3D Geometry Reconstruction Using Stereo Vision

VIBOT - Master's in Computer Vision and Robotics

Universit´e de Bourgogne, France

Batch 2024 - 2026

Submitted By:

Malik Danial Ahmed

Ureed Hussain

Hassaan Ahmed

Submitted in Year - 2025

Submitted To:

Yohan Fougerolle

Contents

List of Figures

1 Probelm Statement

3D geometry reconstruction is a fundamental task in computer vision and robotics, enabling the estimation of a scene's three-dimensional structure using multiple images. The objective of this project is to develop a portable and low-cost software system for 3D reconstruction using two USB cameras connected to a standard computer. The system should be capable of capturing stereo images, processing them, and generating a 3D point cloud.

1.1 Key Challenges

- Accurate camera calibration to determine intrinsic and extrinsic parameters.[1]
- Proper stereo rectification to align epipolar lines.[2]
- Robust feature detection and matching. [3] [4]
- Precise estimation of camera pose and fundamental matrices.
- Reliable triangulation to reconstruct 3D points.[5] [6]
- Effective post-processing for noise reduction and refinement.

To address these challenges, two different setups were considered:

1. **Single-Camera Stereo Setup:** A single camera was used to capture two images of the same scene. The camera was translated by 12 cm (with no rotation) between the captures. This simulates a stereo vision system using only one camera.
2. **Dual-Camera Stereo Setup:** Two cameras were used to capture images simultaneously, providing a true stereo input.

In the following sections, each step of the implementation will be described in detail, including the mathematical foundations, the reasoning behind each function, potential improvements, and validation methods.

2 Methodology

The following flowchart illustrates the workflow of our project by Two cameras. The same process is followed for Approach 1 using a single camera, with the exception that stereo calibration is not required in that case, as the rotation and translation matrices are already known. The remaining steps are identical to those used in the two-camera setup.

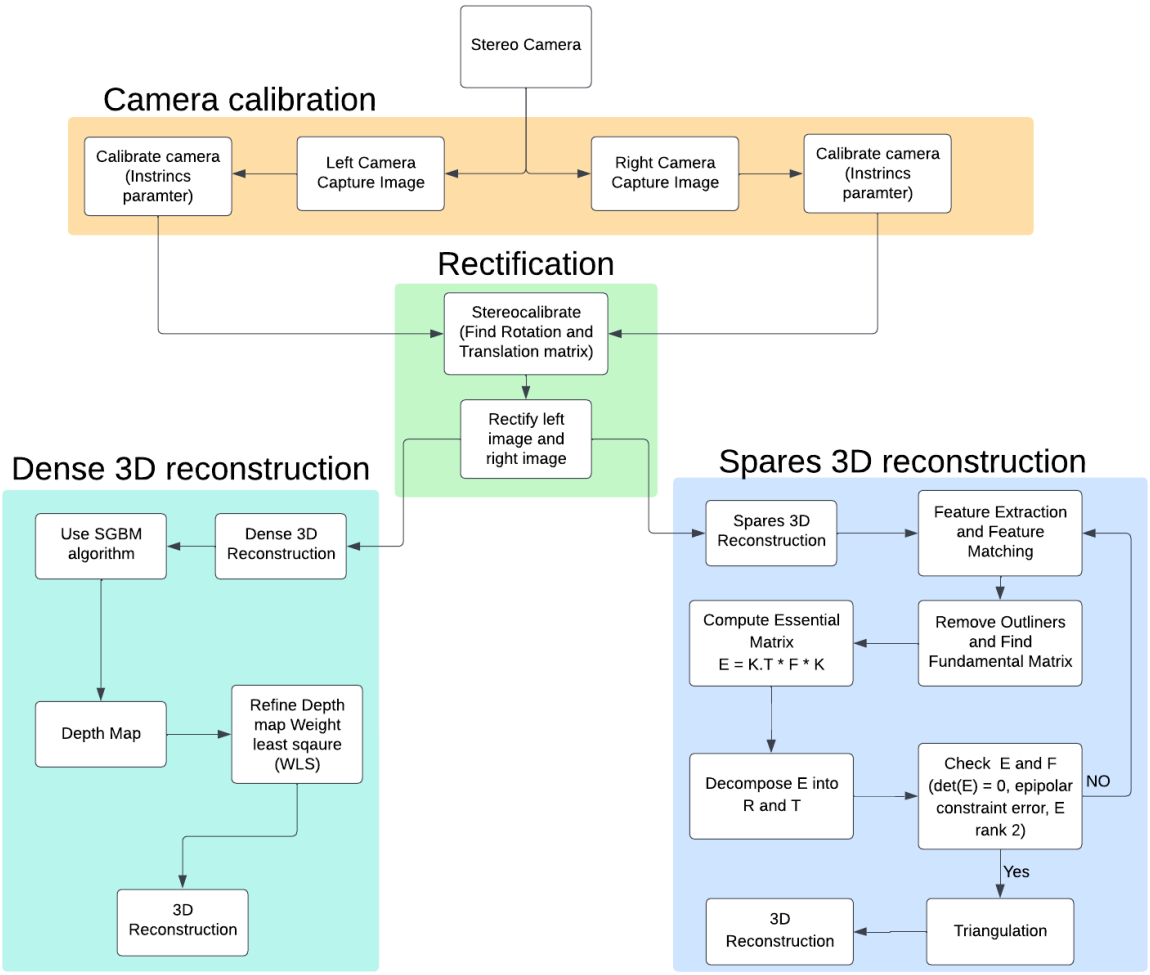


Figure 1: Flow-Chart

3 APPROACH 1: SINGLE-CAMERA STEREO (TRANSLATIONAL MOVEMENT)

3.1 Problem Setup

For 3D reconstruction using a single camera, we first calibrate the camera, ensuring the intrinsic parameters remain the same. We capture one image of the screen and then capture a second image with a 12 cm translation, ensuring there is no rotation between the two captures. The 3D reconstruction is then performed using both sparse and depth-based methods.

3.2 Solution Steps

3.2.1 Camera Calibration (Common Calibration Patterns)

Camera calibration plays a crucial role in 3D reconstruction. In OpenCV, there are several types of patterns available for calibration, as listed in Table 1. For our project, we chose the chessboard pattern due to its compatibility with OpenCV's detection and refinement functions. OpenCV supports several calibration patterns:

Pattern Type	**Detection Method**	**Advantages**	**Disadvantages**
Checkerboard	`findChessboardCorners()`	Simple, robust to lighting	Requires complete visibility
Asymmetric Circles	`findCirclesGrid()`	Rotation-invariant	More sensitive to focus
Charuco Board	`aruco` module	Combines markers + chessboard	More complex setup

Table 1: Common Calibration Patterns

Data Collection: We began by capturing 10-15 images of the chessboard pattern placed on a flat surface.
Code Explanation: The first step was to count the internal corners of the chessboard pattern by specifying its width and height. To detect the corners in each image, we used the cv2.findChessboardCorners function. This function applies the Harris Corner Detection algorithm to find the corners, and we stored these corner coordinates in the variable imagePoints.
Next, we used the cv2.calibrateCamera[2] function to compute the intrinsic matrix and distortion coefficients. This function requires both the image points (detected from the corners) and the object points (the 3D coordinates of the chessboard pattern in the real world). This function applied Zhang's Calibration Method to perform the calibration. [1]
The code for the calibration process is provided in the GitHub.
Finally, we calculated the reprojection error to evaluate the accuracy of our calibration.
Challenges and Solutions: In the following section, we will describe the challenges we encountered during the calibration process and how we overcame them.

Reference: camera-calibration-principles-and-procedures

Challenges and Solutions

Counting Inside Corners of the Chessboard

- **Challenge:** Miscounting internal intersections of the checkerboard due to confusion between squares and corner points.
- **Solution:** Carefully verified the number of internal intersections and checked chessboard dimensions before passing them to `cv2.findChessboardCorners()`[2], ensuring accurate corner detection.

Capturing Images at the Right Angle

- **Challenge:** Images captured from improper angles led to distortions, causing errors in camera parameter estimation.

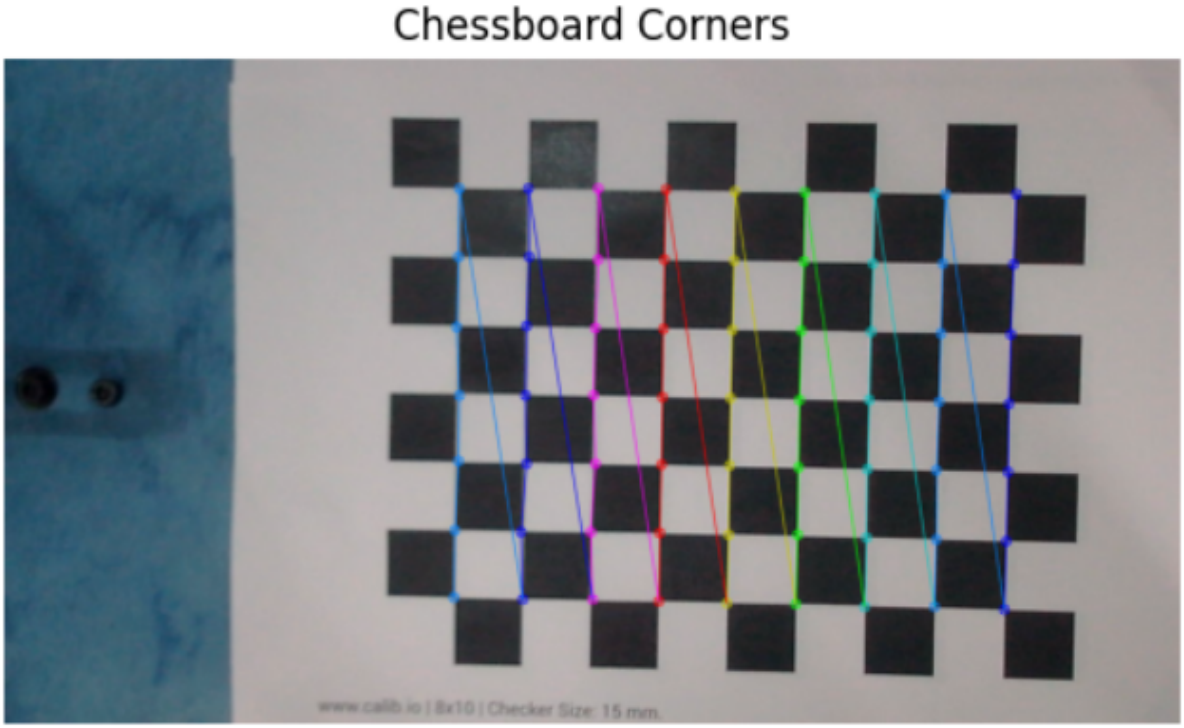
- **Solution:** Fixed the checkerboard securely on a flat surface to maintain a consistent $Z = 0$ reference. Captured images from multiple angles and orientations to enhance calibration accuracy and robustness.

High Reprojection Error

- **Challenge:** Early calibration attempts showed a high reprojection error, indicating inconsistencies in corner detection and dataset quality.
- **Solution:** Eliminated poor-quality images (e.g., blurry or low-angle shots) and ensured good lighting. Recalibrated using a refined dataset to achieve a lower reprojection error.

Ensuring Accurate Calibration

- **Challenge:** Inaccurate calibration led to issues in downstream tasks like 3D reconstruction and distortion correction.
- **Solution:** Continuously monitored the reprojection error and verified the calibration by visually checking undistorted images. Repeated calibration steps until achieving acceptable accuracy before proceeding further.



Camera Calibration Results

Camera Matrix (K):

$$\begin{bmatrix} 1430.57162 & 0 & 675.252736 \\ 0 & 1422.28592 & 375.255856 \\ 0 & 0 & 1 \end{bmatrix}$$

Distortion Coefficients: [0.250151141, -2.15202603, 0.00283665753, -0.000200932362, 6.78654535]

Reprojection Error: 0.1308

Figure 2: Results of Camera Calibration

3.2.2 Rectification

Capture the second image (`image2`). A key assumption here is that there was no rotation during this movement—only translation along the x-axis.
To perform the rectification, we consulted OpenCV's documentation and found that the most suitable function for this task was cv2.stereoRectify(). This function transforms both images so that their epipolar lines are aligned, which is essential for accurate disparity estimation in later stages.
In this function, we passed the camera's intrinsic parameters and distortion coefficients and the Rotation and Translation matrix that we already know. The function then computes new rotation matrices (R1, R2) that align both images into the same plane. Additionally, the new projection matrices (P1, P2) redefine how 3D points project onto the rectified images. The Q matrix, computed during this process, will later be used to convert disparity maps into 3D depth information.
The implemented code for rectification is provided in the GitHub.

Challenges and Solutions in Rectification

Avoiding Camera Rotation

- **Challenge:** Even slight rotation during capture affected rectification accuracy.
- **Solution:** The camera was moved precisely along the x-axis (12 cm) to ensure pure translation without rotation.

Choosing the Correct Rectification Method

- **Challenge:** Multiple OpenCV functions exist for image transformation, making it tricky to pick the right one.
- **Solution:** After reviewing the documentation, `cv2.stereoRectify()` was selected for its suitability in stereo setups.

Dealing with Misalignment and Projection Errors

- **Challenge:** Misalignment due to poor calibration or noisy data led to projection inconsistencies.
- **Solution:** A well-curated image set was used, discarding images with poor alignment or high reprojection error.

Preserving Image Quality

- **Challenge:** Rectification and remapping can degrade image quality.
- **Solution:** High-resolution images were used, and cv2.INTER LINEAR interpolation was applied to minimize quality loss.

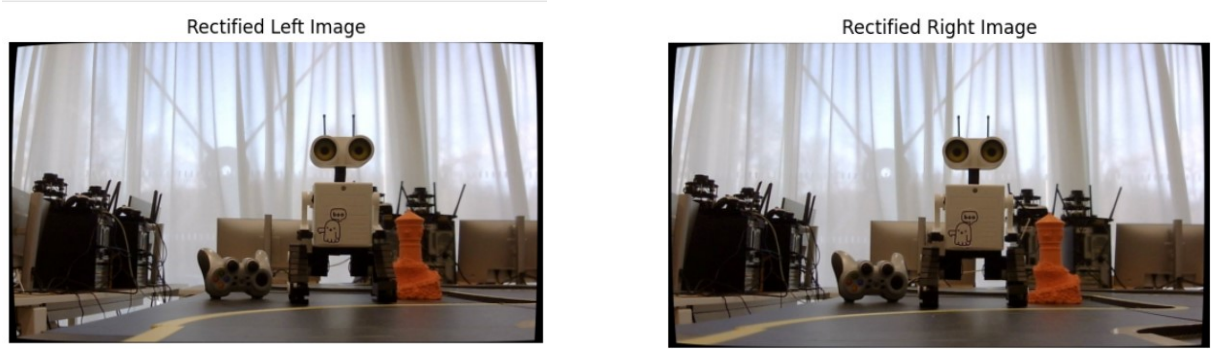


Figure 3: Results of Rectification

Epipolar Lines Visualization:

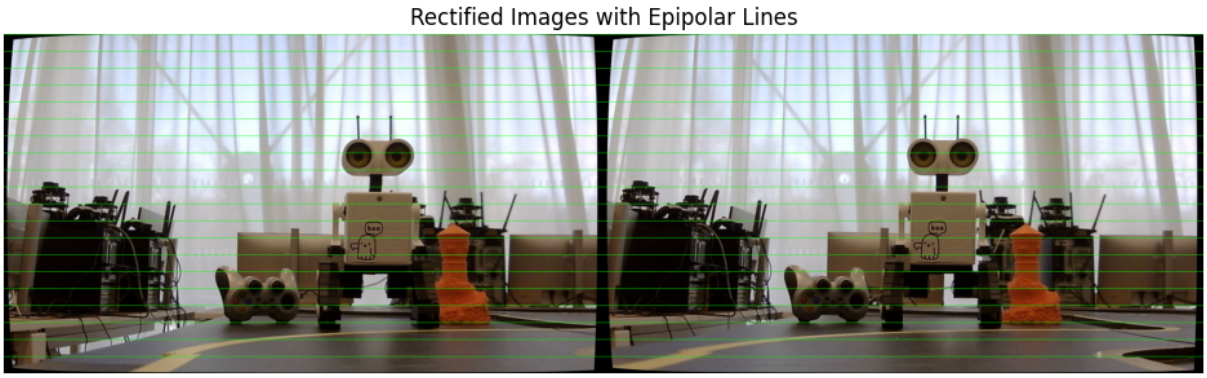


Figure 4: Epipolar Lines Visualization on Rectified Images

3.3 Sparse 3D Reconstruction

3.3.1 Feature Detection and Matching

The next step in our 3D reconstruction pipeline is feature detection and matching, which is both a critical and challenging task. The main difficulty lies in determining which feature detector and matcher pair is best suited for our specific case.
After thoroughly reading two helpful resources — Towards Data Science and Medium Blog by Cristina Popovici [5] — as well as the OpenCV documentation, [7] we narrowed our options down to three suitable detectors: **AKAZE**[3], **SIFT**[8], and **ORB**[4].
To evaluate which performs best with our data, we implemented all three algorithms. One important insight we gained is that choosing the correct matcher for each detector is crucial for obtaining accurate results. The code for all three implementations is provided in the appendix.
After testing, we selected AKAZE due to its good balance between speed and accuracy on our dataset. Feature matching was performed using Brute-Force Matcher with the Hamming norm, which is compatible with AKAZE's binary descriptors. We also fine-tuned the matching threshold iteratively to minimize false correspondences and improve matching stability between the stereo image pair.

Summary of Feature Detection and Matching Techniques:

- **AKAZE:** Fast, real-time performance with binary descriptors.
  AKAZE OpenCV Tutorial
- **SIFT:** Robust but slow; high computational cost.
  SIFT on GeeksforGeeks
- **ORB:** Lightweight, rotation-invariant, and efficient.
  ORB OpenCV Tutorial

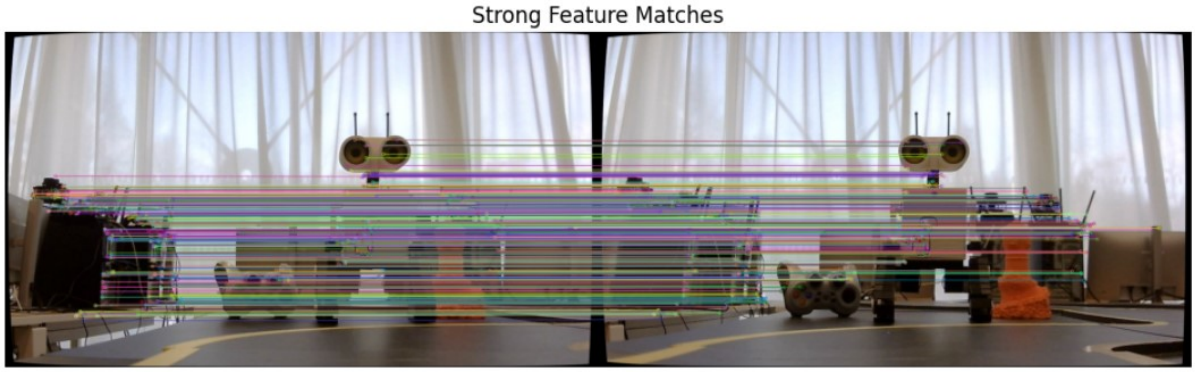
If this system were to be extended for broader use, we suggest adding support for **user-defined feature detector selection and threshold control**, as optimal configurations can vary across applications.

3.3.2 Outlier Removal and Fundamental Matrix Estimation

After feature detection and matching, the next crucial step is to remove outliers and estimate the fundamental matrix, which defines the epipolar geometry between the two views.
To filter out incorrect or noisy matches, we used the well-known RANSAC (Random Sample Consensus) algorithm. RANSAC is robust against outliers and helps in identifying a consistent set of feature correspondences that align with a valid geometric model.
We implemented this step using OpenCV's function:

cv2.findFundamentalMat()

This function estimates the fundamental matrix from the set of matched keypoints while internally applying RANSAC to discard mismatched points. The output includes the estimated fundamental matrix and a mask that indicates which matches were considered inliers.
This step is essential for the next phases, such as computing the essential matrix and camera pose recovery, ensuring that only geometrically consistent matches are used.



Fundamental Matrix

Total matches found: 836
Total strong matches after filtering: 492
Fundamental Matrix =

$$\begin{bmatrix} -5.00186320 \times 10^{-8} & -7.08535095 \times 10^{-6} & 3.48674030 \times 10^{-3} \\ 7.93311714 \times 10^{-6} & 3.99541918 \times 10^{-6} & -2.33413925 \times 10^{-2} \\ -3.70426825 \times 10^{-3} & 1.90617086 \times 10^{-2} & 1 \end{bmatrix}$$

Figure 5: Visualize Feature Matches After Filtering:

3.3.3 Computing the Essential Matrix and Decomposing it into Rotation and Translation

After computing the **Fundamental Matrix (F)**, we calculated the **Essential Matrix (E)** using the camera's intrinsic matrix (K) with the following relation:

$$E = K^T F K \tag{1}$$

To extract **rotation (R)** and **translation (t)**, we performed **Singular Value Decomposition (SVD)** on the Essential Matrix:

$$\text{U, S, Vt} = \text{np.linalg.svd(E)}$$

The translation vector **t** was then extracted directly from the decomposition results:

$$\text{t} = \text{U[:, 2]}$$

This translation vector represents the relative displacement between the two camera positions and is crucial for reconstructing 3D points in space.

Essential Matrix and Rotation Matrices

Essential Matrix:

$$\begin{bmatrix} -1.02364889e-01 & -1.44164351e+01 & 1.13608222e+00 \\ 1.61413696e+01 & 8.08232242e+00 & -2.34467091e+01 \\ -1.08880062e+00 & 2.24388509e+01 & 1.77081984e-03 \end{bmatrix}$$

Possible Rotation Matrices:

$$\begin{bmatrix} 0.38817409 & -0.00898396 & 0.92154227 \\ 0.14921747 & -0.98614533 & -0.07246748 \\ 0.90942565 & 0.16564021 & -0.38145551 \end{bmatrix}$$

$$\begin{bmatrix} 0.9956282 & 0.07605321 & 0.05422535 \\ -0.08210167 & 0.98939826 & 0.11979319 \\ -0.04453981 & -0.12372147 & 0.9913169 \end{bmatrix}$$

Translation Vector:

$$\begin{bmatrix} 0.8472794 \\ 0.0410939 \\ 0.52955538 \end{bmatrix}$$

Figure 6: Essential Matrix and Rotation Matrices with Translation Vector

3.3.4 Test the Essential Matrix, Fundamental Matrix

This verification step was necessary to ensure that the **Essential Matrix** (E), the candidate **Rotation Matrices** (R_1, R_2), and the matched features were correct. We checked the following:

1. Verified that the Essential Matrix satisfied its theoretical constraints.
2. Confirmed the validity of the rotation matrices.
3. Calculated the epipolar constraint error to evaluate how well the matched features adhered to the expected geometric relationship between the two views.

Output:

- Essential Matrix E was validated by checking if it satisfied the singularity constraint: $\det(E) = 0$, ensuring it represented a valid epipolar geometry.
- Rotation matrices R_1 and R_2 were validated by ensuring they were proper orthogonal matrices (i.e., $R^T R = I$ and $\det(R) = 1$).
- The epipolar constraint error was computed and found to be within acceptable limits, confirming that the matched feature points adhered to the expected epipolar geometry.

Essential Matrix Results

Singular values of E:

$$[3.20264290e+01, 2.37433369e+01, 6.94092028e-15]$$

Essential matrix has the correct rank: 2
Det(E):

$$8.237717708962974e-13$$

Essential matrix satisfies det(E) = 0 condition.
Det(R1):

1.0000000000000009, Det(R2): 1.0000000000000009

Deviation of R1 from Identity:

2.821144002567274

Deviation of R2 from Identity:

0.21751614679610887

Mean Epipolar Constraint Error:

0.037000004225239407

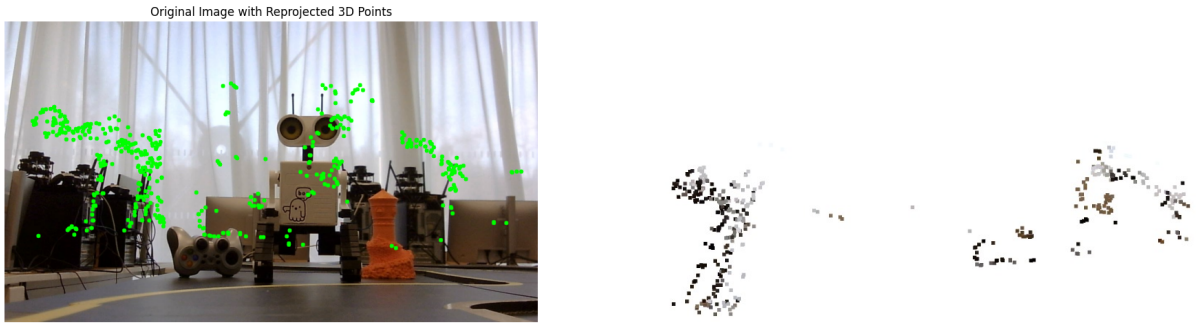
Figure 7: Essential Matrix Results and Evaluation Metrics

If any validation checks fail—whether it's the rank of the Essential Matrix, determinant conditions, validity of the rotation matrices, or the epipolar constraint error—we return to the feature matching step and adjust the threshold to improve match quality. After refining the matches, we recompute the Fundamental Matrix (F), derive the Essential Matrix (E), and revalidate all conditions. This iterative process continues until all constraints are satisfied. Ensuring correctness at each stage is critical, as any error in feature matching propagates through the entire 3D reconstruction pipeline, leading to incorrect depth estimation and faulty 3D structure.

3.3.5 Triangulation for 3D Reconstruction

After validating the Essential Matrix and ensuring the correct Rotation and Translation, we moved on to triangulation. We constructed two projection matrices based on the intrinsic matrix and the computed extrinsics. Using `cv2.triangulatePoints()`, we calculated 3D points from the matched features between the two images. To ensure validity, we filtered out points with negative depth values. This process was repeated with multiple image pairs, refining the feature matching as needed to improve the accuracy of the 3D reconstruction.

3D Point Reprojection and Visualization After triangulation, we obtained a set of 3D points (X, Y, Z). The next step involved reprojecting these points onto the original images to verify the accuracy of the 3D reconstruction. This process helps to check how well the reconstructed points align with the actual image features.



(a) 3D Point Reprojection and Visualization on Original Image

(b) 3D Point Cloud

Figure 8: Left image shows points in 3d space with original colors

3.4 APPROACH 1: SINGLE-CAMERA STEREO (TRANSLA- TIONAL MOVEMENT)
Dense 3D Reconstruction Using StereoBM and Disparity Map

After completing sparse 3D reconstruction through feature-based triangulation, we proceeded to dense 3D reconstruction, which aims to estimate depth for every pixel in the image, not just the matched feature points. This requires the generation of a high-quality disparity map, which is only possible after proper rectification of stereo images.

3.4.1 Using StereoBM — Initial Attempt

As a starting point, we used StereoBM (Block Matching), an OpenCV algorithm designed for computing disparity maps from rectified stereo pairs. However, despite refining and tuning its parameters (such as block size, number of disparities, and uniqueness ratio), StereoBM did not yield satisfactory results. The disparity maps it produced were noisy and lacked accuracy, especially in low-texture or edge regions. These limitations made it unsuitable for reliable depth estimation in our case.

To improve your disparity map, try adjusting the following parameters:

- **Adjust numDisparities and blockSize**: Modify `numDisparities` based on the scene depth:

  numDisparities = 16 * 6 Experiment with values like 16*4, 16*6, etc. blockSize = 15 Try values like 9, 11, 15, 17 stereo = cv.StereoBM$_c$*reate*($numDisparities = numDisparities, blockSize = blockSize$)

  - Larger `numDisparities` helps for farther objects.
  - Larger `blockSize` reduces noise but may blur fine details.

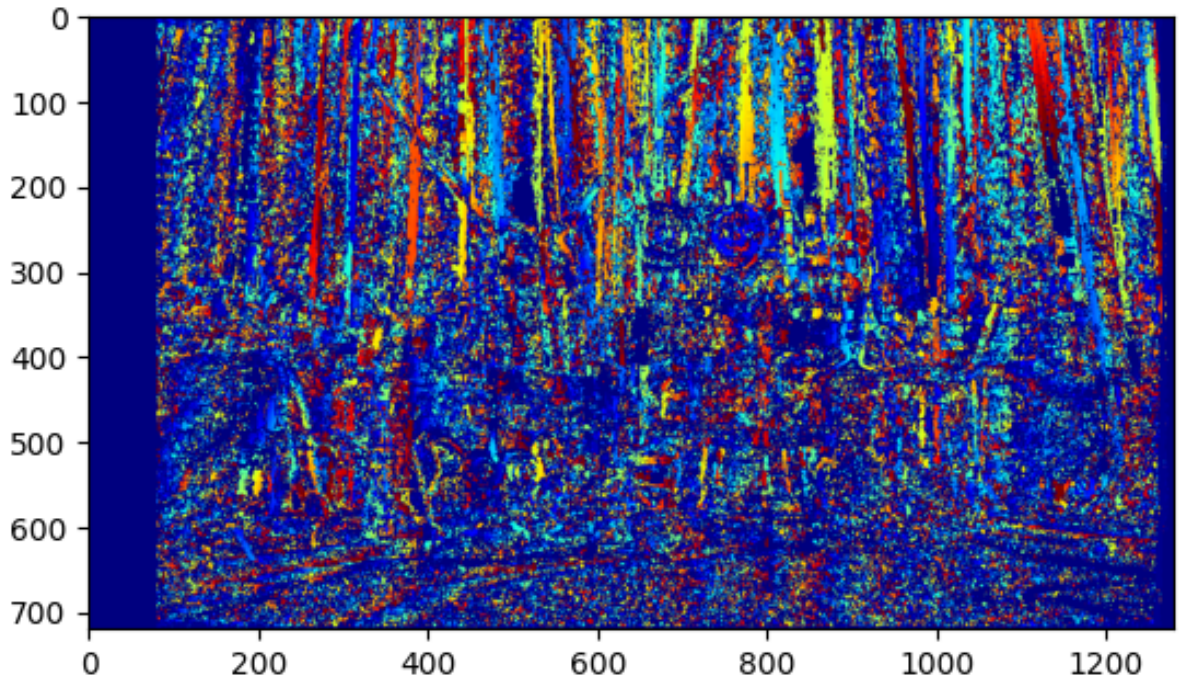


Figure 9: Disparity Map by StereoBM function

3.4.2 Improving Disparity Mapping with StereoSGBM

After initially using StereoBM for disparity mapping, we encountered issues with noise and inaccurate depth estimation. To address this, we explored StereoSGBM (Semi-Global Block Matching), which

refines depth estimation by considering pixel intensities over a larger region, reducing noise. By studying a detailed YouTube explanation[9], OpenCV documentation[7], and a forum discussion on SGBM parameter adjustments, we implemented StereoSGBM effectively. This improvement significantly enhanced the quality of the disparity map, leading to more accurate dense 3D reconstruction.

Advantages of StereoSGBM for Disparity Mapping

- **Better Accuracy:** StereoSGBM improves depth estimation by considering global smoothness constraints.
- **Reduced Noise:** The algorithm applies speckle filtering and uniqueness constraints to eliminate invalid disparities.
- **Higher Detail:** StereoSGBM's 3-way dynamic programming mode refines disparity calculations for greater detail.
- **Improved 3D Point Cloud:** Reprojecting the disparity map into 3D space using the Q matrix results in a more accurate and realistic reconstruction.

This approach offers a dense 3D reconstruction with significantly enhanced depth perception, making it a valuable improvement over traditional StereoBM methods.

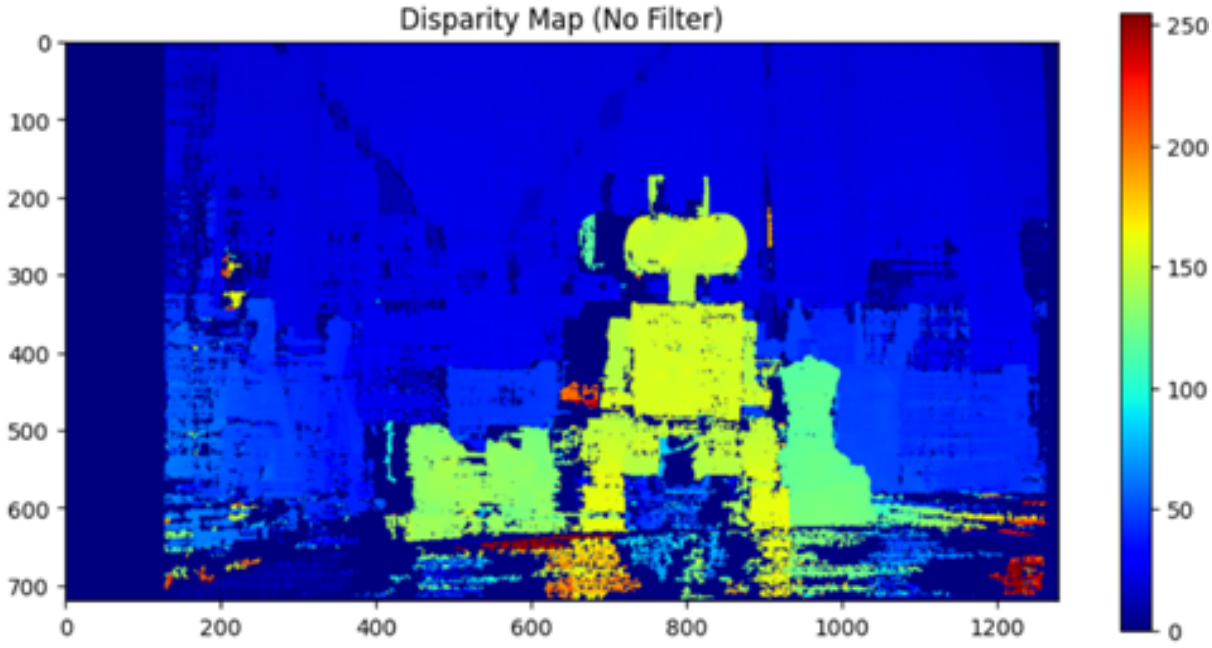


Figure 10: Disparity Map (No Filter)

We initially used the default parameters of the StereoSGBM algorithm and obtained reasonably good disparity maps. However, a significant issue was the loss of detail in the front planar region of the scene. We suspected two potential causes:
(1) inaccurate camera calibration, possibly due to poor-quality chessboard images, and **(2)** suboptimal parameter settings in the StereoSGBM configuration.
To address this, we manually examined and adjusted each parameter of the algorithm. After several experiments, we achieved a notable improvement in the disparity map—though not perfect, it was significantly better than the initial results. The improvement stemmed from a combination of enhanced camera calibration (e.g., avoiding shiny surfaces) and careful parameter tuning.

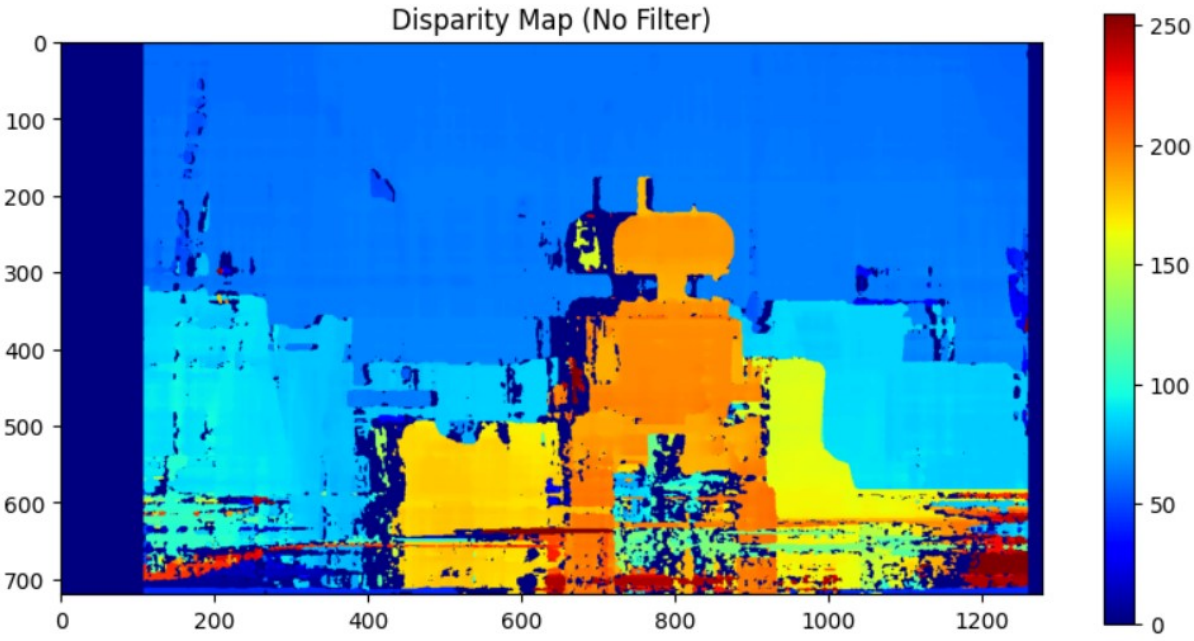


Figure 11: Disparity Map after Tuning the StereoSGBM Parameters

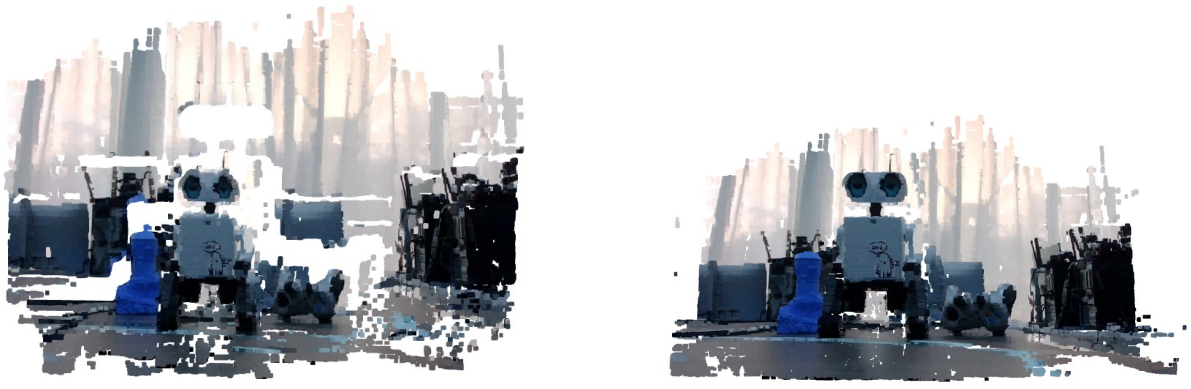


Figure 12: 3D View above figure

The table below explains the role and effect of each key StereoSGBM parameter:

Parameter	Value	Description and Effect
minDisparity	-20	Minimum possible disparity value. Allows handling of objects slightly closer than the baseline. Can affect front-plane accuracy.
numDisparities	128	Number of disparity levels to search. Must be divisible by 16. A larger range helps detect objects at greater depths.
blockSize	9	Size of the matching window. Larger values smooth the disparity map but reduce detail. Smaller values increase sensitivity to noise.
uniquenessRatio	2	Ensures the best match is significantly better than alternatives. Helps reduce ambiguous or incorrect disparity matches.
speckleWindowSize	2	Filters out small noisy regions (speckles). Higher values remove more speckle noise but risk removing fine detail.
speckleRange	1	Controls the allowed disparity variation in speckle filtering. Smaller values enforce stricter consistency.
disp12MaxDiff	1	Acceptable difference in left-right disparity consistency check. Low values enforce stricter validation.
preFilterCap	2	Clamps image pixel values before matching. Helps reduce noise and improve contrast sensitivity in matching.
P1	Derived	Penalty for disparity change of 1 pixel. Controls smoothness between neighboring pixels. Computed as $16 \times 3 \times \text{blockSize}^2$.
P2	Derived	Penalty for disparity change greater than 1 pixel. Larger than P1, it strongly penalizes disparity jumps. Computed as $64 \times 3 \times \text{blockSize}^2$.
mode	SGBM_3WAY	Uses three-way (left-right, right-left, and center) block matching. Increases accuracy and reduces artifacts in the final disparity map.

Table 2: StereoSGBM Parameter Descriptions and Effects

3.4.3 Weighted Least Squares (WLS) Filtering for Disparity Map Refinement

We applied the Weighted Least Squares (WLS) filter to refine the disparity map. Given that disparity maps from algorithms like StereoSGBM can often be noisy or contain inconsistencies due to occlusions or low-texture regions, the WLS filter was used to smooth the map. The filter was effective in preserving edges while reducing noise, as it considers the confidence (weight) of each disparity measurement.

By leveraging both left and right disparity maps, we implemented a cross-checking mechanism that helped to minimize inconsistencies and artifacts. The result was a refined disparity map that improved the quality of our 3D reconstruction.

Summary of Key WLS Parameters in the Code:

- **Lambda:** Controls the smoothing effect of the WLS filter. Higher values result in more smoothing, reducing noise but potentially blurring edges.
- **SigmaColor:** Determines the strength of edge preservation. Higher values allow the filter to better preserve object boundaries and sharp changes in disparity.
- **Disparity Maps:** The left and right disparity maps (disparity_left and disparity_right) are essential inputs for the WLS filter, enabling it to check for consistency and refine the disparity estimation.

By applying WLS filtering in this context, we achieve a cleaner, more accurate disparity map with reduced noise and better edge preservation, which is crucial for generating high-quality 3D reconstructions from stereo images.

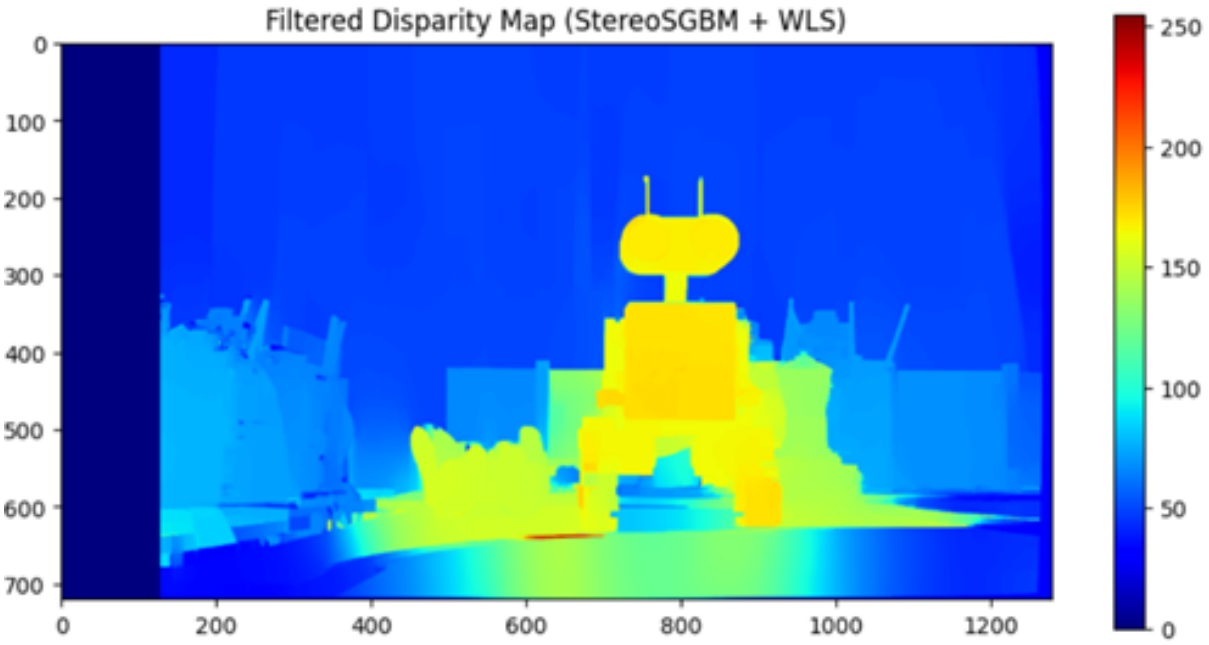


Figure 13: Filtered Disparity Map (StereoSGBM + WLS)

We applied the Weighted Least Squares (WLS) filter to the disparity map shown in Figure 10. However, the planar surface in the scene was not clearly visible, which posed a significant problem for our reconstruction. This issue was likely due to suboptimal parameter settings used during disparity estimation. After tuning the parameters of the StereoSGBM algorithm and reapplying the WLS filter, the results showed noticeable improvement. The planar region became more distinguishable, indicating that both proper disparity computati on and filtering are essential for preserving structural detail.

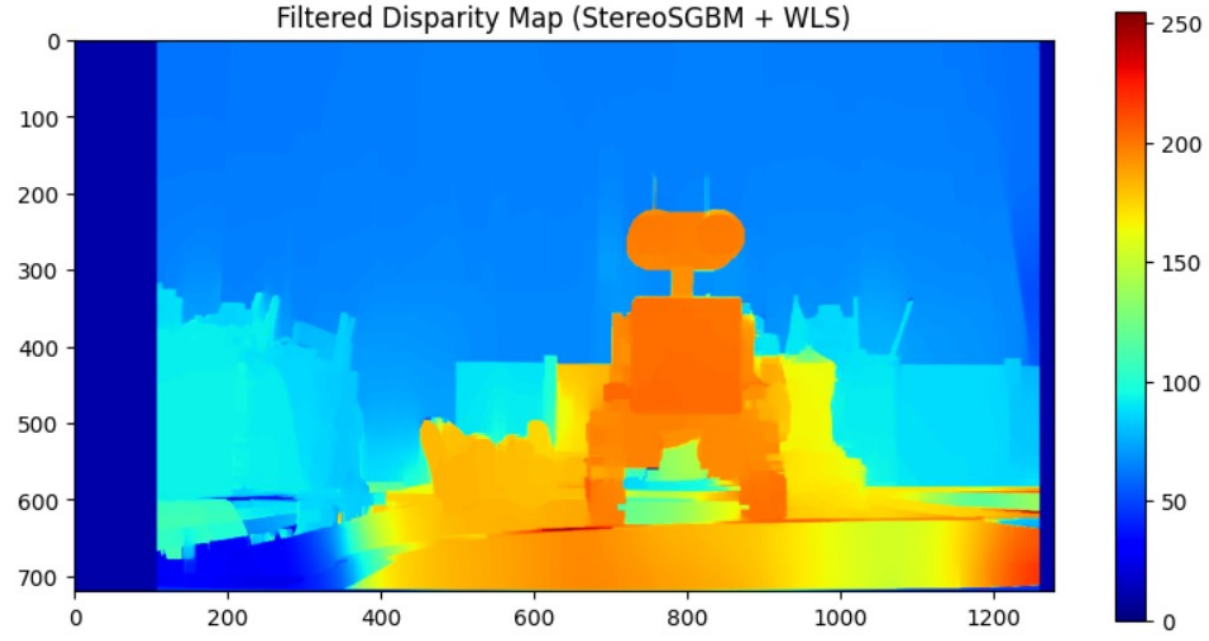


Figure 14: Filtered Disparity Map after tuning the StereoSGBM Parameters

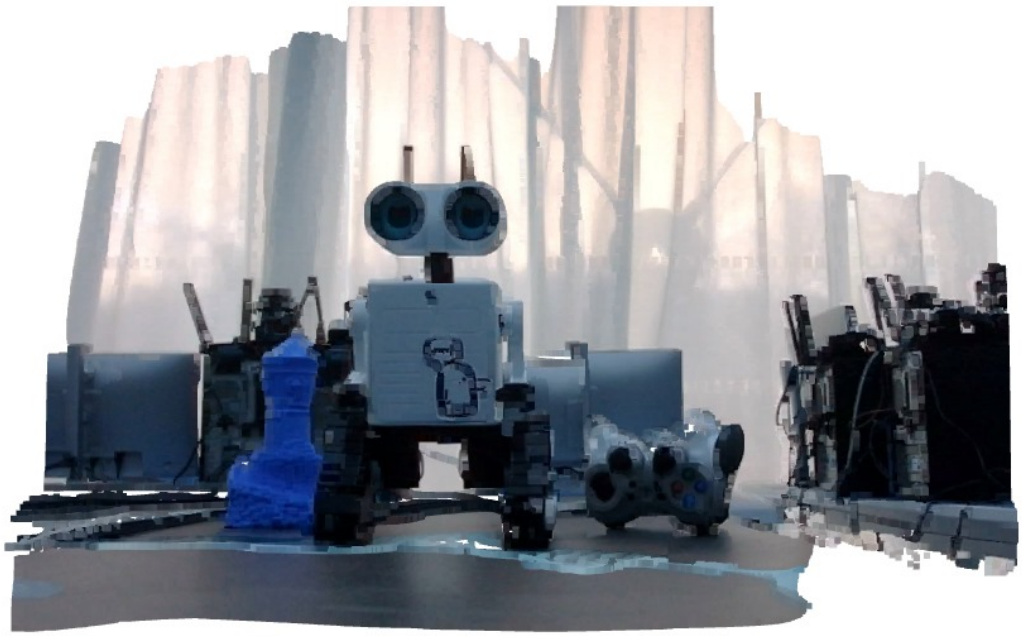


Figure 15: 3D View

Following this, we experimented with several other post-processing filters for disparity refinement. However, the Weighted Least Squares filter consistently produced the most reliable and visually coherent results, making it the best choice for our application.

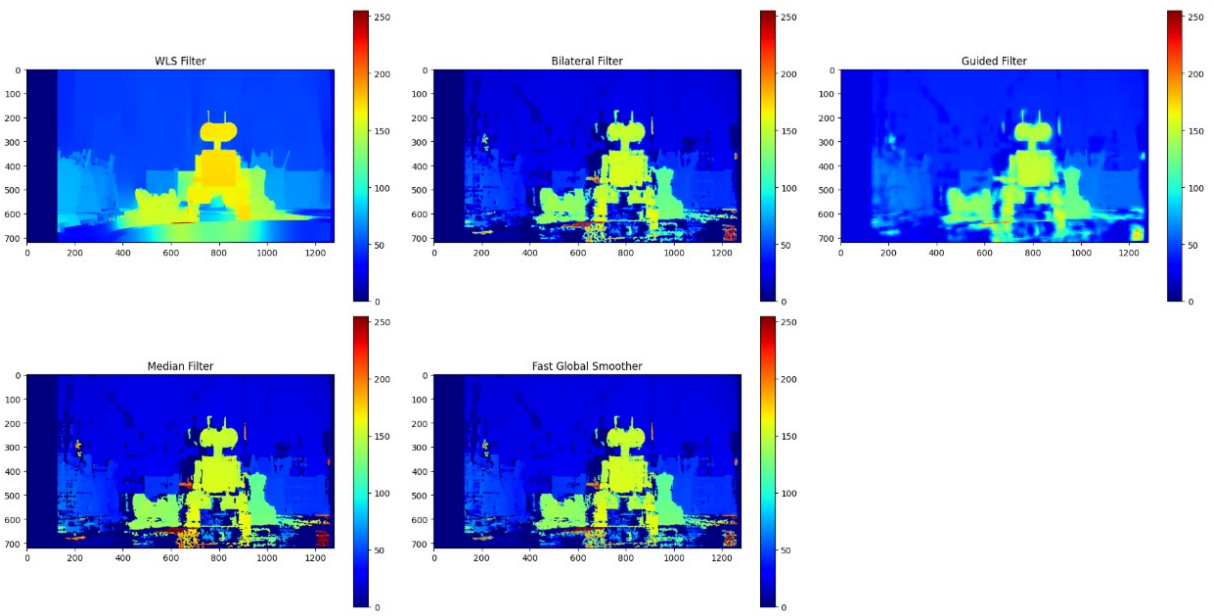


Figure 16: Disparity Map using Different Filters

4 APPROACH 2: DUAL-CAMERA STEREO SETUP

In this setup, we constructed a stereo vision system using two high-resolution USB cameras, aiming to obtain accurate depth perception by capturing synchronized images from two slightly different viewpoints. One of the key challenges in this configuration was manually aligning both cameras, particularly ensuring proper focus. Even a minor mismatch in focus between the two cameras could result in blurry or distorted images, complicating feature matching and ultimately degrading the quality of the disparity map and 3D reconstruction. This focus mismatch made the stereo calibration process more sensitive, requiring extra care during the image acquisition phase.

Figure 17: Basler aCA2440-75uc USB 3.0 cameras with Fujifilm 16mm lenses, mounted on a metal rail for rigid alignment.

4.1 Image Capture with Two Cameras

First, the challenging part was accessing the cameras. We initially attempted to use *Pylon Viewer*, the official software provided for interfacing with these cameras. However, it did not run properly on our PC due to compatibility issues. As a workaround, we used the Python-based `pypylon` library, which provided direct programmatic access to the cameras and allowed us to capture synchronized images efficiently.

Using this setup, we captured multiple image pairs of a calibration pattern (chessboard) under consistent lighting conditions. It was essential to ensure that both cameras remained stationary and that the chessboard appeared in both views with varied positions, orientations, and distances. These image pairs were then used for stereo calibration to accurately estimate each camera's intrinsic parameters, distortion coefficients, and the relative pose (rotation and translation) between the two cameras.

4.2 Calibration

To achieve high calibration accuracy, we captured a total of 149 image pairs, each taken under different focus levels and viewpoints. This large and diverse dataset greatly enhanced the robustness of the calibration results. The primary goal of this calibration step was to estimate the intrinsic parameters (camera matrix and distortion coefficients) for each camera, as well as the extrinsic parameters (rotation and translation) that define the spatial relationship between the two cameras. By using a substantial number of calibration images from various poses, we minimized reprojection errors, leading to more stable and reliable calibration outcomes, particularly important when dealing with manual setup imperfections, such as focus mismatches.

Camera Calibration Results

Camera Matrix1 (K):

$$\begin{bmatrix} 3.56487210e+03 & 0.00000000e+00 & 1.25615010e+03 \\ 0.00000000e+00 & 3.54243868e+03 & 9.94841216e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

Distortion Coefficients1:

$$[-2.23291029e-01, 7.09951120e-02, -2.82582523e-03, 5.79697269e-04, -6.58818514e-01]$$

Reprojection Error1:

0.1143498658630856

Camera Matrix (K2):

$$\begin{bmatrix} 3.95627076e+03 & 0.00000000e+00 & 1.41195677e+03 \\ 0.00000000e+00 & 3.93491245e+03 & 1.00009426e+03 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

Distortion Coefficients2:

$$[-2.29653887e-01, -3.31835273e-02, -5.15949943e-04, -1.19966003e-03, 6.63499465e-01]$$

Reprojection Error2:

0.09727264263886887

Figure 18: Camera Calibration Results for Two Matrices

4.3 Stereo Calibration Function

Once the individual camera calibrations were completed, the next step was to align and optimize the two camera. For this we used the `cv2 stereoCalibrate()` function in OpenCV to perform the stereo calibration. To indicate that the intrinsic parameters for both cameras were already reliable and should be kept fixed during stereo calibration we applied the `cv2.CALIB_FIX_INTRINSIC` flag. This function takes 3D object points (from a known calibration pattern) and their corresponding 2D image points from both the left and right cameras to estimate the rotation matrix (**R**), translation vector (**T**), essential matrix (**E**), and fundamental matrix (**F**). These matrices are crucial for computing the relative pose between the two cameras and are used in the subsequent steps of stereo rectification and 3D reconstruction.

Reference: OpenCV Stereo Calibration Documentation

Outputs of Stereo Calibration

The stereo calibration function returns several important matrices. The rotation matrix (R) and translation vector (T) define how the right camera is positioned relative to the left camera in 3D space. The essential matrix (E) combines this rotation and translation assuming known intrinsics, and is used in computing the epipolar geometry between the two images. The fundamental matrix (F), on the other hand, relates corresponding points in the two image planes without requiring knowledge of the intrinsic parameters.
Additionally, the function outputs a stereo RMS error value, which indicates how well the model fits the observed data — lower values reflect better calibration quality. By printing and analyzing these outputs, we could verify the correctness and accuracy of our stereo setup.

Stereo Calibration Results

Left Camera Matrix:

$$\begin{bmatrix} 3.56465081e+03 & 0.00000000e+00 & 1.25639072e+03 \\ 0.00000000e+00 & 3.54214646e+03 & 9.97879134e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

DisffL:

$$[-0.21789392, -0.01751589, -0.00285498, 0.00074935, -0.2258291]$$

Right Camera Matrix:

$$\begin{bmatrix} 3.99608987e+03 & 0.00000000e+00 & 1.41226888e+03 \\ 0.00000000e+00 & 3.97421433e+03 & 1.00625105e+03 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

DisffR:

$$[-0.23951713, 0.05142432, -0.00106711, -0.00134095, 0.33493019]$$

Stereo Calibration RMS Error:

$$140.10786060075975$$

Rotation Matrix (R):

$$\begin{bmatrix} 0.98878929 & -0.11385833 & 0.09660242 \\ 0.11649968 & 0.99294395 & -0.02213917 \\ -0.09340006 & 0.03314513 & 0.9950768 \end{bmatrix}$$

Translation Vector (T):

$$\begin{bmatrix} -205.31253649 \\ -0.80652978 \\ 205.72714656 \end{bmatrix}$$

Essential Matrix (E):

$$\begin{bmatrix} -23.89181756 & -204.30225733 & 3.75206961 \\ 184.24459431 & -16.61863942 & 224.17548207 \\ -23.12135755 & -203.95567037 & 4.62336228 \end{bmatrix}$$

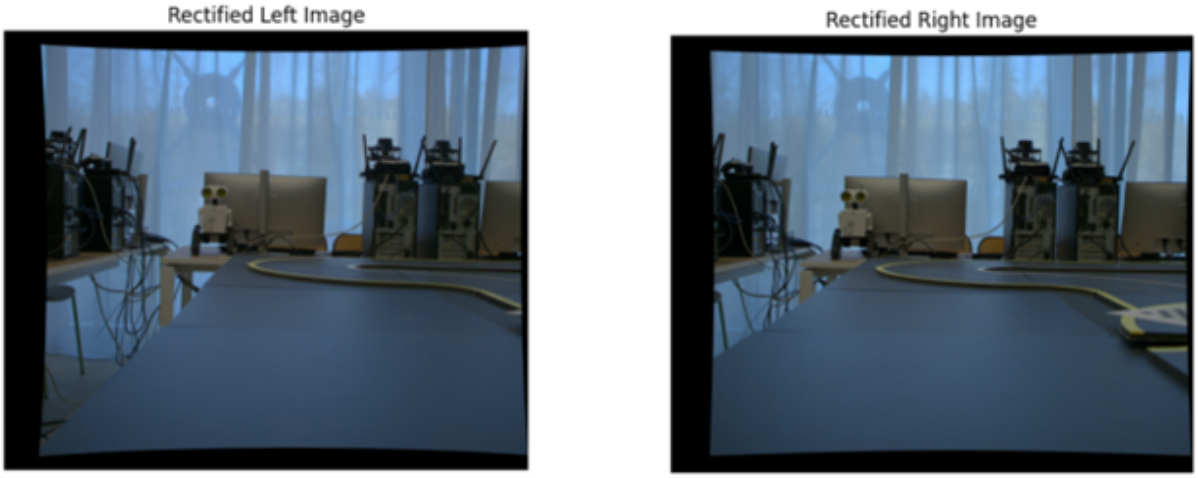
Fundamental Matrix (F):

$$\begin{bmatrix} -4.10682732e-07 & -3.53411678e-06 & 4.27250262e-03 \\ 3.18446125e-06 & -2.89059440e-07 & 1.00991928e-02 \\ -4.21257539e-03 & -8.81670049e-03 & 1 \end{bmatrix}$$

Figure 19: Stereo Calibration Results

4.4 Stereo Rectification

Once we obtained the rotation and translation matrices from stereo calibration, rectifying the stereo images became a more straightforward process. Stereo rectification aligns the image planes of both cameras so that corresponding points lie on the same horizontal scanlines. This simplifies the disparity calculation and makes matching features between the left and right images more accurate. In our case, we reused the same `cv2.stereoRectify()` function and rectification code that we implemented earlier in Case 1. Using the calibration outputs — including the intrinsic parameters, distortion coefficients, rotation, and translation matrices — the function generates rectification transforms and projection matrices for both cameras. These were then used with `cv2.initUndistortRectifyMap()` and `cv2.remap()` to produce the final rectified image pairs, ensuring proper alignment and significantly improving the quality of disparity estimation.



(a) Rectified Left Image (b) Rectified Right Image

Figure 20: Rectified stereo image pair captured using dual-camera setup.

4.5 SPARSE 3D RECONSTRUCTION USING FEATURE-BASED PIPELINE

For sparse 3D reconstruction in the dual-camera stereo setup, we followed a similar pipeline as previously implemented in Case 1. The process begins with detecting and extracting robust features (such as SIFT or ORB) from both stereo images. Once keypoints and descriptors are obtained, we match the features between the two views to establish correspondences. These matched features are then used to compute the fundamental matrix, which encapsulates the epipolar geometry between the two images. Given the intrinsic parameters from the stereo calibration, we computed the essential matrix, which directly relates the camera poses. To validate the computed essential and fundamental matrices, we performed standard checks like the epipolar constraint and determinant conditions. Once validated, we recovered the rotation matrix (R) and translation vector (T) that describe the relative pose between the two camera views. Using this relative pose and the matched keypoints, we applied triangulation to estimate the 3D coordinates of the corresponding scene points. This yielded a sparse 3D point cloud representing the structure of the environment, which serves as the foundation for denser reconstructions or further spatial analysis.

Output:

Matching Results:
Total matches found: 1320
Total strong matches after filtering: 722

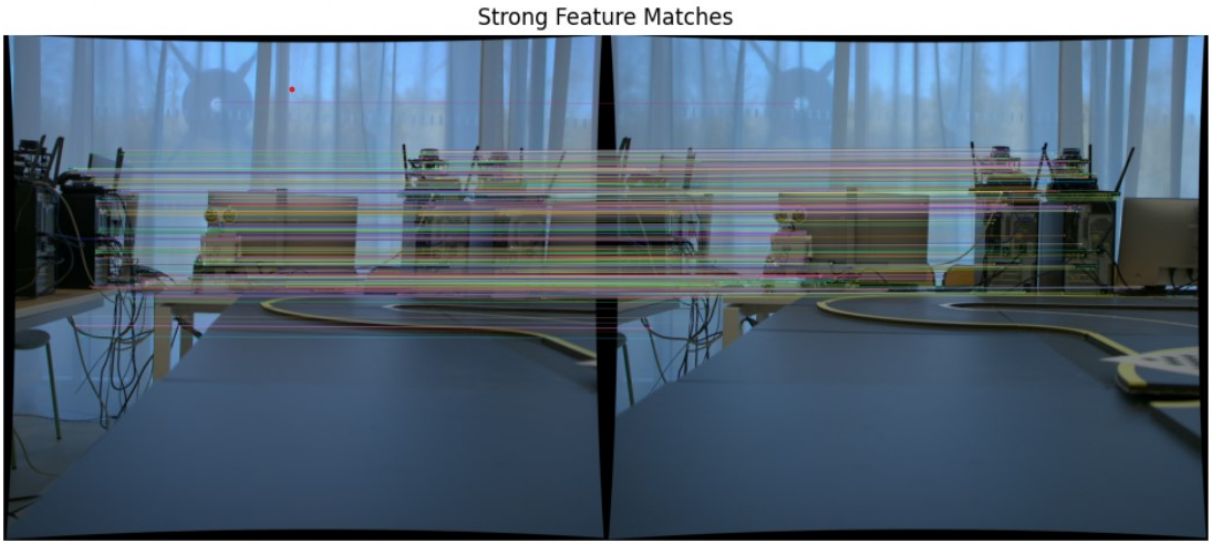


Figure 21: Approach 2 Features Matched

Essential Matrix Results

Essential Matrix:

$$\begin{bmatrix} 1.34121397e+00 & -1.16817021e+02 & 6.67675521e-01 \\ 1.14100527e+02 & -2.65464442e+00 & -8.73205776e+01 \\ -1.50363137e+00 & 8.34859047e+01 & -5.98208646e-02 \end{bmatrix}$$

Possible Rotation Matrices:

$$\begin{bmatrix} -0.29264082 & 0.01305024 & 0.95613338 \\ 0.00536906 & -0.99986868 & 0.01529047 \\ 0.95620737 & 0.00960815 & 0.29253232 \end{bmatrix}$$

$$\begin{bmatrix} 9.99456800e-01 & 3.41434576e-04 & -3.29543499e-02 \\ -6.42203610e-04 & 9.99958236e-01 & -9.11668563e-03 \\ 3.29498608e-02 & 9.13289684e-03 & 9.99415277e-01 \end{bmatrix}$$

Translation Vector:

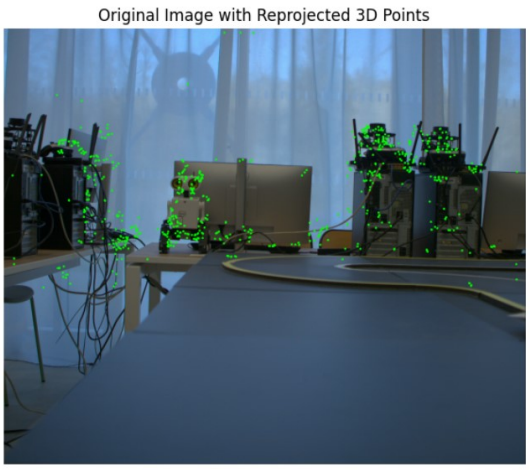
$$\begin{bmatrix} 0.58138363 \\ 0.00388802 \\ 0.81362028 \end{bmatrix}$$

Figure 22: Essential Matrix Results

Essential Matrix Analysis

Singular values of E: [1.37198061e+02 1.33504804e+02 8.41925951e-15]
Essential matrix has the correct rank (2).
Det(E) = 2.543935152915132e-10
Essential matrix satisfies det(E) = 0 condition.
Det(R1) = 0.9999999999999994, Det(R2) = 0.9999999999999998
Deviation of R1 from Identity: 2.8282959013218925
Deviation of R2 from Identity: 0.041557720435104084
Mean Epipolar Constraint Error: 0.0817846243910779

Figure 23: Essential Matrix Analysis Results



(a) 3D Point Reprojection and Visualization on Original Image

(b) 3D Point Cloud

Figure 24: Left image shows points in 3d space with original colors

4.6 DENSE 3D RECONSTRUCTION USING STEREOBM AND DISPARITY MAPS:

After successfully rectifying the stereo image pairs, we moved on to computing the depth map using the StereoSGBM (Semi-Global Block Matching) algorithm. In this step, we first loaded the rectified left and right images in grayscale. We then configured several key parameters for the StereoSGBM object, such as numDisparities, which defines the range of disparity search, and blockSize, which controls the size of the window used for matching. Parameters like uniquenessRatio, speckleWindowSize, and speckleRange were fine-tuned to suppress noise and improve the consistency of the disparity output. The disparity map was computed using the stereo.compute() function and normalized for better visualization using a color map. The result was a dense disparity map that reflects the depth variation across the scene, with warmer colors representing closer objects and cooler colors indicating greater depth. This map serves as a foundation for further steps like 3D reconstruction or filtering with WLS.

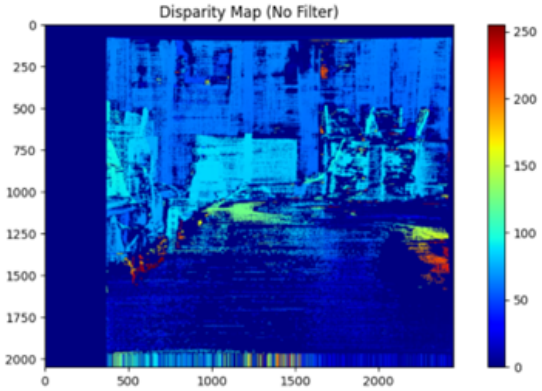


Figure 15: Approach 2 Disparity Map (No Filter)

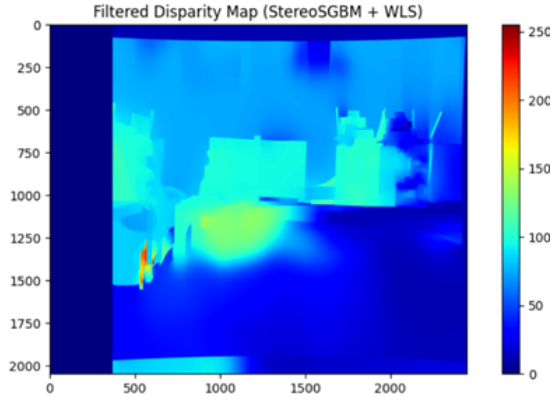
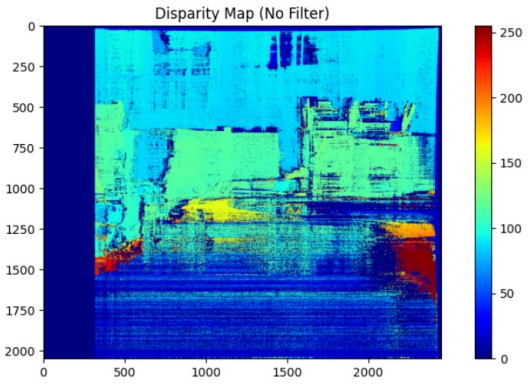
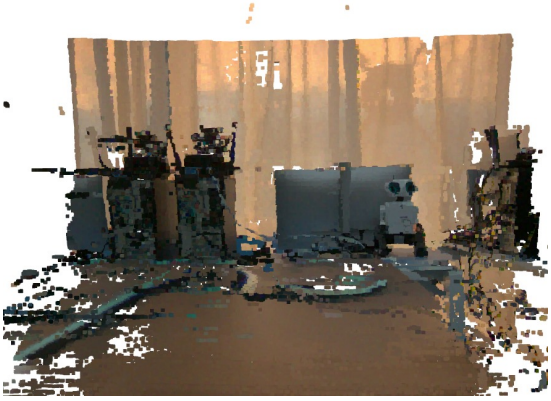


Figure 16: Approach 2 Filtered Disparity Map (StereoSGBM + WLS)

The same issue was observed in both camera outputs: the planar parts of the scene were not reconstructed well in the disparity map. This problem significantly affected the quality of depth perception, especially in flat surfaces where smooth and consistent disparity values are expected. After tuning the parameters of the StereoSGBM algorithm, the results improved, and the disparity maps began to show better structure and detail in these regions.



(a) Disparity map improvement after StereoSGBM tuning, especially in planar regions.



(b) 3D View of Depth Map

Figure 26: Improved Disparity Map and 3D View

Observations and Improvements

We encountered a recurring issue in both camera views: the planar regions of the scene were poorly reconstructed in the disparity map. Initially, this led to inaccurate depth estimation in flat surfaces.

After tuning the parameters of the StereoSGBM algorithm, the results improved significantly, especially in terms of depth continuity and overall quality of the disparity map.

However, two key challenges remained:

1. **Camera Calibration:** Achieving perfect focus alignment between the two cameras was difficult, as the focus had to be matched manually. Even minor inconsistencies in focus can degrade image sharpness and make stereo calibration and matching less reliable.

2. **Lighting Conditions:** Inconsistent or poor lighting in the scene negatively impacted feature detection and matching. Shadows, reflections, and low contrast areas particularly affected the accuracy of disparity computation.

5 User Manual

5.1 Elite Snakes 3D Reconstruction System

This user manual explains how to use the *Elite Snakes* desktop application through a series of annotated screenshots. Each step is illustrated with visual guidance to help users operate the system effectively.

Step 1: Launching the Application and Configuring Basic Settings

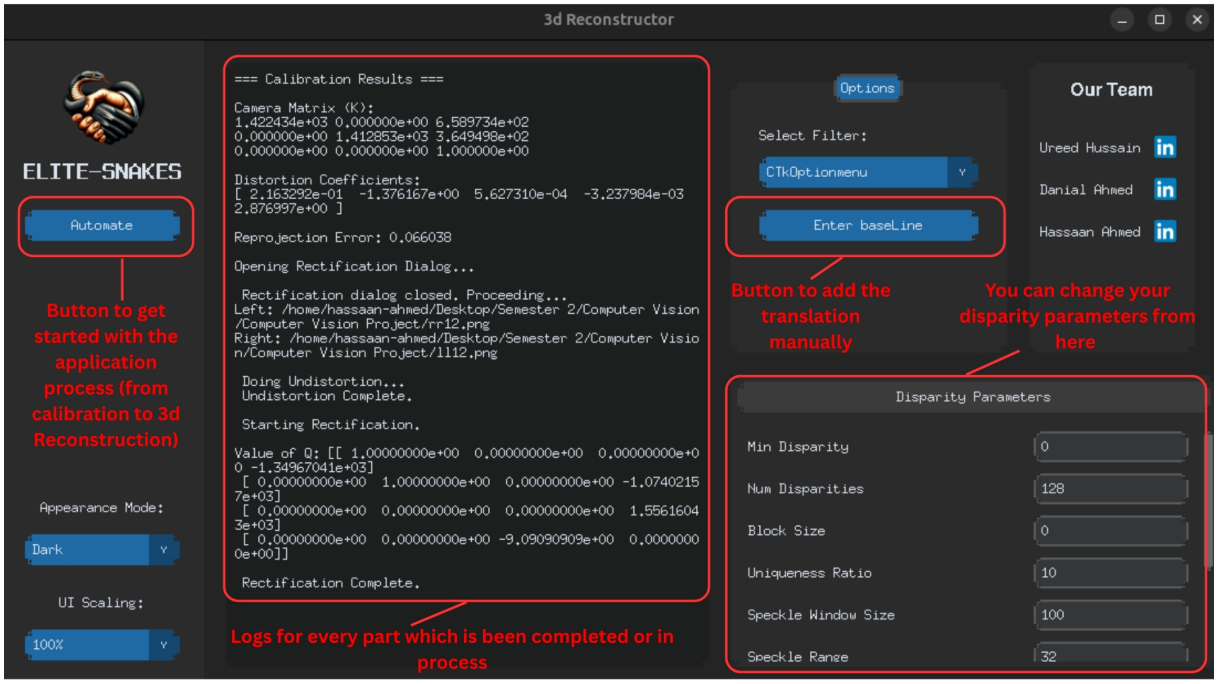


Figure 27: Welcome screen of the Elite Snakes application.

Description: After launching the *Elite Snakes* application, you'll be greeted with the welcome screen. The main button labeled "Start" initiates the entire process, guiding you through calibration to 3D reconstruction. Logs will be displayed for every completed or ongoing task to keep track of progress. You can also adjust the disparity parameters through the settings available on this screen. Additionally, there is a button that allows you to manually add translations to further refine the system's performance.

Step 2: Selecting Calibration Mode

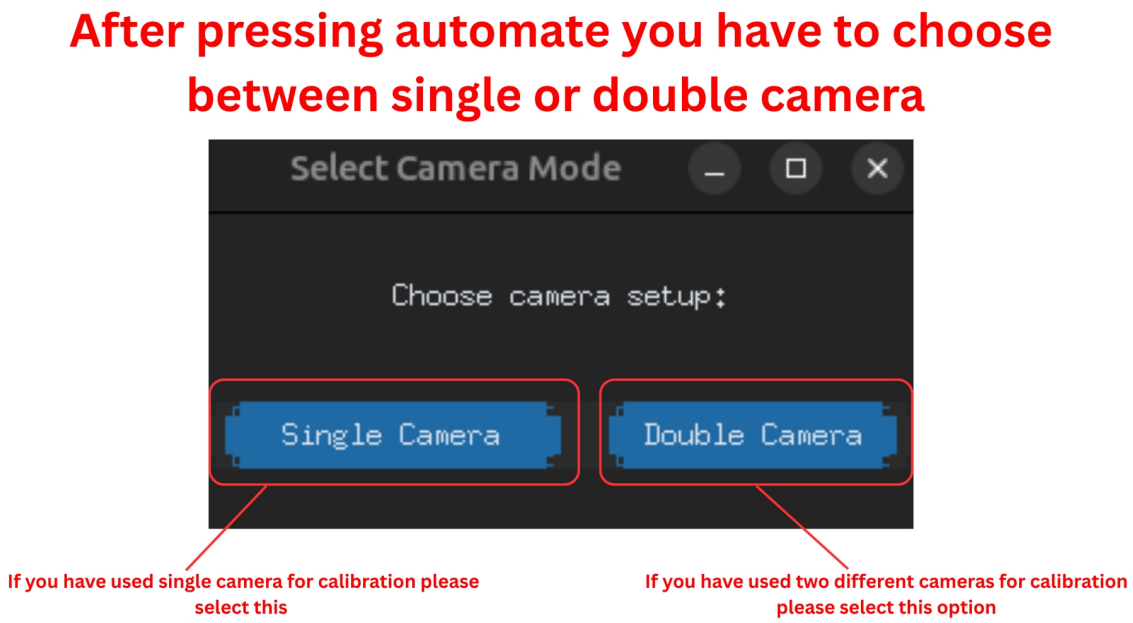


Figure 28: Plug in the USB cameras. The system will automatically detect both cameras.

Description: After pressing the "Automate" button, you will be prompted to choose between using a single camera or a dual camera setup for calibration. If you have used only one camera for calibration, select the "Single Camera" option. If you've used two different cameras for calibration, select the "Double Camera" option to proceed with the setup.

Step 3: Capturing Stereo Images

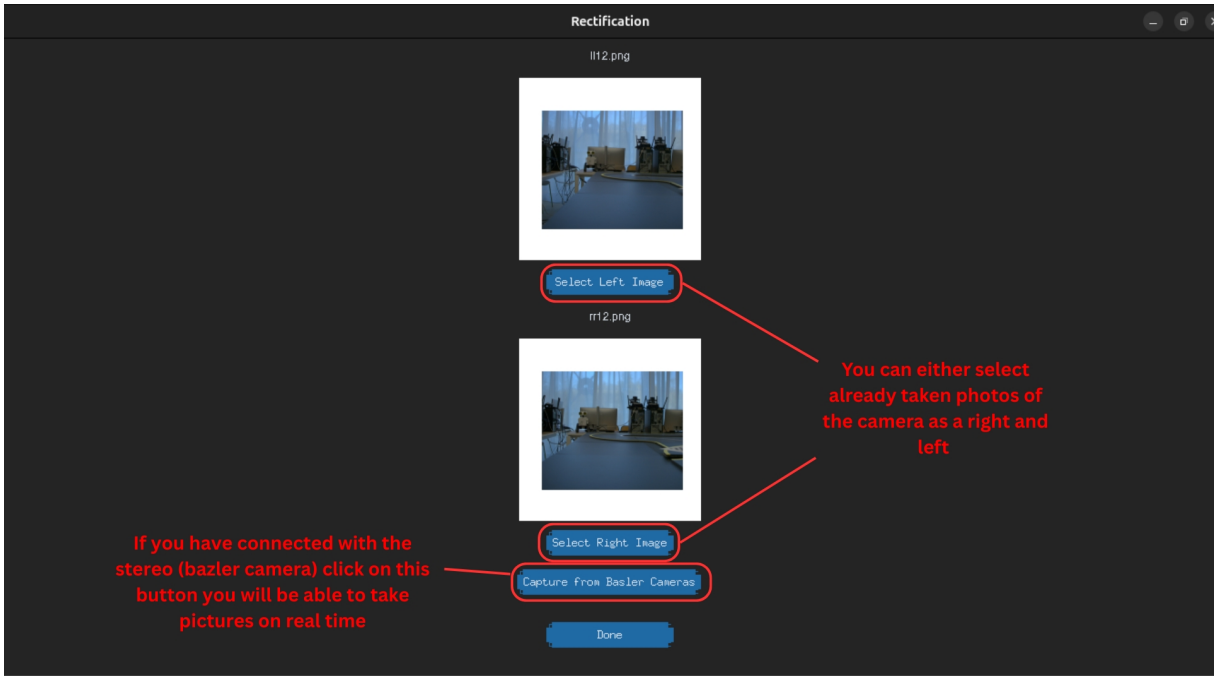


Figure 29: Select images or capture new ones in real time using a stereo camera.

Description: In this step, you have two options for capturing stereo images:
1. Select Pre-Captured Photos: If you already have stereo images, you can manually select them as the left and right images for processing.
2. Capture Real-Time Images: If you have connected a stereo camera (such as the Bazler camera), click the corresponding button to start capturing images in real time. This allows you to take live stereo images, which will be automatically labeled as left and right for the reconstruction process.

Step 4: Image Rectification and Dense 3D Reconstruction

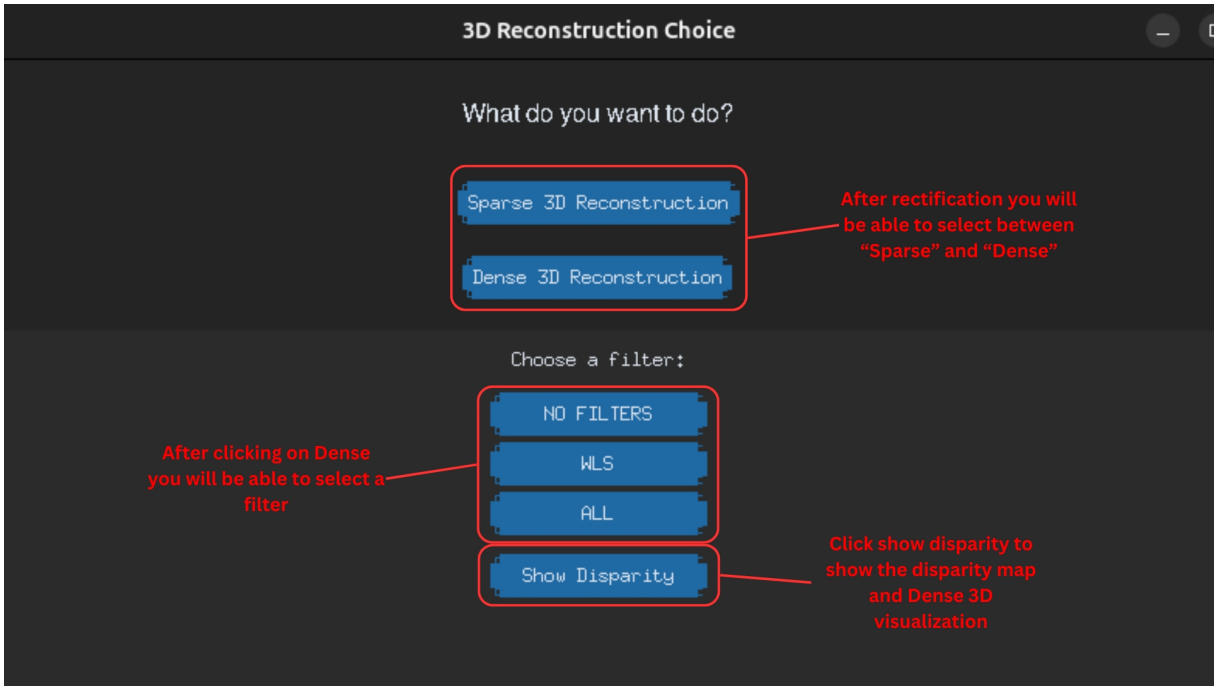


Figure 30: Choose between Sparse or Dense processing, and visualize the disparity map.

Description: After rectifying the stereo images, you will be prompted to choose between two processing modes
1. Sparse: Choose this option for a quicker but less detailed reconstruction.
2. Dense: Selecting this option allows for a more detailed 3D reconstruction. After choosing "Dense," you will be able to select a filter to refine the disparity calculation.

Step 5: Sparse Reconstruction and Real-Time Analysis

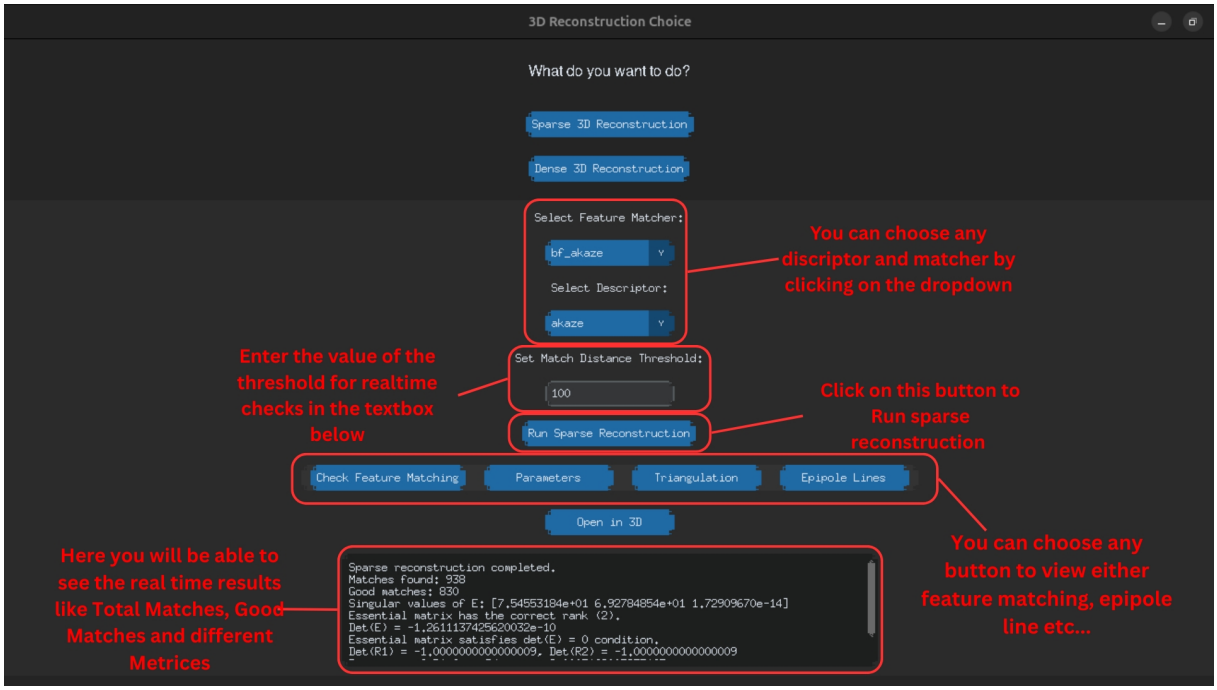


Figure 31: Select descriptors and matchers, run sparse reconstruction, and view real-time results.

Description: In this step, you can configure various parameters to run sparse reconstruction and view real-time results:
1. Descriptor and Matcher Selection: Choose any descriptor and matcher from the dropdown menus. These parameters will define how features are detected and matched between the stereo images.
2. Threshold Input: Enter a threshold value in the textbox below to perform real-time checks on the feature matching process.
3. Running Sparse Reconstruction: Click the corresponding button to run the sparse reconstruction. This process will analyze the images and generate a sparse 3D model.
4. View Different Visualizations: You can select any button to view additional results such as feature matching, epipolar lines, etc.
5. Real-Time Results: Once the reconstruction process is running, you will be able to see real-time results such as:
- Total Matches: The total number of feature matches found.
- Good Matches: The number of matches that meet the quality criteria.
- Metrics: Various metrics related to the quality of the reconstruction.

6 Key Learnings

Through this project, we gained extensive hands-on experience in the field of 3D computer vision and stereo reconstruction. Key takeaways include:

- **Camera Calibration:** We understood the importance of accurate camera calibration and how errors in intrinsic/extrinsic parameters can significantly affect downstream tasks like depth estimation and triangulation.
- **Stereo Rectification:** We learned how to align images properly to ensure correct disparity calculations, particularly how minor rotation or misalignment can disrupt the entire pipeline.
- **Feature Matching & Outlier Removal:** By experimenting with different detectors like AKAZE, ORB, and SIFT, we understood their trade-offs in terms of speed and accuracy. We also saw the effectiveness of RANSAC in refining match quality.
- **Sparse vs. Dense Reconstruction:** The contrast between sparse and dense 3D reconstruction helped us appreciate their respective strengths—sparse for structure, dense for completeness.
- **Algorithm Tuning:** Fine-tuning parameters in algorithms like StereoSGBM and WLS filtering significantly impacted the quality of the disparity map and final 3D model.
- **System Design:** Building a usable software interface (Elite Snakes 3D Reconstruction System) taught us how to integrate complex computer vision tasks into a user-friendly application.

7 Future Work

Although the current system achieves good 3D reconstruction results, there is room for improvement and further development:

- **Real-time Processing:** Implement GPU acceleration (e.g., using CUDA or OpenCL) to enable faster, possibly real-time depth estimation and visualization.
- **Point Cloud Post-processing:** Apply advanced filtering and meshing techniques to convert raw point clouds into clean 3D models.
- **SLAM Integration:** Extend the system for simultaneous localization and mapping (SLAM) for dynamic or large-scale environments.
- **Automatic Calibration Assistance:** Use AI-based tools to automatically guide users during calibration to minimize manual effort and error.
- **Support for More Sensors:** Expand compatibility to include depth cameras like Intel RealSense or LiDAR for hybrid fusion.
- **User Customization:** Add options to allow users to define their own matching algorithms, reconstruction settings, or filtering pipelines via GUI.

8 Conclusion

This project presents a comprehensive and pragmatic exploration into low-cost 3D reconstruction using both single and dual stereo camera setups. By methodically addressing key challenges such as camera calibration, stereo rectification, feature matching, and disparity map refinement, the project successfully demonstrates the feasibility of building a portable 3D vision system with consumer-grade hardware and open-source software. Through rigorous experimentation with both sparse and dense reconstruction techniques—using algorithms like AKAZE, SGBM, and WLS filtering—the system achieves credible 3D scene reconstructions across various setups.

The development of the *Elite Snakes 3D Reconstruction System* further underscores the project's practical value by offering a user-friendly interface that bridges advanced computer vision techniques with accessible application design. Moreover, the dual-mode (single and stereo camera) flexibility and parameter tuning capabilities make the system adaptable for a wide range of applications in robotics, AR/VR,

and automated inspection.
Beyond technical execution, the project also fosters critical insights into the intricate dependencies between hardware precision (e.g., focus alignment), algorithm tuning, and image quality. These learnings, combined with well-identified paths for future improvement such as GPU acceleration and SLAM integration, position this work as a solid foundation for continued innovation in real-time 3D perception and vision-based systems.

References

[1] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[2] "Stereo calibration – opencv documentation," https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga9d2539c1ebcda647487a616bdf0fc716, accessed: 2025-05-01.

[3] OpenCV, ""AKAZE feature detection and matching"," *OpenCV Documentation, 2025, available: https://docs.opencv.org/4.x/db/d70/tutorial_akaze_matching.html, Accessed: May 1, 2025.*

[4] ——, *""ORB feature detection (Python)","* OpenCV Documentation, 2025, available: https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html, Accessed: May 1, 2025.

[5] C. Popovici, "Stereo vision and disparity maps in opencv," https://medium.com/@cristinapopovici, 2021, accessed: 2025-05-01.

[6] "Feature detection and matching with opencv," https://towardsdatascience.com/feature-detection-and-matching-with-opencv-dc8303f3b606, 2019, accessed: 2025-05-01.

[7] G. Bradski, "The opencv library," https://docs.opencv.org/, 2000, accessed: 2025-05-01.

[8] GeeksforGeeks, "SIFT interest point detector using Python-OpenCV," https://www.geeksforgeeks.org/sift-interest-point-detector-using-python-opencv/, n.d., accessed: May 1, 2025.

[9] "Stereosgbm parameters explained – semi-global block matching in opencv," https://www.youtube.com/watch?v=8r0CiLBM1oE, accessed: 2025-05-01.

[10] "Camera calibration principles and procedures," https://eikosim.com/en/technical-articles/camera-calibration-principles-and-procedures/, 2022, accessed: 2025-05-01.

[11] "Sift interest point detector using python-opencv," https://www.geeksforgeeks.org/sift-interest-point-detector-using-python-opencv/, accessed: 2025-05-01.

[12] Basler AG, "Pypylon – python wrapper for basler's pylon camera software suite," https://github.com/basler/pypylon, accessed: 2025-05-01.

[13] *stereoCalibrate — OpenCV Documentation*, 2025, https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html.

[14] *stereoRectify — OpenCV Documentation*, 2025, https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html.

[15] *initUndistortRectifyMap — OpenCV Documentation*, 2025, https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html.

[16] *remap — OpenCV Documentation*, 2025, https://docs.opencv.org/3.4/d1/da0/tutorial_remap.html.

[17] *BFMatcher — OpenCV Documentation*, 2025, https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html.

[18] *findFundamentalMat — OpenCV Documentation*, 2025, https://docs.opencv.org/4.x/da/de9/tutorial_py_epipolar_geometry.html.

[19] N. Developers, *numpy.linalg.svd — NumPy Documentation*, 2025, https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html.

[20] *triangulatePoints — OpenCV Documentation*, 2025, https://docs.opencv.org/4.x/d0/dbd/group__triangulation.html.

[21] *createDisparityWLSFilter — OpenCV ximgproc*, 2025, https://docs.opencv.org/3.4/d9/d51/classcv_1_1ximgproc_1_1DisparityWLSFilter.html.