Journal of Transportation Technologies, 2023, 13, 87-114
https://www.scirp.org/journal/jtts
ISSN Online: 2160-0481
ISSN Print: 2160-0473

# Stereoscopic Camera-Sensor Model for the Development of Highly Automated Driving Functions within a Virtual Test Environment

**René Degen[1,2*], Martin de Fries[1], Alexander Nüßgen[1,2], Marcus Irmer[1,2], Mats Leijon[2], Margot Ruschitzka[1]**

[1]CAD CAM Center Cologne, Institute of Automotive Engineering Cologne (IFK), Faculty of Automotive Systems and Production, Cologne University of Applied Sciences, Cologne, Germany
[2]Division of Electricity, Department of Electrical Engineering, Uppsala University, Uppsala, Sweden
Email: *rene.degen@th-koeln.de

**How to cite this paper:** Degen, R., de Fries, M., Nüßgen, A., Irmer, M., Leijon, M. and Ruschitzka, M. (2023) Stereoscopic Camera-Sensor Model for the Development of Highly Automated Driving Functions within a Virtual Test Environment. *Journal of Transportation Technologies*, **13**, 87-114.
https://doi.org/10.4236/jtts.2023.131005

**Received:** November 15, 2022
**Accepted:** January 28, 2023
**Published:** January 31, 2023



## Abstract

The need for efficient and reproducible development processes for sensor and perception systems is growing with their increased use in modern vehicles. Such processes can be achieved by using virtual test environments and virtual sensor models. In the context of this, the present paper documents the development of a sensor model for depth estimation of virtual three-dimensional scenarios. For this purpose, the geometric and algorithmic principles of stereoscopic camera systems are recreated in a virtual form. The model is implemented as a subroutine in the Epic Games Unreal Engine, which is one of the most common Game Engines. Its architecture consists of several independent procedures that enable a local depth estimation, but also a reconstruction of a whole three-dimensional scenery. In addition, a separate programme for calibrating the model is presented. In addition to the basic principles, the architecture and the implementation, this work also documents the evaluation of the model created. It is shown that the model meets specifically defined requirements for real-time capability and the accuracy of the evaluation. Thus, it is suitable for the virtual testing of common algorithms and highly automated driving functions.

## Keywords

Sensor Model, Virtual Test Environment, Stereoscopic Camera, Unreal Engine, OpenCV, ADAS/AD

## 1. Introduction

The digital age has also ushered in the age of automated driving. Although the

current state of the art has not yet reached fully autonomous driving, technical developments are increasingly moving in that direction. As demand increases, investment in the associated technologies is also expanding. For example, the British investment bank JP Bullhound has recorded a steadily growing number of investments in the area of sensor technology and algorithms for highly automated driving functions in recent years [1].

To develop the necessary sensor technology, the authors' research group sets up a virtual test field with high-resolution visualization of the virtual environment [2].

Due to the high visualization quality in this test environment, it is particularly suitable for developments in the field of optical sensor technology. Therefore, it is obvious to implement camera-based principles for distance measurement in this environment in addition to lidar and radar sensor technology. Stereo cameras are of particular interest here as they offer all the advantages and functional possibilities of single cameras. One major advantage of camera sensors is their excellent suitability for object detection. For example, traffic signs, lane markings and road user can be identified. Other non-visual sensor technologies either do not allow this at all or not at this level [3] [4].

In addition, stereo cameras enable the estimation of spatial positions and thus the use of 3D data-based evaluation algorithms. Another advantage is the increasing demand for camera systems in the entertainment industry. This not only leads to a steady increase in their performance but also makes camera sensor technology very cost-effective [3] [5].

From a developer's perspective, the virtual approach using sensor models enables time- and cost-efficient testing of software for measurement data evaluation and sensor data fusion. These models are also a necessary component of virtual environments if sensor-specific training data for AI algorithms should be created. Computer-aided development also makes it possible to accelerate the above-mentioned virtual traffic scenarios compared to real scenarios. Furthermore, such models contribute to the design of safe test procedures for the designated software systems by minimizing the need for real tests in road traffic.

From this analysis, it can be seen that stereo cameras are an essential sensor principle that should be represented in virtual form for safe, reproducible and accelerated development. The model described in this paper must meet a number of requirements. For example, it must be linkable to virtual vehicle models in order to enable realistic test operations. Another requirement is the estimation of the spatial depth within the virtual test environment. This environment is developed within the C++-based Epic Games Unreal Engine, which is why the stereo camera model has to be programmed in C++ as well. Furthermore, the OpenCV function library is to be used for the implementation, as it contains mature and generally accepted algorithms. In order to be suitable for use in safety-critical vehicle systems, real-time capability is also desirable. Therefore, a measurement frequency of 30 frames per second (FPS) will be referred to as

real-time in the following. The following sections introduce the reader to the state of the art and the basics of stereoscopic depth estimation. Based on the basics, a software architecture for the model of a stereoscopic camera system is developed. Afterwards, the implementation of the model within the Unreal Engine is described, followed by an evaluation of the model's performance in terms of computing speed and measurement accuracy. The work closes with a discussion and conclusion.

## 2. State of Research

The need thus identified has already led to research activities in this area. Our own investigation shows that there are only a small number of scientific publications on this topic. In a relevant publication, the authors deal with the generation of stereo image pairs in the virtual environment of an early version of the Unreal Engine [6]. These image pairs are to be used for the development and testing of algorithms for stereo correspondence analysis. Due to the changeability of the geometric and optical parameters of virtual systems, variable scenarios can easily be achieved in the virtual environment. Image pairs with several base widths are generated and stereo correspondence analysis methods are applied to these image pairs. Furthermore, their evaluation quality is assessed. In this work, the image pair is not generated by two virtual camera modules, as in the model from the present paper. Instead, a single camera module is used and spatially shifted for both images. Also, the software is obviously only used to generate single image pairs, whereas the model discussed in this paper evaluates video image sequences in real time [6].

Another publication presents a tool called "UnrealStereo", which is used to test stereo correspondence analysis methods [7]. This tool is again used in the virtual environment of the Unreal Engine and for the generation of stereo image pairs. The special feature of the tool is that it identifies those features of the virtual environment that have a problematic effect on the evaluation by stereo correspondence analysis methods and makes them adjustable via variables. Such features are, for example, regions with reflections, shiny surfaces, surfaces with transparency effects, texture-less surfaces and regions with disparity discontinuities. The open-source tool's ability to identify and control features should facilitate the validation of stereo correspondence analysis techniques. But it is not clear whether two separate virtual camera modules are used to generate the images. In contrast to the stereo camera model in this paper, the programming is done in the Python programming language instead of C++, which is more suitable for the Unreal Engine [7].

In addition to the scientific view, the industry also deals with the simulation of camera sensor technology for ADAS. The simulation environments of the company IPG represent commercial tools that are suitable, among other things, for the virtual generation of camera data. Although there are models for mono cameras, there is no information on possible model approaches for stereo cam-

eras. All information on IPG products described here are taken directly from the manufacturer's website [8].

An alternative is the open-source simulation environment "CARLA Simulator", which is intended for function development in the field of autonomous driving. Here, the Unreal Engine 4 is used for visualization, physics simulation and rendering. CARLA has various models for radar, lidar and camera sensors. The camera sensor technology is represented by a total of five models, whereby one of these models provides the spatial depth in grey scale. However, based on the explanations of this sensor model on the website, it can be assumed that it is a mono camera, which determines the distance for each of its pixels based on the ground truth data of the virtual environment. According to the results of our own research, a stereo camera model is not implemented. Since CARLA is based on the Unreal Engine, it can be assumed that CARLA is also based on the C++ programming language [9].

In summary, this analysis shows that there is currently no model that fulfils the defined requirements. This illustrates the need to answer the research question posed here.

## 3. Basis of Stereoscopic Image Analysis

To understand the behaviour of the stereo camera model, the knowledge of some basics is necessary. For example, a stereo camera system is composed of two separate cameras, each of which generates the image of a three-dimensional scene in a two-dimensional plane. This process can be described with the help of so-called projective geometry. In this process, a spatial point $M$ of the scenery, which lies at a distance $z_M$ from the optical centre $C$, is mapped onto the image point $\boldsymbol{m}$ of the image plane, which lies parallel to the focal plane of the camera at a distance of the focal length $f$. This process is illustrated by the camera model with virtual image plane in **Figure 1**, which emerges from the construction of the pinhole camera by shifting the image plane [10].

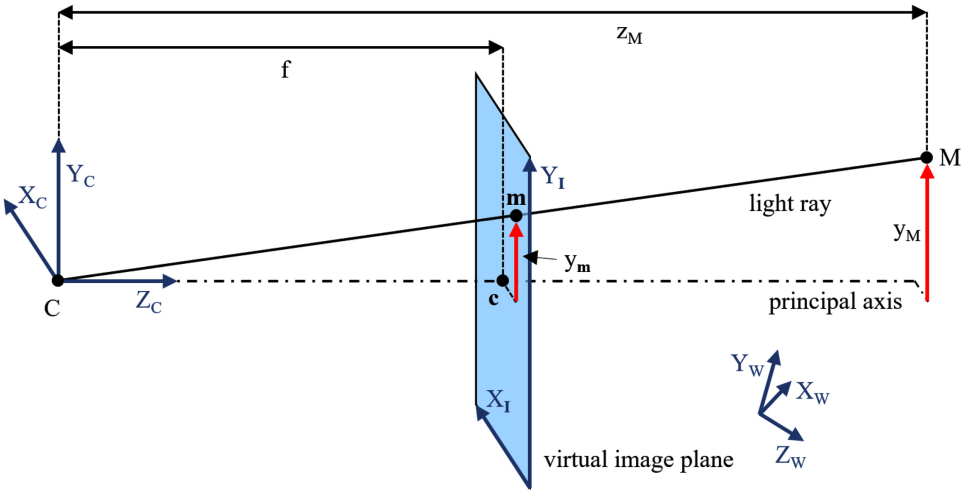


**Figure 1.** Camera model with virtual image plane.

The figure shows that an object with height $y_M$ in space leads to an image with height $y_m$ in the image plane. Furthermore, it must be taken into account that the focal lengths can differ in vertical and horizontal direction due to the nature of the sensor. Therefore, the focal lengths $f_x$ and $f_y$ are introduced. It must also be considered that the principal axis does not intersect the image plane exactly in the centre, which is respected by the correction parameters $c_x$ and $c_y$. All these requirements are reflected in the equations for the central projection. They map a 3D spatial point $M$ in camera coordinates ($C$) onto the 2D image point $\boldsymbol{m}$ in camera coordinates ($C$), where the intrinsic matrix $A$ contains all internal respectively intrinsic parameters of the camera.

The following representation emerges from these equations. It uses homogeneous coordinates for the 2D vector $p_{m,I}$, which can be divided by the scaling factor $z_M$ to obtain the coordinates of the image point [11].

$$
\begin{gathered}
{}^{(C)}\underline{p}_{\boldsymbol{m},I} \cdot {}^{(C)}z_M = \underline{A} \cdot {}^{(C)}\underline{r}_{M,C} \\
\text{with } \underline{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \; {}^{(C)}\underline{r}_{M,C} = \begin{bmatrix} {}^{(C)}x_M \\ {}^{(C)}y_M \\ {}^{(C)}z_M \end{bmatrix}, \; {}^{(C)}\underline{p}_{\boldsymbol{m},I} = \begin{bmatrix} {}^{(C)}x_{\boldsymbol{m}} \\ {}^{(C)}y_{\boldsymbol{m}} \\ 1 \end{bmatrix}
\end{gathered} \tag{1}
$$

In addition to the imaging process, the relative position of the camera and spatial points must also be included in the consideration. If a known vector $r_{M,W}$ represents the relative position of a spatial point $M$ to a world coordinate system ($W$), the spatial point can be transferred from world coordinates ($W$) to camera coordinates ($C$) by a translation, described by the vector $r_{C,W}$, and a rotation, described by the rotation matrix $S$. After that its position can be described from the origin $C$ of the camera coordinate system. In homogeneous coordinates, the following expression results.

$$
{}^{(C)}\underline{r}_{M,C} = \underline{D} \cdot {}^{(W)}\underline{r}_{M,W} \quad \text{with } \underline{D} = \begin{bmatrix} {}^{(C,W)}\underline{S} & {}^{(C)}\underline{r}_{W,C} \\ \underline{0}_3^{\mathrm{T}} & 1 \end{bmatrix} \tag{2}
$$

The matrix $D$ is called the extrinsic matrix because it contains all the external or extrinsic parameters of the camera. These form six degrees of freedom consisting of three displacements and three angles. The combination of the Expressions (3) and (4) transforms a three-dimensional spatial point in the world coordinate system into a two-dimensional point on the virtual image plane described in the camera coordinate system.

In addition to the imaging process of the individual cameras, the geometric relationships between the two cameras of a stereo camera system must also be known. This is because only through the interaction of two cameras is it possible to estimate the depth of a recorded scenery through stereo triangulation. The so-called epipolar geometry describes these relationships. In the application case of a virtual model, it is possible to arrange the principal axes of both cameras exactly in parallel as shown in **Figure 2**.

In such an axis-parallel epipolar geometry, the image planes are only shifted

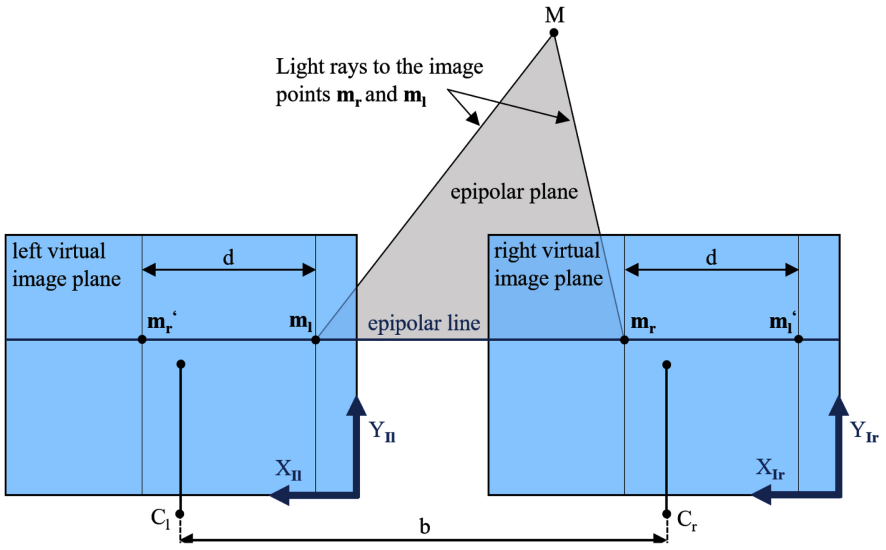


**Figure 2.** Epipolar geometry of a parallel stereoscopic camera system.

horizontally to each other by the base width $b$ and the image points $\boldsymbol{m_l}$ and $\boldsymbol{m_r}$ are in the same plane. Thus the image points $\boldsymbol{m_l'}$ and $\boldsymbol{m_r'}$, which are projections of the same spatial point $M$ and which thus represent a point correspondence, lie on a common epipolar line [3] [10].

This property significantly simplifies the algorithmic search for point correspondence, since instead of examining the entire image planes, only the corresponding epipolar line in both image planes needs to be examined. It also allows the definition of the term disparity as a measure of the horizontal displacement of both point correspondences relative to the two image coordinate systems (Ir) and (Il) as follows.

$$d = {}^{(\mathrm{Il})}x_{m_r'} - {}^{(\mathrm{Il})}x_{m_l} = {}^{(\mathrm{Ir})}x_{m_r} - {}^{(\mathrm{Ir})}x_{m_l'} \tag{3}$$

Here the virtual points $\boldsymbol{m_l'}$ and $\boldsymbol{m_r'}$ each represent the image points $\boldsymbol{m_l}$ and $\boldsymbol{m_r}$ shifted by the base width into the other image plane. The disparity $d$ is a measure for the distance and thus a basis for the depth estimation via triangulation.

In order to calculate the disparity, the point correspondences in both images must be determined. This task is performed by so-called stereo correspondence analysis methods. Due to the multitude and complexity of known methods for stereo correspondence analysis, the focus of this work is limited to the so-called block-matching method. This is a local stereo correspondence analysis method. In its described form, it goes back to [12].

In addition to the pure correspondence search, the OpenCV implementation also contains a pre- and post-processing of the image. Before the search, preparatory steps adjust the lighting conditions in both images and improve the textures.

For the actual correspondence search by means of block-matching, some preconditions must be fulfilled. Both images must have sufficient texture and be

available in grey scale. Rectification and correction of lens distortion must also have taken place for both images.

This is because the method uses the property of rectified and distortion-free images in which corresponding points lie on the common epipolar line and thus on the image line of the same index. This is because it examines the images exclusively row by row and thus uses a very compact search-space.

In addition to the restriction to line-by-line examination, the search-space is also reduced by further observations. Assuming an accurate axis-parallel structure of the stereo system, the point $\boldsymbol{m_r}$ of the search image must lie to the left of the virtual point $\boldsymbol{m_l'}$ or on top of it. The virtual point $\boldsymbol{m_l'}$ marks the relative position of the image point $\boldsymbol{m_l}$ within the search image. If $\boldsymbol{m_r}$ lies on $\boldsymbol{m_l'}$, the minimum $d_{\text{min}}$ disparity is zero. By choosing a range $d_{\text{num}}$ of disparity values the maximum disparity $d_{\text{max}}$ and thus the search-space is defined. **Figure 3** shows all the properties mentioned above.

After the search-space has been defined, a comparison of a reference image pixel $\boldsymbol{m_l}$ with all the pixels within the search-space can be made. For the evaluation of the pixel similarity in both images based on their intensity values, OpenCV uses the sum of absolute differences (short: SAD) [11].

However, the SAD does not only include two individual pixels in the comparison, but calculates the similarity of a square area around the examined pixel in the reference image (left) to an area in the search image (right). Therefore, it is also called "block-matching".

The SAD is calculated for all possible disparity values d and thus for all pixels of the search-space. To identify the most similar point, the algorithm subsequently determines the minimum of the SAD curve. In the described manner, the procedure successively examines all pixels of the stereo image pair and determines their disparity. The resulting disparity maps are enhanced by post-processing steps before output.

After the stereo correspondence analysis has been completed and the disparity has been determined, the next step is to estimate the spatial depth, based on the principle of triangulation. This is oriented on the geometry of a triangle as shown in **Figure 4** [11].

Triangulation makes it possible to infer the distance $z_M$ in the Z-direction of

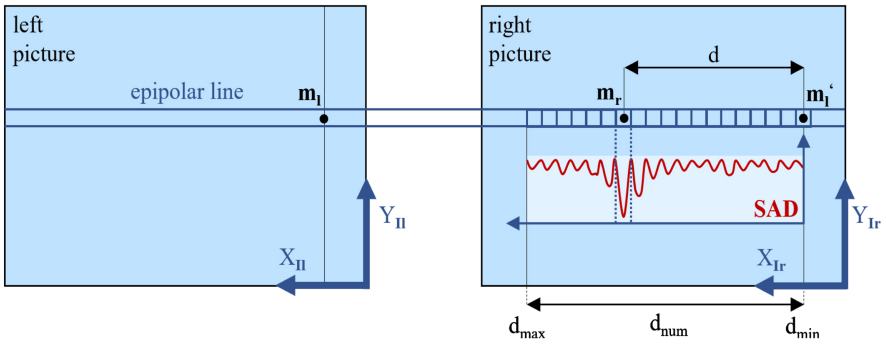


**Figure 3.** Search-space and SAD-course of block-matching methods as described in [11].

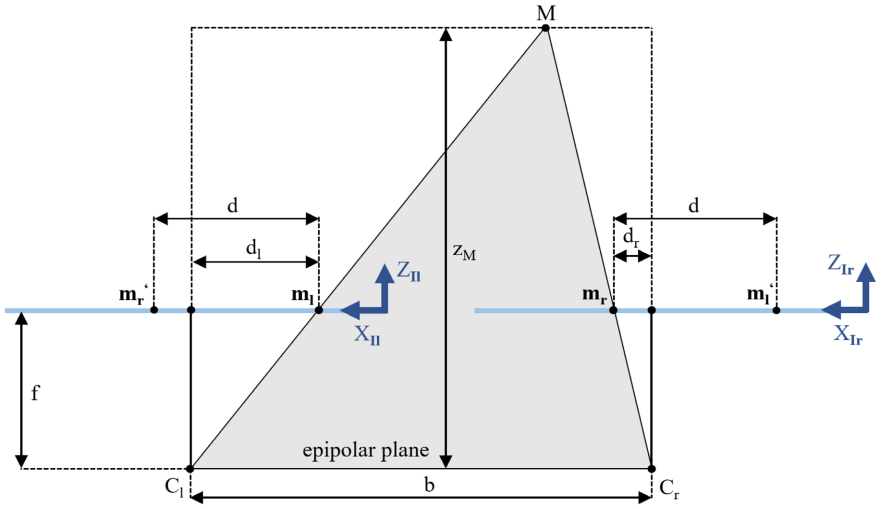


**Figure 4.** Top view of a parallel stereoscopic camera system.

the camera coordinate systems of the spatial point *M*, which forms a triangle with the image points, on the basis of two image points $\boldsymbol{m_l}$ and $\boldsymbol{m_r}$ with the known distance *b*. The following formula clarifies this relationship.

$$z_M(d) = \frac{f \cdot b}{d} = \frac{f \cdot b}{d_l + d_r} \tag{4}$$

As the expression makes clear, disparity and depth are inversely proportional to each other and thus have a non-linear relationship. Thus, the disparity is very large at low depths, which is why small absolute changes in disparity only have a minor effect on the depth estimate. Therefore, the resolution of a depth measurement with stereoscopic camera sensors decreases with increasing distance. In order to achieve a reconstruction of the three-dimensional position of the spatial point, it is necessary to reconstruct the X and Y coordinates in addition to determining the depth in the Z direction. For this purpose, the reprojection matrix $Q$ is used. It determines the position of the spatial point in homogeneous coordinates from the two-dimensional position of the image point in the image plane and the disparity. This matrix is constructed from the relationships presented in the Equations (1) and (4), while assuming an equal focal length f for the horizontal and vertical direction. It allows the reconstruction of all three spatial coordinates [11].

$$w \cdot {}^{(C)}\underline{r}_{M,C} = \begin{bmatrix} w \cdot {}^{(C)}x_M \\ w \cdot {}^{(C)}y_M \\ w \cdot {}^{(C)}z_M \\ w \end{bmatrix} = \underline{\underline{Q}} \cdot \begin{bmatrix} {}^{(I)}x_m \\ {}^{(I)}y_m \\ d \\ 1 \end{bmatrix} \text{ with } \underline{\underline{Q}} = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{1}{b} & 0 \end{bmatrix}, w = \frac{d}{b} \tag{5}$$

The calibration of the cameras plays an important role for the practical application. At this point, however, only a brief description is given. One step is the calibration of the two individual cameras, which involves determining the intrinsic and extrinsic matrices. There are various procedures for this. One possi-

bility is the analysis of a chessboard pattern that is recorded several times by both cameras [13].

In addition to the calibration data of the individual cameras, the calibration data of the stereo system, which include the geometric relationship of both individual cameras, is also of interest [11].

Within real and thus non-ideal stereo camera systems, a number of realisation effects occur, which shall only be briefly mentioned here. They have a negative effect on the application of some methods for stereo correspondence analysis. On the one hand, these are distortions of the images of real camera lenses. These distortions can be corrected algorithmically, which [14] deals with in more detail for the application in the library OpenCV. Secondly, the epipolar geometry of real stereo camera systems is never fully axis-parallel. A rectification can transform the views of a non-axis-parallel system into those of an axis-parallel system. Relevant functions for rectification in OpenCV are based on [15].

## 4. Software Architecture

Based on the fundamentals now known, a model software architecture can be synthesised. To keep the mode of representation formal, the following section uses activity diagrams based on the conventions of the Unified Modelling Language (UML) [16].

The four main procedures of the model software are shown in **Figure 5**.

The measurement procedure and the "mode for free-shots" are part of a common programme sequence. Before free-shots in form of stereo image pairs can be created or the measurement procedure is started, the programme environment and the models parameters are initialised. The calibration procedure is included in a separate programme sequence.

For a calibration of the stereo system, the provision of a large number of stereo image pairs is essential. The programme part "mode for free-shots" serves to generate such image pairs free of any further image processing. During its execution, the programme continuously records image pairs, the storage of which

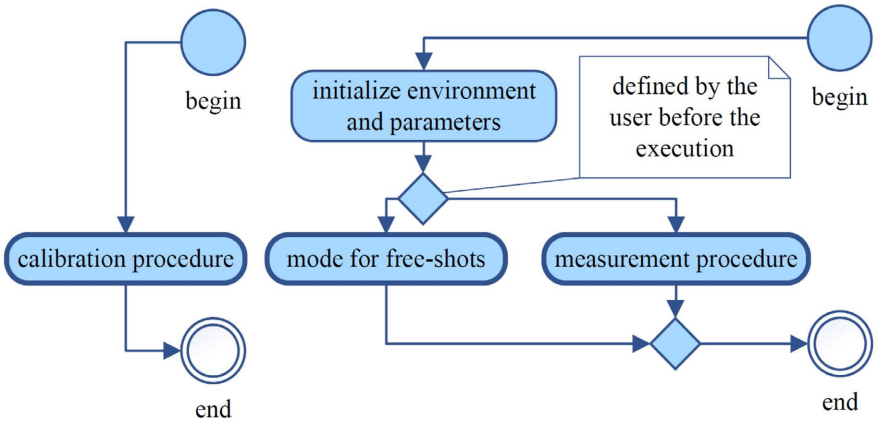


**Figure 5.** Activity diagram with rough programme structure at the highest hierarchical level.

is triggered by the user by pressing a key. The user also terminates the measurement by pressing a button. With regard to the recording of the images, it should be noted that the recording processes by the left and right camera modules in Figure 6 are assumed to be synchronous in time.

The measurement procedure performs a stereo analysis for acquired stereo image pairs as well as a depth estimation. At the beginning, the procedure loads previously determined calibration and correction data from the hard disk memory and makes them available to the subsequent, iterative process. Each iteration loop starts with the acquisition of a stereo image pair by the camera modules, assuming the recording to be synchronous. This is followed by lens distortion correction and rectification of the image pairs. In the ideal conditions of virtual systems, these steps are not necessary and are skipped. Programme parts with such a status are marked in orange. Figure 6 shows the measurement procedure.

Subsequently, methods for stereo correspondence analysis search for corresponding image points or pixels in both images. For the purpose of performance comparison, several different methods are implemented. Before starting the programme, the user can select one of the methods included in OpenCV, which

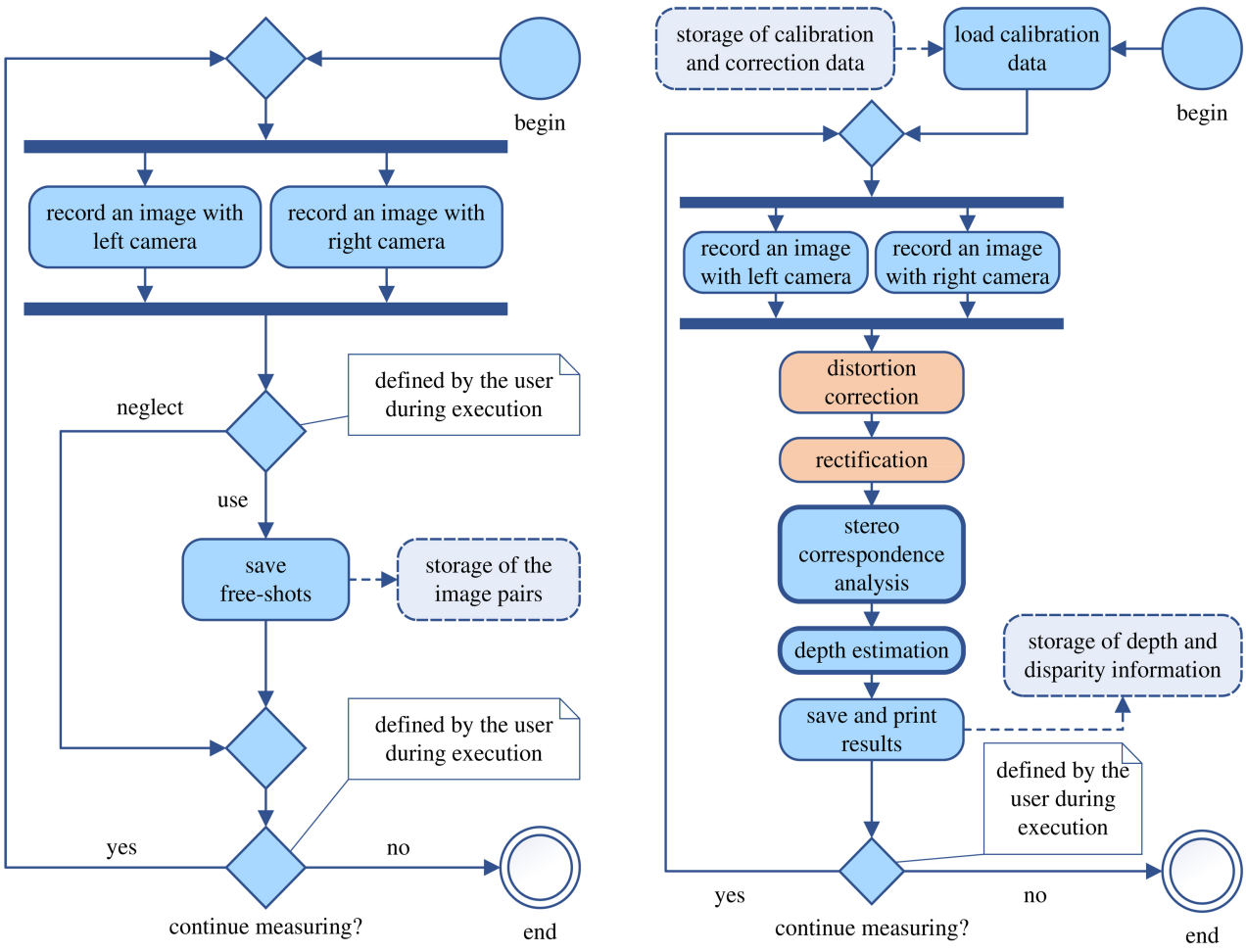


**Figure 6.** Activity diagram of the mode for free-shots (left) and the measurement procedure (right).

are shown in Figure 7. The methods marked "(CUDA)" are methods that can be executed on NVIDIA GPUs [11].

The process of stereo correspondence analysis determines the disparity for each common pixel of both stereo images and visualizes all disparities in a disparity map. Although objects can already be recognised in this map, the depth information is still missing.

The local depth estimation is done for a user-defined area in the centre of the disparity map. A simple method is the direct depth estimation via stereotriangulation. Depth can also be determined using the reprojection matrix from the calibration data, which map pixels from the image plane to a corresponding point in 3D space. Both approaches calculate the average depth of the defined area and output it. The measurement procedure concludes with the output and storage of the disparity images as a video.

In addition to the local depth estimation, the programme also enables the estimation of the three-dimensional position of a spatial point for each pixel from the image plane for which a disparity value is available. The reprojection matrix is also used for this purpose. Since such a global 3D reconstruction is computationally expensive, it is carried out at the user's keystroke instead of at each iteration. In the course of this, the programme saves the calculated positions on the hard disk. Figure 8 shows all three forms of depth estimation.

Based on the findings of the basic chapters, a multi-part calibration procedure can be designed. At the beginning of this procedure, the parameters are initialised. This is followed by the separate calibration of the two camera modules of the stereo camera model in the "mono-calibration" part of the programme, which leads to the intrinsic and extrinsic parameters of the camera modules. This process picks up previously generated stereo image pairs of a calibration pattern, detects the corners of the pattern in the images and determines the named parameters based on this. The user can generate the necessary stereo image pairs in a previous step with the help of the programme part "free-shots".

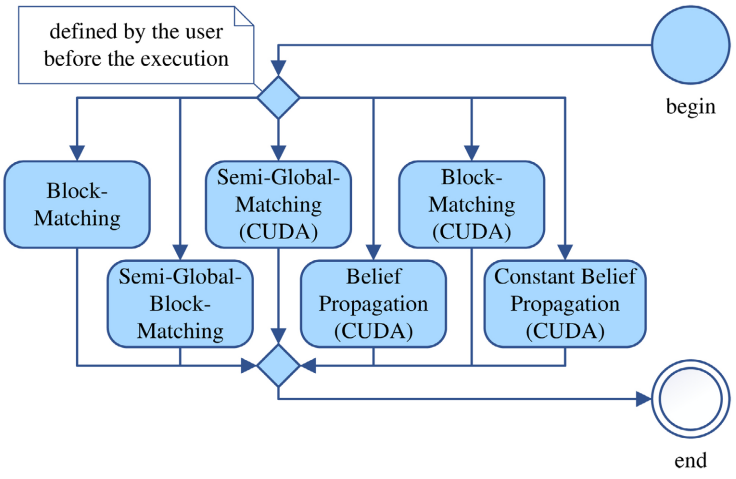


Figure 7. Activity diagram for the selection of the stereo correspondence analysis method.

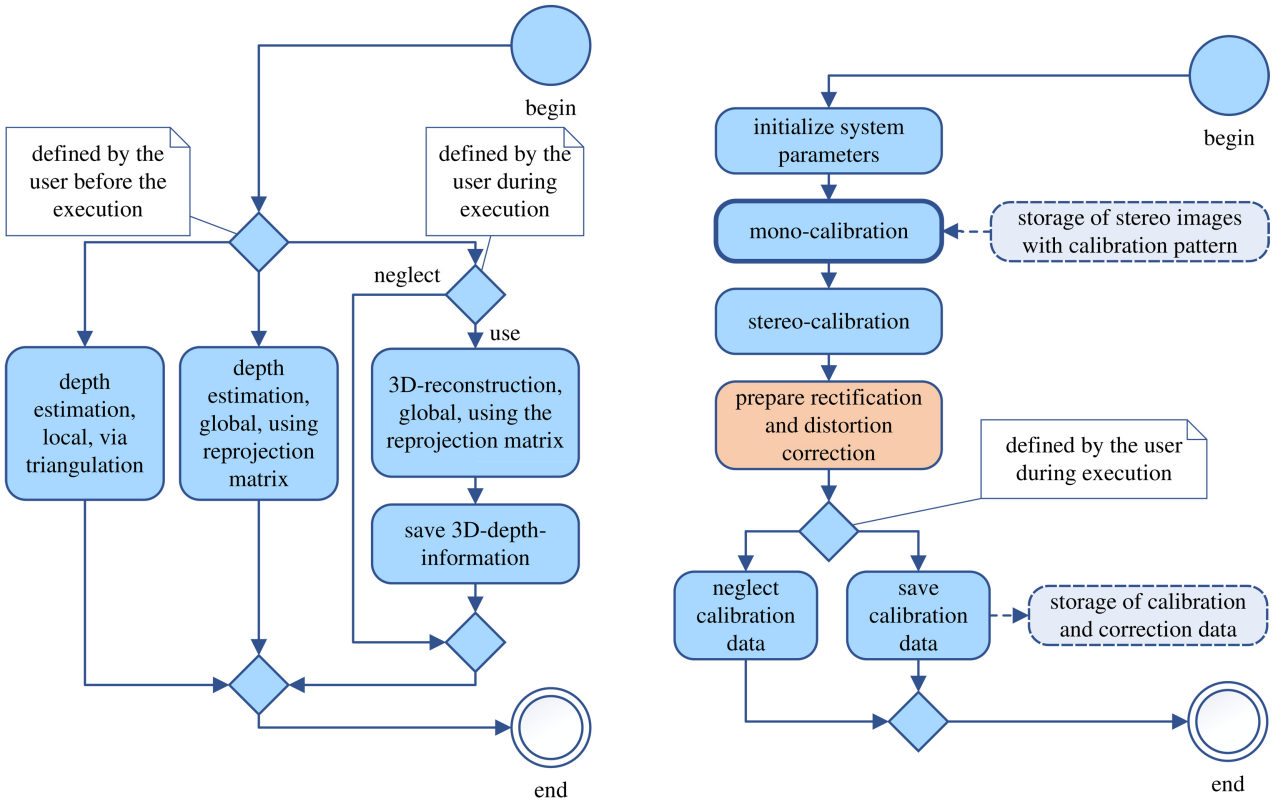


**Figure 8.** Activity diagram for depth estimation (left) and for the calibration procedure (right).

The mono-calibration is followed by the calibration of the stereo system (in short: stereo-calibration). It leads to the extrinsic parameters of the stereo system. Another programme part of the calibration procedure deals with the determination of correction data for distortion correction and rectification. This part of the programme is not necessary for the present virtual system. Nevertheless, it has been fully implemented and is thus available for subsequent developments in the field of real systems.

After this step, the calibration of the system is completed. Finally, the user decides based on the displayed reprojection error, which is a measure for the quality of the calibration, whether he wants to save the calibration data on the hard disk or discard it. Within **Figure 8**, the entire calibration process is shown sequentially.

## 5. Implementation

Based on the previously defined architecture, the software of the stereo camera model can be integrated into the virtual environment of the Unreal Engine. The possibility of creating one's own C++ projects within the Unreal Engine provides great scope for designing technical applications. The so-called "Actors" of the Unreal Engine serve as a framework for the integration of such applications. In relation to the graphical user interface, the Unreal Editor, an actor is an independent object that can be placed in the virtual environment. Its appearance and

position can be changed. It is also possible to link it to other elements, which, for example, allows the actor to be linked to a vehicle model. Each actor has an underlying programme consisting of a header file and a source code file that the user can edit. The header file defines the class of the actor, which inherits the properties of the base class "AActor". It also contains the declaration of methods and attributes. The definition of the methods and attributes takes place within the source code file. The functions of the stereo camera model are also implemented here with the addition of further classes and methods from external function libraries [17].

When creating an Actor object in the Unreal Editor, the Unreal Engine first executes the object's default constructor, which performs initialisations and definitions and sets up the object for operation. After the actor has been created, the simulation can begin. Initially, the predefined method BeginPlay() is executed. It is suitable for a one-time execution of preparatory functionalities. Iterative functionalities can be implemented in the predefined method Tick(), because this method is called repeatedly during the calculation of each new frame. In this context, it is important to note that the Unreal Engine cannot start calculating a new frame until the execution of the Tick() methods of all actors within the virtual environment have been completed. Thus, the repetition frequency of the frames, also called frame rate, depends directly on the computational efficiency of the operations within the Tick() method. As soon as the user aborts the simulation, the Unreal Engine executes the predefined method EndPlay(). It is suitable for one-time executions after leaving the Tick() functionalities, such as closing output windows after the simulation [17].

The implementation within the Unreal Engine makes use of the methods and attributes of two classes. On the one hand, the class "AStereoCam_Actor" of the Actor plays a role. It is created when the C++ class of the type Actor is created. It contains the predefined elements mentioned above. On the other hand, it contains the methods and attributes relevant for the stereo camera model. In addition, there are some enumerations to set up drop-down menus in the Unreal Editor. The corresponding UML class diagram can be found in **Figure 9**.

For reasons of clarity, only the essential functions are listed. Furthermore, the attributes are not shown. Within the diagram the prefix "u::" refers to methods that are specified by the Unreal Engine. Prefixed characters symbolise the availability of the class elements, where "#" stands for public and "+" for protected elements.

In addition to this class, the Actor's header file contains another class of a task-based storage function for video recordings. A later section looks at this in more detail. **Figure 10** shows the corresponding class diagram.

Now that the class structure of the model is known, the description of the implementation of the programme sequence begins. Before starting the measurement procedure or the procedure for the free-shots of stereo image pairs, an initialisation of the actor is necessary. As mentioned before, this is done by calling

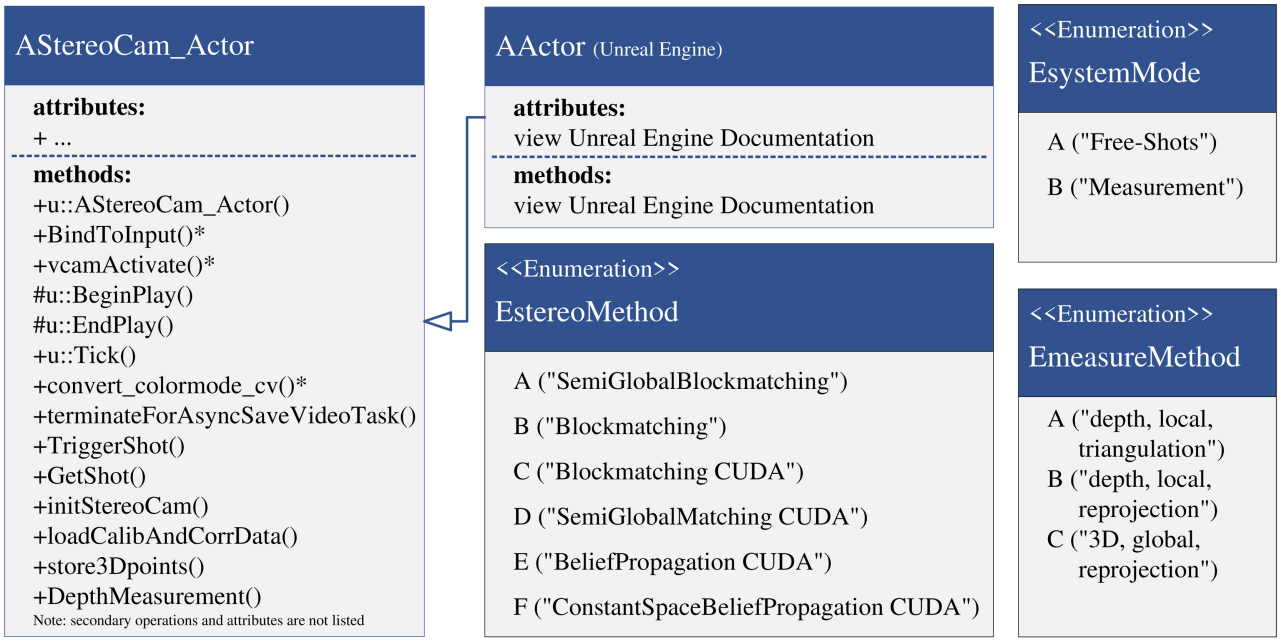


**Figure 9.** Class diagram for the "AStereoCam_Actor".

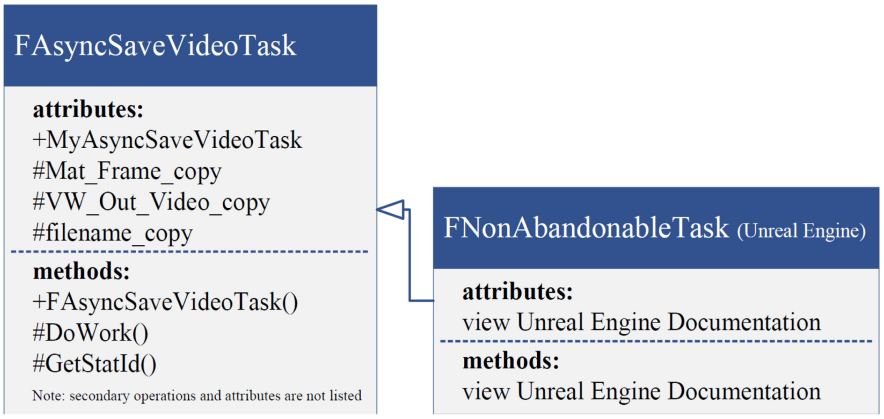


**Figure 10.** Class diagram for the "AsyncTask".

the constructor, which in this case is realised via the method AStereoCam_Actor(). This creates two "USceneCaptureComponent2D", which as ideal camera modules enable the recording of the scene, and positions them parallel to each other. In addition, two "UStaticMeshComponents" are included, which are given the same position as the USceneCaptureComponent2Ds. If geometries are assigned to the "UStaticMeshComponents", the position of the invisible USceneCaptureComponent2Ds is visualised during the simulation [17].

In addition, the initStereoCam() method defines the attributes of the model parameters declared in the header. As soon as the user starts the simulation, the BeginPlay() method executes the BindToInput() method. It sets up the use of keystrokes for programme control during the simulation, so that when a defined key is pressed, the TriggerShot() method is executed as an interrupt. This method sets a flag that can be used to control iterative parts of the programme.

Then BeginPlay() calls the method vcamActivate() if the user selects the activation of the actor in the Unreal Editor. If the user does not select activation, the actor is not part of the simulation. This allows the user to exclude individual model actors from the simulation if multiple stereo camera models are placed in the same virtual environment. Within vcamActivate(), the programme performs further setup steps for the recording. For example, the convert_colormode_cv() method sets the colour space specified by the user. If the procedure is to be run to generate free captures and the programme is to save these captures, vcamActivate() specifies the associated storage location. If the user selects the measurement procedure, vcamActivate() prepares the storage location for the measurement results. In this case, the loadCalibAndCorrData() method also reads the XML file with previously determined calibration and correction data from the hard disk.

After initialisation, the iterative execution of the free-shots procedure or the measurement procedure begins. The recordings generated by the USceneCaptureComponent2D are read out and are then available for the selected operating mode. The process does not generate the recordings of both virtual camera modules exactly at the same time, but sequentially and immediately one after the other. In the present case, this does not lead to errors in the evaluation, as the execution speed of the programme is sufficiently high. If the user has chosen the "mode for free-shots" before starting the simulation, the programme displays the currently captured image at each iteration using the method cv::imshow(). Methods with the prefix "cv::" come from the OpenCV library. The iterative output with cv::imshow() appears to the user like a live video. If he has selected that the programme should save the free-shots and if the desired motif has been found, the user initiates a shot by pressing a key. This is followed by the execution of the method GetShot(), which saves both stereo shots with individual file names in JPEG format on the hard disk. Such a stereo image pair is shown in **Figure 11**. When comparing the bottom of the virtual environment at the lower edges of the image, the disparity becomes clear.

Alternatively, the user can select the measurement mode before starting the

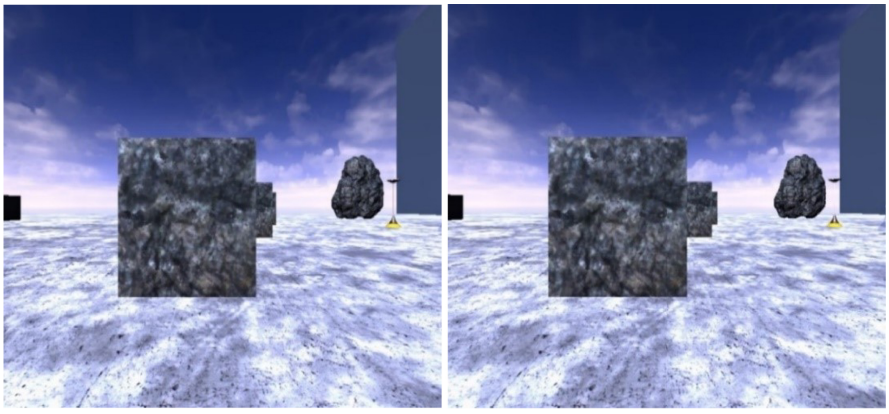

**Figure 11.** Left and right image of a stereo image pair recorded as a "free-shot".

simulation. In this case, the programme executes the DepthMeasurement() method. In the first iteration of Tick(), the selected algorithm for stereo correspondence analysis is set up. The chosen parameters are based on the author's own preliminary experiments. Although there are a large number of possible parameters, the programme is limited to the use of the parameters described in the following **Table 1**. For descriptions of the other parameters, please refer to the literature [11].

Once the selected algorithm for stereo correspondence analysis is set up after the first iteration, the programme performs the following steps at each Tick() iteration. First, the algorithm calculates a disparity map from the two stereo images. According to its own measurement, this process is very computationally intensive compared to the rest of the programme, which is why GPU-accelerated methods have also been implemented. Although these methods require the stereo images to be uploaded to the GPU before the calculation and then the disparity map to be downloaded, these two operations are not significant in terms of execution frequency. The procedures mentioned in **Figure 7** are implemented in the model. After their creation, the programme cuts the disparity map because its resolution is different from the resolution of the stereo images used. This is partly because the correspondence search can only take place in the area where the two images overlap. In addition, a part of the disparity map remains empty due to the properties of the stereo correspondence analysis method. This is because if the point is located at the left edge of the right image, no determination of the SAD for the disparity area up to $d_{\max}$ can take place, since there are less than a number $d_{\text{num}}$ of pixels on the left side for an SAD calculation. In this case, the method does not provide any disparity values and an empty border area is created in the disparity map. A determination of the disparity is only possible from a distance of $d_{\max}$ to the edge. The programme then converts the images into the file format "CV_32F". In the process, each pixel is represented by a floating point number with the word length of 4 bytes, which enables the execution of various floating point operations in the further processing steps. Some methods generate disparity maps with pixel values in fixed-point representation.

**Table 1.** Parameter definitions used according to [11].

| Parameter | Description |
|---|---|
| Uniqueness Ratio [%] | Percentage difference that the pixel with the best SAD rating must have to the pixel with the second best SAD rating in order to berecognized as a corresponding pixel. |
| BlockSize [px] | Size of the block-shaped area around the current pixel that is used to calculate the similarity measure. |
| MinDisparity [px] | Smallest disparity studied. |
| NumDisparities [px] | Number of disparities that the algorithm distinguishes in the calculation; Defines the minimum distance with MinDisparity. |

In this case, the pixel values are also scaled. Figure 12 shows an example of the disparity maps determined.

Based on the disparity map, the programme either performs an estimation of the spatial depth or generates data for the three-dimensional reconstruction of the recorded scene, depending on the user's wishes. The estimation of the spatial depths takes place at each iteration and thus for each recorded stereo image pair. It is implemented via two different approaches, each of which determines the depth for a small region-of-interest (RIO) in the centre of the disparity map. For this reason, both procedures are also referred to as "local".

The first local method determines the mean value of the disparities of all pixels of the RIO. This mean value is then assumed to be the disparity value for the whole RIO. The spatial depth of the RIO can then be estimated via stereo triangulation by using the calibration data.

The second local procedure is carried out by the method cv:: reprojectImageTo3D(). This generates a multidimensional field that contains three spatial coordinates for each pixel of the disparity map. From this field, the programme extracts the values for the depth coordinate of the RIO and generates an average value for the depth. The calculation of the depth coordinate is mathematically identical for both approaches. However, the execution of the former approach is somewhat more time-efficient.

The question arises why the depth is not calculated for a single pixel, but instead the mean value of the RIO is used. The reason for this is the smoothing influence of this method, which avoids strong outliers in the numerical values of the output.

The execution per Tick() iteration enables the continuous output of a disparity map with depth value and marked RIO. Figure 12 shows such an output, which the programme saves to the hard disk as a video. Note that the lettering of the depth value looks blurred because of the limited resolution of the frame.

A third, global evaluation method is the generation of data for the three-dimensional reconstruction of the entire recorded scene. Here the programme again uses the method cv::reprojectImageTo3D(). However, the multidimensional field with the three-dimensional pixel coordinates is transferred

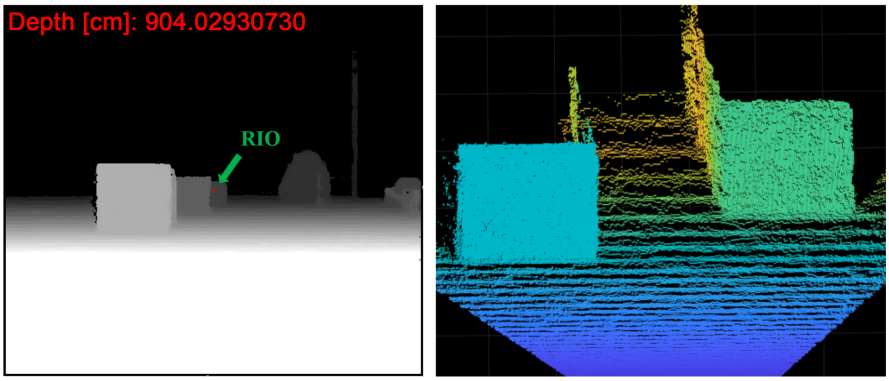


Figure 12. Disparity map with RIO and depth value (left) and 3D reconstruction (right).

into the field format representation of the numerical simulation software Mathworks MATLAB and stored on the hard disk via the method store3Dpoints(). A visualisation in parallel to the execution of the stereo camera model does not take place, since the generation of a visual point cloud for such a large number of pixels slows down the programme considerably. Instead, the three-dimensional pixel coordinates are recorded and saved during the simulation by pressing a key. The visualisation of the point cloud from the three-dimensional pixel coordinates takes place using MATLAB, with the output as in Figure 12.

Once the user has achieved his goals by running the stereo camera model, he terminates the simulation within the Unreal Editor. The method EndPlay() closes all output windows.

If the user wants to save the output of the depth estimation as a video, this does not take place in the method Tick(), but is outsourced to a task running in parallel for reasons of efficiency. For this task, the class "FAsyncSaveVideoTask" is created in the header file, which organises the data transfer and inherits various methods from the base class "FNonAbandonableTask" of the Unreal Engine. The process of task-based storage begins in each iteration immediately after the DepthMeasurement() method in order to add the currently recorded disparity map to a video sequence. To do this, the programme first creates a new instance of the task and starts it with the StartBackgroundTask() method inherited from the base class "FNonAbandonableTask". The method Tick() passes the image data of the disparity maps to be saved to the task instance, which assembles them into a video file of type AVI via OpenCV methods. Meanwhile, the task instance runs in parallel with the operations of the next Tick() iteration. Once this iteration has completed the DepthMeasurement() method and the task instance has been fully executed, the method terminateForAsyncSaveVideoTask() terminates the task instance and deletes it. This is immediately followed by the creation of a new instance of the task for the next Tick() iteration and the start of this instance. The parallel processing is shown in Figure 13. At the end of the simulation, the method EndPlay() closes the task instance. The method terminate ForAsyncSaveVideoTask() is executed one last time to terminate the task instance from the last iteration of Tick().

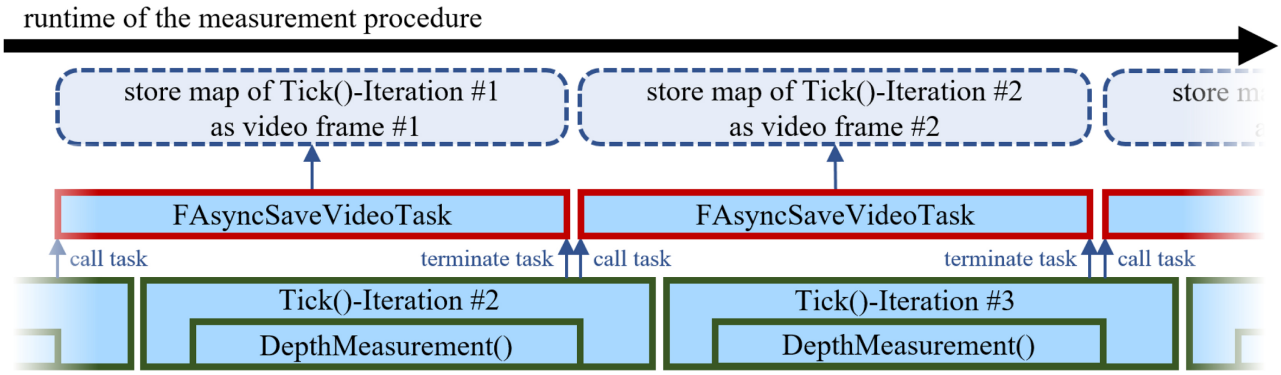


Figure 13. Diagram of the parallel programme sequence of storage.

The calibration procedure is implemented as a separate C++ console application within a project of the Visual Studio development environment. The reason for the separate implementation is that the calibration programme cannot be adequately operated when implemented in an Unreal Engine project. This is because the Unreal Engine does not allow the control of the calibration procedure via the console without further ado. A separate implementation has the advantage that the programme can be used autonomously for future applications, such as for real systems. The named console application consists of a header file (.h) and a source code file (.cpp). The header file includes a number of necessary libraries, such as OpenCV. It also contains the declaration of the class "StereoCamCalib" and all attributes and methods for the operation of the calibration procedure. Figure 14 shows all methods, while the attributes are not shown for reasons of clarity. They can be taken from the software if necessary.

Within the calibration procedure, the initStereoCam() method initialises the application by defining all attributes. These attributes include parameters for controlling the programme flow, model parameters, parameters of the calibration pattern, the file paths for reading in stereo image pairs and the file paths for storing the calibration and correction data. This is followed by the call of the MonoCalibration() method. First, the iterative evaluation of all stereo images of the calibration pattern is carried out. For this purpose, the method cv::imread() reads the stereo image pairs from the hard disk memory at each iteration. After a conversion of the colour space to a grey scale representation by cv::cvtcolor(), cv::findChessboardCorners() detects the corner points within the patterns, which serve as fixed points for the calibration. If the detection process for the current stereo image pair is not successful, the programme skips further processing of the currently detected fixed points and loads the next stereo image pair from the hard disk.

If the detection process has been successful, the position of the corner points is first determined to subpixel accuracy by cv::cornerSubPix(). Then the method

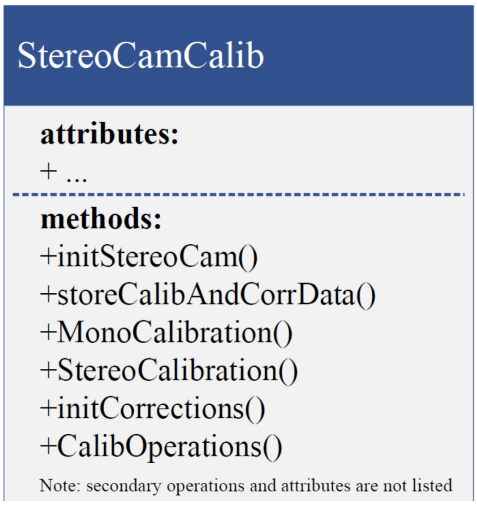


**Figure 14.** Class diagram of "StereoCamCalib".

cv::drawChessboardCorners() inserts markers for the detected corners into the stereo images, which is shown in Figure 15.

For visual assessment of the stereo images, cv::imshow() outputs both images with corner markers on the screen. If the user is satisfied with the detection result, he confirms the transfer of the points into the further calibration process.

Based on the confirmed fixed points, the method cv::calibrateCamera() determines the extrinsic, intrinsic and distortion-related calibration data of both cameras. Furthermore, this method yields the reprojection errors, which are used to assess the calibration quality.

After that, the calibration procedure continues with the call of the StereoCalibration() method. In it, cv::stereoCalibrate() determines the extrinsic calibration data of the stereo camera system. In addition, this method outputs the reprojection error of the stereo calibration. The further calibration procedure terminates if this error is too high.

Subsequently, the method initCorrections() determines the data for rectification and correction of the lens distortion. In it, the method cv::stereoRectify() determines matrices for projecting points in space onto rectified views of the two-dimensional image planes. The method also determines the reprojection matrix, which allows a depth estimation and reconstruction of the three-dimensional scene. The rectification and distortion correction within the later measurement procedure is based on maps. Therefore cv::initUndistortRectifyMap() is used to determine these maps.

After the calibration has been carried out and if the user confirms this by pressing a key, the method storeCalibAndCorrData() saves the calibration data as an XML file on the hard disk. To improve the portability of the calibration procedure described, the method CalibOperations() summarises the previously described procedure.

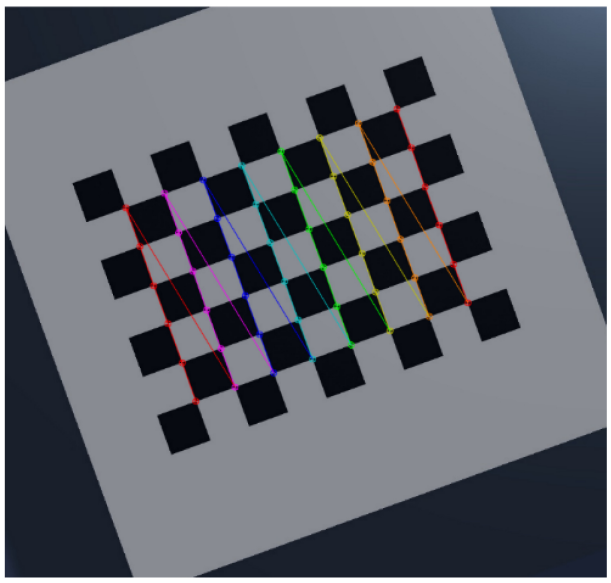

**Figure 15.** Image with markers in the inner corners.

## 6. Experimental Evaluation

### 6.1. Quantitative Evaluation of Measurement Frequency and Depth Estimation

After the implementation is completed, an analysis of the model behaviour should prove that the requirements mentioned in the introduction are fulfilled. An essential requirement for the stereo camera model is the ability to estimate depth in real-time. To check this requirement, the stereo camera model is supplemented with a function that measures the execution frequency. It stores the duration of a large number of Tick() iterations. When the simulation is terminated, the function calculates the mean value and the standard deviation of the execution frequencies of all Tick()-Iterations and outputs them. Since all methods for image processing are executed within the method Tick(), its execution frequency corresponds to the measurement frequency of the model.

To create a reproducible environment, the experimental computer is restarted before the evaluation and all unnecessary programmes are deactivated. The simulation also starts after a short waiting period so that the internal execution frequency of the Unreal Engine and the task management of the computer have stabilised.

The measurement frequency is determined for the block-matching method and for the semi-global matching method, each in a CPU- and GPU-supported variant. Table 2 lists all relevant model parameter, which are used to measure the measurement frequency. For the remaining parameters of the stereo correspondence analysis methods, which are not shown in the table, the default values from OpenCV are applied.

The results of the measurement within Table 3 show that the local block-matching method does not experience a significantly higher measurement frequency by running it on the GPU. A different picture emerges for the semi-global matching method according to [18] and [19], which has properties of both local and global stereo correspondence analysis. Here, the use of the GPU enables an increase in the measurement frequency by more than six times. Thus, only a GPU-accelerated execution guarantees the real-time capability of this

Table 2. Model parameters used for the evaluation of the measurement frequency.

| Stereo system | | Stereo correspondence analysis methods | |
|---|---|---|---|
| Parameter | Value | Parameter | Value |
| mode of operation | "measurement" | UniquenessRatio [%] | 21 |
| resolution [px$^2$] | 720 × 576 | BlockSize [px] | 19 |
| base width [cm] | 10 | MinDisparity [px] | 0 |
| horizontal field of view [˚] | 90 | NumDisparities [px] | 64 |
| focal length [px] | 360 | | |

**Table 3.** Results of the measurement frequency evaluation.

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| CPU-based block-matching method | | CPU-based semi-global matching method | |
| number of iterations | 1000 | number of iterations | 1000 |
| arithmetic mean [FPS] | 37.5585 | arithmetic mean [FPS] | 5.6977 |
| standard deviation [FPS] | 3.7410 | standard deviation [FPS] | 1.2734 |
| GPU-based block-matching method | | GPU-based semi-global matching method | |
| number of iterations | 1000 | number of iterations | 1000 |
| arithmetic mean [FPS] | 39.2316 | arithmetic mean [FPS] | 35.7616 |
| standard deviation [FPS] | 2.8043 | standard deviation [FPS] | 4.1839 |

method as defined in Section 1. The strong increase in speed can be attributed to the global evaluation steps of the method, which place higher demands on computing speed and memory capacity. They obviously benefit from a parallelised calculation using GPUs.

In summary, this study demonstrates the real-time capability of a depth estimation using the developed stereo camera model. In addition to these main findings, the high standard deviations from the mean execution frequency illustrate that for real sensor applications within safety-critical systems, dedicated real-time hardware with stable clocking of computation and data transmission is necessary.

Another important object of investigation is the ability of the model to estimate depth. In order to set up an appropriate test scenario, several cube actors are placed in the virtual environment of the Unreal Engine at different distances from the stereo camera actor. The spectrum of distances ranges from 50 centimetres to 600 centimetres. The cubes are staggered within this range at a distance of 50 centimetres from each other. In order to be able to measure the distances of the cubes in the virtual environment clearly and without further programming, the "Distance Measuring Tool" designed by [20] is used. In addition, the generation of a sufficiently varied surface texture is also necessary for the robust operation of the stereo correspondence analysis method. Therefore, the cubes are given the texture-rich material emulation "M_Basalt_Rock", which is shown in **Figure 16** on the left.

The parameter set from **Table 2** is reused for the investigation of the depth estimation. In doing so, the programme uses the parameters for focal length and base width, which can be set to a desired value within the Unreal Editor and are therefore known. In this way, a higher precision of the depth measurement is achieved than with the numerically determined values for focal length and base width from the calibration data. The GPU-based block-matching method is used

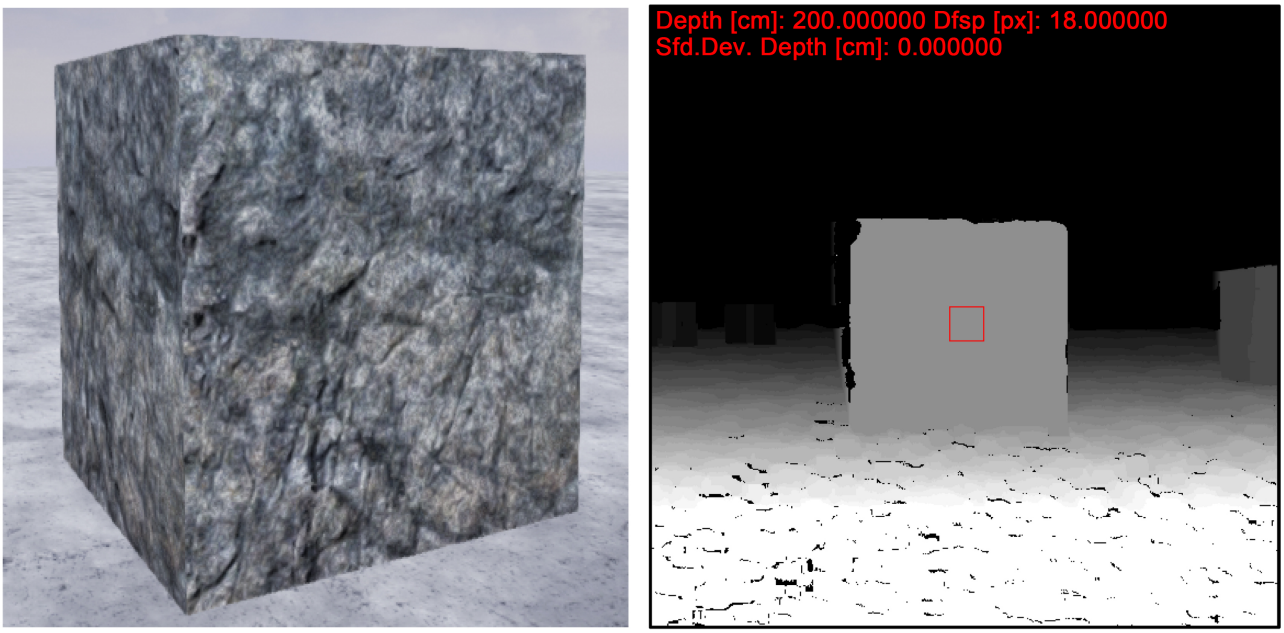


**Figure 16.** Cubic object with "M_Basalt_Rock" as material (left) and output of the depth estimation (right).

for the stereo correspondence analysis.

The experimental procedure starts with the measurement of the distances to all cubes. An output for the mean depth, the mean disparity in the RIO and the standard deviation is printed within the disparity map. **Figure 16** shows a recorded disparity map with the described measures on the right-hand side.

After a disparity map has been generated for all distances, the measured numerical values are documented. From this data, **Figure 17** can be generated, which compares the accurate depth retrieved from the virtual environment with the depth measured by the model on the left-hand side and with the disparity measured by the model on right-hand side.

**Figure 17** shows that the measured depth values are a good approximation of the accurate depth values of the cubes. At greater depths, the decreasing depth resolution becomes noticeable. The values of the disparity become denser in the range of smaller distances. Here a larger number of different distances can thus be distinguished. However, such a distribution of disparity values leads to greater deviations from the accurate distance for more distant objects. **Figure 18** clearly shows the narrower scaling of the values at low distances using the example of a rock object.

In summary, this study confirms the ability of the stereo camera model to estimate spatial depth. A higher depth resolution can be achieved by increasing the image resolution of the camera modules, which, however, increases the computational effort and thus decreases the execution frequency of the evaluation. In order to design the model to measure greater spatial depths, it is advisable to increase the base width and focal length. This also has undesirable side effects. For example, the ability to capture very close objects degrades and the field of view of the individual cameras is reduced.

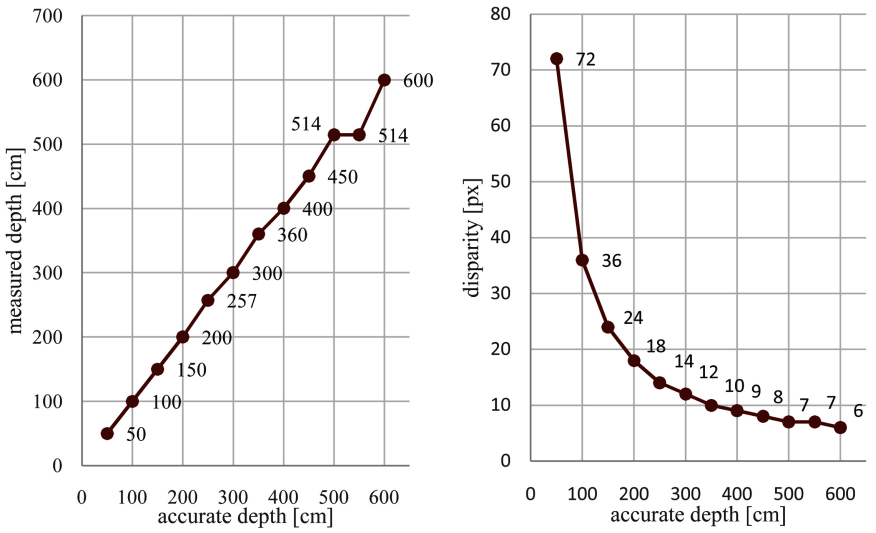


**Figure 17.** Depth (left) and disparity (right) measured by the model as a function of the accurate distance.

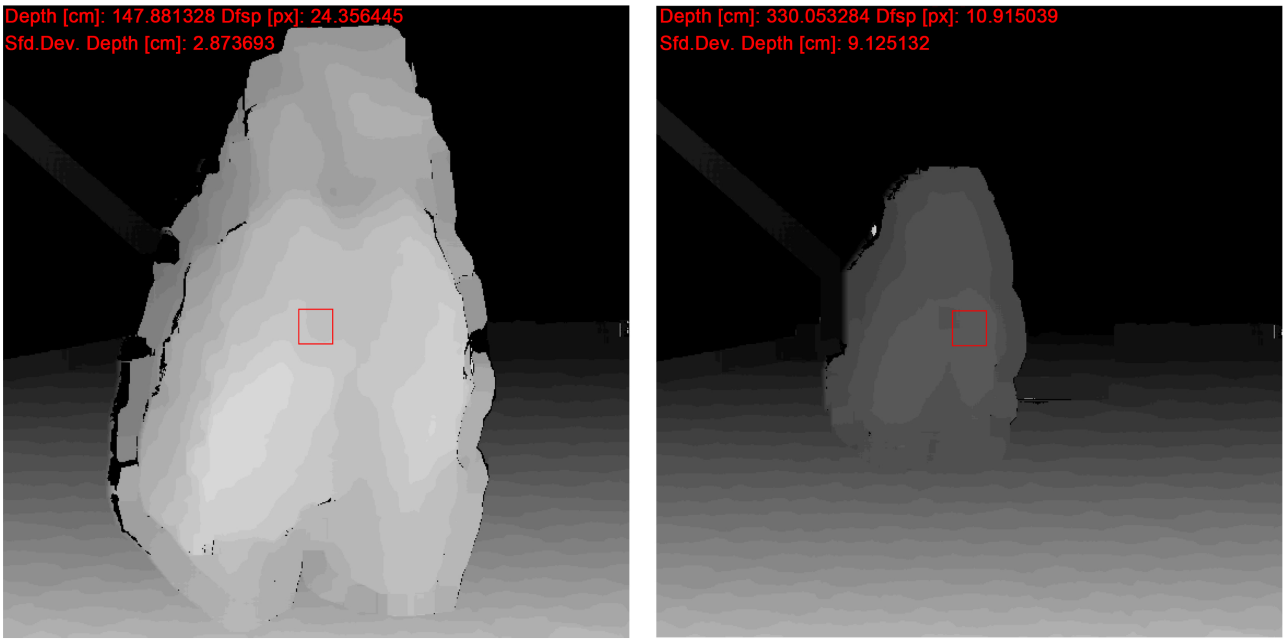


**Figure 18.** Different depth resolution for close (left) and far away objects (right).

## 6.2. Qualitative Evaluation of Further Phenomena

On the periphery of the investigations, other phenomena appear during the operation of the stereo camera model. These are, for example, process-related mismatches of correspondences in areas of repetitive textures. According to [3], such features can lead to defects in the disparity maps, especially when using local procedures for stereo correspondence analysis. Thus, these defects also appear during the evaluation using the block-matching method, although a material emulation as rich in texture as possible is used. The following **Figure 19** shows a disparity map of the block-matching procedure with defects on the left

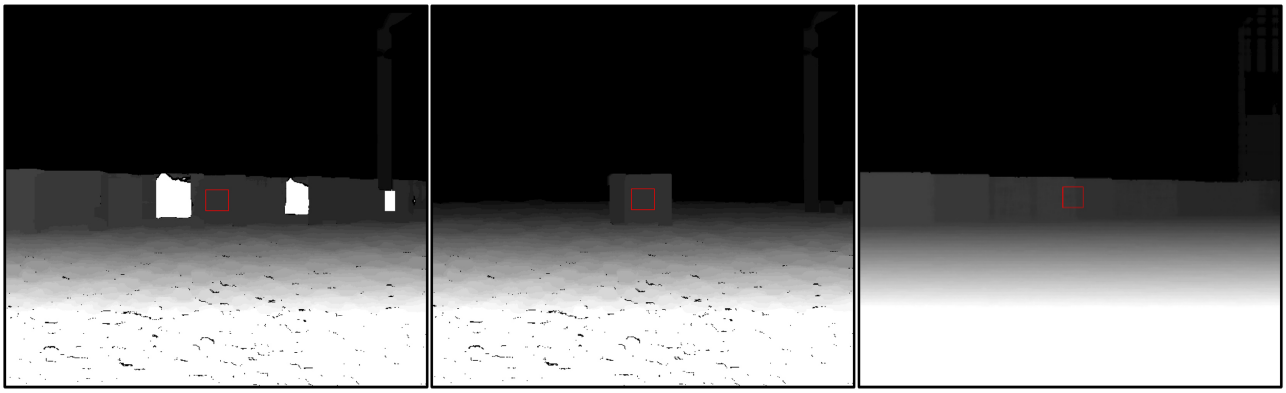

**Figure 19.** Disparity map of block-matching with a group of cubes (left), with a single cube (middle) andof semi-global matching (right).

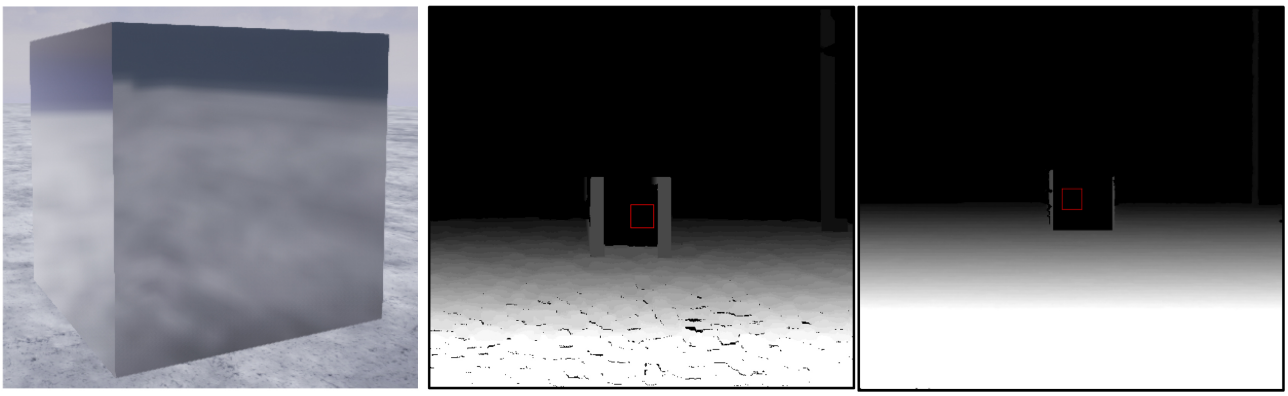

**Figure 20.** Material emulation "M_MetalSpec" (left), disparity map of block-matching (middle) and semi-global matching (right).

which contains much smaller disparities than expected. In the present case, the cause of the defects is that the textures of all cubes placed next to each other are chosen to be the same. Thus, the frontal view appears the same for all cubes in the virtual environment and the stereo image pairs therefore contain areas with repetitive textures. If a single cube is detached from the group at a constant distance, the disparity map shows no such defects. This supports the above hypotheses on the cause of the defect. This scenario is reproduced in **Figure 19**. A parameter-based solution to the problem is to increase the resolution of the camera modules in combination with a reduction of the search area. A reduction of defects can also be expected when using a virtual environment with a more complex texture. Alternatively, global or hybrid methods such as semi-global matching can be used. According to [19], they are more robust against repetitive structures, which are also shown in **Figure 19**.

According to [21], defects in disparity maps can also occur in areas of reflective or transparent surfaces. In order to prove this effect, images of a cube with the reflective material emulation "M_MetalSpec" are generated. The disparity maps show clear gaps in the area of the cube with both the block-matching method and the semi-global matching method. The results of both methods are illustrated via **Figure 20**.

## 7. Discussion

The previous sections describe how the model of a stereoscopic camera system is developed, implemented and evaluated. Therefore, the research objective is achieved by the work documented here. By running the model as an actor within the virtual environment of the Unreal Engine, it can be easily linked to virtual vehicles models. Furthermore, it is possible to estimate spatial depth with sufficient accuracy. In doing so, the model achieves a real-time execution frequency. Both characteristics are proven in chapter 6. Besides, the model is suitable to reproduce phenomena, which are characteristic for common stereo correspondence analysis methods. Since no virtual stereo camera models for the Unreal Engine or tools based on them are known to date, the present work contributes to scientific progress. However, there are also some aspects that the work leaves either unaddressed or unnoticed. For example, the presented model has an ideal geometry that neglects some realisation effects. Also, the evaluated distance range is not yet adapted to road traffic applications.

## 8. Conclusions and Future Research

The discussion in the previous section indicates that the model meets the requirements from the introduction. However, it is also clear that further development on this topic is worthwhile. Due to its behaviour, the model presented in this work is suitable for the virtual development of methods for stereo correspondence analysis and algorithms for higher-level scene evaluations, as for example methods for object detection methods based on 3D-Data.

In addition to that, the successful implementation will allow a comparison with models of other sensor principles as Lidar or Radar. A validation of the stereo camera model through a comparison with real stereo camera also suggests itself as a subsequent research topic.

Furthermore, the implementation of a combined sensor model would be conceivable, which obtains depth information by models with different sensor principles and combines their measurement results via sensor data fusion. For instance, a combination with object recognition according to [22] would be worthwhile.

As mentioned in the previous chapter, the model itself also has potential for optimization. For example, an adaption of the stereo systems parameters to the distance spectrum of objects in real road traffic situations should be investigated. Furthermore some realisation effects can be integrated in the model: A camera actor with lens distortion is available for the Unreal Engine and a non-parallel epipolar geometry can also be arranged by changing the models parameters. By integrating both feature, an evaluation of the software parts for distortions correction and rectification is possible.

Further optimization of the programmes performance by using more GPU-accelerated feature would also be desirable. Relevant performance improvements in terms of measurement accuracy and computational efficiency can be expected

for both approaches.

## Acknowledgements

The project is supported by the Ministry of Economic Affairs, Innovation, Digitization and Energy of North Rhine-Westphalia.

## Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

## References

[1] GP Bullhound. Autotech: The Mother of All Tech Battles. https://www.gpbullhound.com/insights/autotech-2019/

[2] Degen, R., Ott, H., Klein, F., Shankavaram, R., Leijon, M. and Ruschitzka, M. (2021) CityInMotion—A Virtual Urban Test Environment for Automated Mobility. *International Symposium for Development Metodology*, Wiesbaden, 9-10 November 2021.

[3] Winner, H., Hakuli, S., Lotz, F. and Singer, C. (2016) Handbook of Driver Assistance Systems: Basic Information, Components and Systems for Active Safety and Comfort. Springer, Cham. https://doi.org/10.1007/978-3-319-12352-3

[4] De Ponte Müller, F. (2017) Survey on Ranging Sensors and Cooperative Techniques for Relative Positioning of Vehicles. *Sensors*, **17**, Article No. 271. https://doi.org/10.3390/s17020271

[5] Sivaraman, S. and Trivedi, M.M. (2013) Looking at Vehicles on the Road: A Survey of Vision-Based Vehicle Detection, Tracking, and Behavior Analysis. *IEEE Transactions on Intelligent Transportation Systems*, **14**, 1773-1795. https://doi.org/10.1109/TITS.2013.2266661

[6] Miknis, M., Plassmann, P. and Jones, C. (2014) Virtual Environment Stereo Image Capture Using the Unreal Development Kit. 2014 *Global Summit on Computer & Information Technology* (*GSCIT*), Sousse, 14-16 June 2014, 1-5. https://doi.org/10.1109/GSCIT.2014.6970136

[7] Zhang, Y., Qiu, W., Chen, Q., Hu, X. and Yuille, A. (2018) UnrealStereo: Controlling Hazardous Factors to Analyze Stereo Vision. 2018 *International Conference on* 3*D Vision* (3*DV*), Verona, 5-8 September 2018, 228-237. https://doi.org/10.1109/3DV.2018.00035

[8] IPG Automotive GmbH (2022) Products and Solutions. (In Chinese) https://ipg-automotive.com/en/products-solutions/

[9] CARLA Team. CARLA: Open-Source Simulator for Autonomous Driving Research. https://carla.org/

[10] Hartley, R. and Zisserman, A. (2003) Multiple View Geometry in Computer Vision. 2nd Edition, Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511811685

[11] Kaehler, A. and Bradski, G.R. (2017) Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library. O'Reilly Media, Sebastopol.

[12] Konolige, K. (1998) Small Vision Systems: Hardware and Implementation. In: Shirai, Y. and Hirose, S., Eds., *Robotics Research*, Springer, London, 203-212. https://doi.org/10.1007/978-1-4471-1580-9_19

[13] Zhang, Z. (2000) A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**, 1330-1334. https://doi.org/10.1109/34.888718

[14] Brown, D.C. (1971) Close-Range Camera Calibration. *Photogrammetric Engineering*, **37**, 855-866. https://www.asprs.org/wp-content/uploads/pers/1971journal/aug/1971_aug_855-866.pdf

[15] Bouguet, J.-Y. (1999) Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the Algorithm. http://robots.stanford.edu/cs223b04/algo_tracking.pdf

[16] Object Management Group (2017) OMG® Unified Modeling Language® (OMG UML®): Version 2.5.1. https://www.omg.org/spec/UML/2.5.1/PDF

[17] Epic Games, Inc. Unreal Engine 4 Documentation. https://docs.unrealengine.com/4.27/en-US/

[18] Hirschmüller, H. (2011) Semi-Global Matching: Motivation, Development and Applications. In: *Photogrammetric Week*, Vol. 11, Wichmann Verlag, Heidelberg, 173-184. https://elib.dlr.de/73119/1/180Hirschmueller.pdf

[19] Hirschmüller, H. (2007) Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **30**, 328-341. https://doi.org/10.1109/TPAMI.2007.1166

[20] Burkart, B. (2015) Distance Measuring Tool. https://www.unrealengine.com/marketplace/en-US/product/distance-measuring-tool

[21] Geiger, A., Lenz, P. and Urtasun, R. (2022) Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite. 2012 *IEEE Conference on Computer Vision and Pattern Recognition*, Providence, 16-21 June 2012, 3354-3361. https://doi.org/10.1109/CVPR.2012.6248074

[22] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2016) You Only Look Once: Unified, Real-Time Object Detection. 2016 *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), Las Vegas, 27-30 June 2016, 779-788. https://doi.org/10.1109/CVPR.2016.91