Technische Hochschule Rosenheim

Master's Degree - Engineering Sciences

(Specialization in Mechatronics)

Master Project

Stereo Vision (3D-Vision) System for Bin-Picking

Submitted by:

Selvam Dasari Gnanaprakash

Matriculation Nr.: 1047785

Examiner:

Prof. Dr. Ing. Michael Wagner

Faculty of Engineering

TH Rosenheim

Date of Submission: July 23, 2024

Declaration

I hereby declare that I have independently authored this project, utilizing only the sources and resources specified. All content directly quoted or paraphrased from these sources has been clearly marked and appropriately referenced.

Place, Date: Rosenheim, 23.07.2024

Signature: D. G. Selvam

Acknowledgment

I would like to express my heartfelt gratitude to Prof. Dr.-Ing Michael Wagner for providing me with the opportunity to undertake this project. His expert guidance, insightful feedback, and unwavering support have been invaluable throughout the entire process. His encouragement and patience have greatly contributed to the successful completion of this project.

I extend my sincere thanks to Technische Hochschule Rosenheim for offering this opportunity and for providing excellent facilities that were instrumental in the successful execution of this project. The resources and support from the faculty and staff have been exceptional and crucial for my research.

I am also deeply grateful to my friends and colleagues for their continuous help, encouragement, and valuable discussions that enriched my understanding and contributed to the development of this work. Their collaboration and moral support have been a source of motivation throughout the project.

Finally, I would like to express my profound appreciation to my family for their endless support, patience, and understanding. Their belief in me has been a constant source of strength and inspiration, enabling me to persevere and complete this project.

Thank you all for your unwavering support and encouragement.

Abstract:

This project focuses on developing a stereo vision-based system for robotic grasping of various objects randomly organized in a bin. The core challenge addressed is the bin-picking problem, where objects are placed in a bin in a non-oriented, interlocked, jumbled, and/or heavily occluding manner. The increasing demand for automated solutions in industrial applications necessitates the development of reliable bin-picking technologies. This paper outlines the design and implementation of a stereo vision system using Alvium G1-240m/c cameras. The system captures high-quality stereo images, computes disparity maps, and converts these maps into depth information to obtain precise XYZ coordinates. These coordinates are then used to localize parts within the bin, enabling a robotic arm to accurately pick the parts from their calculated positions. The project also involves familiarization with the OpenCV library, camera mounting design, and camera calibration using the chessboard algorithm. The system's performance is validated through extensive testing in various bin-picking scenarios, highlighting its accuracy, efficiency, and reliability in real-world conditions.

Table of Contents:

1. Introduction

1.1 Introduction to Stereo Vision

Stereo vision, also known as stereoscopic vision or 3D vision, is a technique used to infer depth information from two-dimensional images. By capturing two images from slightly different perspectives, similar to human binocular vision, a stereo vision system can compute the disparity between corresponding points in the images. This disparity is then used to calculate the depth information, allowing for the reconstruction of a three-dimensional scene.

In industrial applications, stereo vision plays a crucial role in automation and robotics, enabling machines to perceive and interact with their environment in a more human-like manner. This technology is particularly valuable in scenarios where precise spatial awareness is required, such as in bin-picking systems where robots need to identify and grasp objects from a cluttered environment.

1.2 Objective

The primary objective of this project is to develop a comprehensive stereo vision system for bin-picking applications. This system will process stereo images to extract XYZ coordinates, which will be used to accurately localize parts within a bin. By feeding these coordinates to a robot, the system will enable the robot to pick parts from the correct calculated positions.

Specific objectives include:

1. **Get Familiar with OpenCV Library:** Understand and utilize the functionalities of the OpenCV library for image processing and computer vision tasks.
2. **Design a Camera Mount for Installing the Alvium Cameras:** Create a stable and precise mounting system for the Alvium G1-240m/c cameras to ensure accurate image capture.
3. **Calibrate the Camera Using Chessboard Algorithm from OpenCV:** Perform camera calibration using the chessboard pattern to obtain intrinsic and extrinsic camera parameters.
4. **Capture High-Quality Images Using the Calibrated Cameras:** Ensure the images captured are of high resolution and free of distortions.
5. **Stereo Calibration**: Establishment of the spatial relationship between calibrated camera pairs to facilitate accurate depth estimation.
6. **Compute the Disparity Map from the Stereo Pair Images:** Implement algorithms to generate disparity maps from the stereo image pairs.
7. **Convert Disparity Map into Depth Information to Obtain XYZ Coordinates:** Transform the disparity maps into depth information and subsequently derive XYZ coordinates for object localization.

2. Literature Review

This section explores essential concepts, methodologies, and technologies pivotal for developing robust stereo vision systems, particularly in bin-picking applications. We delve into camera calibration techniques, examine the principles of stereo vision, and scrutinize various algorithms and techniques employed for disparity map computation and depth estimation.

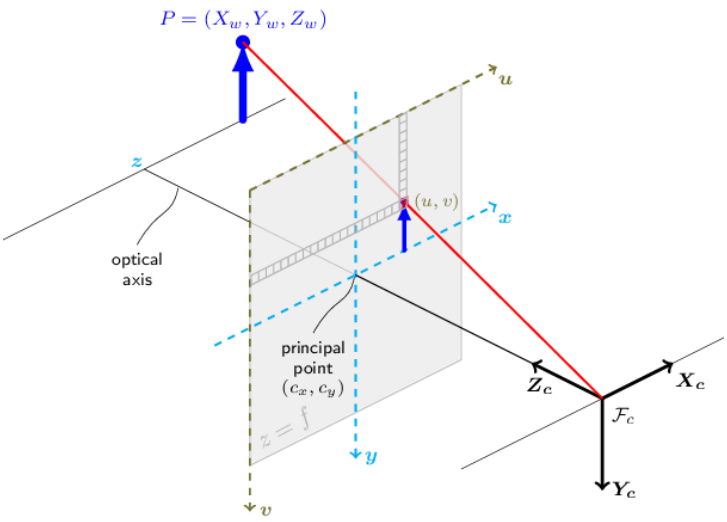
2.1 Camera Calibration

Camera calibration is a critical step in stereo vision systems to ensure precise 3D reconstruction. It involves estimating intrinsic parameters (focal length, optical center, distortion coefficients) and extrinsic parameters (rotation, translation between cameras). The process typically includes:

- **Calibration Pattern:** Utilizing a known pattern, such as a chessboard, to capture multiple images from different angles.
- **Parameter Estimation:** Applying algorithms to estimate intrinsic and extrinsic parameters from the captured images.
- **Rectification:** Adjusting images to remove distortions and align them accurately for disparity calculation.

2.1.1 Linear Camera Model

The foundational pinhole camera model assumes a linear relationship between 3-D world coordinates (X, Y, Z and 2-D image coordinates (x, y). Key components include transformations from world to camera coordinates and from camera coordinates to the image plane, incorporating rotation R, translation T, and projection parameters.



Camera calibration is a very important part of machine vision technology and photogrammetry. The essence of machine vision technology and photogrammetry is to obtain geometric information of three-dimensional objects from the image information taken by the camera. It can also be said that camera calibration is the foundation of machine vision technology and photogrammetry. The process of camera calibration is the process of obtaining the internal and external parameters of the camera through calculations. Among them, the internal parameters include the focal length of the camera, and the external parameters include the position information of the camera itself in the world coordinate system. The projection relationship between the world coordinate system and the image coordinate system is determined by these internal and external parameters of the cameras.

The world coordinate system XYZ and the camera coordinate system xyz are separated, but the camera coordinate system and the image coordinate system x′y′ are also separated.

The ultimate goal of imaging is for computer processing, so it is necessary to establish a connection from the world coordinate system to the pixel coordinate system.

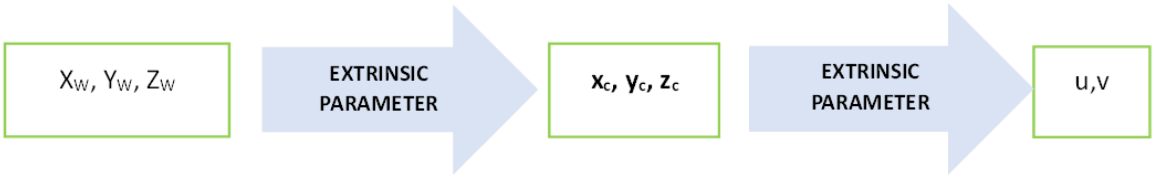


Fig1: Conversion from 3-D world coordinates to computer image coordinates under the linear camera model

Conversion from the world coordinate system X_W, Y_W, Z_W to the camera coordinate system $\mathbf{x_c}$, $\mathbf{y_c}$, $\mathbf{z_c}$. This conversion can be represented as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \boldsymbol{R} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \boldsymbol{T}$$

where R and T are, respectively, 3 × 3 rotation matrix (actually a function of the angles between the three pairs of corresponding coordinate axes of the two coordinate systems) and 1 × 3 translation vector:

$$\boldsymbol{R} = \begin{bmatrix} \boldsymbol{r_1} & \boldsymbol{r_2} & \boldsymbol{r_3} \\ \boldsymbol{r_4} & \boldsymbol{r_5} & \boldsymbol{r_6} \\ \boldsymbol{r_7} & \boldsymbol{r_8} & \boldsymbol{r_9} \end{bmatrix}$$

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{T_x} & \boldsymbol{T_y} & \boldsymbol{T_z} \end{bmatrix}^{\boldsymbol{T}}$$

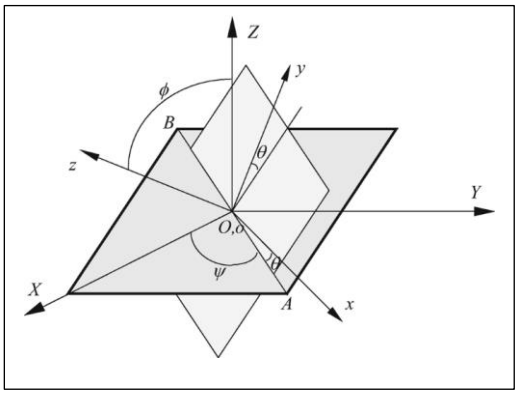
Conversion C2 from the camera coordinate system xyz to the image plane coordinate system x′y′. This conversion can be represented as

$$\boldsymbol{x'} = \boldsymbol{\lambda} \frac{\boldsymbol{x}}{\boldsymbol{z}}$$

$$\boldsymbol{y'} = \boldsymbol{\lambda} \frac{\boldsymbol{y}}{\boldsymbol{z}}$$

2.1.2 Intrinsic and Extrinsic Parameters

The calibration parameters involved in camera calibration can be divided into external parameters (outside the camera) and internal parameters (inside the camera).



The first step of transformation is to transform from the 3-D world coordinate system to the 3-D coordinate system whose center is at the optical center of the camera. The transformation parameters are called external parameters or camera attitude parameters. The rotation matrix R has a total of nine elements, but in fact there are only three degrees of freedom, which can be represented by the three Euler angles of the rigid body rotation.

Using Euler angles, the rotation matrix can be represented as a function of θ, ϕ, and ψ:

$$\boldsymbol{R} = \begin{bmatrix} cos\Psi\ cos\theta & sin\Psi\ cos\theta & -sin\theta \\ -sin\Psi cos + cos\Psi sin\theta sin\phi & \cos\cos\phi + sin\Psi\ \sin\theta\ sin\phi & \cos\theta\ sin\phi \\ \sin\Psi\ sin\phi + cos\Psi sin\ \theta cos\phi & -cos\Psi sin\phi + sin\Psi sin\theta cos\phi & cos\theta cos\phi \end{bmatrix}$$

It can be seen that the rotation matrix has three degrees of freedom. In addition, the translation matrix also has three degrees of freedom (translation coefficients in three directions). In this way, the camera

has six independent external parameters, namely, the three Euler angles θ, ϕ, ψ in R and the three elements Tx, Ty, Tz in T.

In addition to this, we need to some other information, like the intrinsic and extrinsic parameters of the camera. Intrinsic parameters are specific to a camera. They include information like focal length (fx,fy) and optical centres (cx,cy). The focal length and optical centres can be used to create a camera matrix, which can be used to remove distortion due to the lenses of a specific camera. The camera matrix is unique to a specific camera, so once calculated, it can be reused on other images taken by the same camera. It is expressed as a 3x3 matrix:

$$Camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

For stereo applications, these distortions need to be corrected first. To find these parameters, we must provide some sample images of a well-defined pattern (e.g. a chess board). We find some specific points of which we already know the relative positions (e.g. square corners in the chess board). We know the coordinates of these points in real world space and we know the coordinates in the image, so we can solve for the distortion coefficients. For better results, we need at least 10 test patterns.

2.1.3 Non-Linear Camera Model

In actual situations, a camera usually uses a lens (often containing multiple lenses) for imaging. Due to the levels of the processing technology of the lens and the manufacturing technology of the camera, the projection relationship of the camera cannot be simply described as a pinhole model. The real optical system does not work exactly according to the idealized pinhole imaging principle, but there is lens distortion.

2.1.3.1 Type of Distortion

Due to the influence of various distortion factors, when projecting a 3-D space object point onto a 2-D image plane, the actually obtained image point coordinates (xa, ya) and the undistorted ideal image point coordinates (xi, yi) will be different (there are deviations), which can be expressed as

$$x_a = x_i + d_x$$

$$y_a = y_i + d_y$$

where dx and dy are the total non-linear distortion deviation values in the X and Y directions, respectively.

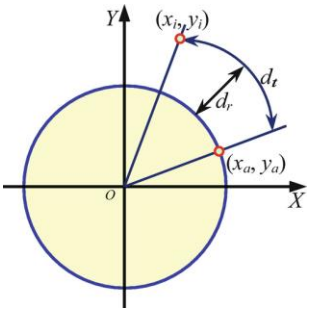
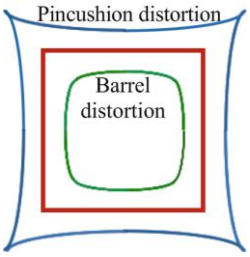


Fig: Schematic diagram of radial and tangential distortion

There are two common basic distortion types: radial distortion and tangential distortion. The influence of these two basic distortions is shown in Fig. 2.3, where dr represents the deviation caused by radial distortion and dt represents the deviation caused by tangential distortion.

2.1.3.2 Radial Distortion

Radial distortion is mainly caused by irregular lens shape (surface curvature error). The deviation caused by it is often symmetrical about the main optical axis of the camera lens, and it is more obvious at the distance from the optical axis along the lens radius. Generally, the positive radial distortion is called pincushion distortion, and the negative radial distortion is called barrel distortion, as shown in Fig. 2.4. The mathematical model is



$$d_{xr} = x_i(1 + k_1 r^2 + k_2 r^4 + \cdots)$$
$$d_{yr} = y_i(1 + k_1 r^2 + k_2 r^4 + \cdots)$$

where $r = (x_i^2 + y_i^2)^{1/2}$ is the distance from the image point to the image center and k1, k2, etc. are the radial distortion coefficients.

2.1.3.3 Tangential Distortion

Tangential distortion is mainly caused by the non-collinear optical centers of the lens group, which produces the actual image point to move tangentially on the image plane. Tangential distortion has a certain orientation in space, so there is a maximum axis of distortion in a certain direction, and a minimum axis in the direction perpendicular to that direction. The solid line represents the absence of distortion, and the dashed line represents the result caused by tangential distortion. Generally, the influence of tangential distortion is relatively small, and independent modelling is relatively small.

$$d_{xt} = x_i + [\,2p_1 x_i y_i + p_2(r^2 + 2x^2)]$$
$$d_{yt} = y_i + [\,p_1(r^2 + 2y^2) + 2p_2 x_i y_i\,]$$

In short, we need to find five parameters, known as distortion coefficients given by:

$$Distortion\ coefficients = (k_1\ k_2\ k_3\ k_4\ k_5\ )$$

2.2 Stereo Vision

Stereo vision is essential for reconstructing objective scenes in computer vision applications. The process involves several interconnected modules to accurately perceive depth and spatial information from images. Fig. 2 illustrates the flow diagram of the stereo vision process, detailing the data flow through each module.

The complete stereo vision system can be segmented into six functional modules, each contributing distinct tasks crucial for achieving accurate 3D reconstruction:

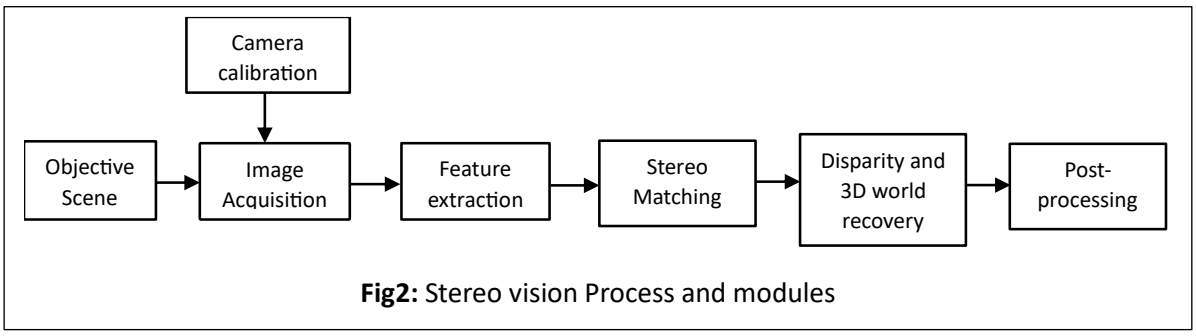


Fig2: Stereo vision Process and modules

2.2.1 Image Acquisition

Image acquisition involves capturing synchronized images of the scene using the stereo camera setup. These images serve as input for subsequent processing modules, providing the raw data necessary for depth estimation and 3D reconstruction.

2.2.2 Feature Extraction

Feature extraction identifies distinctive points or regions in the stereo images that can be reliably matched across views. Various features, from point-like to volumetric features, are considered, each

balancing complexity with accuracy in matching algorithms. Feature extraction is critical for establishing correspondences essential for depth estimation.

2.2.3 Stereo Matching

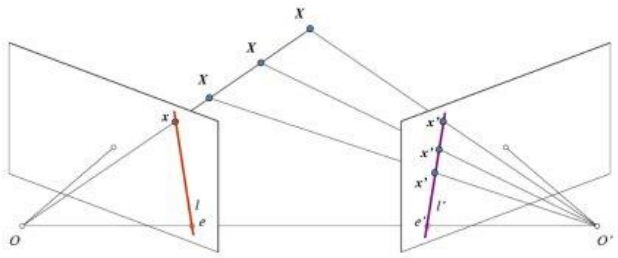
Stereo matching computes the disparity (horizontal shift) between corresponding features in the left and right images. This process, often challenging due to variations in lighting, texture, and occlusion, determines the depth information by leveraging parallax. Grayscale correlation and feature-based matching methods are commonly employed, each offering unique advantages in different scene conditions.

2.2.3.1 Template Matching

The region-based method needs to consider the nature of the neighbourhood of the point, and the neighbourhood is often determined with the help of templates (also called mask, sub-images, or windows).

2.2.3.2 Epipolar Line Constraint

The epipolar line constraint is a fundamental concept in stereo vision that significantly reduces the complexity of finding corresponding points in stereo images. When a point in one image is projected into 3D space, it corresponds to a line (epipolar line) in the other image rather than an area. This constraint helps transform the search for matching points from a two-dimensional problem to a one-dimensional one, which greatly enhances both accuracy and computational efficiency.



Schematic diagram of epipoles and epipolar lines

Understanding Epipolar Geometry:

1. **Epipolar Plane:** The plane that contains the baseline (the line connecting the optical centers of the two cameras) and the point in 3D space.
2. **Epipoles:** The points of intersection of the baseline with the image planes.
3. **Epipolar Line:** The intersection of the epipolar plane with the image plane. Every point in one image has a corresponding epipolar line in the other image, along which its matching point must lie.

2.2.4 3-D Information Recovery

After stereo matching, the computed disparities are converted into depth information, enabling the reconstruction of the 3D structure of the scene. Factors influencing the accuracy of depth measurement include calibration precision, feature detection robustness, and baseline length between cameras. Techniques like triangulation transform disparities into spatial coordinates, facilitating precise 3D information recovery.

2.3 Algorithms and Techniques

In the realm of stereo vision, various algorithms and techniques are employed to compute disparity maps and subsequently convert them into depth information. These methods differ in terms of complexity, accuracy, and suitability for various applications, ensuring that the diverse needs of stereo vision systems are met. This section reviews the primary algorithms and techniques used in stereo vision.

2.3.1 Block Matching Algorithms

Block matching algorithms are widely used in stereo vision due to their simplicity and efficiency. These methods divide images into smaller blocks and match them between the stereo pair to determine disparity.

1. **StereoBM (Block Matching):** This algorithm segments the image into fixed-size blocks and searches for the best match in the corresponding image by minimizing a cost function, such as sum of absolute differences (SAD). StereoBM is computationally efficient and suitable for real-time applications. However, its performance can degrade in regions with low texture or repetitive patterns, where distinctive features are lacking.

2. **StereoSGBM (Semi-Global Block Matching):** An extension of block matching, StereoSGBM enhances accuracy by considering multiple paths of pixels, integrating both local and semi-global matching strategies. This method improves disparity estimation in low-texture and high-disparity variation regions by aggregating cost functions over different directions, resulting in more consistent and accurate disparity maps.

2.3.2 Global Methods

Global methods for disparity estimation address the limitations of local approaches by formulating the problem as an optimization task. These methods aim to find the disparity map that minimizes an energy function defined over the entire image.

1. **Graph Cuts and Belief Propagation:** These techniques approach disparity estimation as an energy minimization problem. Graph cuts partition the image into regions of similar disparity by minimizing a global energy function, while belief propagation iteratively updates the disparity values based on probabilistic models. Both methods provide high accuracy and can effectively handle large disparity variations and occlusions. However, they require significant computational resources, making them less suitable for real-time applications.
2. **Dynamic Programming:** This approach optimizes a cost function along individual rows or columns of the image, balancing accuracy with computational efficiency. Dynamic programming solves the stereo matching problem by minimizing a cost function that includes both data and smoothness terms. It is particularly effective for structured environments where disparities vary smoothly, offering a good trade-off between computational complexity and accuracy.

2.3.3 Post-Processing Techniques

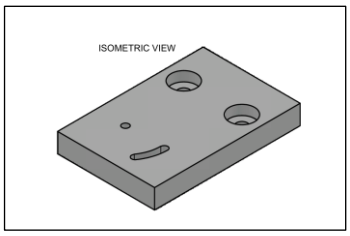
Post-processing techniques are essential for refining disparity maps obtained from initial matching algorithms. These methods improve the quality of disparity maps by reducing noise, handling occlusions, and enhancing edge preservation.

1. **Filtering and Smoothing:** Techniques such as Weighted Median Filtering (WMF) and Joint Bilateral Filtering (JBF) are employed to smooth disparity maps while preserving important edges. WMF uses weights based on the spatial and intensity differences between pixels, ensuring that edges are maintained. JBF incorporates both spatial and range information, filtering disparity maps to reduce noise without blurring sharp discontinuities.
2. **Occlusion Handling:** Occlusions occur when parts of the scene are visible in one image but not in the other. Effective occlusion handling techniques detect and manage these regions, improving the reliability of disparity maps. Methods include identifying occluded areas using left-right consistency checks and inpainting techniques to fill occluded regions with plausible disparity values.

3. Methodology

3.1 Camera Mounting

Proper camera mounting is crucial for ensuring accurate stereo vision. The cameras must be securely fixed to avoid any movement that could affect calibration and depth estimation. The following steps outline the camera mounting process:



- **Mount Design:** Designed a robust mount using CAD software (e.g., AutoCAD) to ensure the cameras are held firmly in the desired position.
- **Alignment:** Ensure the cameras are aligned parallel to each other with a known baseline distance.
- **Stability:** Use materials and mounting techniques that minimize vibrations and movement.

3.2 Camera Calibration

Camera calibration is essential to determine the intrinsic and extrinsic parameters of the cameras, which are crucial for accurate 3D reconstruction.

3.2.1 Intrinsic Parameters

Intrinsic parameters are the internal characteristics of the camera, including the focal length, optical center, and distortion coefficients.

```
import cv2 as cv
import numpy as np
import glob
import pickle
boardsize = (7,9)
framesize = (640,480)
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)             # termination criteria
# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objP = np.zeros((boardsize[0]*boardsize[1],3), np.float32)
objP[:,:2] = np.mgrid[0:boardsize[0],0:boardsize[1]].T.reshape(-1,2)
 # Arrays to store object points and image points from all the images.
objPoints = [] # 3d point in real world space
imgPoints = [] # 2d points in image plane.
 images = glob.glob("C:/Users/localuser/Documents/test/CamR/*.bmp")
for image in images:
    img = cv.imread(image)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    ret, corners = cv.findChessboardCorners(gray, boardsize, None)                  # Find the chess board corners
    # If found, add object points, image points (after refining them)
    if ret == True:
        objPoints.append(objP)
        corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
        imgPoints.append(corners2)
        cv.drawChessboardCorners(img, boardsize, corners2, ret)                     # Draw and display the corners
        cv.imshow('img', img)
        cv.waitKey(1000)
cv.destroyAllWindows()
################ Get Intrinscic and Extrinsic Parametes by CALIBRATION ########################
calibration, cameraMatrix, distotion, rotVector, transVector = cv.calibrateCamera(objPoints, imgPoints, framesize, None, None)
```

3.2.2 Extrinsic Parameters

Extrinsic parameters describe the camera's position and orientation in the world, represented by rotation and translation vectors.

3.3 Stereo Calibration Process

Stereo calibration is essential for determining the relative positions and orientations of the two cameras in a stereo vision setup. This process involves calculating the extrinsic parameters (rotation and translation) between the two cameras, allowing the system to understand their spatial relationship. Stereo calibration ensures that the disparity between corresponding points in the two images can be accurately computed, which is crucial for depth estimation.

3.3.1 Parameters Out of Calibration:

I. Intrinsic Parameters of Each Camera

Intrinsic parameters define the internal characteristics of each camera. They include:

$$Camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Where, f_x and f_y are the focal lengths in the x and y directions, respectively.c_x and c_y are the coordinates of the optical center (principal point).

```
                        #Camera1 Caliberation
calibrationLeft, cameraMatrixLeft, distortionLeft, rotVectorLeft, transVectorLeft = cv.calibrateCamera(objPoints,
imgPointsLeft, grayLeft.shape[::-1], None, None)
heightLeft, widthLeft, channelsLeft = imgL.shape
                        # Camera2 Caliberation
calibrationRight, cameraMatrixRight, distortionRight, rotVectorRight, transVectorRight = cv.calibrateCamera(objPoints,
imgPointsRght, grayRight.shape[::-1], None, None)
heightRight, widthRight, channelsRight = imgR.shape
#### Returns the new camera intrinsic matrix based on the free scaling parameter.
newCameraMatrixLeft, roi_Left = cv.getOptimalNewCameraMatrix(cameraMatrixLeft, distortionLeft,(widthLeft,
heightLeft), 1, (widthLeft, heightLeft))

newCameraMatrixRight, roi_Right = cv.getOptimalNewCameraMatrix(cameraMatrixRight,
distortionRight,(widthRight, heightRight), 1, (widthRight, heightRight))
```

II. Extrinsic Parameters

Extrinsic parameters describe the position and orientation of the cameras relative to each other. They include:

- Rotation Matrix (R):The 3x3 rotation matrix describes the rotation of the right camera relative to the left camera.
- Translation Vector (T):The 3x1 translation vector describes the translation (shift) of the right camera relative to the left camera.

III. Essential Matrix (E)

The essential matrix encapsulates the rotation and translation between the two cameras, focusing on the epipolar geometry. It is used in the process of determining the relative pose of the cameras when the intrinsic parameters are known.

$$E = [T] \times R$$

IV. Fundamental Matrix (F)

The fundamental matrix is a 3x3 matrix that relates corresponding points in the two images. It is used to compute the epipolar lines in each image.

$$F = (mtxR^{-1})^T E(mtxL^{-1})$$

```
################ Stereo Caliberation ######################
    flags = 0
    flags |= cv.CALIB_FIX_INTRINSIC
    # here we fix the intrinsic camera matrices so that only Rotation, translation essential and fundamental matrices
    # so the intrinsic parameters are the same
ret, newCameraMatrixLeft, distCoeffsLeft, newCameraMatrixRight, distCoeffsRight, rotMatrix, transVector,
essentialMatrix, fundamentalMatrix = cv.stereoCalibrate(objPoints,  imgPointsLeft, imgPointsRight,
newCameraMatrixLeft, distortionLeft, newCameraMatrixRight, distortionRight, grayRightImage.shape[::-1],
criteria=criteria, flags=flags)
```

V. Rectification Parameters

Rectification aligns the images so that corresponding points lie on the same horizontal line. This step is essential for simplifying the stereo matching process.

```
############# Stereo Rectification ########################################
# Computes rectification transforms for each head of a calibrated stereo camera
# # # alpha = 1
# # #alpha=0 means that the rectified images are zoomed and shifted
rectifyScale = 1
rectTransLeft, rectTransRight, projMatrixLeft, projMatrixRight, disp2DepthMapMatrix, roiLeft, roiRight =
cv.stereoRectify(newCameraMatrixLeft, distCoeffsLeft, newCameraMatrixRight, distCoeffsRight, grayR.shape[::-1],
rotMatrix, transVector, rectifyScale, (0,0))
```

- Rectification Transforms (R1 and R2): Rotation matrices used to align the left and right images.
- Projection Matrices (P1 and P2): These matrices project the 3D points onto the rectified images.
- Disparity-to-Depth Mapping Matrix (Q): This 4x4 matrix is used to reproject disparity images to 3D space, allowing for depth calculation.

3.4 Image Processing Workflow

The image processing workflow in stereo vision involves several key steps to transform raw stereo images into meaningful 3D information. This process includes undistorting images, calculating disparity, estimating depth, and computing XYZ coordinates. Below, each of these steps is detailed to provide a clear methodology for their implementation.

3.4.1 Undistort Images

Undistorting images is a crucial step to correct lens distortions, ensuring that the images captured by the stereo cameras are accurately represented. This involves using the camera matrices and distortion coefficients obtained from the camera calibration process.

The steps are

1. **Load Calibration Parameters:** Load the intrinsic and distortion parameters for both the left and right cameras.
2. **Compute Rectification and Undistortion Maps:** Using the calibration parameters, compute the rectification transforms and undistortion maps.
3. **Apply the Maps to Undistort the Images:** Use the maps to rectify and undistort the stereo images.

```
import sys
import cv2 as cv
import numpy as np
## Load the Parameters
camMatrix0 = np.load('C:/Users/localuser/Documents/test/pixel2WCos/calibParams/cameraMatrixLeft.npy')
camMatrix1 = np.load('C:/Users/localuser/Documents/test/pixel2WCos/calibParams/cameraMatrixRight.npy')
distcoef0 = np.load('C:/Users/localuser/Documents/test/pixel2WCos/calibParams/Distortion_CoefficentLeft.npy')
distcoef1 = np.load('C:/Users/localuser/Documents/test/pixel2WCos/calibParams/Distortion_CoefficentRight.npy')
newCamMat0 = np.load('C:/Users/localuser/Documents/test/pixel2WCos/calibParams/newcameraMatrixLeft.npy')
newCamMat1 = np.load('C:/Users/localuser/Documents/test/pixel2WCos/calibParams/newcameraMatrixRight.npy')
rectTransL = np.load('C:/Users/localuser/Documents/test/pixel2WCos/calibParams/rotationMatrixLeft.npy' )
rectTransR = np.load('C:/Users/localuser/Documents/test/pixel2WCos/calibParams/rotationMatrixRight.npy')
projMatrixL = np.load('C:/Users/localuser/Documents/test/pixel2WCos/calibParams/projectionMatrixLeft.npy' )
projMatrixR = np.load('C:/Users/localuser/Documents/test/pixel2WCos/calibParams/projectionMatrixRight.npy')
def undistort(image0, image1):
    #The function is simply a combination of initUndistortRectifyMap (with unity R ) and remap (with bilinear interpolation)
    undistorted0 = cv.undistort(image0, camMatrix0, distcoef0, newCamMat0)
    undistorted1 = cv.undistort(image1, camMatrix1, distcoef1, newCamMat1)
    return undistorted0, undistorted1
```

3.4.2 Calculate Disparity

Disparity calculation is the process of determining the horizontal shift (disparity) between corresponding points in the left and right images. This step is fundamental for depth estimation.

Steps:

- **Choose a Stereo Matching Algorithm:** Select a suitable stereo matching algorithm, such as StereoBM or StereoSGBM.
- **Configure Algorithm Parameters:** Set the parameters for the chosen algorithm, such as the number of disparities and block size.

- **Compute Disparity Map:** Apply the stereo matching algorithm to the rectified images to compute the disparity map.

```
# Create Stereo block matching object
stereo = cv.StereoBM_create(numDisparities = 16, blockSize = 15)
#Compurte Disparity map
disparity = stereo.compute(imageL, imageR)
# Normalize the disparity map for visualization
disparityNormalized = cv.normalize(disparity, None, 0,255, cv.NORM_MINMAX)
image = np.array(disparityNormalized, dtype = np.uint8)
```

3.4.3. Calculate Depth

Depth calculation involves converting the disparity values into depth information. This step uses the disparity-to-depth mapping matrix (Q) obtained from the stereo rectification process.

Steps:

- **Use Q Matrix:** Apply the Q matrix to the disparity map to convert disparity values to depth information.
- **Normalize Depth Values:** Normalize the depth values for visualization and further processing.

```
# Reproject disparity map to 3D
points_3D = cv.reprojectImageTo3D(disparity_map, Q)
points_3D = points_3D[mask]
# Normalize depth values
depth_map = points_3D[:, 2]
depth_map = cv2.normalize(depth_map, depth_map, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX)
depth_map = np.uint8(depth_map)
```

3.4.4 Calculate XYZ Coordinates

Calculating the XYZ coordinates involves extracting the 3D coordinates of each point in the scene from the depth information.

Steps:

- **Filter Points:** Filter out points with invalid disparity values to improve the accuracy of the 3D reconstruction.
- **Extract Coordinates:** Extract the X, Y, and Z coordinates from the 3D points.

```
 # Extract X, Y, Z coordinates
xyz_coordinates = points_3D[mask]
# Example usage of XYZ coordinates
for point in xyz_coordinates:
   x, y, z = point
   print(f"X: {x}, Y: {y}, Z: {z}")
```

4. Experimental Setup

4.1 Equipment

The experimental setup for this project involved a stereovision system utilizing the Alvium G1-240m/c cameras. These cameras are equipped with the Sony IMX392 sensor, providing high-resolution images necessary for accurate depth and disparity calculations.

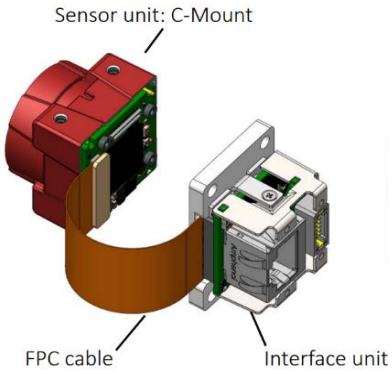


Fig: Alvium G1-240 camera

Camera Specifications:

- Camera Model: Alvium G1-240m/c
- Sensor Model: Sony IMX392
- Resolution: 1936 (H) × 1216 (V); 2.4 MP
- Sensor Type: CMOS
- Shutter Type: Global shutter (GS)
- Sensor Size: Type 1/2.3; 6.7 mm × 4.2 mm; 7.9 mm diagonal
- Pixel Size: 3.45 µm × 3.45 µm
- Relative Humidity: 0% to 80% (non-condensing)
- Digital Interface: 1000BASE-T
- Camera Controls: GenICam (GenICam Access)
- Image Buffer (RAM): 32 MByte
- Non-volatile Memory (Flash): 1024 KByte
- Inputs and Outputs:
    - 1 opto-isolated input
    - 1 opto-isolated output
    - 2 non-isolated GPIOs
- Power Requirements:
    - External power: 12 to 24 VDC
    - Power over Ethernet (PoE): IEEE 802.3af
- Power Consumption (Typical):
    - External power: 3.6 W at 12 VDC
    - Power over Ethernet: 4.0 W

4.2 Setup Configuration

The cameras were securely mounted in a fixed position using a custom-designed camera mount with a baseline of 150mm. This ensured stable and consistent positioning throughout the calibration and imaging processes.

The following diagram illustrates the camera mounting setup:

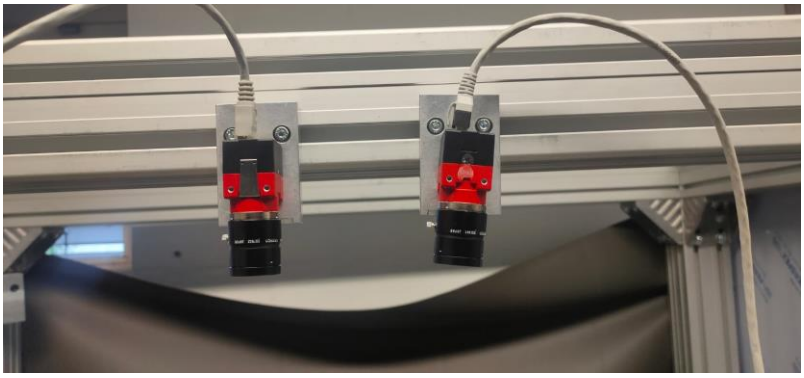


Fig: Camera Set-up

The cameras were connected via Ethernet cables from a PoE (Power over Ethernet) connector, utilizing the 1000BASE-T digital interface to provide a robust and reliable connection for data transfer. This setup allowed for both power and data to be transmitted over a single Ethernet cable, simplifying the installation and ensuring a stable connection.

Accessing the Camera using vmbpy Library

To verify the camera setup and capture images for calibration and processing, the following code demonstrates how to access the camera using the vmbpy library

4.2.1 Streaming Video from Synchronized Cameras

The script below shown initializes the Vimba system, accesses two cameras, and configures their settings. Both cameras are triggered and start streaming video with custom handlers. The cameras are continuously triggered to maintain synchronization, allowing real-time video streaming and processing

```
with VmbSystem.get_instance() as vmb:
    ## get access to the cameras
    print( "test" )
    with get_camera()[0] as camL, get_camera()[1] as camR:
## setup cameras parameters
        setup_camera(camR)
        setup_camera(camL)
        handlerR = Handler()
        handlerL = Handler()
## Trigger the camera with software trigger
        triggerCamera(camL)
        triggerCamera(camR)
## video start streaming
        try:
            camR.start_streaming(handler = handlerR, buffer_count = 10)
            camL.start_streaming(handler = handlerL, buffer_count = 10)
            msg = 'Stream from \'{}\. Press Esc key to stop stream'
            escKey = 27
            counter = 0
            size = (640,480)
            while True:
                ## trigger the cameara with speicified time
                trigger(camR, 0.001)
                trigger(camL, 0.001)
```

4.2.2 Camera Initialization and Configuration

The code below detects connected cameras using the Vimba API and configures their settings. It sets the cameras' exposure and white balance to 'Continuous' and adjusts the packet size for optimal streaming. Robust error handling ensures the script runs smoothly even if certain features are unavailable.

```
def get_camera():
    with VmbSystem.get_instance() as vmb:
        cameras = vmb.get_all_cameras()
        if not cameras:      #If no cameras are detected
            print ("No cameras Detected")
            exit ()
    return cameras
def setup_camera(cam: Camera):
    with cam:
        try:
            cam.ExposureAuto.set('Continuous')
            cam.ExposureTime.set(20000)
        except (AttributeError, VmbFeatureError):
            pass
        try:
            cam.BalanceWhiteAuto('Continuous')
        except (AttributeError, VmbFeatureError):
            pass
        try:
            stream = cam.get_stream()[0]
            stream.GVSPAdjustPacketSize.run()
            while not stream.GVSPAdjustPacketSize.is_done():
                pass
        except(AttributeError, VmbFeatureError):
            pass
```

4.2.3 Camera Trigger Functions

These functions enable software-controlled triggering of camera image capture. triggerCamera(camera) configures the camera for software triggering, setting up continuous acquisition mode, while trigger(camera) executes the software trigger command after a short delay to capture images. These functions are crucial for synchronized and continuous image acquisition in camera systems.

```
def triggerCamera(camera):     #Trigger the camera with software

    camera.TriggerSource.set('Software')
    camera.TriggerSelector.set('FrameStart')
    camera.TriggerMode.set('On')
    camera.AcquisitionMode.set('Continuous')

def trigger(camera, sec):     # Trigger with the specified time
    time.sleep(sec)
    camera.TriggerSoftware.run()
```

4.2.4 Handler Class for Camera Frame Processing

The Handler class manages and processes frames from a camera stream using the Vimba API. It initializes a queue to store frames, processes frames to ensure they are in the correct format for display with OpenCV, and handles continuous frame acquisition by re-queuing processed frames. This setup ensures efficient and smooth handling of real-time video data.

```
class Handler:
    def __init__(self):
        self.display_queue = Queue(10)

    def get_image(self):
        return self.display_queue.get(True)

    def __call__(self, cam:Camera, stream:Stream, frame:Frame):

        if frame.get_status() == FrameStatus.Complete:
            print('{} acquired {}'.format(cam, frame), flush = True)

            if frame.get_pixel_format() == opencv_display_format:
                display = frame
            else:

                display = frame.convert_pixel_format(opencv_display_format)
            self.display_queue.put(display.as_opencv_image(), True)

        cam.queue_frame(frame)
```

4.3 Calibration Procedure

The calibration procedure is a critical step in setting up the stereo vision system, as it ensures that the cameras are accurately aligned and any lens distortions are corrected. This procedure involves determining the intrinsic and extrinsic parameters of each camera, which are then used to rectify and undistort the captured images, facilitating precise depth calculations.

Purpose and Importance of Calibration

Calibration is essential to correct lens distortions, align stereo camera pairs, and obtain accurate depth measurements. Without proper calibration, the 3D reconstruction would be inaccurate, leading to errors in depth estimation and overall system performance. The Detailed Steps for Calibration are

4.3.1 Capture Calibration Images:

Capture multiple images (a minimum of 7 to 8) of a chessboard pattern from different angles and positions for both cameras simultaneously. The number of images ensures a robust calibration by covering various perspectives and reducing the influence of noise.

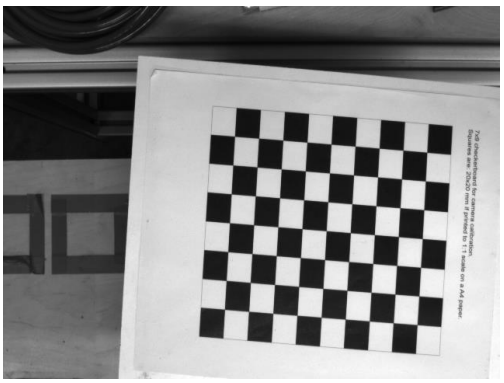


Fig: Left Camera Image

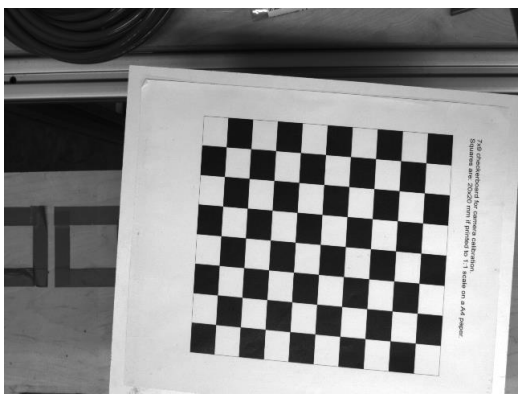


Fig: Right Camera Image

4.3.2 Detect Chessboard Corners:

Use OpenCV's *cv2.findChessboardCorners( )* function to detect the corners in each calibration image.

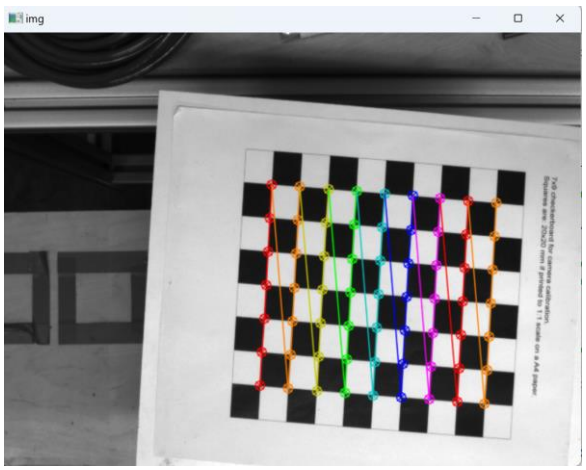


Fig: Left Chessboard detected Image

Fig: Right Chessboard detected Image

4.3.3 Calculate Intrinsic Parameters:

Use *cv2.calibrateCamera( )* to calculate the intrinsic parameters, including the camera matrix and distortion coefficients for each camera.

```
   6.05374696e+01]]
PS D:\Project\26.06.2024> python -u "d:\Project\26.06.2024\test\stereoVision\stereoCaliberate.p
y"

Camera_Matrix of Left Camera:
 [[1.14854703e+03 0.00000000e+00 3.16193995e+02]
 [0.00000000e+00 1.37075805e+03 2.36066505e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

 Distortion coefficients of Left camera:
 [[-1.68306523e-01 -3.76536928e-01  1.19290496e-03 -1.53532173e-03
   2.94195970e+00]]


 Camera_Matrix of Right Camera:
 [[1.14642415e+03 0.00000000e+00 3.22328367e+02]
 [0.00000000e+00 1.36855010e+03 2.48691616e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

 Distortion coefficients of Right camera:
 [[-1.43702810e-01 -3.30738115e+00  1.47874633e-03 -1.24707221e-03
   6.05374696e+01]]
PS D:\Project\26.06.2024>
```

Fig: Camera calibration Output (Intrinsic Parameters of both cameras)

4.3.4 Stereo Calibration:

Performed stereo calibration using *cv2.stereoCalibrate( )* to obtain the rotation and translation between the two cameras, ensuring that the cameras are accurately aligned.

```
New_Camera_Matrix of Left Camera:
 [[1.12578530e+03 0.00000000e+00 3.15618922e+02]
 [0.00000000e+00 1.34248105e+03 2.36330653e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

New_Camera_Matrix of Right Camera:
 [[1.14351458e+03 0.00000000e+00 3.21830151e+02]
 [0.00000000e+00 1.35724116e+03 2.48929215e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]

Rotation Matrix:
 [[ 0.99988394  0.00283046 -0.01496998]
 [-0.00291321  0.99998058 -0.00550913]
 [ 0.01495409  0.0055521   0.99987277]]

Translational Vector:
 [[-0.05216245]
 [-0.00706746]
 [ 0.63555307]]

Essential Matrix:
 [[ 1.74581263e-03 -6.35579964e-01 -3.56521986e-03]
 [ 6.36259344e-01  2.08851633e-03  4.26415944e-02]
 [ 7.21860355e-03 -5.21414293e-02  1.81569931e-04]]

Fundamental Matrix:
 [[ 6.95104063e-07 -2.12211935e-04  4.83347334e-02]
 [ 2.13437661e-04  5.87519336e-07 -5.14001051e-02]
 [-5.00679729e-02  4.82421035e-02  1.00000000e+00]]
PS D:\Project\26.06.2024>
```

Fig: Stereo calibration output

4.3.5 Rectification:

Use *cv2.stereoRectify( )* and *cv2.initUndistortRectifyMap( )* to rectify the images and compute undistortion maps.

```
PS D:\Project\26.06.2024> python -u "d:\Project\26.06.2024\test\stereoVision\stereoCaliberate.p
y"

Projection Matrix of Left Camera :
 [[ 1.34986110e+03  0.00000000e+00 -1.19062787e+03  0.00000000e+00]
 [ 0.00000000e+00  1.34986110e+03  2.64126312e+02  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]

Projection Matrix of Right Camera w.r.t Left Camera:
 [[ 1.34986110e+03  0.00000000e+00 -1.19062787e+03 -8.60845881e+02]
 [ 0.00000000e+00  1.34986110e+03  2.64126312e+02  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]

Disparity to Depth matrix:
 [[ 1.00000000e+00  0.00000000e+00  0.00000000e+00  1.19062787e+03]
 [ 0.00000000e+00  1.00000000e+00  0.00000000e+00 -2.64126312e+02]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.34986110e+03]
 [ 0.00000000e+00  0.00000000e+00  1.56806361e+00 -0.00000000e+00]]
PS D:\Project\26.06.2024>
```

Fig: Output of Stereo rectify function which computes rectification transforms

4.3.6. Store Calibration Parameters:

Save the intrinsic and extrinsic parameters, and stereo calibrate parameters in .npy files for further usage. This allows for easy loading and application of these parameters in the image processing workflow.

5. Experiments

5.1 Experiment_1:

Compute disparity using OpenCV function Stereo_BM

Objective:

The objective of this experiment was to compute the disparity map from a pair of stereo images using the StereoBM (Block Matching) algorithm provided by OpenCV. Additionally, a graphical user interface (GUI) was developed to allow dynamic adjustment of the StereoBM parameters for real-time observation of their effects on the disparity map.

StereoBM Algorithm

The StereoBM algorithm in OpenCV computes the disparity map by comparing blocks of pixels in the left and right images. The key parameters of StereoBM include:

- numDisparities: The maximum disparity minus minimum disparity. The value must be divisible by 16.
- blockSize: The size of the block window. It must be an odd number between 5 and 255.
- preFilterType, preFilterSize, preFilterCap: Parameters for the pre-filtering step.
- textureThreshold: A threshold for texture filtering.
- uniquenessRatio: A margin in the best (minimum) cost function for the best match compared to the next best match.
- speckleWindowSize, speckleRange: Parameters for post-filtering to remove small speckles

Fig: Undistorted Left Image

Fig: Undistorted Right Image

Experiment Procedure:

1. GUI Development: Developed a GUI to adjust StereoBM parameters dynamically.
2. Parameter Adjustment: Varied parameters such as numDisparities, blockSize, and preFilterCap to observe their impact on the disparity map quality.

3. Observations:
    - numDisparities: Increasing improved depth perception range but increased computation time.
    - blockSize: Larger sizes led to smoother maps but reduced edge accuracy.
    - preFilterCap: Adjustments affected intensity normalization, influencing map quality.

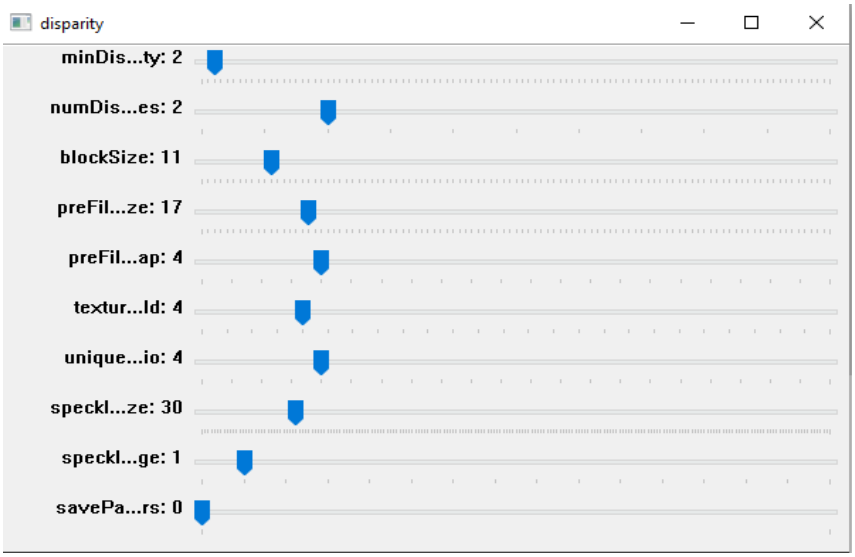


Fig: GUI for Adjusting StereoBM Parameters

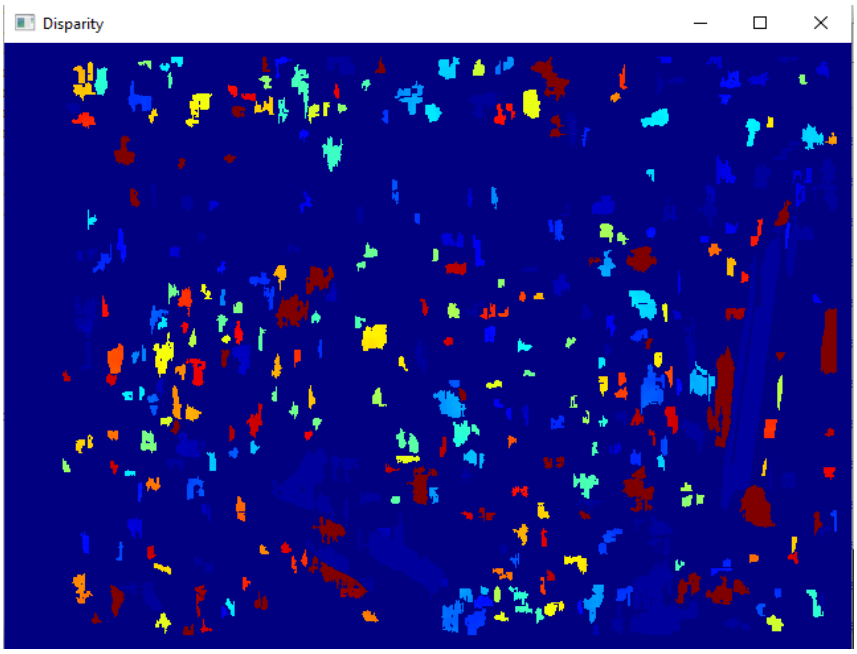


Fig: Disparity Map

Results and Observations: The disparity maps generated using StereoBM showed limitations in handling textureless regions and edge details, often resulting in noisy or incomplete depth maps. While useful for real-time applications due to its efficiency, StereoBM's performance was suboptimal for scenarios requiring high accuracy and robustness. Additionally, despite adjustments through the GUI, the output images primarily displayed noise, particularly in color images, which obscured the disparity information.

5.2 Experiment_2:

Compute disparity using OpenCV function Stereo_SGBM

Objective:

The objective of this experiment was to evaluate the disparity map quality using the 'StereoSGBM' (Semi-Global Block Matching) algorithm and assess its performance with parameter adjustments through a GUI. The experiment aimed to achieve smoother and more accurate disparity maps compared to the StereoBM algorithm.StereoBM Algorithm.

StereoSGBM Algorithm Parameters:

The StereoSGBM algorithm in OpenCV provides several parameters that can be adjusted to fine-tune the disparity map generation. These parameters include:

- minDisparity: Minimum possible disparity value. Usually set to 0.
- numDisparities: The number of disparity values to be considered. It must be divisible by 16.
- blockSize: The size of the block window. Must be an odd number.
- P1: The penalty on the disparity changes by plus or minus 1 between neighbour pixels.
- P2: The penalty on the disparity changes by more than 1 between neighbour pixels.
- disp12MaxDiff: Maximum allowed difference in the left-right disparity check.
- preFilterCap: The maximum value of the pre-filtered image pixels.
- uniquenessRatio: Margin in percentage by which the best (minimum) computed cost function value should "win" the second-best value to consider the found match correct.
- speckleWindowSize: Maximum size of smooth disparity regions to consider their noise speckles and invalidate.
- speckleRange: Maximum disparity variation within each connected component.

Fig: Undistorted Images from Left camera and Right Camera

Experiment Procedure:

1. **GUI Development**: Created a GUI to adjust StereoSGBM parameters dynamically.
2. **Parameter Adjustment**: Varied parameters such as numDisparities, blockSize, P1, P2, and preFilterCap to observe their impact on the disparity map quality.
3. **Initial Observations**:
    - The disparity maps obtained using StereoSGBM were better than those from StereoBM, exhibiting clearer contours and better object definition.
    - Despite improvements, the maps still contained noise and inaccuracies, particularly around edges and texture less region

Fig: GUI for Adjusting StereoSGBM Parameters

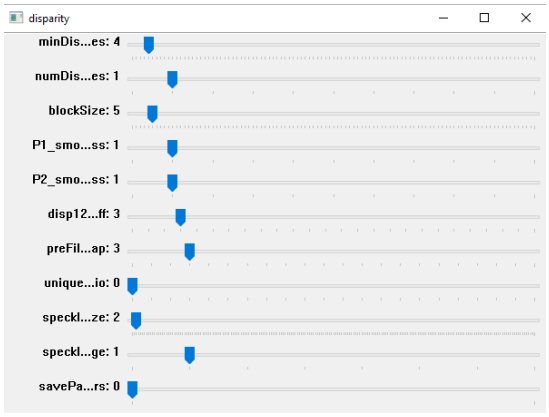


Fig: GUI for Adjusting StereoSGBM Parameters

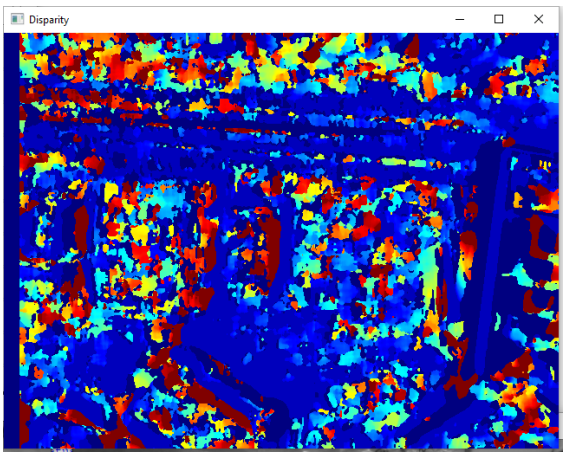


Fig: Disparity map without filter

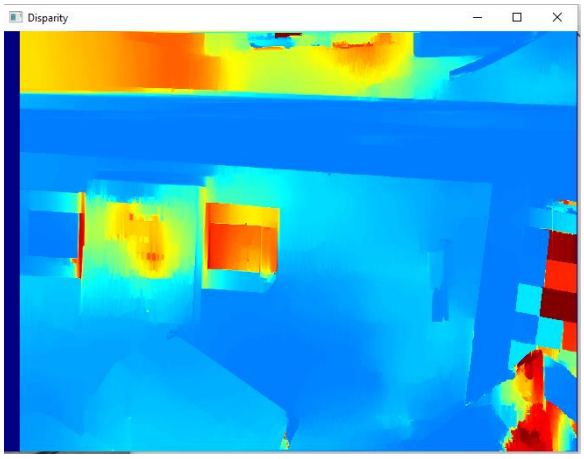


Fig: Disparity Map After Applying WLS Filter

Post-Processing

To address the remaining noise and inaccuracies, post-filters were applied to the disparity maps. A weighted least squares (WLS) filter was used to enhance the clarity and accuracy of the disparity maps.

1. **Application of WLS Filter**: Applied the WLS filter to the disparity maps obtained from the StereoSGBM algorithm.

Results and Observations: The disparity maps still exhibited some noise and inaccuracies, highlighting the necessity of post-processing. The application of the WLS filter was essential for achieving acceptable disparity maps, demonstrating the importance of post-processing steps in disparity map generation.

5.3 Experiment_3:

Computing Disparity Using OpenCV's StereoBM and StereoSGBM with Online Images

Objective:

The objective of this experiment was to evaluate the performance of the 'StereoBM' and 'StereoSGBM' algorithms on online stereo image datasets and compare the results. The aim was to understand the reasons behind the discrepancies in disparity map quality observed in previous experiments and to assess the algorithms' performance with standard stereo datasets.

Procedure

1. Selection of Online Images: Selected an online stereo image dataset, specifically the tsukuba.png image pair, known for its use in disparity map testing.
2. Disparity Calculation: Computed disparity maps using both the StereoBM and StereoSGBM algorithms.
3. Parameter Adjustment: Utilized the GUI developed in previous experiments to fine-tune the parameters for both algorithms to achieve optimal results.

Fig: Left camera Image

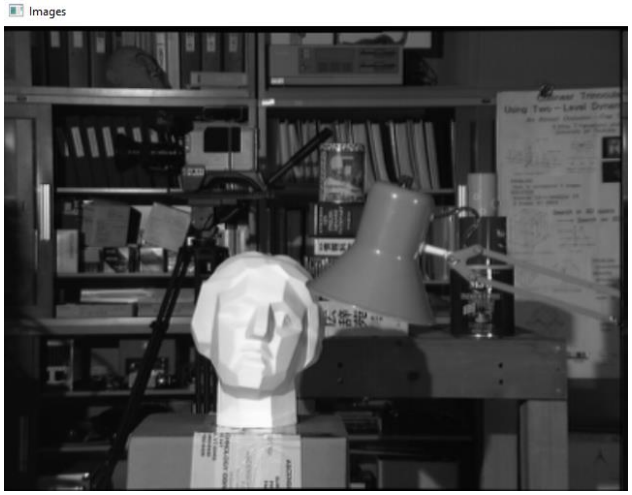


Fig: Right Camera Image

StereoBM Algorithm Results

- **Disparity Map Quality**: The disparity map generated using the StereoBM algorithm showed noticeable improvements compared to previous experiments with local images.
- **Observation**: The disparity map was clearer, with defined contours and objects. The results were more satisfactory and demonstrated the algorithm's effectiveness when applied to well-calibrated and high-quality stereo image pairs.

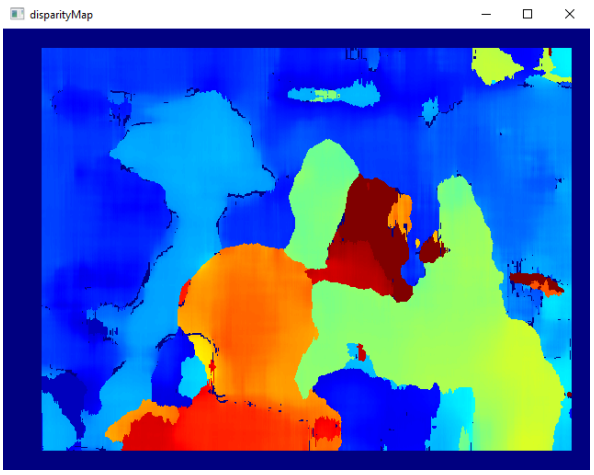


Fig: Disparity Map

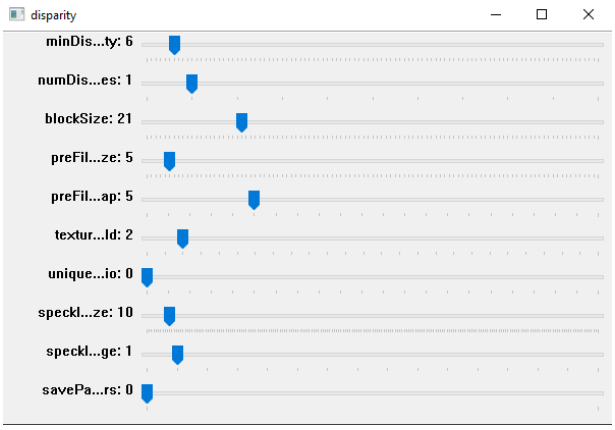


Fig: GUI for Adjusting StereoBM parameters

StereoSGBM Algorithm Results

- **Disparity Map Quality**: The disparity map obtained using the StereoSGBM algorithm was superior to that from the StereoBM algorithm. The map exhibited smoother transitions and fewer artifacts.
- **Observation**: The disparity map quality was significantly enhanced post-application of the WLS filter, providing a more accurate and visually appealing result.

Conclusion

This experiment demonstrated that both StereoBM and StereoSGBM algorithms can produce good disparity maps when high-quality stereo images are used. The StereoSGBM algorithm, especially when combined with post-processing filters like WLS, provided superior results compared to StereoBM. The findings highlight the critical role of image quality and post-processing in achieving accurate and reliable disparity maps. Future work will involve exploring additional online datasets and further refining the algorithms and post-processing techniques to enhance disparity map quality.

5.4 Experiment_4:

Compute disparity using Projector and Gaussian filtered Images

Objective

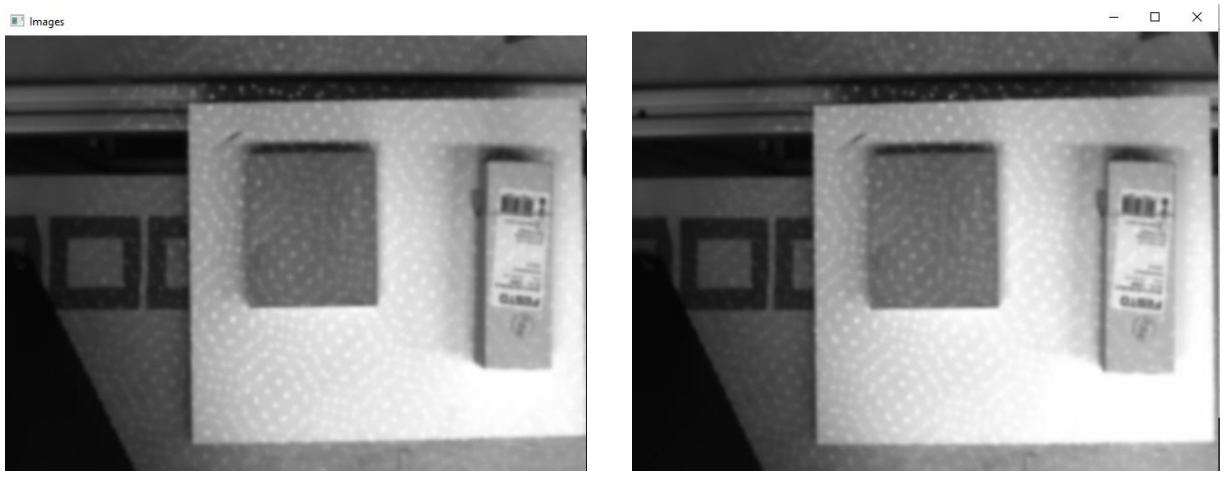
The objective of this experiment was to further improve the quality of disparity maps by using live camera images with additional preprocessing steps. The experiment aimed to evaluate the performance of the StereoBM and StereoSGBM algorithms when using a projector to enhance feature detection and applying Gaussian filtering to reduce noise.

Procedure

1. **Setup**: A projector was used to project a structured light pattern onto the scene to enhance feature detection for disparity calculation.
2. **Image Capture**: Images were captured from live cameras.
3. **Pre-Processing**:

    - Applied Gaussian filtering to the captured images to reduce noise and improve the quality of the input for disparity calculation.
    - Undistorted the images to correct any lens distortion.

4. **Disparity Calculation**: Computed disparity maps using both the StereoBM and StereoSGBM algorithms.
5. **Parameter Adjustment**: Used the GUI developed in previous experiments to adjust parameters dynamically for real-time observation of their effects on the disparity map



Undistorted Images from Left Camera and Right Camera

StereoBM Algorithm: The disparity map generated using the StereoBM algorithm showed improvements compared to earlier experiments. The map had clearer contours and better-defined objects, though some noise and artifacts were still present.

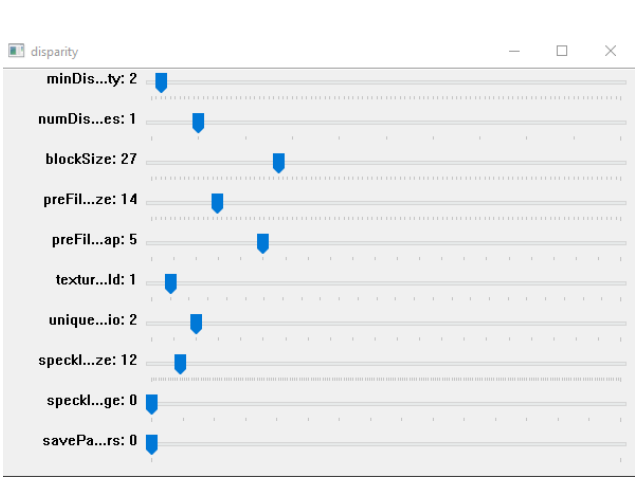


Fig: GUI

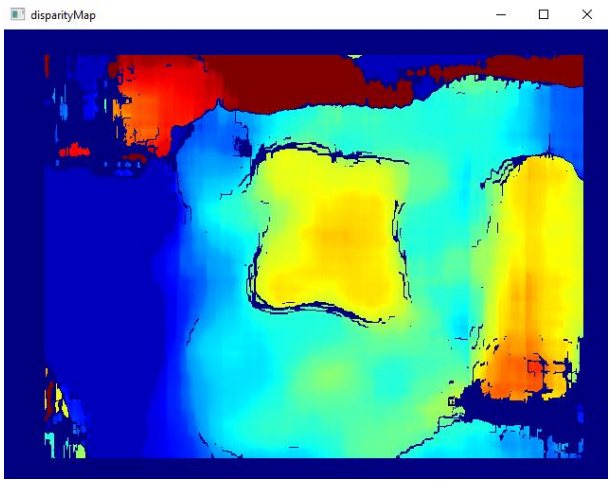


Fig: Disparity map

StereoSGBM Algorithm: The disparity map obtained using the StereoSGBM algorithm exhibited smoother transitions and fewer artifacts compared to the StereoBM algorithm. The use of the projector and Gaussian filtering contributed to the enhanced quality of the disparity map.

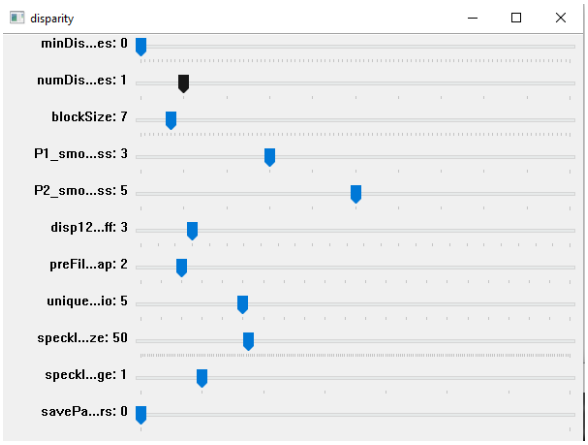


Fig: GUI for StereoSGBM

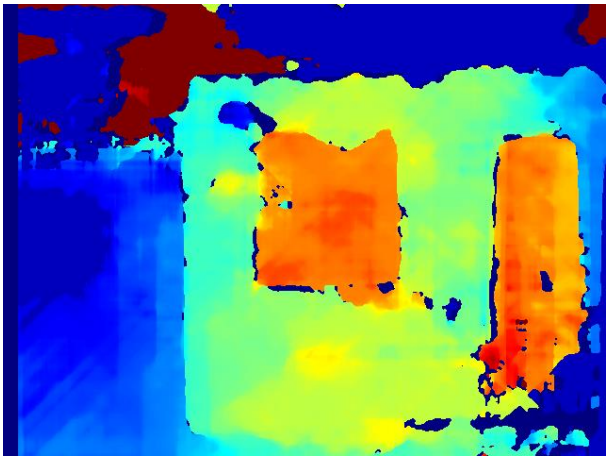


Fig: Disparity Map

Result:

This experiment demonstrated that using a projector to project structured light patterns and applying Gaussian filtering to captured images significantly improves the quality of disparity maps. Both StereoBM and StereoSGBM algorithms benefited from these enhancements, with StereoSGBM providing superior results

5.5 Experiment_5:

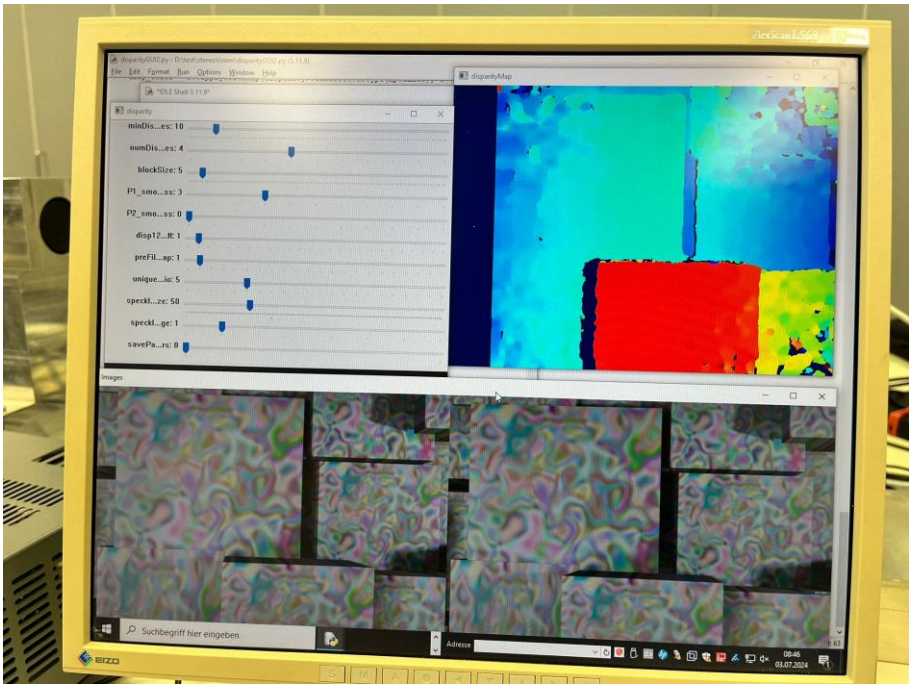
Compute disparity using OpenCV function StereoSGBM with Blender Images

Objective

The objective of this experiment is to evaluate the performance of the StereoSGBM algorithm using synthetic stereo images generated in Blender. These images come with known ground truth disparity, allowing for a precise comparison between the computed disparity maps and the actual disparity.

Procedure

1. **Blender Image Creation**: Synthetic stereo images were created using Blender. These images included known ground truth disparity values for accurate evaluation.
2. **Disparity Calculation**: Disparity maps were computed using the StereoSGBM algorithm



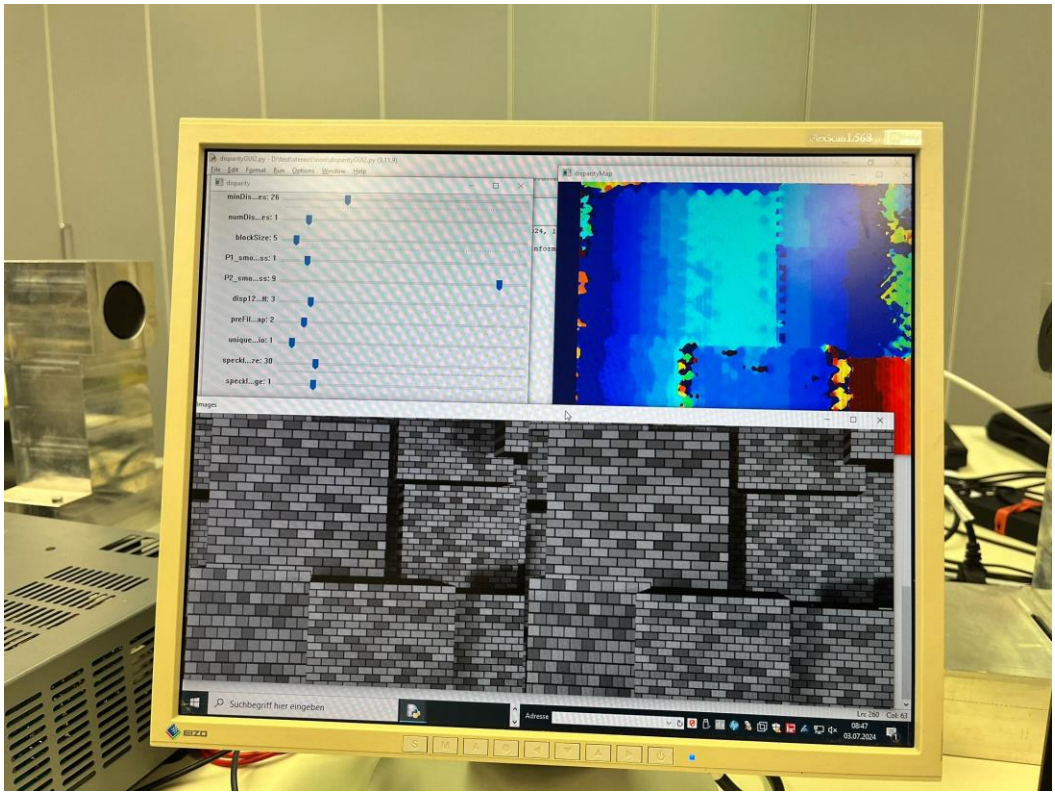


Fig: Disparity map with Blender Images

Result: The disparity maps generated from Blender images were more accurate than those from real-world images. The smooth, noise-free nature of the synthetic images contributed to this increased accuracy. The computed disparity maps showed clear contours of objects, indicating that the StereoSGBM algorithm performs well with high-quality input images.

6. Future Work

The results from Experiments 4 and 5 lead to several key areas for future work to further improve the accuracy and quality of disparity maps:

1. **Advanced Pre-Processing Techniques**: Future work should focus on incorporating advanced pre-processing techniques such as adaptive filtering, edge-preserving smoothing, and deep learning-based denoising to further improve the input image quality.
2. **Enhanced Calibration**: Improving camera calibration procedures will help ensure more accurate undistortion and alignment of stereo image pairs, leading to better disparity maps.
3. **Real-Time Optimization**: Optimizing the algorithms and the GUI for real-time performance will facilitate dynamic adjustments and immediate feedback on disparity map quality, enhancing the overall workflow.
4. **Post-Processing Enhancements**: Applying advanced post-processing techniques such as bilateral filtering or deep learning-based disparity refinement methods will help reduce artifacts and improve the accuracy of disparity maps.
5. **Integration with Synthetic Data**: Utilizing synthetic data with known ground truth for training and evaluation can help fine-tune the algorithms for better performance in real-world scenarios.
6. **Hybrid Approaches**: Combining multiple algorithms and techniques, such as integrating elements of both StereoBM and StereoSGBM, can leverage the strengths of each approach to produce more accurate and robust disparity maps.

References:

1. Hartley, R., & Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press.
2. Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer.
3. Zhang, Z. (2000). A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11), 1330-1334.
4. OpenCV's `calibrateCamera` and `stereoCalibrate` functions.
5. Scharstein, D., & Szeliski, R. (2002). A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1-3), 7-42.
6. Hirschmuller, H. (2008). Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2), 328-341.
7. OpenCV Documentation: Comprehensive resource for the functions and algorithms used in this project, including StereoBM and StereoSGBM.
8. Blender Documentation: Essential for creating and understanding the synthetic images with known ground truth disparities used in Experiment 5.
9. Semi-Global Matching and Mutual Information Matching for Real-Time Stereo Vision: Hirschmuller, H. (2007). A seminal paper on the StereoSGBM algorithm, providing the theoretical foundation for its use.
10. Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering: Dabov, K., Foi, A., Katkovnik, V., Egiazarian, K. (2007). Discusses advanced image filtering techniques, useful for pre-processing steps like Gaussian filtering.
11. Disparity Map Post-Processing Using Weighted Least Squares: A study on the WLS filter, which was crucial for refining disparity maps in your experiments.
12. Hartley, R., Zisserman, A. (2004). Multiple View Geometry in Computer Vision: A comprehensive textbook on the geometric principles underlying stereo vision and disparity map computation.
13. Brown, M. Z., Burschka, D., Hager, G. D. (2003). Advances in Computational Stereo: A review of computational stereo techniques, providing context and background for the algorithms used.
14. Scharstein, D., Szeliski, R. (2002). A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms: A foundational paper evaluating various stereo correspondence algorithms.
15. Camera Calibration Toolbox for Matlab: A valuable resource for understanding camera calibration, a critical step in your experiments.
16. Zhang, Z. (2000). A Flexible New Technique for Camera Calibration: This paper outlines the camera calibration method used in many computer vision applications.