Semi Global Block Matching (SGBM) and 3D space transformation

Final Submission - Mobile Robotics

Aniruth Suresh
2022102055
IIIT , Hyderabad, India
aniruth.suresh@students.iiit.ac.in

Aryan Garg
2022102074
IIIT , Hyderabad, India
aryan.garg@students.iiit.ac.in

Pratyush Jena
2022111016
IIIT , Hyderabad, India
pratyush.jena@research.iiit.ac.in

***Abstract*—This report presents the final submission of our project, which investigates the Semi-global Block Matching (SGBM) algorithm for disparity estimation and transformation in stereo vision. The project is divided into two phases.**

In Phase 1, we focus on implementing SGBM from scratch and benchmarking its performance on toy datasets and the KITTI dataset, comparing it to existing disparity estimation methods. This phase evaluates the algorithm's accuracy and efficiency in creating disparity maps.

Phase 2, we focus on post-processing the disparity output to reconstruct a 3D representation of the scene in the camera's coordinate space. This transformation will leverage the geometric model of the binocular camera setup, completing the stereoscopic depth estimation pipeline.

I. Introduction

Accurate depth perception is a critical challenge in computer vision, particularly for applications like autonomous driving, robotics, where understanding the 3D structure of the environment is essential. Traditional depth estimation techniques often struggle with achieving both **high accuracy and real-time performance**, especially in complex, dynamic scenes.

One promising approach to address this challenge is **stereo vision**, which uses two spatially separated cameras to estimate depth by **comparing disparities between two images**. However, finding the optimal disparity values across the entire image, especially in real-time, remains difficult due to the computational load and the need for precise matching, particularly in areas with low texture or repeating patterns.

To overcome these issues, the **Semi-global Block Matching (SGBM) algorithm** has emerged as an effective solution. SGBM balances **accuracy and computational efficiency**, making it suitable for real-time applications. By performing **block matching and aggregating costs along multiple paths**, SGBM provides accurate disparity maps while significantly reducing computational demands compared to traditional global matching methods.

II. Semi Global Block Matching Model and Pipeline

The Semi-Global Block Matching (SGBM) algorithm follows a three-stage pipeline, as shown in Fig. 1. Each stage contributes to constructing an accurate and efficient disparity map, and they will be explored in detail in the following sections.

1) **Pixelwise Cost Volume Calculation**: This stage involves calculating the matching cost for each pixel across the stereo image pair, forming a cost volume essential for disparity estimation.
2) **Cost Aggregation**: In this stage, SGBM applies a semi-global optimization method to refine the disparity map
3) **Final Disparity Map Generation**: The minimum costs are selected for each pixel to produce the disparity map. This preliminary map may then undergo post-processing to enhance quality and resolution.

The Semi-Global Block Matching (SGBM) algorithm takes a pair of rectified stereo images, denoted by I_L and I_R, each of size $H \times W$, as input. These images are assumed to be **rectified**, meaning that **corresponding pixels lie along the same horizontal row in both images**.

The objective is to compute a disparity map D that provides the horizontal shift between corresponding pixels in the left and right images.

A. STAGE 1 : Pixelwise Cost Volume Calculation

The first step in SGBM is to compute the *cost volume*, which measures the **similarity between corresponding pixels** in the left and right images across a range of disparity values.

For each pixel (h, w) in the left image, we calculate the matching cost for each disparity $d \in [d_{\text{min}}, d_{\text{max}}]$, where d_{min} and d_{max} are the minimum and maximum allowable disparities, respectively.

The **Cost Volume** C is defined as:

$$\boxed{C(h,w,d) = \text{cost}(I_L(h,w), I_R(h-d,w)) \quad \text{for} \quad d \in [d_{\text{min}}, d_{\text{max}}]} \tag{1}$$

where :

- h and w are the row and column indices of a pixel in the left image I_L.
- d is the disparity value at which the matching cost is calculated.

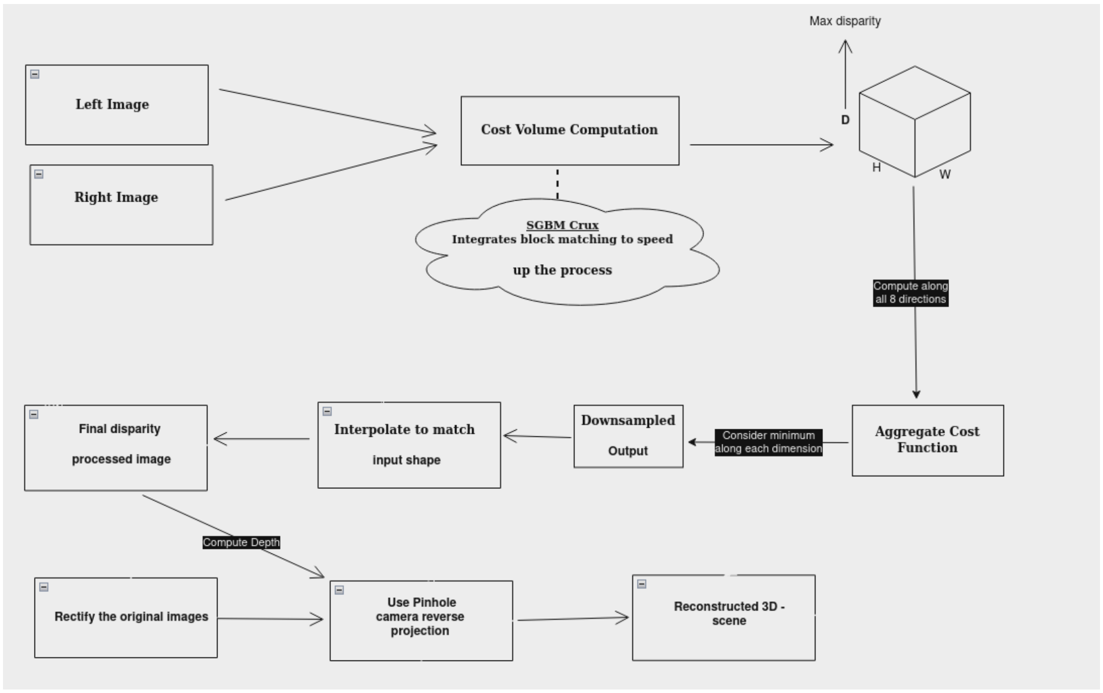
Max disparity
Left Image
Right Image
Cost Volume Computation
D
H
W
SGBM Crux
Integrates block matching to speed up the process
Compute along all 8 directions
Aggregate Cost Function
Consider minimum along each dimension
Downsampled Output
Interpolate to match input shape
Final disparity processed image
Compute Depth
Rectify the original images
Use Pinhole camera reverse projection
Reconstructed 3D - scene

Fig. 1. The figure above illustrates the **complete architecture of the Semi-global Block Matching (SGBM)** algorithm. Initially, stereo images are input, and a **matching cost** is computed using metrics such as census transform with Hamming distance, intensity difference, or similar methods. **A cost volume is then constructed**, and cost aggregation is performed along multiple directions, followed by selecting the **minimum cost** for each pixel. Due to **block matching** , which is the main curx of SGBM, the resulting output is **downscaled**. To restore this output to the original input image dimensions, **upscaling (interpolate)** is required, along with **post-processing** techniques to enhance the image quality. For the 3D reconstruction , we first need to **rectify the left and right images** and **compute the depth** from the disparity map and use the pinhole model to construct the 3D - scene .

- $\text{cost}(I_L(h,w), I_R(h-d,w))$ is the matching cost function, which measures the **dissimilarity between the pixel** (h,w) in the left image and the corresponding pixel $(h-d,w)$ in the right image.

Common choices for the cost function include the **Sum of Absolute Differences (SAD), Sum of Squared Differences (SSD), or the Census transform with Hamming** .

The resulting cost volume C is a 3D array of size $H \times W \times D$, where $D = d_{\text{max}} - d_{\text{min}} + 1$ is the number of disparity levels. The original cost volume C, with dimensions $H \times W \times D$, is **transformed into a new cost volume** C', where each voxel represents the aggregated matching cost over a local block surrounding the corresponding pixel. The size of the cost volume C' is adjusted based on the chosen grid size, which determines the dimensions of the local block.

Block Aggregation and Formula: For each pixel (h,w) in the image and each disparity d, the aggregated cost in the new cost volume C' is calculated by **summing the costs of all pixels within a block of size grid size $\times$ grid size**. The new cost at pixel (h,w) and disparity d is:

$$\boxed{C'(h,w,d) = \sum_{i=h-\left\lfloor \frac{\text{grid size}}{2} \right\rfloor}^{h+\left\lfloor \frac{\text{grid size}}{2} \right\rfloor} \sum_{j=w-\left\lfloor \frac{\text{grid size}}{2} \right\rfloor}^{w+\left\lfloor \frac{\text{grid size}}{2} \right\rfloor} C(i,j,d)} \tag{2}$$

where:

- $C(h,w,d)$ is the cost at pixel (h,w) and disparity d in the original cost volume.
- grid size is the size of the local block used for aggregation.
- The indices i and j range over the local block around pixel (h,w), and are bounded within the image dimensions $H \times W$.

The resulting aggregated cost volume C' will have dimensions:

$$\left\lfloor \frac{H + \text{grid size} - 1}{\text{grid size}} \right\rfloor \times \left\lfloor \frac{W + \text{grid size} - 1}{\text{grid size}} \right\rfloor \times D \tag{3}$$

Here, $\lfloor x \rfloor$ denotes the floor of the value x, which represents the greatest integer less than or equal to x.

This **accounts for the reduction** in the cost volume dimensions after block aggregation. The height and width of the new cost volume are determined by how many full blocks of size grid size fit into the original height H and width W. This results in a cost volume with reduced height and width dimensions, as it is **downsampled** by the grid size.

STAGE 2 : Aggregation of Costs

The pixelwise cost calculation in stereo matching can be ambiguous and prone to errors due to noise. To address this, a **smoothness constraint is added**, penalizing disparity changes between neighboring pixels.

The energy function $E(D)$, which depends on the disparity image D, is defined as:

$$\boxed{\begin{aligned}E(D) = \sum_p \Bigg(C(p, D_p) + \sum_{q \in N_p} P_1 T[|D_p - D_q| = 1] \\ + \sum_{q \in N_p} P_2 T[|D_p - D_q| > 1] \Bigg)\end{aligned}} \tag{4}$$

where:

- $C(p, D_p)$: The pixel matching cost.
- The second term **penalizes small disparity** changes, $|D_p - D_q| = 1$, with a constant penalty P_1.
- The third term **penalizes larger disparity** changes, $|D_p - D_q| > 1$, with a larger constant penalty P_2.

Dynamic Programming (DP) can efficiently solve the problem along individual image rows, but it suffers from **streaking** [1].

To resolve this, the idea of **aggregating matching costs from all directions** is introduced.

The aggregated cost $S(p, d)$ for a pixel p and disparity d is computed by **summing the costs of all minimum cost paths** that end at p with disparity d, as shown in the following recursive relation:

$$\boxed{\begin{aligned}L'_r(p, d) = C(p, d) + \min \big(& L'_r(p - r, d), \\ & L'_r(p - r, d - 1) + P_1, \\ & L'_r(p - r, d + 1) + P_1, \\ & \min_i L'_r(p - r, i) + P_2 \big)\end{aligned}} \tag{5}$$

This equation calculates the minimum cost along a path in direction r and incorporates penalties for disparity discontinuities. The cost is **normalized by subtracting the minimum cost** from the previous pixel:

$$\boxed{\begin{aligned}L_r(p, d) = C(p, d) + \min \big(& L_r(p - r, d), \\ & L_r(p - r, d - 1) + P_1, \\ & L_r(p - r, d + 1) + P_1, \\ & \min_i L_r(p - r, i) + P_2, \\ & - \min_k L_r(p - r, k) \big)\end{aligned}} \tag{6}$$

The **aggregated cost** $S(p, d)$ is then obtained by summing the costs over all directions:

$$\boxed{S(p, d) = \sum_r L_r(p, d)} \tag{7}$$

[1]Streaking refers to the artifact where disparity values vary abnormally across rows, causing visible vertical stripes or bands in the disparity map due to the lack of smoothness in the vertical direction.

An efficient implementation involves pre-calculating the pixelwise matching costs $C(p, d)$ and storing them in a 16-bit integer array. The cost calculation is performed for each direction, and the **results are aggregated into the array** $S[p, d]$**.**

B. *STAGE 3: Final Disparity Map Generation*

After aggregating the cost volumes for each direction, the final disparity map is obtained by selecting the **disparity with the least aggregated cost for each pixel**. The steps for determining the final disparity map are as follows:

The disparity D_p for each pixel p is chosen as the disparity d that minimizes the aggregated cost $S(p, d)$:

$$D_p = \arg\min_d S(p, d) \tag{8}$$

Reprojection: Once the disparity values D_p are computed for all pixels, the **disparity map is reprojected back into the original image dimensions**, adjusting for the dimensions lost due to SGBM block matching .

III. Post Processing

To enhance the quality of the disparity maps, we apply two post-processing techniques: the **Bilateral Filter and the Median Filter**. These methods effectively reduce noise while preserving edges, which is crucial for disparity maps.

A. *Bilateral Filter*

The Bilateral Filter is a **non-linear filter** that smooths an image while preserving edges. It achieves this by combining both s**patial proximity and intensity similarity when averaging the pixel values**. The filter output for a pixel p is given by:

$$\boxed{I'(p) = \frac{1}{W_p} \sum_{q \in \mathcal{N}(p)} G_s(\|p - q\|) \cdot G_r(|I(p) - I(q)|) \cdot I(q),} \tag{9}$$

where:

- $\mathcal{N}(p)$ is the neighborhood of the pixel p,
- $G_s(\|p - q\|)$ is the spatial Gaussian kernel, weighting based on the distance between p and q,
- $G_r(|I(p) - I(q)|)$ is the range Gaussian kernel, weighting based on the intensity difference,
- W_p is the normalization factor:

$$W_p = \sum_{q \in \mathcal{N}(p)} G_s(\|p - q\|) \cdot G_r(|I(p) - I(q)|). \tag{10}$$

This filter is **highly effective in removing noise** while retaining edge details in disparity maps.

B. Median Filter

The Median Filter is a non-linear filter that replaces each pixel value with the median value of the intensities within its neighborhood. This approach is robust against **salt-and-pepper noise**, making it suitable for disparity map refinement. For a pixel p, the output of the filter is given by:

$$\boxed{I'(p) = \text{median}\{I(q) \mid q \in \mathcal{N}(p)\},} \tag{11}$$

where:

- $\mathcal{N}(p)$ is the neighborhood of pixel p,
- $\text{median}\{\cdot\}$ computes the median of the intensities within $\mathcal{N}(p)$.

The Median Filter effectively removes noise while maintaining sharp edges, making it an essential step in post-processing disparity maps.
By applying these filters, the final disparity maps are smoother and more accurate, particularly in regions with noise or weak texture.

IV. Image Rectification

Image rectification is a preprocessing step in stereo vision that aligns the stereo image pair such that the **corresponding points in one image lie on the same horizontal line (row) in the other image**. This process simplifies the correspondence search for creating the disparity map, as it reduces the problem to a **one-dimensional search along the epipolar lines.**

A. Epipolar Geometry and Rectification

In a stereo setup, the geometric relationship between two cameras can be described using epipolar geometry:

- **Epipoles**: The epipole is the intersection point of the line joining the camera centers with the image plane. Each image has an epipole corresponding to the other camera.
- **Epipolar Lines**: An epipolar line is the projection of a 3D point's possible positions onto the image plane. In rectified images, these lines are horizontal, simplifying the disparity calculation.

B. Rectification Homographies

To rectify the images, a **pair of homographies** H_L and H_R are applied to the left and right images, respectively. These homographies align the epipolar lines and ensure the epipoles are at infinity. The rectification homographies are computed as:

$$\boxed{H_L = K \cdot R_L \cdot K^{-1}, \quad H_R = K \cdot R_R \cdot K^{-1},} \tag{12}$$

where:

- K: The intrinsic camera matrix.
- R_L and R_R: Rectification rotation matrices for the left and right cameras.
- K^{-1}: The inverse of the intrinsic camera matrix.

The rectification rotations R_L and R_R are calculated as follows:

$$\boxed{\mathrm{R}_L = R, \quad R_R = R \cdot R',} \tag{13}$$

where:

- R: A rotation matrix that aligns the baseline of the cameras with the x-axis.
- R': A rotation matrix that aligns the principal axes of the two cameras.

The homographies transform the left and right images into a common rectified coordinate system.

C. Epipolar Constraint

The epipolar constraint governs the relationship between a point in the left image $\mathbf{x}_L$ and its corresponding point in the right image $\mathbf{x}_R$:

$$\boxed{\mathrm{x}_R^\top F \mathbf{x}_L = 0,} \tag{14}$$

where F is the fundamental matrix, encapsulating the stereo camera geometry.

After rectification, the fundamental matrix becomes simplified **such that epipolar lines align with the rows of the rectified images**. This means:

$$y'_L = y'_R, \tag{15}$$

where y'_L and y'_R are the vertical coordinates of corresponding points in the rectified left and right images.

D. Rectification in Practice

In practice, image rectification involves:

1) Computing the fundamental matrix F or the essential matrix E.
2) Determining the rectification transformations H_L and H_R using the camera matrices P_L and P_R.
3) Applying the transformations to align the stereo image pair.

E. Advantages of Rectification

Rectification simplifies the disparity calculation by ensuring that:

- Corresponding points lie on the **same horizontal line**, reducing the search space to one dimension.
- The **epipolar lines are parallel** and aligned with the image rows.

This preprocessing step is crucial for algorithms like SGBM and SGM, which assume rectified image pairs for efficient stereo matching.

V. 3D Projection

To reconstruct a 3D representation of the scene from the 2D disparity map, we project the disparity values into the 3D space using a **pinhole camera model**. The process involves mapping the pixel coordinates in the image plane to 3D points in the camera coordinate frame. The relationship is governed by the camera intrinsics and the baseline between the stereo cameras.

A. Projection Formula

The 3D coordinates (X, Y, Z) of a point in the camera coordinate frame are calculated using the following equations:

$$\boxed{Z = \frac{f \cdot B}{d},} \tag{16}$$

$$X = \frac{(x - c_x) \cdot Z}{f}, \tag{17}$$

$$Y = \frac{(y - c_y) \cdot Z}{f}, \tag{18}$$

where:

- f: Focal length of the camera, obtained from the intrinsic matrix.
- B: Baseline distance between the stereo cameras.
- d: Disparity value at the pixel location (x, y).
- (x, y): Pixel coordinates in the image.
- (c_x, c_y): Principal point (optical center) coordinates, derived from the intrinsic matrix.

B. Practical Considerations

1. **Depth Calculation**: The depth Z is inversely proportional to the disparity d. Higher disparity corresponds to closer objects, while lower disparity indicates farther objects.
2. **Invalid Disparities**: Pixels with zero or undefined disparity values are typically excluded from reconstruction or assigned a large depth value.
3. **Scaling and Units**: Ensure that f, B, and d are in consistent units (e.g., millimeters or meters).

C. Camera Projection Matrix

The camera intrinsic matrix K is typically of the form:

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{19}$$

The intrinsic matrix encodes th**e focal length and principal point**, which are critical for the projection process.
Using these formulas and the intrinsic parameters, we project each pixel of the disparity map into the 3D space, creating a dense point cloud that represents the reconstructed scene.

VI. Datasets

For testing the functionality of the implemented SGBM algorithm, we utilized two available datasets:

1) **KITTI Dataset:** This dataset is available at KITTI RAW.

2) **TOY Dataset:** A very basic dataset consisting of the original images and their corresponding depth maps. This can be accessed at TOY Dataset.

For both datasets, we load the left and right stereo images, along with the corresponding ground truth disparity maps. These ground truth values are used to **compute the RMSE** of the disparity estimation produced by the SGBM algorithm.

VII. Achievements - Computational Time vs Accuracy

To evaluate the effectiveness of the SGBM algorithm compared to existing methods, we plotted the **grid size used to reduce the total computational cost versus the accuracy and time taken**.

As shown in Fig. 2, it is evident that as the kernel size increases, the computation time significantly decreases, while the **left and right disparity accuracy only dips slightly**. This highlights the advantage of SGBM over other techniques, such as SGM, where we can achieve **excellent performance with much lower computational time.**

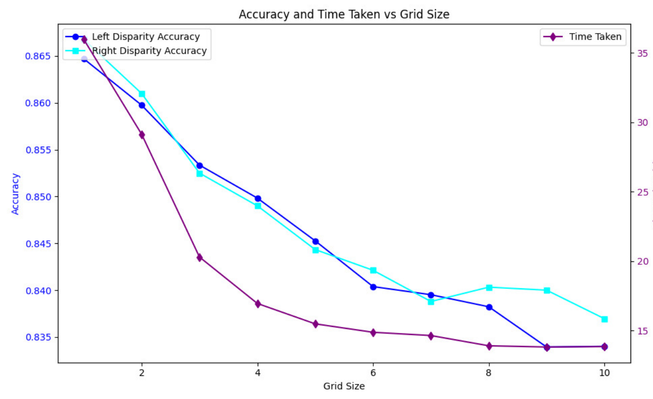

Fig. 2. This plot illustrates the relationship between grid size, accuracy, and computation time. As the **grid size increases from 2 to 9**, the computation time decreases from **28.14 seconds to 13.7 seconds**. Meanwhile, the **left disparity accuracy slightly decreases from 0.860 to 0.838**, and the **right disparity accuracy dips from 0.861 to 0.841**.

VIII. Results

For the same sequence, we ran **SGM, SGBM, and the built-in OpenCV SGBM algorithm** (available at OpenCV StereoBM). The results are presented in a 1x3 subplot, where the first image refers to the original disparity, the second corresponds to the SGBM output, and the third shows the result obtained using the OpenCV implementation. This comparison is illustrated in Figure 3 to 5.

Similarly, for the 3D reconstruction, we visualized the reconstructed 3D scenes using both **SGBM and the OpenCV built-in function**. We tested the reconstruction on three datasets: **two toy datasets and the KITTI dataset** . This comparison is illustrated in Figure 6 to 8.

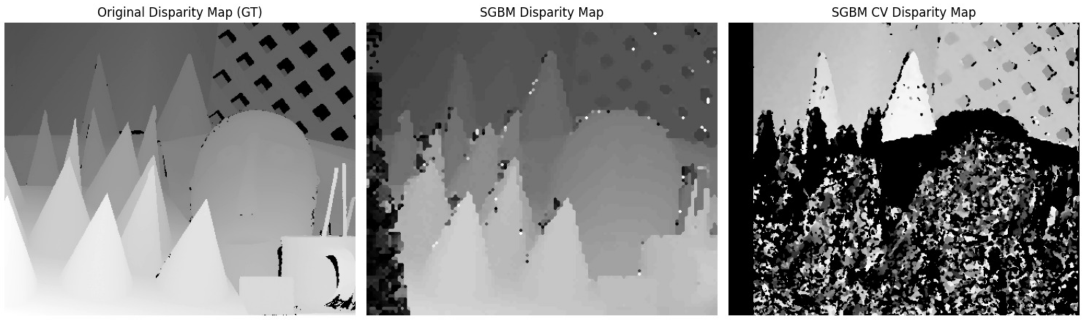

Fig. 3. Observe the **occlusions** in the OpenCV result, which occur due to **missing regions** that it tries to fill. The occlusions are significantly reduced in our own implementation of SGBM. Although the disparity map produced by SGBM appears slightly worse than original ground truth in terms of quality, its **computational efficiency is much higher**. For the OpenCV implementation, we used the following parameters: **min_disp = 0, num_disp = 64, block_size = 4, P1 = 10, P2 = 120.**

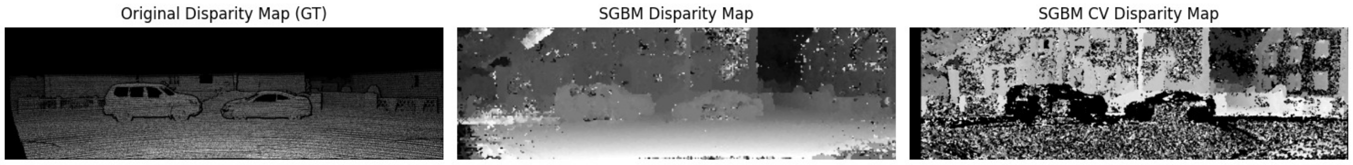

Fig. 4. Results obtained on the **KITTI** dataset. Clearly, our implementation outperforms the OpenCV method, with noticeable improvements in several areas.

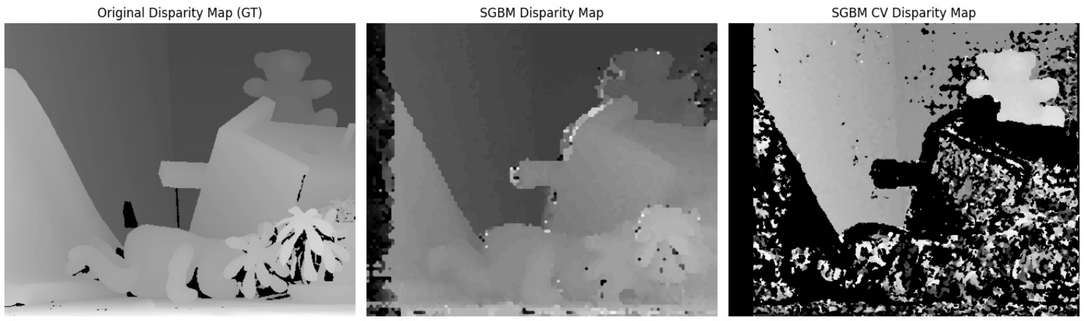

Fig. 5. This is the **disparity** comparison between Original ground truth, SGBM, and OpenCV on the KITTI dataset. Even though there is a **little bit of salt-and-pepper noise** in the SGBM output, it is **still better than OpenCV** . We used the same parameters for OpenCV as in Fig. 3.

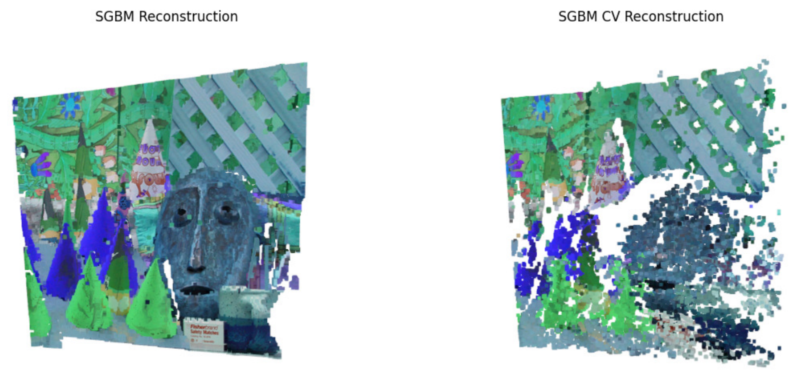

Fig. 6. This figure shows the 3D reconstructed scene of cones. The left image corresponds to the SGBM reconstruction, and the right image shows the result obtained using OpenCV.

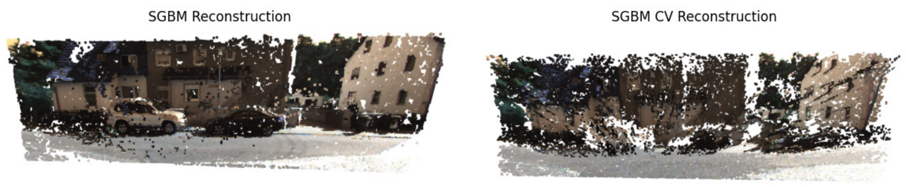

Fig. 7. This figure depicts the performance on the KITTI dataset. It is evident from the output that our **implementation clearly outperforms OpenCV**.

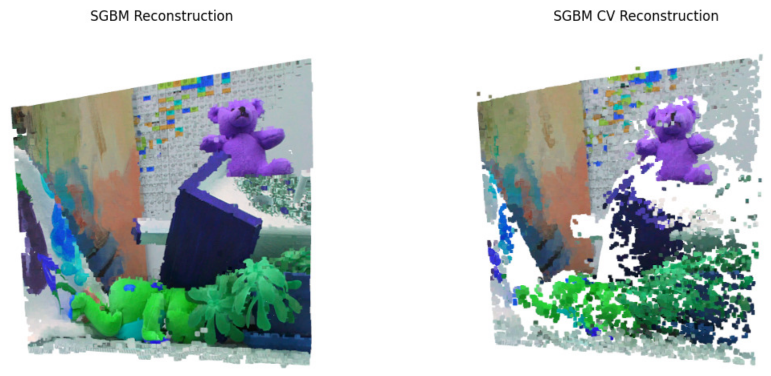

Fig. 8. This figure shows the performance on the top dataset. Even in this case, our **implementation handles occlusions better**, while OpenCV clearly shows occlusions in the output.

IX. Performance Metrics

In this case, we used a **grid size of 4** for the SGBM algorithm.

A. Test 1: Cones Dataset

This dataset is available in CONES.

Algorithm	Disparity Map MSE	Time (s)
SGM	23.81	42.71
SGBM	24.53	17.37

TABLE I
Performance comparison of SGM and SGBM on the Cones Dataset.

B. Test 2: Teddy Dataset

This dataset is available in TEDDY. The Teddy dataset is **more complex with challenging features to detect**, leading to lower accuracy for both methods compared to the Cones dataset.

Algorithm	Disparity Map MSE	Time (s)
SGM	22.31	44.03
SGBM	23.56	20.34

TABLE II
Performance comparison of SGM and SGBM on the Teddy Dataset.

X. Contributions

1. **Aniruth Suresh**: Contributed to the implementation of the SGBM algorithm and assisted in preparing the report and also assisted in the 3-D reconstruction phase.

2. **Aryan Garg**: Led the setup and core implementation of the SGBM algorithm and the 3D reconstruction, and contributed to drafting the pipeline and results sections of the report.

3. **Pratyush Jena**: Primarily focused on the DSO project but provided support in debugging and troubleshooting during the SGBM implementation

XI. Troubleshooting

During the implementation and setup of Semi-Global Block Matching (SGBM), we encountered several challenges. Here are a few of the major issues:

1. **Disparity Discrepancy in OpenCV**: While benchmarking, the OpenCV left disparity map appeared accurate, but the **right map was significantly flawed**. After investigation, we realized that different values for `min_disp` were needed for left and right disparities, as the right axis shifts negatively. We resolved this by setting `min_disp` to **0 for the left disparity and -64 for the right**, which corrected the issue.

2. **Interpolation Methods**: We experimented with various interpolation methods such as cv2.INTERPOLATE and SCIPY_ZOOM. However, these methods were either **computationally expensive or produced unsatisfactory results**. As a solution, we developed a **custom function** that mapped a smaller dimension to a larger dimension by **replicating pixel values into blocks within the larger image**, achieving acceptable quality and efficiency.

XII. Conclusion

In this project, we successfully implemented the **Semi-Global Block Matching (SGBM)** algorithm, exploring the impact of different kernel sizes on performance. We also **benchmarked SGBM** against the traditional Semi-Global Matching (SGM) algorithm and the built-in OpenCV SGBM function, demonstrating SGBM's computational efficiency and effectiveness.

Additionally, we successfully **reconstructed the 3D scene** and benchmarked our results against OpenCV's implementation. The comparison clearly shows that **our implementation outperforms the existing OpenCV** code in terms of accuracy and handling occlusions.

References

[1] Semi-Global Matching – Motivation, Developments and Applications HEIKO HIRSCHMÜLLER, Oberpfaffenhofen 1
[2] MathWorks 2
[3] An Introduction to the SGM Algorithm for Dense Matching in Binocular Stereo -Avinash Kak , Purdue University 3
[4] KITTI Dataset KITTI