific caravan parts such as:

- **Side panels** (second most common type of damage):  
Regularly damaged while reversing, parking, or maneuvering, particularly when on the driveway.
- **Doors** (third most common type of damage):  
In most cases recorded in conjunction with a more serious incident such as a road traffic accident.
- **Back panels** (fifth most common type of damage):  
Typically includes instances where customers have accidentally hit something when reversing or when a third party has run into them.
- **Front panels** (sixth most common type of damage):  
Collisions with structures like lamp posts, railings, fences, and wheel bins, mostly when driving in tight spaces.

This once more shows the challenge drivers encounter when driving leisure vehicles they are not accustomed to, particularly while navigating tight spaces and executing precise maneuvers. Therefore, it would be useful to develop a system to assist drivers of leisure vehicles, which this project aims to address. Such a system could help mitigate the challenges associated with operating these larger vehicles, especially in situations requiring precise maneuvering, ultimately with the goal of reducing the frequency of accidents and damage.

Different systems to serve such a purpose could be considered. One of those systems is a 360° surround view camera system. Such a camera system would provide the driver with a bird's-eye view of the vehicle, eliminating the vehicle's blind spots by providing visual context on objects in the full surroundings of the vehicle. However, with only the visual information, it would still be hard for the driver to gauge distances with certainty. Additionally, a camera-based system might underperform in low visibility conditions such as at night or during fog or mist. The system would also lack a mechanism to alert users when objects are too close, requiring the user to constantly monitor the display while maneuvering.

A second solution to the problem could be a proximity sensor system. Such a system could use proximity detection to alert the user, and can give the user precise input on the exact distances to objects. The flexibility in sensor types also means a system that performs well in all kinds of weather conditions can be designed. This system however, would be unable to visually inform the user. Therefore, while the user would be alerted to the presence of an object, they would not have insight into what this object actually is.

By combining these two systems, a composite system can be created wherein the subsystems account for each others weaknesses. This would result in a surround view system that includes both visual context and a warning mechanism, capable of functioning effectively in various weather conditions. This thesis aims to define the camera subsystem that works as a part of this greater, composite camera-sensor system.

The next part; chapter 2, contains a list of predefined requirements. It is followed by a quick description of the system design in chapter 3. Then the image processing and camera calibration are presented in chapter 4. Following that, in chapter 5 the wireless and wired communication are elaborated upon. Chapter 6 then shows what user interaction will look like. This is followed by chapter 7 which demonstrates the feasibility of the design. Finally, the thesis is concluded in chapter 8.

# 2

## Programme of Requirements

This chapter outlines the functional and non-functional requirements for a 360° camera-based surround view system designed for leisure vehicles. The system aims to provide a comprehensive bird's eye view via an in-car display, ensuring compatibility, efficiency, and ease of installation. Below are the detailed requirements categorized into functional and non-functional sections.

### Functional Requirements

- 360° Camera-Based Surround View System
  - Display: Provides the bird's eye view via an in-car display
- Wireless Communication
  - Communication between System/Cameras and Display
- Compatibility
  - Designed for leisure vehicles
- Performance
  - Speed Range: Designed for speeds up to 20 km/h

### Non-Functional Requirements

- 360° Camera-Based Surround View System
  - Latency: Sufficiently low for real-time feedback
- Installation Time
  - Easy, intuitive setup within 30 minutes
- Power Requirements
  - Power Source: 12V, 10A (maximum)
  - Consumption Target: Less than 2A<sup>1</sup>
- Cost Efficiency
  - Cost Price Target: Approximately €100<sup>1</sup>

---

<sup>1</sup>Requirement of combined system



# 3

## System Outline

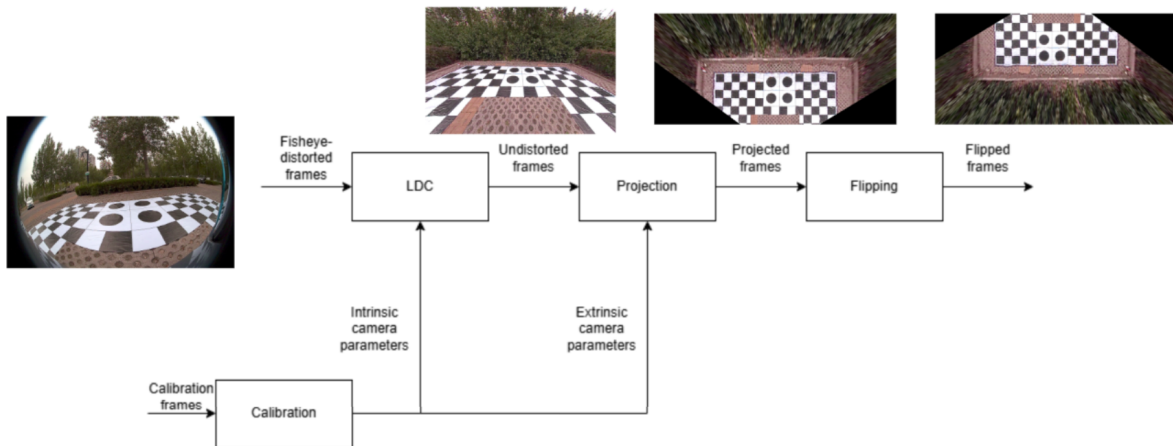
To create a camera-based surround view system, cameras should be mounted in a way that captures information on the entire vehicle surroundings. To limit the number of cameras necessary for this, extremely wide-angle fisheye cameras are often employed [5][6][7][8]. Typically, four to six fisheye cameras, positioned at the front, back, and sides of the vehicle, are sufficient for this purpose. Frames from the mounted cameras are then captured synchronously, after which their information is processed in two stages to achieve a bird's-eye view of the vehicle. In the first stage, the input frames from each camera get processed separately. This process involves correcting the distortion introduced by the fisheye lenses, projecting the corrected images onto the ground plane to create a top-down view, and adjusting the images to their proper orientation in the final composite image. In the second stage, the processed frames from each of the camera streams get combined through a stitching procedure to generate the corresponding bird's-eye view frame. This image undergoes brightness correction to account for the difference in scene illumination between the mounted cameras, in order to ensure seamless stitching. Subsequently, the output frame gets encoded and transmitted. The stitched bird's-eye view image is received at the display embedded system, where it is subsequently decoded. Following the decoding process, the video, along with sensor data from the sensing subsystem, is provided as input to the Graphical User Interface (GUI) which is finally shown on the display. Flowcharts for the transmitting and receiving end systems are included in Figure A.1 and Figure A.2 respectively.

## Image Processing

As described previously, input streams of multiple fisheye cameras have to be processed and combined in order to generate a bird's-eye view of the vehicle. In this chapter, the process of generating a frame in the top-down view output stream will be described. This process can be divided into two parts. The first part, pre-processing, concerns the individual processing of the images from each fisheye camera to prepare them for generating a stitched image. The second part then concerns the stitching of these prepared images to create a composite bird's eye view image, as well as performing color correction on this composite image to account for the discrepancies in scene illumination between the different cameras. This chapter will outline the steps taken in each of these two processing stages.

### 4.1. Pre-processing

The input images of each of the fisheye cameras undergo pre-processing before being stitched together to generate the bird's-eye view image. This pre-processing consists of the steps as outlined in the flowchart shown in Figure 4.1.



**Figure 4.1:** Pre-processing flowchart

The pre-processing of frames consists of three stages, namely Lens Distortion Correction (LDC), projection to the ground plane, and flipping of the frames. In the LDC stage, the distortion introduced by the wide-angle fisheye cameras is corrected for based on the estimated intrinsic parameters of the camera, converting the wide-angle images into undistorted images. In the next stage, these undistorted images are referred to the ground plane through the camera's extrinsic parameters/transformation matrix. Lastly, the undistorted and transformed images are flipped in orientation based on their position in the composite top-down view image. The intrinsic and extrinsic parameters necessary for the LDC and projection stages are obtained through processes of calibration, which will also be discussed in detail in the remainder of this chapter.



### 4.1.1. Lens Distortion Correction

#### Pinhole camera model

Before considering the distortion introduced by camera lenses, the simplest model for a camera, the pinhole model, will briefly be discussed. This discussion aims to introduce part of the intrinsic parameters which are necessary to perform distortion correction.

The pinhole model describes an optical system that uses a small opening, the pinhole, to project an (inverted) image of a scene onto a surface behind the pinhole. It assumes a single ray of light from each point of the scene enters the pinhole and falls onto the image plane, resulting in an inverted image of the observed scene or object. The relation between the size of the observed object and the projected image is then given by the 'focal length'  $f$ , which is a parameter of the camera.

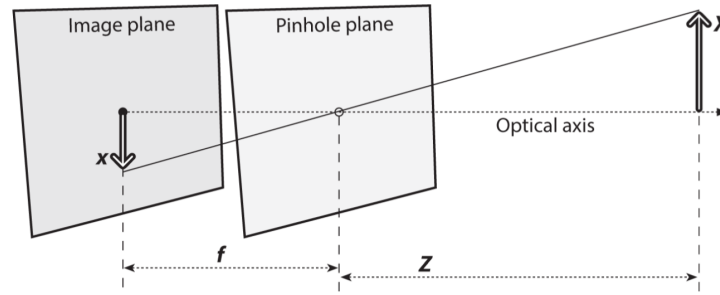


Figure 4.2: Pinhole camera model [9]

In the figure it can be seen that:

$$-\frac{x}{f} = \frac{X}{Z} \quad (4.1)$$

A mathematically equivalent form of the pinhole model that projects the observed scene upright can be achieved by putting the image plane in front of the pinhole:

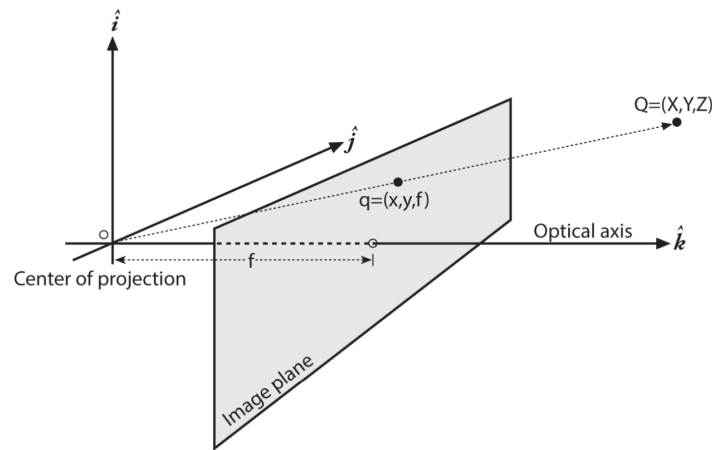


Figure 4.3: Equivalent pinhole camera model [9]

As the projection is now upright, Equation 4.1 reduces to:

$$\frac{x}{f} = \frac{X}{Z} \quad (4.2)$$

The pinhole is now represented by the center of projection to which the singular rays of light from the scene travel, with the illustrated image point as a result. In this equivalent model, the intersection of optical axis (which passes perpendicularly through the center of the pinhole) and the image plane is referred to as the 'principal point'.

Ideally, this principal point would align with the center of a camera's imager (device that captures the image at the image plane). In reality, the center coordinates of the imager are often displaced from the optical axis. This displacement is captured by two of the camera's intrinsic parameters, namely  $c_x$  and  $c_y$ . Moreover, cheap imagers usually have rectangular rather than square individual pixels, meaning cameras have two different effective focal lengths in terms of pixels for the horizontal and vertical directions, namely  $f_x$  and  $f_y$ . These are two more of the camera's intrinsic parameters. When the pinhole model accounts for the imperfections introduced through the inclusion of the imager, it describes the projection of physical world coordinates  $(X, Y, Z)$  onto the imager at a pixel location  $(x_{imager}, y_{imager})$  by the following equations [9]:

$$x_{imager} = f_x \cdot \frac{X}{Z} + c_x \quad (4.3)$$

$$y_{imager} = f_y \cdot \frac{Y}{Z} + c_y \quad (4.4)$$

Usually, the described camera defining parameters are grouped into a single 3x3 'intrinsic camera matrix'  $\mathbf{K}$ :

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (4.5)$$

This allows the mapping of a physical world point  $\mathbf{Q} = (X, Y, Z)$  to a homogeneous point on the imager screen ( $\mathbf{q} = x, y, w$ ), to be written in terms of a single matrix multiplication:

$$\mathbf{q} = \mathbf{KQ} \quad (4.6)$$

In reality, cameras rely on lenses to gather light effectively, introducing distortions to this ideal pinhole model. Consequently, in order to properly perform distortion correction, both the intrinsic camera matrix  $\mathbf{K}$ , which describes the fundamental properties of the camera, and the intrinsic parameters characterizing distortion introduced by the lens have to be found through a process of calibration.

#### Distortion model

In order to generate a bird's-eye view, wide-angle fisheye cameras must be mounted in a way that captures the full vehicle surroundings. While these types of cameras have extremely wide fields-of-view, they also introduce high levels of nonlinear distortion. An important type of distortion to consider for fisheye lenses is radial distortion. The lenses bend rays further from their center more than those closer in, resulting in a nonlinear displacement of image points along a radial axis from a Center of Distortion (COD), causing them to deviate from their positions in the ideal pinhole camera model. A second important type of distortion introduced by fisheye cameras is tangential distortion. This type of distortion is not a feature of the lens, but rather of the assembly of the camera as a whole. It is a result of the lens and imaging plane not being exactly parallel due to various manufacturing defects. With tangential distortion, image points get elliptically displaced as a function of location and radius [9]. These distortions can pose significant problems in the usage of fisheye cameras for the application at hand. Not only does distortion visually deteriorate the captured scene, it also creates problems for further computer vision processing [10], which is an essential component in various ground plane projection processes. Additionally, radial distortion disrupts the rectilinearity of real-world objects, meaning straight lines will

appear curved. All these factors degrade the quality of the composite image stitched together from the fisheye inputs. Therefore, to effectively utilize the wide-angle Field of View (FOV) of fisheye cameras in the bird's-eye view application, it is essential to correct the distortion that is introduced by these cameras.

To perform this correction, it is necessary to obtain the mapping between the distorted and undistorted image points. In some implementations, this relation is found by simply utilizing lens distortion curves provided by the lens supplier [6][8]. As such distortion curves are not available for the acquired cameras, they must be approximated using radial and tangential distortion models instead. A forward radial distortion model, for example, describes to what distorted radius  $r_d$  (measured from the COD) an undistorted radius  $r_u$  gets mapped. To undistort images the inverse of such a model needs to be found and applied, mapping distorted points to their corresponding place in the undistorted image.

While different distortion models exist [10], this implementation will use OpenCV to model and correct for distortion, as it offers models for both radial and tangential distortion. OpenCV simplifies the undistortion process by enabling the correction of both types of distortion simultaneously through a single function, making it more accessible and user-friendly compared to developing distortion models from scratch. This is particularly beneficial considering the time constraints that exist for the project. Moreover, OpenCV offers a calibration procedure that is compatible with these models, meaning straightforward and fairly dependable undistortion can be achieved when this calibration is performed accurately.

OpenCV models radial distortion with the initial terms of a Taylor series expansion around  $r = 0$  from the COD [9], giving the inverse models as seen in Equation 4.7 and Equation 4.8:

$$x_{\text{corrected}} = x \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (4.7)$$

$$y_{\text{corrected}} = y \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (4.8)$$

Where  $(x, y)$  refers to the distorted image point and  $(x_{\text{corrected}}, y_{\text{corrected}})$  to its undistorted counterpart. Since both coordinates are scaled by the same factor, these equations describe the mapping between the distorted and undistorted radial distances for the point under consideration. The  $k$ -terms are called radial distortion terms, with  $k_3$  and higher order terms typically only being included for lenses introducing significant distortion [9], like the fisheye lenses under consideration. The OpenCV inverse models for tangential distortion introduce two other distortion parameters, namely  $p_1$  and  $p_2$ :

$$x_{\text{corrected}} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \quad (4.9)$$

$$y_{\text{corrected}} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \quad (4.10)$$

### Undistortion

The mentioned distortion parameters that define the distortion models, together with the described camera intrinsics matrix  $\mathbf{K}$ , make up the total intrinsic parameters of the camera, and get obtained through a calibration process described in the subsequent section. Based on these intrinsic parameters, an undistortion map can be computed using the `cv::initUndistortRectifyMap()` function, which specifies the mapping of each pixel from the input image to its corresponding location in the undistorted output image. The computed undistortion map can then be applied to distorted images using the `cv::remap()` function, which returns the corresponding corrected images as output. Applying this process to example images taken by cameras with known estimated intrinsics gives the undistortion result as seen in Figure 4.4:



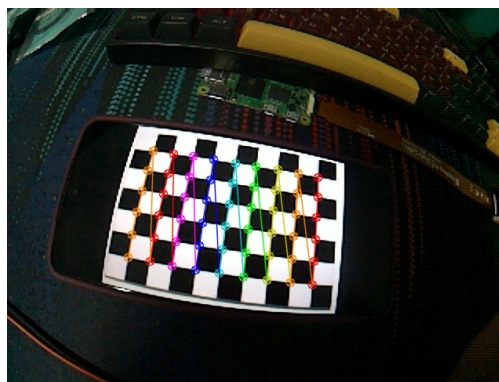
**Figure 4.4:** Undistortion using OpenCV

### 4.1.2. Intrinsic calibration

As previously stated, OpenCV offers a (fisheye-specific) calibration process to find the intrinsic parameters necessary for the undistortion of fisheye images. In this section, this well-documented calibration process will be briefly described, without going into detail on the underlying mechanisms that allow the intrinsics to be estimated.

The general calibration process in OpenCV involves capturing multiple images of a known calibration pattern, such as a chessboard, from different angles and positions. Usually 10 or more of such images are needed for accurate results [9]. In these images, the entirety of the used chessboard should be visible. The captured images are then processed to detect the calibration points, namely the internal chessboard corners, using the dedicated `cv::findChessboardCorners()` function. In order to improve the calibration accuracy, the locations of the detected corners are refined using the sub-pixel corner detection function `cv::cornerSubPix()`. Once the corner points have been accurately detected and refined, these points, along with the known dimensions of the calibration pattern, are used to perform the calibration using the `cv::fisheye::calibrate()` function. This function estimates the described intrinsic parameters of the fisheye by minimizing a re-projection error, defined by the coordinate difference between the localized corner positions and their projected positions based on the estimated parameters.

For this project, to avoid having to take and upload individual pictures with each of the fisheye cameras, the general calibration process has been adapted to extract calibration images directly from camera streams. Only the frames where all internal chessboard corners can be localized are extracted, which can be achieved through the fact that `cv::findChessboardCorners()` sets a flag when this is the case. To visualize to the user that the way in which the camera views the pattern generates usable calibration frames, the `cv::drawChessboardCorners()` function is used to highlight the localized internal corners, as illustrated in Figure 4.5:



**Figure 4.5:** Highlighted internal chessboard corners for usable camera stream frame

After the number of usable extracted frames reaches a certain threshold, which is set higher than the specified minimum of 10, the intrinsic calibration can automatically be completed, giving the intrinsic parameters needed for the Lens Distortion Correction.

#### 4.1.3. Top-down View Projection

After performing the intrinsic calibration and using the obtained parameters to undistort the fisheye inputs, the undistorted images must be registered with the ground plane to generate a bird's-eye view of the partial scenes captured by the fisheye cameras. For this, geometric manipulations of these images are needed. Specifically, perspective transforms or homographies must be performed. Such methods effectively compute the way in which a plane in three dimensions is observed by a particular observer, in this case for one looking at the captured plane from a top-down perspective. Perspective transformations are specified completely by a single 3x3 matrix, typically called the projection or homography matrix. The mapping between a pixel coordinate in the undistorted image and its corresponding coordinate in the projected image is then given by:

$$\mathbf{p}_{proj} = \mathbf{M}\mathbf{p}_i \quad (4.11)$$

$$(x, y, 1) = \begin{pmatrix} M_{00} & M_{01} & M_{02} \\ M_{10} & M_{11} & M_{12} \\ M_{20} & M_{21} & M_{22} \end{pmatrix} \times (u, v, 1) \quad (4.12)$$

Where  $\mathbf{M}$  is the projection matrix. The parameters of the projection matrix are defined by 'extrinsic' camera parameters, which relate to the position and orientation of the camera. The process of obtaining the bird's-eye view generating projection matrix will thus be referred to as extrinsic calibration.

It is important to highlight the difference in process that exists between intrinsic and extrinsic calibration. The parameters found through intrinsic calibration characterize the internal properties of the camera. These parameters usually remain constant after production [9], and are unaffected by how the camera is installed by individual customers. Therefore, intrinsic calibration can be conducted beforehand, separate from the installation process of the surround view camera system. On the other hand, the projection matrix obtained during extrinsic calibration is dependent on the camera's position and orientation. This means that the way in which individual customers mount the camera directly affects the extrinsic parameters. Consequently, unlike intrinsic calibration, the extrinsic calibration can only be done after the cameras have been mounted. This means that the process of extrinsic calibration has to be performed by the customer buying the surround view system. Therefore, while considering the duration and complexity of the process is less critical for intrinsic calibration, as it can be performed by experts and within any timeframe before product delivery, accounting for these factors becomes crucial in selecting the procedure for extrinsic calibration. Here, they directly impact the user experience and installation time for customers tasked with performing this calibration. With this in mind, the remainder of this section will outline different implementations of the extrinsic calibration process, along with their respective advantages and disadvantages concerning customer usability.

##### Manual calibration approach

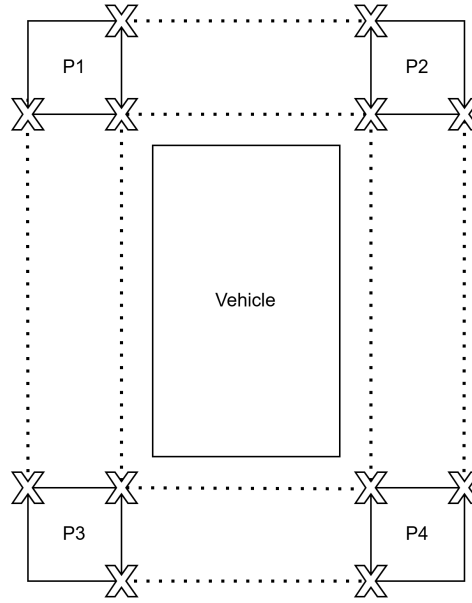
The first possible implementation of the extrinsic calibration process falls under the category of manual calibration methods. These types of methods rely on putting patterns with recognizable features, usually checkerboards or calibration charts, in the field of view of the cameras. These patterns have a known geometry, meaning the real-world coordinates of feature points in the patterns can be easily obtained, as well as their corresponding pixel coordinates in the projected image. By manually specifying the mapping between a sufficient number of pixel coordinates of these feature points in the undistorted image to their corresponding coordinates in the projected image, the projection matrix can then be estimated.

OpenCV offers functions which are directly compatible with this type of extrinsic calibration process. The function `cv::getPerspectiveTransform()` takes a list of at least 4 source pixel coordinates from

the undistorted image and their corresponding destination pixel coordinates in the projected image, based on which it can compute the 3x3 projection matrix  $M$  necessary to achieve the specified mapping relationships.  $M$  can then be applied to the undistorted images using the `cv::warpPerspective()` function. The value at each pixel coordinate in projected image is then computed from the undistorted image as [9]:

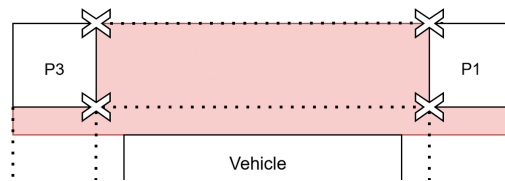
$$\mathbf{I}_{\text{proj}}(x, y) = \mathbf{I}_{\text{und}} \left( \frac{M_{00}x + M_{01}y + M_{02}}{M_{20}x + M_{21}y + M_{22}}, \frac{M_{10}x + M_{11}y + M_{12}}{M_{20}x + M_{21}y + M_{22}} \right) \quad (4.13)$$

Here the division by the  $M_{20}x + M_{21}y + M_{22}$  term follows from the conversion to inhomogeneous coordinates. The function also allows the user to specify the size or projection area of the output image. To correctly utilize these two functions to generate projected images suitable for the generation of the composite bird's-eye view, the layout of the final image must first be defined based on the calibration site setup. An example of a possible calibration site setup that would provide the information needed by the two functions is given in Figure 4.6



**Figure 4.6:** Possible calibration site layout

In this setup, 4 patterns of known dimensions must be placed at a fixed distance (with regard to both width and height), in the overlapping regions of view of the camera, with the four necessary feature points in the projected images defined and observable in the camera frames. Placing the patterns enables the sharing of feature points on the patterns to select, eliminating the need to surround the entire vehicle with calibration patterns. Besides this, the setup requires vehicle width and length to be known. To show how the extrinsic calibration can now be done, the projection area of the left-viewing fisheye camera as seen in Figure 4.7 is taken:



**Figure 4.7:** Projection area of left-viewing fisheye camera



Here, the upper left point of the highlighted area is the origin point with respect to which the destination pixel coordinates of the feature points will be defined. Defining each pixel in the final image to correspond to 1cm in real-world distances, combined with the fact that all included distances of the calibration site are known, means all the destination pixel coordinates can be defined through these distances. Moreover, the size of the projected image can also be defined through this information. By finally selecting the four pixel coordinates of the annotated features in the undistorted image, the input requirements of both OpenCV functions are fulfilled, enabling the extrinsic calibration and the creation of a top-down view projection suitable for composite bird's-eye view generation.

Completing the described extrinsic calibration process on the undistorted example images and subsequently applying the obtained projection matrix to said images, yields the top-down view projection result as seen in Figure 4.8:



**Figure 4.8:** Top-down view projection for manual extrinsic calibration using OpenCV

The requirements for the described manual calibration implementation to function correctly make this process sub-optimal with regard to practicality and customer experience. The major disadvantage of this method is the significant responsibility it places on the customer for the calibration process, going beyond the initial installation of the cameras. Firstly, it requires the customer to 'build' the calibration site, while keeping in mind the specific pre-defined geometry required. This also means that a significant area around the caravan needs to be available during calibration, and that calibration patterns have to be shipped with the product. To add to this, the process requires the customer to correctly select the pre-defined feature points in the images, leaving potential for human error.

Despite these drawbacks, this solution offers rather reliable results [12]. While the process will never be optimal with regard to duration and complexity, creating supporting modules for this method can help in still making the method relatively easy to complete. The requirement of only four calibration charts, rather than a pattern surrounding the entire vehicle, simplifies the site setup. Additionally, implementing a dedicated GUI and providing clear instructions makes the selection of the necessary points straightforward and easy.

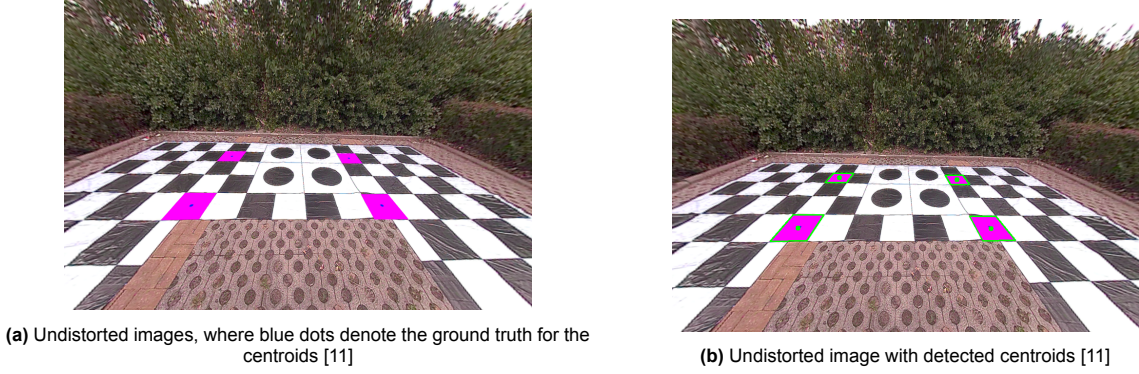
#### Computer vision approach

The next possible implementation of extrinsic calibration focuses on simplifying the previously described process for users, effectively automating the more intricate steps. When examining the manual calibration approach, two main areas, if simplified, would significantly enhance the user experience. Firstly, eliminating the need for manual distance measurements during the setup of the calibration site would significantly speed up this process. Secondly, removing the requirement for users to manually select image points in the undistorted images would not only reduce the risk of human error but also accelerate the overall procedure.

A practical way to achieve this is to utilize computer vision to find the source pixel coordinates and the sensor subsystem to find real-world coordinates. Therefore, an object must be designed that can be detected by both computer vision and the sensor subsystem while minimizing inaccuracies.

The first step is to determine the optimal object for each detection method. For computer vision, two

considerations are crucial: the object should be as flat as possible to be close to the ground plane for accurate projection, and it should be recognizable at both close and far distances from the camera. A flat square with a distinct color is optimal, as the centroid can be calculated by finding the four corners of the polygon, even if distorted due to perspective. A distinct color simplifies the process by allowing for easy masking and contour detection. To find the centroid, `cv2.inRange()` is used to create a mask around the specified color's HSV value, and `cv2.findContours()` identifies contours within that mask. The contour is then approximated to a polygon using `cv2.arcLength()` and `cv2.approxPolyDP()`, and the centroid is calculated using `cv2.moments()`, giving the results as shown in Figure 4.9:



**Figure 4.9:** Centroid detection using computer vision

However, a flat object presents a challenge for the sensor subsystem, as it is difficult, if not impossible, to detect. Cones were initially considered but proved unsuitable due to the varying radius from the centroid and field of view blockage. A cylinder with a known radius could work, as the real-world coordinates can be determined using Equation 4.14, where  $r_{cylinder}$  is the known cylinder thickness,  $(x_{real}, y_{real})$  are the real centroid coordinates,  $(x_{measured}, y_{measured})$  are the coordinates measured by the sensor subsystem, and  $(x_{camera}, y_{camera})$  are the camera coordinates.

$$\begin{pmatrix} x_{real} \\ y_{real} \end{pmatrix} = \begin{pmatrix} x_{measured} \\ y_{measured} \end{pmatrix} + \frac{r_{cylinder}}{|r_{measured}|} \cdot \begin{pmatrix} x_{measured} - x_{camera} \\ y_{measured} - y_{camera} \end{pmatrix} \quad (4.14)$$

Thus, the optimal object is a cylinder with a flat square base. For improved consistency in centroid detection and reduced packaging size, the cylinder should be designed to be removable. This would allow for the detection of all centroids using only the base. After the centroids are detected, the cylinders can be added to determine the real-world coordinates, enabling the extrinsic calibration process.

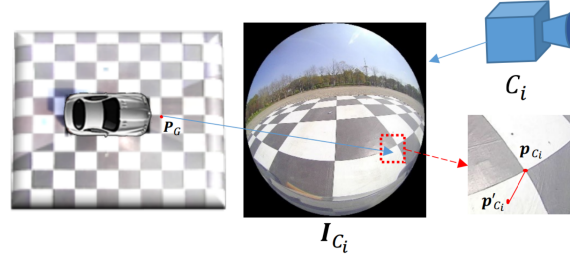
#### Deep learning approach: WESNet

The final possible implementation of the extrinsic calibration process falls under the category of self-calibration methods. These types of methods aim to remove responsibility from the user by allowing the system to autonomously determine its extrinsic parameters, offering significant benefits in terms of customer usability. One of the existing self-calibration schemes, based on a weakly supervised deep learning framework, is Weakly-supervised Extrinsic Self-calibration Network (WESNet). This calibration scheme is described in detail in [12]. In this section the scheme's main features described in [12] will be outlined, and it will be discussed how this scheme could be modified to be compatible with use in leisure vehicle applications and what the advantages of such an implementation would be.

The WESNet implementation offers extrinsic self-calibration specially designed for surround view camera systems, which contrary to other existing self-calibration schemes can be applied stably in various environmental conditions. Essentially, the trained WESNet model can take the group of four synchronously captured fisheye frames, and directly output the extrinsic parameters necessary for top-down view projection. The model relies completely on these natural images and the customer does not need any other objects to assist in the calibration. To this end, the multi-layered Convolutional Neural

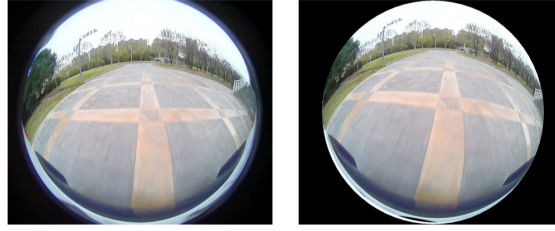
Network (CNN), which the WESNet architecture is based on, undergoes two stages of training. These training stages are specifically adapted to optimize the CNN for estimating extrinsic parameters that generate seamless bird's-eye view images.

The first training stage uses weak supervision to do a first rough optimization of the network based on a geometric loss. This training stage relies on a small amount of undistorted images of a calibration site filled with chessboard markings, whose known dimensions are used to provide weak supervision information used in the geometric loss function that is optimized. To this end, the pixel coordinates ( $p_C$ ) of 20 to 30 corners in each calibration site image are marked, and their real-world coordinates ( $P_G$ ) are extracted from the marking dimensions. Only this process in the scheme requires dedicated manual labor. As the output of the WESNet model estimates the extrinsics (which describe the camera pose), the extracted  $P_G$  coordinates can be mapped to corresponding pixel coordinates based on these estimated parameters. The loss term is then established as the distance between this predicted/mapped point  $p'_C$  and the actual marked point  $p_C$ , as illustrated in Figure 4.10. Optimizing for this reprojection loss thus allows WESNet to roughly learn to estimate the extrinsic parameters/poses of cameras in a set reference system. This process is weakly supervised in the sense that the weights are not optimized based on ground truth labels of the complete extrinsic parameters of the mounted cameras. This would require first fully calibrating the cameras using for example manual calibration methods as described before, which would be too labor consuming in preparing the dataset. Using the corner information thus serves as a more efficient alternative for this.



**Figure 4.10:** Loss based on distance between point  $p_C$  and  $p'_C$ , where  $p'_C$  is the point mapped to the undistorted image through extracted  $P_G$  and network estimated extrinsic parameters [12]

In the second training stage, the model is fine-tuned based on photometric information from groups of synchronously captured natural fisheye images. The loss function in this stage is defined by the amount photometric discrepancy in the overlapping regions of view of adjacent cameras. Minimizing this loss will optimize the extrinsics estimated by the model for generating top-down view projections that allow for seamless composite bird's-eye view images. To facilitate this training phase, the researchers captured over 19,000 fisheye image groups in a wide variety of environments. This ensures WESNet can perform the extrinsic calibration stably under different environmental circumstances, although it is noted that for WESNet to operate optimally the calibration should be performed on flat ground with some larger textures in view. To generalize the model to accommodate individual customers, each of whom will mount the cameras differently, causing their extrinsic parameters to change, the variety of extrinsics in the dataset needs to be increased. However, mounting the cameras in different poses and capturing their frames would be labor-intensive process. Instead, the extrinsic variety in the dataset necessary for generalization is achieved by data augmentation through applying a range of homography transformations to frames captured by cameras mounted in a single fixed position, simulating the pose of the cameras being adjusted. The result of this data augmentation process is illustrated in Figure 4.11:



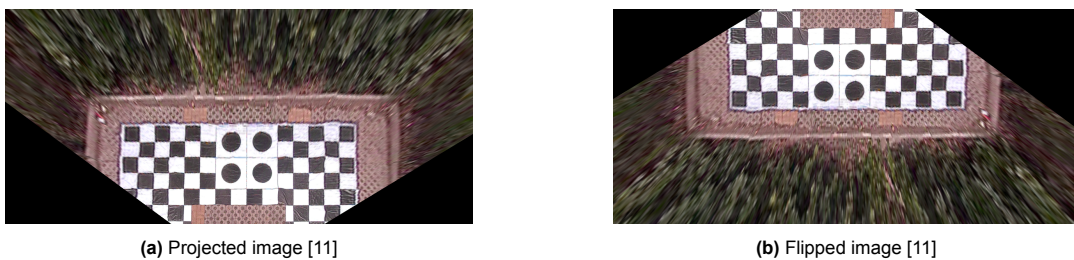
**Figure 4.11:** Applying homography to original (left) to simulate cameras being mounted differently during individual installations [12]

After both training stages are completed, WESNet can be used to directly generate the extrinsic parameters necessary for the top-down view projection, by taking only a group of synchronously captured natural fisheye frames as input. This has major benefits with regard to customer usability, as it effectively removes any labor-intensive responsibility of the customer. It removes much of the restrictions on the calibration environment, as a sizeable and user-made calibration site as needed in the described manual calibration methods is no longer necessary. Moreover, due to the online nature of the calibration, a possible re-calibration can be done quickly, without having to go through a labor-intensive process all over again. This is particularly useful in making the method flexible against potential changes in the camera's orientation over time, for example due to vehicle damage.

While the datasets and source codes for WESNet are publicly available, they can not be used directly for extrinsic calibration for the prototyping of this project. This is due to several factors. Firstly, the natural images in the dataset were captured by cameras mounted on a car, meaning that even with data augmentation, the general extrinsics within the dataset differ significantly from those of cameras mounted on the larger leisure vehicles. Besides this, the quality of cameras used for creating the dataset is better than the ones acquired for prototyping. In order to fine-tune the WESNet solution for use in leisure vehicles, the described process of dataset generation should be repeated with the product fisheye cameras mounted around a leisure vehicle. Capturing frames from the camera streams for the leisure vehicle driving on various surfaces and subsequently performing the outlined data augmentation would enable the creation of a purpose-fitted test dataset. Subsequently, training the WESNet model with this fitted dataset would tune the model for leisure vehicle applications, thereby unlocking all the customer usability benefits associated with this specific extrinsic calibration method.

#### 4.1.4. Flipping

Before the projected images from the different camera views (front, left, right, back) are combined into a single bird's-eye view, they have to be changed in orientation in accordance with the layout as provided in Figure 4.6. This done by transposing the image arrays appropriately, giving the result for the example projected back camera image as seen in Figure 4.12:



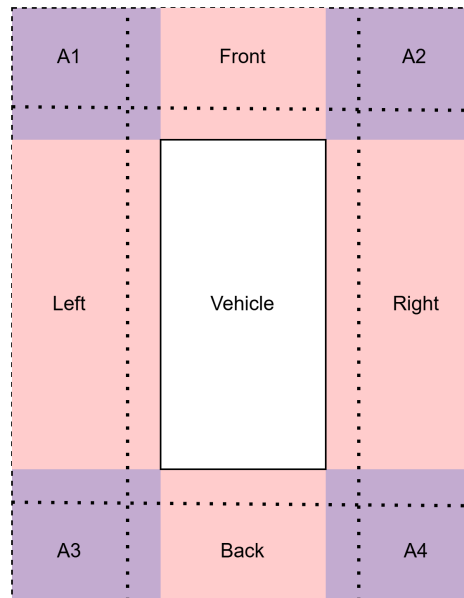
**Figure 4.12:** Adjusting the orientation of projected frames to fit final composite image

With this, the pre-processing of the individual fisheye input frames is complete, meaning the different camera views can be combined in order to generate a single bird's-eye view image.



## 4.2. Bird's-eye view

In order to generate the composite bird's-eye view image, the undistorted, projected and flipped images must be put in the appropriate positions in a final image layout such as presented in Figure 4.6. In the projection stage, the size of the projected frames was set to fit the projection area as presented Figure 4.7. Consequently, after placing these projected images in the final image layout, two types of regions of regions can be distinguished: overlapping regions and non-overlapping regions of view, as illustrated in Figure 4.13:



**Figure 4.13:** Overlapping regions (purple) and non-overlapping regions (pink)

In the non-overlapping regions, the bird's-eye view's visual information is derived from a single camera view. Thus, the sections of the projected images that fall into these non-overlapping regions can directly be copied to form those parts of the composite image. In the overlapping regions of view, the visual information of two adjacent camera views is present. At the points in these regions where the non-zero valued parts of the images actually overlap, information must be merged to create the bird's-eye view visual, which is achieved through a process of stitching. After the projected images have been appropriately stitched, the remaining pixels are filled with a placeholder, such as a schematic drawing of a caravan, which matches the vehicle's dimensions according to the established pixel-distance ratio, resulting in the final bird's-eye view image.

### 4.2.1. Stitching

At the actual image intersections within the defined regions of overlap, each coordinate has two corresponding non-zero pixel values from which a single pixel value has to be computed. Simply taking the average of these pixel values would result in a final composite images with clear seams that show where the stitching took place. To merge images from adjacent cameras seamlessly, it is essential to accurately process the exact region where the non-zero valued parts of the images intersect within the defined regions of overlap.

One simple implementation of such a process begins by identifying and isolating the common region shared by both images [11]. The overlapping region is then converted into a simpler gray-scale format to facilitate further processing. Techniques like thresholding are applied to highlight important features and remove noise, resulting in a clear mask representing the intersecting area.

Next, the boundaries of the parts of the overlapping region outside the intersecting area are determined using contour-finding algorithms to trace the outer edges of these parts. These contours, representing

the overlapping regions excluding the intersecting part, are approximated into polygonal shapes to simplify handling in subsequent steps.

For each pixel in the intersecting region, its distance to the these outer boundaries of the adjacent images is determined. The boundaries are denoted as polygons  $polyA$  and  $polyB$ . Using the function `cv2.pointPolygonTest`, the distance of each pixel in the overlapping area to both  $polyA$  and  $polyB$  is calculated, denoted as  $d_A$  and  $d_B$ , respectively. The weight  $w$  for each pixel is then computed using Equation 4.15, ensuring that pixels closer to  $polyA$  receive a higher weight from the image associated with  $polyA$ , and vice versa.

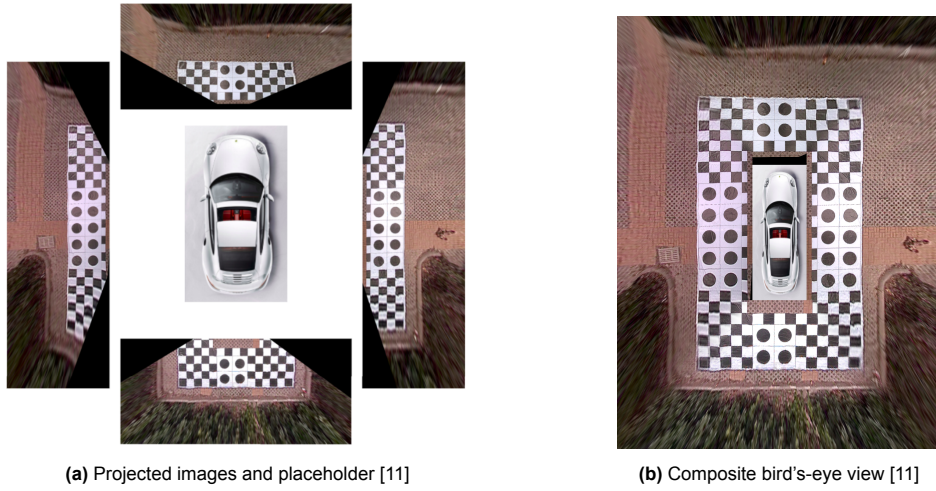
$$w = d_B^2 / (d_A^2 + d_B^2) \quad (4.15)$$

The weights are dynamically assigned based on each pixel's proximity to the boundaries. Using the distances  $d_A$  and  $d_B$ , a weight matrix  $G$  is computed, where each element corresponds to a pixel in the region of overlap. For pixels in the region of overlap that are outside the actual intersecting region, weights are straightforwardly set to 1 if the pixel belongs to the respective image and 0 otherwise. The weight matrix  $G$  contains values ranging from 0 to 1, providing a continuous gradient that dictates the blending of the two images.

Using the weight matrix  $G$ , the final blended image that fills the region of overlap is computed as shown in Equation 4.16. This formula combines the pixel values from the two images, weighted by the matrix  $G$ .

$$image_{fused} = image_A \cdot G + (1 - G) \cdot image_B \quad (4.16)$$

Through this process, each pixel in the intersecting region is blended based on its distance to the defined boundary around the non-intersecting image parts, and each pixel in the remainder of the region of overlap is taken directly from these non-intersecting image parts. Performing this process on the projected example images leads to the composite bird's-eye view image as shown in Figure 4.14:



**Figure 4.14:** Stitching of image elements based on weighted averages and resizing of placeholder based on vehicle dimensions

#### 4.2.2. Brightness Correction

In the shown composite image, differences in scene illumination between the cameras make the stitching seams visible. To correct for this, a consistent overall brightness for the image must be achieved. To this end, the brightness levels of the different image areas need to be adjusted to account for the varying exposure levels. The process by which this is done will be briefly outlined in this section.



Each camera captures an image in BGR (Blue, Green, Red) format, resulting in three color channels per camera. With four cameras, this means there are a total of 12 channels based on which color correction can be performed. Firstly, the brightness ratios for the overlapping regions between adjacent camera images are calculated. Let  $I_1, I_2, I_3, I_4$  represent the images from the four cameras, each having B, G, and R channels (e.g.,  $I_{1B}, I_{1G}, I_{1R}$  for the first camera). The brightness ratio for the overlapping regions between two channels from different cameras is given by:

$$r_{ij} = \frac{\text{mean}(I_{iC})}{\text{mean}(I_{jC})}$$

where  $\text{mean}(I_{iC})$  is the mean brightness of channel  $C$  in image  $I_i$ , and  $r_{ij}$  is the ratio of mean brightnesses between image  $I_i$  and image  $I_j$ .

Using these brightness ratios, adjustment coefficients for each channel can be computed by taking the geometric mean of the brightness ratios for that channel across all images:

$$k_C = \left( \prod_{i=1}^4 r_i \right)^{\frac{1}{4}}$$

To refine these coefficients, an exponential tuning function can be applied:

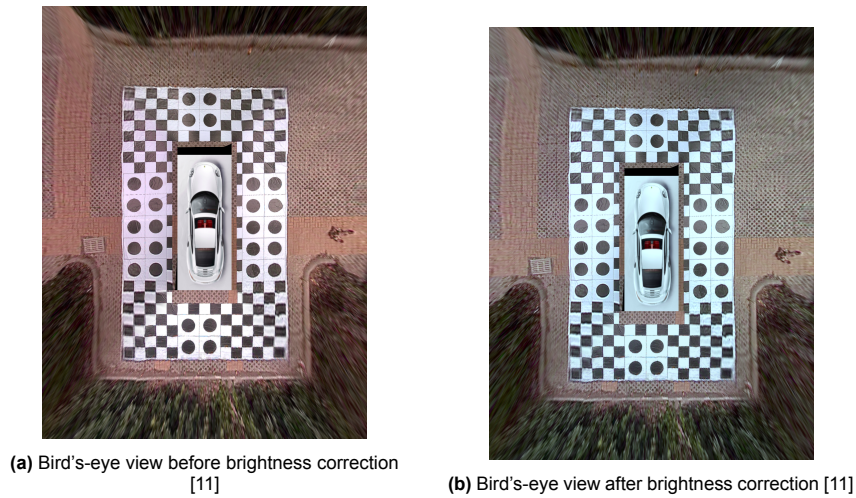
$$k_C = k_C \times \exp((1 - k_C) \cdot p)$$

where  $p$  is a parameter that depends on whether  $k_C$  is greater than or less than 1 (e.g.,  $p = 0.5$  if  $k_C \geq 1$ , otherwise  $p = 0.8$ ).

The brightness of each channel is then adjusted by multiplying it by its corresponding coefficient:

$$I'_{iC} = I_{iC} \times k_C$$

This adjustment process is applied to all channels across all images. Finally, the adjusted channels are combined back together to form the final stitched image with consistent brightness and thus a more seamless result, as can be seen in Figure 4.15:



**Figure 4.15:** Performing brightness correction on the composite bird's-eye view image

# 5

## Transmission

In order to have a complete system, data from each individual camera needs to be combined in a processing hub and thereafter displayed to the driver in the front of the vehicle. This chapter will detail the process of connecting the unique components of the system in accordance with the requirement defined previously in chapter 2. These requirements define the need for a wireless connection between the camera and display parts, while also aiming for a real-time system. Such a combination means finding a balance between total system latency, hardware price and output video resolution. The steps taken to find this balance are described during this chapter. There is however also the connection between the different cameras and the processing hub which performs the post-processing from section 4.2. Since each camera will need to be connected to power, the possibility of a wired solution for the data transfer between these components is also explored in this chapter.

### 5.1. Wireless Communication

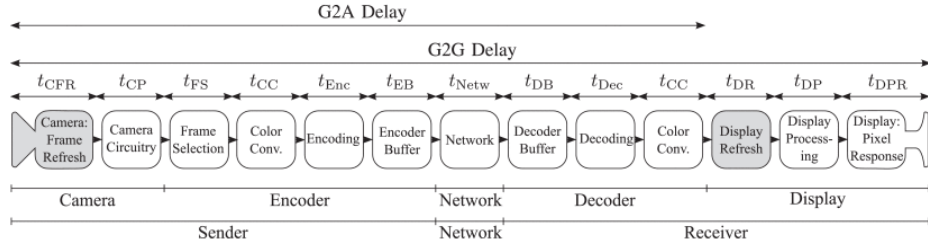
The main challenge concerning wireless communication is obtaining a reliable, high-resolution signal without adding too much latency. Defining when the added latency is "too much" is already challenging on its own, since it depends heavily on the context in which this latency occurs. When controlling a vehicle of any kind remotely and based solely on the video feed of said vehicle's camera(s), the experienced latency should be sufficiently low to be able to act in time when avoiding an obstacle [13], [14]. The latency between sensor (camera) and display is known as the Glass-to-Glass (G2G) delay [15], [16] which together with the network latency and the delay between the action of an operator and that of an actuator in the vehicle makes up the End-to-End (E2E) delay of the system. The latter "actuator delay" is relevant in teleoperation cases such as the scenario described prior, however for this project the G2G latency is most interesting as that will also be the E2E latency in this case.

Unlike Bachhuber and Steinbach [15] and others envisioning applications tailored to the ultra fast "Tactile Internet" (E2E delay downwards of 10 ms) [17], during this design process there is not necessarily a heavy focus on minimizing delays towards the just noticeable difference (JND) since the vehicle is considered to be moving at lower speeds during regular system operation. Table 5.1 shows the effect of larger latencies on the theoretical performance of the system while moving at low speed. From the table it can be seen that at the earlier specified 20 km/h the vehicle moves roughly a meter at upwards of 200 ms latency. It is possible to analyse the table further by using the rule of thumb on braking distance of a vehicle with trailer/caravan being around double the regular distance. Here that regular distance is taken for an average UK car as described in *The Highway Code* and simultaneously 30 km/h is used for simplicity of conversion from miles. Then the braking distance becomes roughly 12 m and the stopping distance, obtained by also adding the estimated thinking distance, becomes roughly 18 m. From that, assuming a latency in the region of 150 ms, the total stopping distance increases by around 7% to almost 20 m due to the additional G2G delay. The impact is emphasized further by considering that this same latency accounts for roughly a 20% increase in thinking distance or processing time and it grows to almost 35% for 250 ms delay.

**Table 5.1:** Distance in meter that the vehicle travels during the G2G delay

Speed [km/h]	Latency [ms]						
	10	50	100	150	200	250	500
5	0.014	0.069	0.139	0.208	0.278	0.347	0.694
10	0.028	0.139	0.278	0.417	0.556	0.694	1.389
<b>20</b>	<b>0.056</b>	<b>0.278</b>	<b>0.556</b>	<b>0.833</b>	<b>1.111</b>	<b>1.389</b>	<b>2.778</b>
30	0.083	0.417	0.833	1.250	1.667	2.083	4.167

Previous work [13], [14], [16] provides a detailed subdivisions of the G2G delay according to the illustration shown in Figure 5.1. In the remainder of this section the encoder, decoder and network components are particularly interesting to delve into further. Both the encoding and decoding can be realized through either hardware or software. The former however does require specific components, such as described by Rommel [18] for within the camera module or more commonly by use of a GPU. Such "*hardware acceleration*" has the potential to significantly reduce the processing required compared to software encoding which is entirely done on the CPU [19]. Additional dedicated hardware might significantly increase the total price, however the benefit is worth it considering the importance of increased performance in this critical part of the system [20].

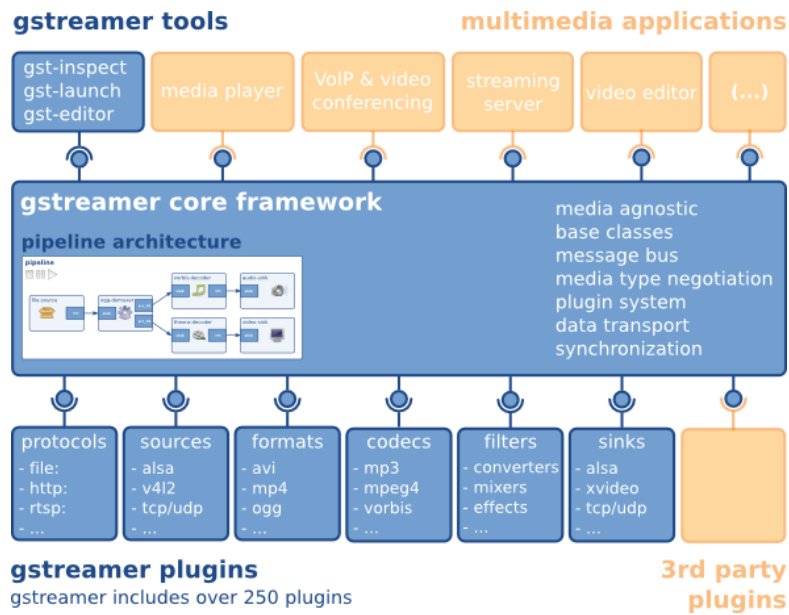
**Figure 5.1:** Overview of the different components that make up the G2G delay [16].

During the project, applications heavily involving connection to the internet were deemed infeasible since the availability of internet is all but guaranteed with leisure vehicles often being in rather remote locations. The emergence of edge computing would allow for the distribution of some of the more demanding computational tasks, thus potentially alleviating the need for significant hardware on the system itself [21], [22]. Such solutions would open the door for more sophisticated Artificial Intelligence (AI) models to further improve upon and even expand the current system. Nevertheless, for the scope of this project only Peer-to-Peer (P2P) networking is explored. In that field, *GStreamer* emerged as the clear optimal solution.

### 5.1.1. GStreamer

GStreamer defines itself as "a framework for creating streaming media applications". It allows for 'easy' implementations of complex multimedia applications. The framework is extremely flexible, allowing a wide variety of input and output file formats and many ways of processing the data in-between because of the pipeline design. Figure 5.2 is an attempt by GStreamer itself to show this flexibility graphically. The earlier mentioned pipeline architecture is important for applications that, similar to the image processing in chapter 4, need to process data in a set order. Furthermore, with the pipeline designed to have minimal overhead and extensive threading support, the framework achieves high performance even on high-end applications for real-time conferencing. That all makes GStreamer especially suitable to be used in media players applications, although its potential goes much further. The framework is designed as a base for plug-ins to add a plethora of extra functionalities like support for a broader selection of input and output drivers for additional hardware and streaming options. Such plug-ins makes it possible to use GStreamer on edge devices like *Raspberry Pi* and *Nvidia Jetson* for Internet of Things (IoT) implementations, and for streaming media from device to device or application like with RTSP,

WebRTC and HLS.



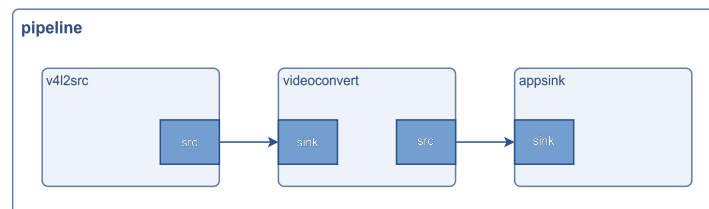
**Figure 5.2:** Overview of the GStreamer project. Source: GStreamer

Another big benefit of GStreamer is the fact that it and the majority of its standard plug-ins are open-source, released under the LGPL. Also, even though GStreamer itself is written entirely in C, language bindings are available for both *C++* and *Python* among others. Furthermore there are some built-in features that are very useful for an implementation of this project. Firstly, GStreamer comes by default with wrappers for a select few OpenCV functions, however it also supports data exchange with external applications like OpenCV via the `libgstapp` library. That way the complete library of OpenCV can be used for the image processing described previously. Secondly the framework has native support for integration with GUI toolkits like *GTK+* and *Qt*. Those toolkits can create a window for the GStreamer video to be played at, while concurrently handling user interaction. Finally, the pipeline allows control signals to travel upstream which can be used to for example update the Lookup Tables (LUTs) after calibration.

The complete implementation of the required system will make for a very complex GStreamer pipeline. Therefore in the next part of this chapter the pipeline is split into more manageable portions to be described.

### Transmitter Pipelines

The transmitter side of the wireless transfer component can be further split into two separate pipelines. The first, shown in Figure 5.3, handles the input stream from one camera. For a final implementation of this project there would thus be four of such pipelines, one for each deployed camera.



**Figure 5.3:** GStreamer pipeline representation of the input streams coming from the different cameras

The video stream of the camera enters the pipeline through a GStreamer source element. In Figure 5.3 this element is denoted as `v4l2src`, where the first part of the name describes the plug-in that supplies

it; in this case V4L version 2. Video4Linux (V4L) adds support for real-time video applications on *Linux* based operating systems such as *Raspberry Pi OS*. By using this specific source, GStreamer can read the data from the camera while it is connected as a Universal Serial Bus (USB) device.

The source is linked to a “filter-like” element, `videoconvert`, to automatically convert the format of the video to one accepted by the next linked element. This is achieved through a process called “*caps negotiation*” which is done by the connecting pads; `src` and `sink` within the darker blocks of each element in the figure. Without this element the source and sink of this pipeline would be unable to transfer data. The final element of this first pipeline is the sink, denoted as `appsink` in Figure 5.3. As mentioned before, `appsink` and its counterpart `appsrc` allow other applications to access the data from the GStreamer pipeline with their use being described as “short-cutting the pipeline”. By extracting the raw pixel data with this sink into the OpenCV application described in section 4.2, the full extend of OpenCV’s functionality can be used to process the video compared to using only GStreamer wrapped functions.

After OpenCV has finished processing the video frame it should be imported back into the next GStreamer pipeline. Therefore, the second pipeline of this implementation starts with a `appsrc` element as shown in Figure 5.4.

Since this source only provides unformatted data, another `videoconvert` filter element is linked to provide the next element with the appropriate data format.

That next element is critical in ensuring real-time performance of the system. As described prior, the realization of the encoding and decoding is largely responsible for the total G2G latency. GStreamer offers support for many different encoders and decoders as their performance is often directly linked to the environment they are used in. Many commonly used encoders within GStreamer are software based, such as `x264enc`. However, in the case of this project, those would likely not work since their implementation on edge devices are lackluster. The pipeline in Figure 5.4 instead uses another V4L plug-in addition in the shape of `v4l2h264enc`. As is in the name, this codec uses the added V4L2 functionality to encode raw data into the AVC/H.264 format using an available Graphical Processing Unit (GPU). This plug-in is ideal in the case of using devices from for example Raspberry Pi, though there exist many other plug-ins designed for other prevalent manufacturers like `nvcodec` for use on Nvidia devices.

Linked to the source pad of the encoder is another version of a filter element. Similar to earlier usage of `videoconvert`, the `h264parse` element applies automatic reformatting to comply with the requirements of the `sink` and `src` pads it connects. Specifically, it allows for having the compressed video stream be NALu aligned where partial frames can be processed instead of waiting for the full image frame. This could slightly reduce latency as it virtually removes any buffer.

The `rtph264pay` element is used to prepare the video data for transmission over the network through “*payload encoding*”. Every Real-time Transport Protocol (RTP) packet contains a single Network Abstraction Layer (NAL) unit which means at most one entire frame is included. While RTP is a relatively bare network protocol, GStreamer also supports the use of Real-Time Control Protocol (RTCP) in conjunction, as well as many other options such as Real-Time Streaming Protocol (RTSP) servers.

To end this second pipeline, `udpsink` is added to send the RTP packets to the network. Unlike Transmission Control Protocol (TCP), User Datagram Protocol (UDP) operates via connectionless communication where no retransmission of lost packets is possible. This does however make UDP ideal for real-time applications as in those cases dropping a package is preferred over waiting for retransmission. To facilitate the P2P nature required, the `udpsink` is supplied with a “*host*” IP address to which it sends the data stream.

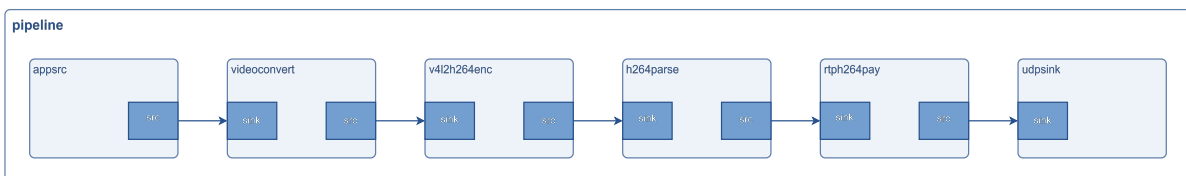
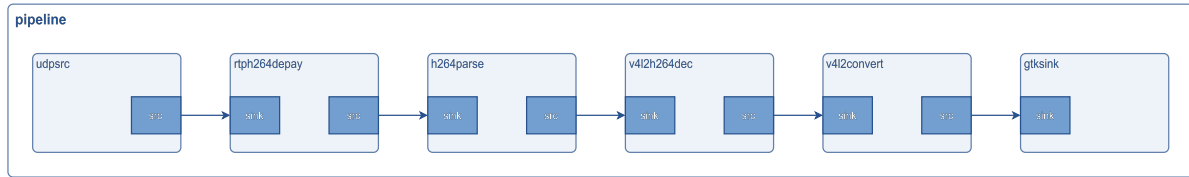


Figure 5.4: GStreamer pipeline representation of the RTP “server” side

### Receiver Pipeline

On the receiving end of the complete camera system the final pipeline is created. It facilitates the use of GUI toolkits to create a video player like the one described in section 6.2. Figure 5.5 contains a visual representation of what the pipeline could look like, although specifically the final sink element has a number of viable alternatives.



**Figure 5.5:** GStreamer pipeline representation of the RTP "client" side

Because the previous pipeline of Figure 5.4 ended in a `udpsink`, this final pipeline has to start with its counterpart; `udpsrc`. This element listens at all times to the network via a specified port and reads whichever packet it receives from there. To ensure the connection of the two UDP elements is created properly, this source element should be initialized before the sink starts its transmission.

Another mirror element of the transmitter pipeline is linked next to finalize the RTP stream. The `rtph264depay` element is a "depayloader" which returns the RTP packets back into a AVC/H.264 formatted video stream.

Following the depayloader the AVC/H.264 is again reformatted by a `h264parse` filter to help facilitate the link between two strict elements. Depending on the type of codec used in the next element this filter might be redundant, however it should not have a measurable negative impact the performance of the pipeline regardless.

To decode the compressed video stream back into raw data, usable for further processing or converting into a displayable video format, the pipeline is then linked to the `v4l2h264dec` element. Once more the V4L plug-in is used to add hardware acceleration to the codec on Raspberry Pi devices.

The next element is another important addition from the same plug-in, `v4l2convert`, which allows the format conversion to also be done on the GPU. This is with the current implementation especially preferred, since the output of the decoder is already on there. The previously used `videoconvert` could still be used instead, however that uses software conversion and would thus require all the data to be copied from the GPU to the Central Processing Unit (CPU). That process would waste time and resources, particularly when considering that afterwards the data has to be copied back to the GPU for displaying. The software converter does offer more flexibility with respect to the possible formats, so the final choice also depends on the implementation of the GUI.

In the pipeline from Figure 5.5, such an implementation is done using GTK+. Thus, to end the final GStreamer pipeline of the system, a `gtxsink` is linked. This element allows GStreamer to output its video stream to a specific window created by GTK in the form of a widget. Here the video is rendered and any additional GUI widgets can be added to improve the User Experience (UX) of the application. The main alternative to GKT+ is Qt, which is a more powerful though complex variant of GTK+. When using it instead of GTK+, `gtxsink` in Figure 5.5 should be replaced by a Qt equivalent. This equivalent element can be an image sink like `xvimagesink`, the previously used `appsink` or `qmlglsink` provided by the `qmlgl` plug-in.

None of the previous pipelines contains a `queue` element. This element simultaneously acts as a buffer and as an automatic threading manager. Since the implementation has an increased emphasis on real-time streaming, no `queue` elements were added initially. It could however be beneficial for performance to add one in the final pipeline between the `udpsrc` and `rtph264depay` elements to ensure the completeness of received frames and possible reduce stutter during the playback.

### 5.1.2. Network Realization

Any pipeline similar to those described prior require the existence of network to use for routing. However, as already mentioned before, this can not be guaranteed for a design aimed at leisure vehicle application. Instead a P2P connection should be created between the two wireless Single-Board Com-



puters (SBCs). This can be realized through the use of a *wireless ad hoc network* which can be created by either of the two SBCs. Such a network can then operate isolated to the internet and does not need any access points to manage it. A shortcoming of an implementation as such, however, stems from the fact that the system will reboot very often. This is because the SBCs are powered through the vehicle and thus lose power every time the vehicle is turned off. After every reboot, the processors have to re-establish their connection which can introduce some delays.

Therefore, using *Wi-Fi Direct* is preferred to create the required P2P connection. This standard is quite similar to the ad hoc network, however it adds a few improvements that make this standard more suitable. Most importantly in the context of this project, Wi-Fi Direct provides easier automatic connection options that make it somewhat similar to Bluetooth. By using the *Dynamic Host Configuration Protocol* it can fix the assignment of Internet Protocol (IP) addresses. For implementation on a SBC, the free `wpa_supplicant` software is used.

## 5.2. Wired Connections

Besides the previously described wireless connection of modules, some parts of the system all but force the use of cables. Mainly the cameras fall into that category, needing to be connected to a nearby SBC where the captured data can be pre-processed before entering the above pipelines. There exist several options of cameras using USB or High-Definition Multimedia Interface (HDMI), among others, interfaces to facilitate such data transfer. The most common one for embedded applications like this project, however, is through MIPI CSI-2. Camera Serial Interface (CSI) is capable of transferring a wide variety of data at very high resolutions in real-time while having low power consumption. The main drawback of the interface, developed by Mobile Industry Processor Interface (MIPI) Alliance, is its lower range compared to the previous two. This is not an issue for this project however, since the design in chapter 3 already used separate processors at each individual camera. Otherwise, the upcoming subsection 5.2.2 describes how this issue can be resolved.

In addition to the cameras, the display also likely requires wired data transfer between it and a nearby receiver SBC. And in similar fashion to the cameras, the by MIPI offered solution is preferred over USB and HDMI. In this case their Display Serial Interface (DSI)-2 offers a low cost option for high-speed high-bandwidth data transfer between processor and display.

As per chapter 2, the system mandates the use of non-battery-powered cameras which guarantees the presence of power cables running to each camera supporting processor. This necessity however also provides the opportunity to transmit both data and power through a single cable. Such a solution also aligns with the design in chapter 3 to combine the individual camera streams at a separate processor. Ethernet and USB are predominantly used for this kind of multifaceted cabling because of the ease of implementation. One such implementation of the USB case is described below.

### 5.2.1. USB Video Class

The selected method for wired transmission in the prototype is USB Video Class (UVC). UVC is a USB specification created by the USB Implementers Forum, designed to standardize video streaming functionality over USB. This specification is compatible with most operating systems and applications, making it the de facto choice for creating an embedded device capable of sending video over USB.

A design decision specified in chapter 3 outlines that each camera has its own SBC where pre-processing takes place, meaning each camera functions as a UVC device. To enable pre-processing on the SBC, software modifications are necessary.

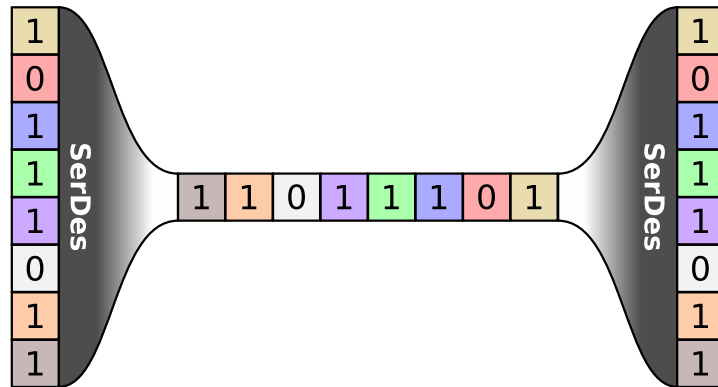
First, a virtual video device is created using `v4l2loopback`. The pre-processing script utilizes the camera's intrinsic parameters file to pre-process the CSI camera output. This pre-processed video is then directed to the created virtual device. The virtual device ultimately serves as the UVC output, allowing the pre-processed video stream to be transmitted over USB and recognized by other systems as a standard UVC camera.

### 5.2.2. Serializer / Deserializer

The basic principle behind Serializer / Deserializer (SerDes) is the conversion of serial and parallel data streams into one another like shown in Figure 5.6. More specific to the use case of this project, it adds the ability to combine multiple channels into a single cable for longer range transmission. Similar



to the above UVC, with SerDes it is possible to deliver power and transfer data over the same cable. Moreover, it has the ability to outperform the USB and Ethernet alternatives by offering the potential for higher data rates, more flexibility and bidirectional control measures.



**Figure 5.6:** Visualization of the SerDes principle. By Xinf - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=36068380>

Both CSI and DSI, introduced at the start of this section, are generally implemented using C-PHY or D-PHY, or sometimes both, as its physical layer where both exclusively support short range applications and have long been industry standards. More recent, however, MIPI Alliance started offering a longer range solution based on SerDes technology in the form of their A-PHY specification. For use in this project, especially a fully implemented combination of the A-PHY specification, CSI-2 and DSI-2 like MIPI Automotive SerDes Solutions should be considered for a final product. Use of any MIPI service or specification requires a licence through membership. Such a membership amounts to an annual fee of either \$4000 or \$8000, depending on the annual turnover of the aspiring member company. Similar annual fees are required for obtaining vendor IDs for USB and Ethernet; \$5000 and \$7000 respectively.

Besides MIPI, some other manufacturers also offer SerDes solutions compatible with CSI-2 and DSI-2; notably *Texas Instruments* and *Analog Devices* in the form of FPD-Link III and GMSL3 respectively. Both are optimized for use in automotive applications and offer similar performance. To use either of these solutions, separate custom PCBs should be designed for both sides of the SerDes process. These PCBs would then be connected by a single coaxial cable.

For a final product, the use of FPD-Link III, GMSL3 or ideally MIPI A-PHY should all be preferable over the prototype solution of UVC. For this project however, none of those three were available due to exorbitant cost or due to requiring designing custom PCBs.

# 6

## Human Machine Interaction

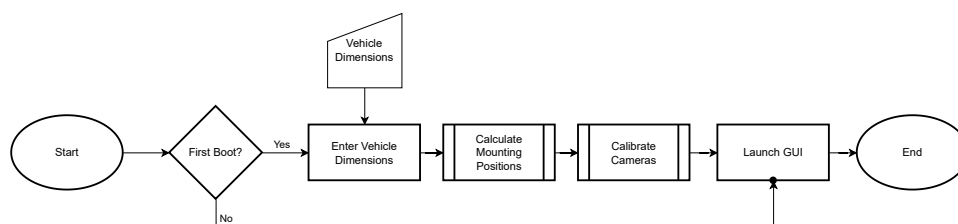
When considering a consumer product, the interaction of the end user is crucial to the overall experience. Even if the system performs exceptionally, suboptimal user experience can significantly diminish product perception or, worse, result in user abandonment. Therefore, it is crucial to carefully consider how an end user interacts with the product and ensure that points of friction are minimized. This chapter discusses all areas where human-machine interaction occurs, identifies potential points of friction, and proposes solutions to address them.

### 6.1. User Experience

UX refers to the interaction between the user and the application. This encompasses everything from initial setup and configuration to everyday real-time use. It is vital for the UX to feel seamless, as any disruptions in the flow can make the system feel cumbersome to interact with. Additionally, the system is designed to minimize user interaction to avoid distracting the driver while operating the vehicle.

#### 6.1.1. Initial Setup

One of the system requirements outlined by the client is an easy and intuitive initial setup process. An intricate setup experience can be a major barrier to entry for many non-technical users; therefore, it is essential that the process be as user-friendly as possible. This encompasses everything from mounting to calibration. The first step involves powering the display, which launches the first boot wizard. The user is prompted to enter their vehicle model or vehicle dimensions and is provided with suitable mounting locations. The camera mount is designed with an optimal pitch angle, which was found to be when the vertical part of the trailer just barely does not showing, simplifying the mounting process by requiring users only to drill and mount at the calculated location.



**Figure 6.1:** Overview of the first boot user flow.

Next, the user goes through the calibration sequence. A camera is shown, and the user is prompted to indicate its pointing direction (front, back, left, or right), after which the calibration process, as discussed in subsection 4.1.3, will proceed. Once the camera has been calibrated, the next camera is prompted, and this sequence is repeated until all cameras have been calibrated. Upon completion of the calibration process, the first boot flag is set to false, and the users are taken to the GUI.

### 6.1.2. Regular Operation

The next challenge to address is everyday operation. It is essential that daily use remains as seamless as possible for users. Any unnecessary complications will accumulate over time and degrade the overall user experience. Measures to achieve this include having the GUI launch automatically upon vehicle startup, thereby eliminating the need for user input. Furthermore, the following section 6.2 on the GUI will highlight several deliberate design decisions aimed at minimizing unnecessary user interactions, which could otherwise become potential points of friction.

## 6.2. Graphical User Interface

As mentioned in section 6.1, the way users interact with a product significantly impacts their perception of it. Therefore, considerable importance was placed on making the User Interface (UI) intuitive, clearly displaying necessary information, and avoiding unnecessary complexity. The following section will present the GUI concept and explain the design choices made.

### 6.2.1. Boot Wizard

The goal of the boot wizard is to guide the user through the initial setup process by obtaining necessary information efficiently. Designed to provide a linear and user-friendly experience, the wizard breaks down the setup into clear, manageable steps.

As shown in Figure A.3, the first page of the wizard informs users of the minimum setup time, ensuring they are aware of the time requirement in advance. This allows users to allocate sufficient time for the setup process if necessary, preventing potential frustration that could arise from unexpected time commitments.

Next, as shown in Figure A.4, users are prompted to enter their vehicle type or select a specific model from a potential database of recreational vehicles.

If the user decides to manually enter the vehicle dimensions, a helpful graphic, as seen in Figure A.5, is displayed to label each measurement, preventing any confusion. The data is further validated to ensure it fits within a logical range, mitigating potential human error.

After the cameras are mounted according to the calculated positions, users are prompted to identify a specific camera direction, as shown in Figure A.6. That camera will then complete the calibration process as outlined in subsection 4.1.3. This camera position will subsequently be greyed out, and users will be prompted to identify the next camera direction. This sequence is repeated until all cameras have been identified. After which the first boot flag is set to false, and the main GUI is finally launched.

### 6.2.2. Camsense View

The "camsense view" is the primary view of the product and, as such, is the first option in the menu. This view provides a Bird's-Eye View (BEV) with an overlay to indicate object proximity, as shown in Figure A.7. The object proximity is calculated based on data provided by the sensor subsystem.

### 6.2.3. Rear Cam View

Another possible view is the "Rearcam view", this view as the name implies shows the undistorted camera feed from the rearview camera, along with a Picture-in-Picture (PiP) of the BEV. This view, which is shown in Figure A.8, is especially useful in parking situations.

## Prototype Implementation

In order to verify whether the design is feasible, a prototype for the system must be created. Hence, a provisional selection of components was made that favored lead times and ease of implementation. Using the prototype, it was possible to demonstrate the workings of a couple key system components. A complete implementation prototype was however deemed unattainable in the limited time available for the project, but as components have been verified on test data throughout the project, this can be attained with more time for integration.

### 7.1. Hardware Selection

The choice of hardware to use for the prototype was made by considering multiple factors: performance, cost, ease of implementation and availability. Each component was assessed separately first and in combination with the whole system afterwards. It was agreed upon, prior to the selection process, that E-trailer would purchase the components that the group decided upon. Moreover, the company mentioned that the cost was not as important (within reason) for a prototype.

**Cameras** To start, the horizontal FOV of the lens needed to be as wide as possible to minimize the area of blind spots. The resolution of the lens was considered too, albeit with lower weight since the video will ultimately be played on a smaller resolution display regardless. Additionally, the presence of some night vision capabilities were considered as a bonus. The image sensor was required to have a high enough data rate to support at least 720p at 30 FPS and be interfaced via MIPI CSI.

From the above requirements, the *Omnivision OV5647* and an accompanying 175° fish-eye lens was ultimately chosen.

**Processors** For the choice of SBC, cost and ease of implementation played a larger role than for the cameras. The prototype featured four smaller processors and one larger processor. The smaller type was required to interface with the chosen camera, thus requiring a CSI port. Furthermore, it needed to be able to do some basic image processing. Additionally, the ability to connect to Wi-Fi was preferred as it allowed for testing an alternative version of the design which omitted the larger SBC. As described in subsection 5.2.1, the smaller SBCs are connected to the larger processor via UVC thus both types also require compliant USB port(s).

This led to the decision in favor of four *Raspberry Pi Zero 2 W* processors as the smaller type. The larger type became the *Raspberry Pi 5* processor since it was already owned by a member of the group.

**Display** During the prototyping laptop screens were used to show the outputs, instead of using a separate display with its own processor. This choice was primarily made to save time as other tests and designs were considered more important. Moreover, adapting to a display was believed to be quite similar to the implemented work since the GStreamer pipeline feeding it was already working as intended. The implementation would then only require modifying the boot of the accompanying SBC to use the screen for outputs, which is comparable to what was done on the camera side.

## 7.2. Proof of concept setup

In making a proof of concept, confirming the workings of certain system components were prioritized. Notably, it was crucial to verify the effectiveness of the most optimal extrinsic calibration solution implemented, which was the one supported by computer vision in the finding of source pixel coordinates. Since the stitching algorithm only depends on the projection outputting properly formatted partial scene bird's eye views, proving the effectiveness of the image processing up until the projection stage more or less verifies the feasibility of creating the surround view system. Because of this, the proof of concept setup was built to show the processing of images all the way until the projection, made to work for a single fisheye camera stream. For this, the calibration site as described in the computer vision part of subsection 4.1.3 was built as shown in Figure A.9, Figure A.10 and Figure A.11, with a fisheye camera mounted at a height of 2.5 meters to show compatibility with the large leisure vehicles. The real-world centroid coordinates of the calibration patterns, which were spread unevenly to show the freedom that the customer is allowed in placing them, were measured with respect to the camera, as would be done by the sensing subsystem in a final implementation.

The goal of this setup was to output a continuous projected stream from the mounted fisheye. For this, intrinsic calibration was performed as described in subsection 4.1.2 for the acquired cameras, allowing the pi to output a continuous undistorted stream. A single undistorted frame of the calibration site was captured by the user after it was observed that the four centroids were detected. The corresponding source pixel coordinates were extracted, enabling the extrinsic calibration to be completed. The found projection matrix could then be applied on the continuous undistorted stream, generating a partial bird's-eye view output. This output was visualized on one laptop and sent wirelessly to another laptop using the GStreamer command line tool `gst-launch-1.0`, in order to simulate the transmission stage.

## 7.3. Validation Results

Performing the described process gives the results in stages as shown in Figure 7.1 below:



**Figure 7.1:** Validation results

It can be seen that the undistortion, centroid detection and projection stages have the same effect as when the implemented systems were employed on example test data as shown in chapter 4, indicating good performance of the ideas for actual hardware implementation. While only a single high-mounted camera was used due to limitations in time for finding and experimenting in suitable testing environments, for instance on a real caravan, the good results on projection show that a complete, caravan-implemented bird's-eye view image would likely give good results.

## Conclusion and Recommendations

In this thesis, the design procedure for a 360° surround view camera system for leisure vehicles was discussed. To this end, a system was introduced that uses inputs of multiple wide-angle fisheye cameras to create a bird's-eye view of the vehicle, which can subsequently be sent to a display via wireless transmission. In order to create the bird's-eye view frames, image processing had to be performed on the incoming fisheye images. This process could be divided into two parts: the pre-processing of the individual fisheye frames and the final image generation through the combining of the processed frames. In the pre-processing stage, undistortion, projection and flipping was applied. For the undistortion stage, necessary parameters were found through intrinsic calibration, after which OpenCV was utilized to compute and apply the undistortion maps that follow from these parameters. The projection matrix necessary for the projection was found through a process of extrinsic calibration. Several methods for extrinsic calibration were discussed and compared based on their advantages and disadvantages for customer usability. Two of these methods were implemented, namely the manual calibration method and the computer vision method, the latter of which is an optimization of the manual method. After the flipping of the projected frames, a bird's-eye view was generated through a stitching process based on weighted averages, and the final frame was obtained by correcting for differences in scene illumination through brightness compensation.

With regard to image processing, a major part of future research would go into actually carrying out the described process for finetuning the WESNet model for leisure vehicle applications. This self-calibration method was the most optimal out of the described techniques in terms of complexity and duration, and thus would perform best in terms of customer usability.

Following the image processing part, potential GStreamer pipelines were discussed to transmit the data throughout the system. The first of these pipelines was designed to receive the output signals of the individual camera processors through for instance a UVC cable and export the data to a separate OpenCV application. The second pipeline was designed to import the processed data back into GStreamer and transmit it over a wireless network to the "*client*". Additional consideration went into enabling hardware acceleration on the Raspberry Pi to improve the latency performance of the pipeline. The third and final pipeline was designed to receive the video data from the network and render it using the GTK+ GUI toolkit.

As already briefly mentioned before, the more flexible Qt GUI toolkit would likely allow the interface to be improved compared to the current implementation [23]. The addition of control over the RTP stream, for instance through adding RTCP, is something to consider in a final application to make it more robust and allow bidirectional signaling. Another option worth looking into further are some of the open source GStreamer projects from *RidgeRun* which offer a variety of features. Also, as a way to provide a more sophisticated solution, an AI to do for example object detection could be added to the final part of the system [24], [25].

In the final stage of the thesis, a small-scale proof of concept was crafted to show the workings of some individual system components. Notably, it was shown that the implemented computer vision method succeeds in generating reliable projected frames when set up in a position simulating the height of the caravan. This implies that the stitching process, which depends on the quality of the projected frames, could also be implemented naturally if a suitable medium for placing the cameras, for instance a real caravan, would be available.

# Glossary

<b>AI</b>	Artificial Intelligence	<b>LUT</b>	Lookup Table
<b>AVC/H.264</b>	Advanced Video Coding	<b>MIPI</b>	Mobile Industry Processor Interface
<b>BEV</b>	Bird's-Eye View	<b>NAL</b>	Network Abstraction Layer
<b>CNN</b>	Convolutional Neural Network	<b>OS</b>	Operating System
<b>COD</b>	Center Of Distortion	<b>P2P</b>	Peer-to-Peer
<b>CPU</b>	Central Processing Unit	<b>PCB</b>	Printed Circuit Board
<b>CSI</b>	Camera Serial Interface	<b>PiP</b>	Picture-in-Picture
<b>DSI</b>	Display Serial Interface	<b>RTCP</b>	Real-Time Control Protocol
<b>E2E</b>	End-to-End	<b>RTP</b>	Real-time Transport Protocol
<b>FOV</b>	Field Of View	<b>RTSP</b>	Real-Time Streaming Protocol
<b>FPD</b>	Flat Panel Display	<b>SBC</b>	Single-Board Computer
<b>FPS</b>	Frames Per Second	<b>SerDes</b>	Serializer / Deserializer
<b>G2G</b>	Glass-to-Glass	<b>TCP</b>	Transmission Control Protocol
<b>GMSL</b>	Gigabit Multimedia Serial Link	<b>UDP</b>	User Datagram Protocol
<b>GPU</b>	Graphical Processing Unit	<b>UI</b>	User Interface
<b>GUI</b>	Graphical User Interface	<b>USB</b>	Universal Serial Bus
<b>HDMI</b>	High-Definition Multimedia Interface	<b>UVC</b>	USB Video Class
<b>HLS</b>	HTTP Live Streaming	<b>UX</b>	User Experience
<b>HTTP</b>	Hyper Text Transfer Protocol	<b>V4L</b>	Video4Linux
<b>IoT</b>	Internet Of Things	<b>WebRTC</b>	Web Real-Time Communication
<b>IP</b>	Internet Protocol	<b>WESNet</b>	Weakly-supervised Extrinsic Self-calibration Network
<b>JND</b>	Just Noticable Difference		
<b>LDC</b>	Lens Distortion Correction		



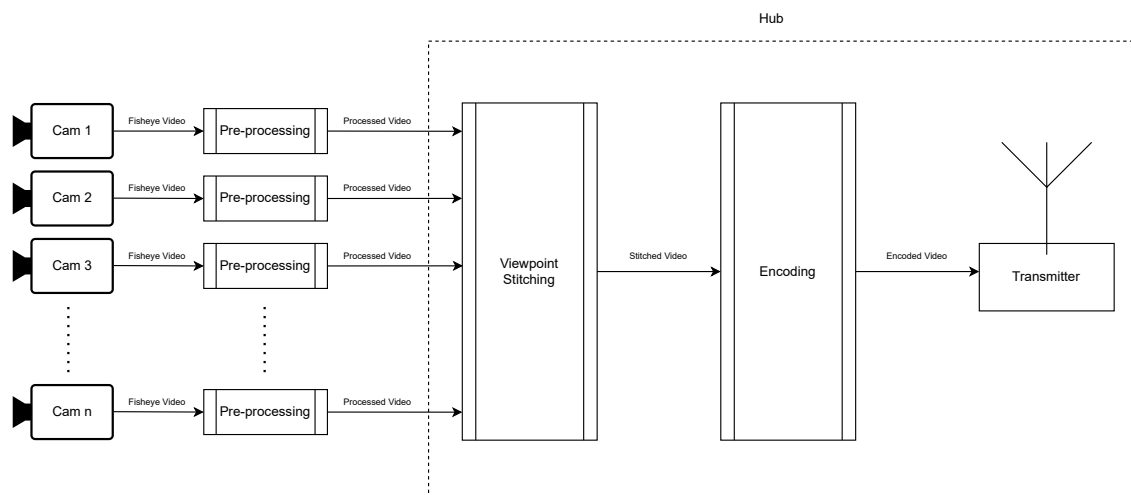
# References

- [1] Y. Dwarkasing and B. Yildiz, *Proximity sensor-based parking aid solution for mobile homes and trailers*, Faculty of Electrical Engineering, Mathematics and Computer Science, 2024.
- [2] *Caravan blijft de camper ruim voor*, <https://www.rdw.nl/particulier/nieuws/2023/caravan-blijft-de-camper-ruim-voor>.
- [3] *Voorkom schade aan je caravan*, <https://www.anwb.nl/verzekeringen/caravanverzekering/top-5-caravanschades>.
- [4] Caravan Guard. "Top 10 most accident prone parts of a caravan revealed!" (), [Online]. Available: <https://www.caravanguard.co.uk/news/accident-prone-part-caravan-7252/> (visited on 04/29/2024).
- [5] B. Zhang, V. Appia, I. Pekkucuksen, *et al.*, "A surround view camera solution for embedded systems," 2014.
- [6] M. Yu and G. Ma, "360° surround view system with parking guidance," *SAE Int. J. Commer. Veh.*, vol. 7, no. 1, 2014. DOI: 10.4271/2014-01-0157.
- [7] Y.-C. Liu, K.-Y. Lin, and Y.-S. Chen, "Bird's-eye view vision system for vehicle surrounding monitoring," in *International Workshop on Robot Vision*, 2008, pp. 207–218.
- [8] Y. Gao, C. Lin, Y. Zhao, X. Wang, S. Wei, and Q. Huang, "3-d surround view for advanced driver assistance systems," *IEEE Transactions on Intelligent Transportation Systems*, DOI: 10.1109/TITS.2017.2750087.
- [9] A. Kaehler and G. Bradski, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. Sebastopol, CA: O'Reilly Media, Inc., 2016.
- [10] C. Hughes, M. Glavin, E. Jones, and P. Denny, "Review of geometric distortion compensation in fish-eye cameras," in *Proceedings of the Irish Signals and Systems Conference (ISSC)*, Connaught Automotive Research Group, Department of Electronic Engineering, National University of Ireland, Galway, Galway, Ireland, Jun. 2008.
- [11] Z. Liang, *Surround-view system introduction*, <https://github.com/neozaoliang/surround-view-system-introduction>, 2020.
- [12] Y. Chen, L. Zhang, Y. Shen, B. N. Zhao, and Y. Zhou, "Extrinsic self-calibration of the surround-view system: A weakly supervised approach," *IEEE Transactions on Multimedia*, 2021, Submitted for publication.
- [13] J.-M. Georg, J. Feiler, S. Hoffmann, and F. Diermeyer, "Sensor and actuator latency during teleoperation of automated vehicles," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, (Las Vegas, NV, USA, Oct. 2020), IEEE, pp. 760–766. DOI: 10.1109/IV47402.2020.9304802.
- [14] D. Barrett and P. Desai, "Low-latency design considerations for video-enabled aerial drones," Texas Instruments, White Paper, Sep. 2016. [Online]. Available: <https://www.ti.com/lit/pdf/spry301> (visited on 04/25/2024).
- [15] C. Bachhuber and E. Steinbach, "A system for high precision glass-to-glass delay measurements in video communication," in *2016 IEEE International Conference on Image Processing (ICIP)*, (Phoenix, AZ, USA, Sep. 2016), IEEE. DOI: 10.1109/icip.2016.7532735.
- [16] C. Bachhuber, E. Steinbach, M. Freundl, and M. Reisslein, "On the minimization of glass-to-glass and glass-to-algorithm delay in video communication," *IEEE Transactions on Multimedia*, vol. 20, no. 1, pp. 238–252, Jan. 2018. DOI: 10.1109/tmm.2017.2726189.
- [17] G. P. Fettweis, "The tactile internet: Applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, Mar. 2014. DOI: 10.1109/mvt.2013.2295069.
- [18] A. Rommel, "Low-latency image data compression," Fraunhofer Institute for Telecommunications, Heinrich Hertz Institut, HHI, Press Release, Feb. 2018. [Online]. Available: <https://www.fraunhofer.de/en/press/research-news/2018/February/low-latency-image-data-compression.html> (visited on 04/25/2024).

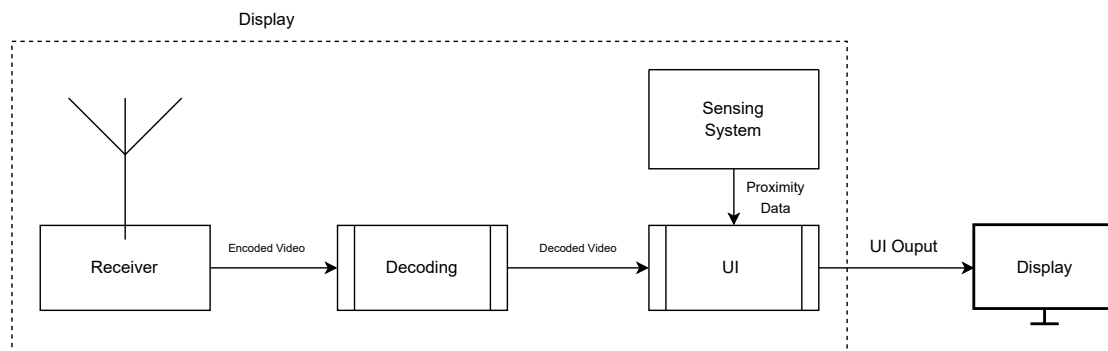
- [19] R. Safin, E. Garipova, R. Lavrenov, H. Li, M. Svinin, and E. Magid, "Hardware and software video encoding comparison," in *2020 59th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, (Chiang Mai, Thailand, Sep. 23, 2020), IEEE, pp. 924–929. DOI: 10.23919/sice48898.2020.9240439.
- [20] U. Jennehag, S. Forsstrom, and F. Fiordigigli, "Low delay video streaming on the internet of things using raspberry pi," *Electronics*, vol. 5, no. 4, p. 60, Sep. 2016, *Raspberry Pi Technology*. DOI: 10.3390/electronics5030060.
- [21] Y. Liu, Z. Yu, D. Zong, and L. Zhu, "Attention to task-aligned object detection for end–edge–cloud video surveillance," *IEEE Internet of Things Journal*, vol. 11, no. 8, pp. 13 781–13 792, Apr. 15, 2024. DOI: 10.1109/jiot.2023.3340151.
- [22] S. Liang, H. Wu, L. Zhen, *et al.*, "Edge yolo: Real-time intelligent object detection system based on edge-cloud cooperation in autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 12, pp. 25 345–25 360, Dec. 2022. DOI: 10.1109/tits.2022.3158253.
- [23] H. Ali. "Differences between gtk+ and qt applications," Baeldung. (Mar. 18, 2024), [Online]. Available: <https://www.baeldung.com/linux/gui-toolkits-gtk-qt> (visited on 06/10/2024).
- [24] A. Dahal, J. Hossen, C. Sumanth, *et al.*, "Deeptrailerassist: Deep learning based trailer detection, tracking and articulation angle estimation on automotive rear-view camera," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, (Seoul, South Korea, Oct. 2019), IEEE. DOI: 10.1109/iccvw.2019.00287.
- [25] V. Appia, H. Hariyani, S. Sivasankaran, *et al.*, "Surround view camerasystem for adas on ti's tdax socs," Texas Instruments, White Paper, Oct. 2015. [Online]. Available: <https://www.ti.com/lit/wp/spry270a/spry270a.pdf> (visited on 06/10/2024).

# A

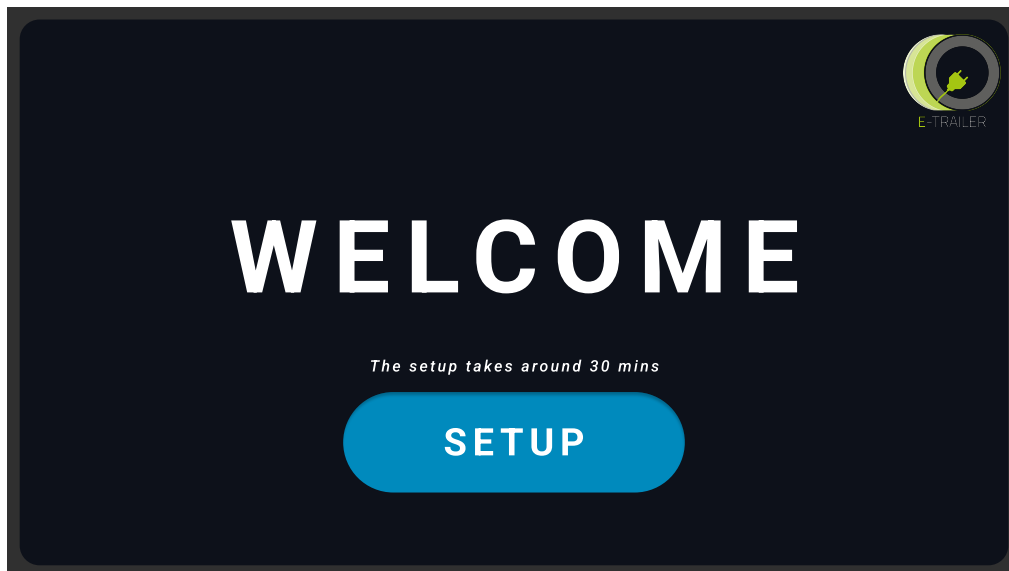
## Additional Figures



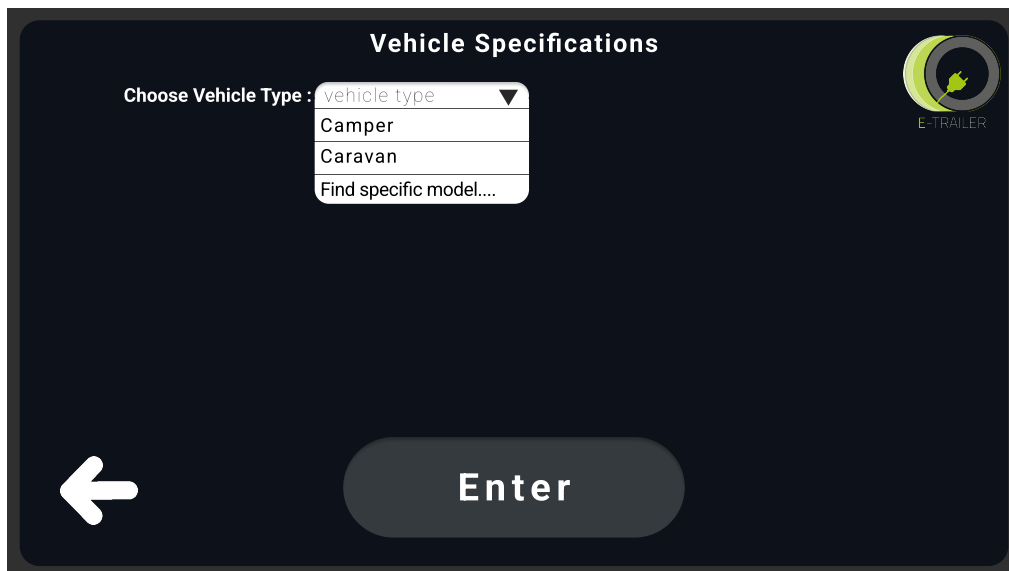
**Figure A.1:** Transmission end system



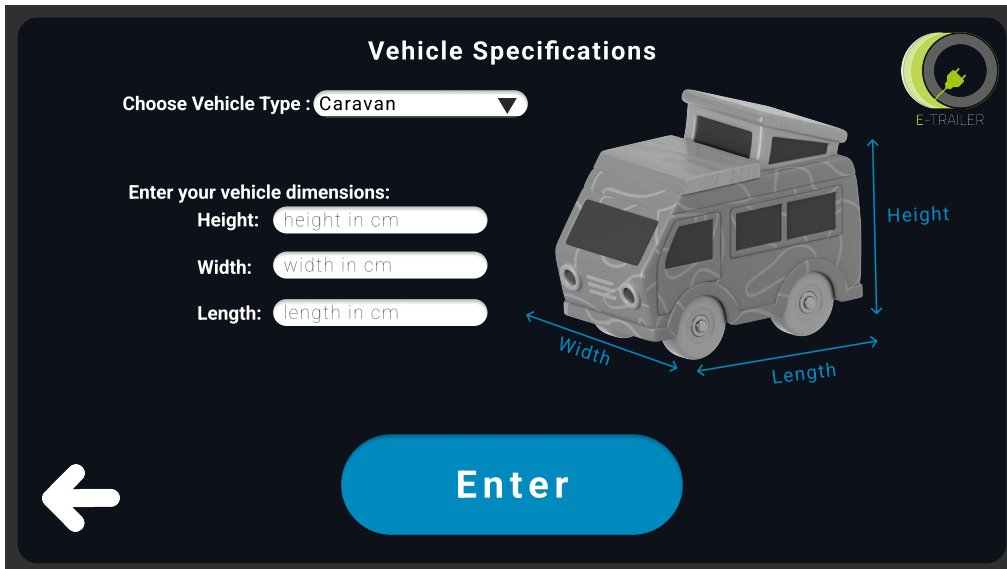
**Figure A.2:** Receiver end system



**Figure A.3:** At first boot user is greeted with a welcome screen



**Figure A.4:** User is prompted to enter their vehicle type



**Vehicle Specifications**

Choose Vehicle Type :

Enter your vehicle dimensions:

Height:

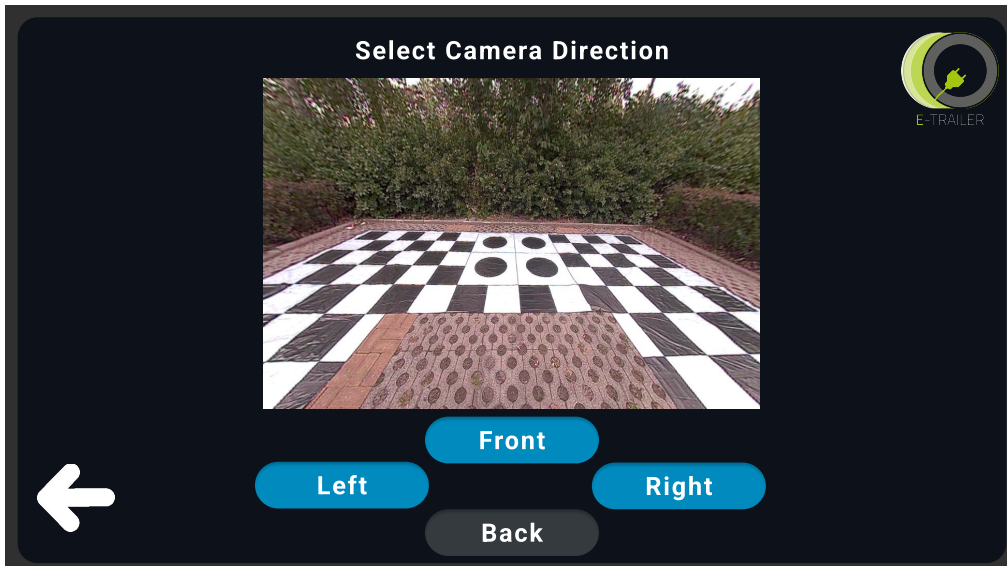
Width:

Length:

**Enter**

The screen features a 3D model of a caravan with dimension lines for Height, Width, and Length. A back arrow is on the left, and the E-TRAILER logo is in the top right.

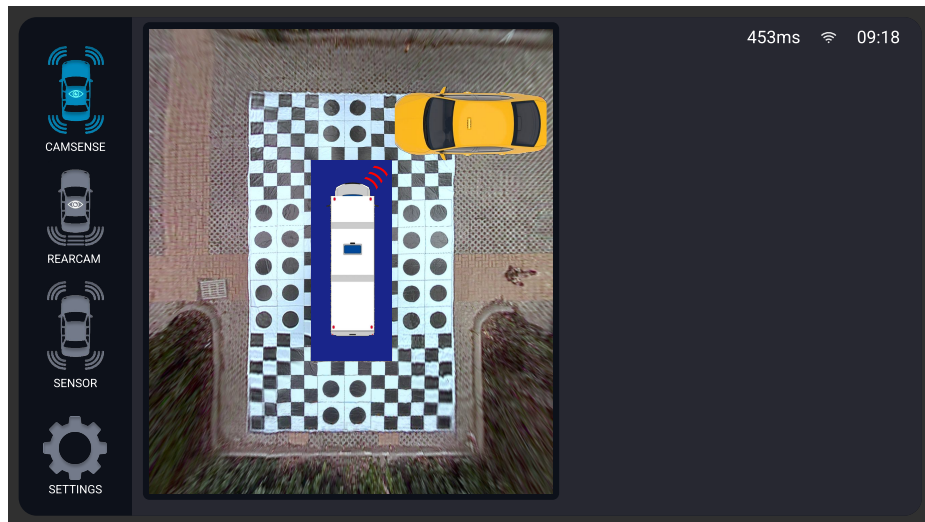
**Figure A.5:** User is prompted to enter their vehicle specifications



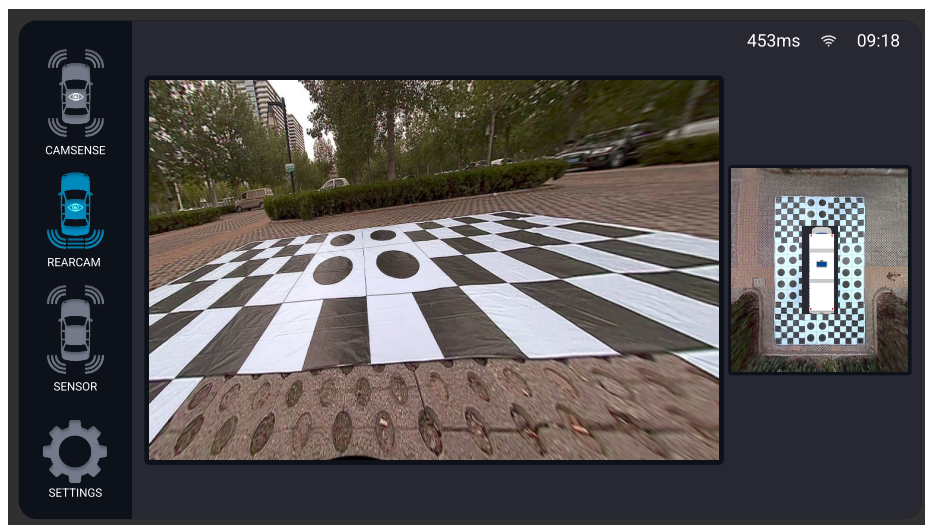
**Select Camera Direction**

The screen shows a photograph of a checkered floor. Below the photo are four buttons: Front, Left, Right, and Back. A back arrow is on the left, and the E-TRAILER logo is in the top right.

**Figure A.6:** User is prompted to select camera direction



**Figure A.7:** Camsense view with object detected in close proximity to the front right corner of the vehicle



**Figure A.8:** Rear Cam View



**Figure A.9:** Setup view 1





Figure A.10: Setup view 2





Figure A.11: Setup view 3