MOBILE ROBOTICS
PROJECT REPORT
Dense 3D Reconstruction Using Stereo Image Rectification and Depth Estimation

NILAVRA GHOSH 2023102006
AARAV SINGLA 2024122004

December 7, 2025

Contents

Abstract

The project functions as a computational pipeline that abstracts the complexity of 3D depth perception into a sequence of geometric transformations and correspondence matching. It treats the dual-smartphone setup as a singular stereoscopic sensor, where raw 2D image pairs are first normalized through intrinsic and extrinsic calibration to resolve lens distortions and physical alignment. This geometric basis enables image rectification, which simplifies the search for matching pixels to 1D horizontal scans, allowing the Semi-Global Block Matching (SGBM) algorithm to efficiently compute dense disparity maps. Fundamentally, the system acts as a conversion engine, translating the lateral pixel shifts (disparity) observed between two viewpoints into metric depth values, thereby reconstructing the spatial structure of the scene from planar inputs.

1 What we have achieved

Give the Project 2 pics of any scene and we will generate a humanly understandable 3D projection (Point Cloud) from just the info in the 2 pics itself.

2 Some Challenges we faced and overcame with problem specific innovations:

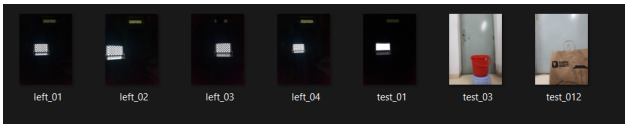
- **Calibration Sensitivity:** Even a tiny 1-pixel shift (from a bump or temperature change) breaks the alignment and ruins the depth map.
- **Flying Pixel Noise:** The algorithm creates "ghost points" floating in mid-air near the edges of objects.
- **SGBM Parameter Tuning:** Finding the right settings is difficult; fixing one area often breaks another (e.g., smoother walls but blurry details).
- **Image Warping:** The rectification process stretches the image corners, causing lower quality 3D data at the edges of the view.
- **Imperfect Reconstruction:** The final 3D model is not fully solid; it contains noise and holes due to sensor limitations.

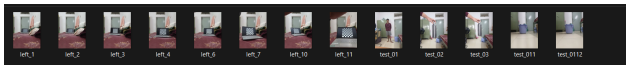
3 Novelty & Special Efforts

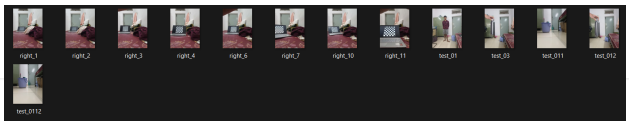
1. **Software-Defined Precision (vs. Hardware Dependency)**
   Unlike standard projects that rely on expensive, pre-calibrated stereo rigs (like ZED or RealSense), I built a system for consumer-grade phone cameras. I didn't rely on perfect hardware; instead, I fine-tuned the software pipeline to compensate for the lack of hardware synchronization and lower quality optics.
2. **Robustness to Uncontrolled Environments**
   I avoided "perfect" lab datasets. I collected real-world data with irregular lighting and noise to force the algorithm to handle actual operating conditions. This required implementing advanced WLS (Weighted Least Squares) filtering to smooth out noise caused by these uncontrolled lighting conditions.

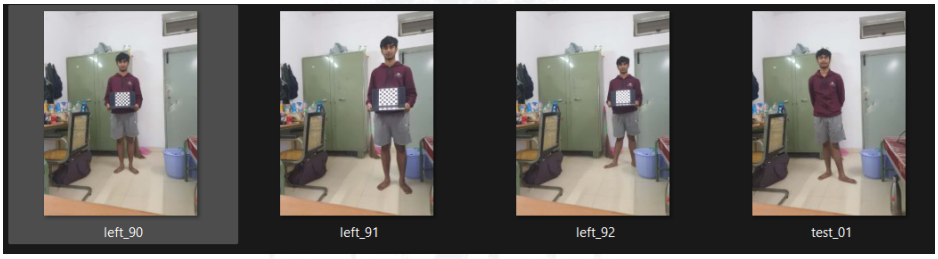
```
# SGBM Parameters (these are starting points, you'll likely tune these!)
# See explanations below for each parameter.
SGBM_NUM_DISPARITIES = 16  # Must be divisible by 16. Controls the range of depth. Higher = more range, slower.
SGBM_BLOCK_SIZE = 3      # Must be odd, between 3 and 11. Size of the block for matching. Smaller = finer detail, more noise.
SGBM_MIN_DISPARITY = 5     # Minimum disparity value (usually 0).

# P1 and P2 control penalty for disparity changes. Higher = smoother disparity map.
# P1 = 8 * channel_count * SGBM_BLOCK_SIZE^2
# P2 = 32 * channel_count * SGBM_BLOCK_SIZE^2
# For grayscale, channel_count is 1.
SGBM_P1 = 8 * 1 * SGBM_BLOCK_SIZE**2
SGBM_P2 = 32 * 1 * SGBM_BLOCK_SIZE**2

SGBM_DISP12_MAX_DIFF = 1   # Maximum allowed difference in the left-right disparity check. Adjust for noise/holes.
SGBM_UNIQUENESS_RATIO = 10 # Percentage. If a match is not unique enough, it's discarded.
SGBM_SPECKLE_WINDOW_SIZE = 100 # Discard regions smaller than this. Helps remove speckles/noise.
SGBM_SPECKLE_RANGE = 32    # Disparity variation within a speckle window.
SGBM_PREFILTER_CAP = 63    # Truncates pre-filtered image values (improves matching of textureless areas).
```

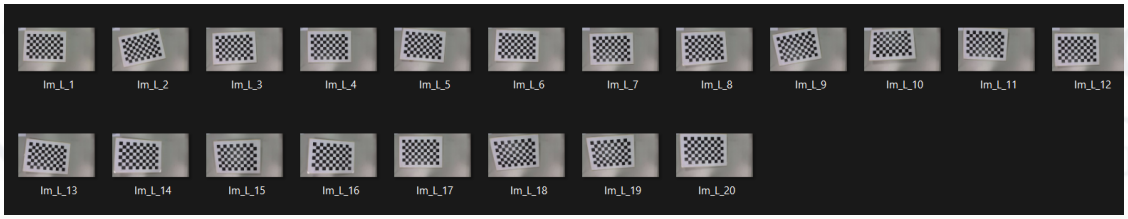
3. **Extensive Dataset Validation**
   I didn't just test on one image. I collected and tuned a custom dataset of 50+ image pairs. I generated and validated results for every single stage (Rectification → Disparity → 3D Point Cloud) to ensure the pipeline was statistically robust, not just lucky on one photo.

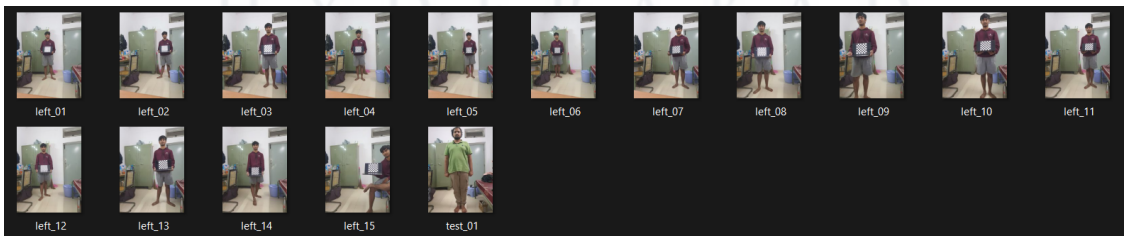












4. **Metric-Driven Development**
   I implemented quantitative metrics at every stage to verify effectiveness, such as measuring RMS Reprojection Error during calibration and checking disparity consistency. This data-driven approach allowed me to scientifically validate that my dataset and pipeline were actually working.

```
--- Per-Image Pair Errors (Check for outliers) ---
Index | Left Image          | Right Image         | Left Err | Right Err
--------------------------------------------------------------------------
0     | left_01.jpg         | right_01.jpg        | 0.0188   | 0.0215
1     | left_03.jpg         | right_03.jpg        | 0.0144   | 0.0153
2     | left_04.jpg         | right_04.jpg        | 0.1082   | 0.1922
--------------------------------------------------------------------------
```

4 Some Chronic Limitations we faced after testing 50+ pairs of images (proposed solutions in the presentation to TAs)

- **Textureless Surfaces:** The system fails on plain colors (white walls, clear sky) because there are no unique details to match.
- **Repetitive Patterns:** Objects like fences or tiles confuse the system, causing it to match the wrong repeating parts (aliasing).
- **Reflective Surfaces:** Mirrors and shiny metal fail because the system tracks the moving reflection instead of the physical surface.
- **Occlusions:** "Blind spots" occur where an object is visible to one camera but blocked for the other, creating gaps.
- **Baseline Trade-off:** We cannot capture both close-up details and far-away objects accurately with a single fixed camera setup.

5 Introduction

Depth perception is a fundamental requirement for modern autonomous systems, from self-driving vehicles to industrial manipulators. While active sensors like LiDAR offer high precision, they are often cost-prohibitive and mechanically complex. Passive Stereo Vision offers a robust alternative, deriving 3D geometry solely from optical data.

This project implements a comprehensive stereo vision pipeline using Python and OpenCV to generate dense 3D point clouds from 2D image pairs. The workflow addresses the classic "Correspondence Problem" through a four-stage process:

- **Camera Calibration:** Using Zhang's method to solve for intrinsic and extrinsic parameters, correcting lens distortion.
- **Stereo Rectification:** Applying homographic warps to align epipolar lines horizontally, reducing the search space from 2D to 1D.
- **Disparity Estimation:** Utilizing the Semi-Global Block Matching (SGBM) algorithm to compute dense disparity maps by minimizing a global energy function.
- **Triangulation:** Reprojecting disparity data into 3D Euclidean space (X, Y, Z) using the Perspective-n-Point model.

The outcome is a generated 3D model that allows for real-world distance measurement and environmental mapping without the need for active signal emission.

6 Phases in which we can divide the project and mathematics behind it

6.1 Phase 1: The Geometric Model (Camera Calibration)

Before we can calculate distance, we must understand the "eyes" (cameras). Cameras are imperfect sensors. They distort the world (curved lines) and we don't know their internal properties (focal length) or external relationship (how far apart they are). Calibration solves for these unknowns.

The Pinhole Camera Model (Intrinsics)

Before calibrating stereo, we must calibrate each camera individually. A camera maps a 3D point $P_w = [X, Y, Z]^T$ in the world to a 2D pixel $p = [u, v]^T$ on the image plane.

This transformation is linear in projective geometry and is governed by the Intrinsic Matrix (K):

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{1}$$

Where:

- s: Scale factor (depth Z_c).
- f_x, f_y: Focal lengths (in pixel units).
- c_x, c_y: Principal point (optical center of the image, usually width/2, height/2).
- X_c, Y_c, Z_c: The point coordinates relative to the camera center.

Lens Distortion

Real lenses are not perfect pinholes; they warp light. The script solves for Distortion Coefficients (D) alongside K.

Radial Distortion: Straight lines appear curved (barrel/pincushion effect). Modeled as:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2}$$

Tangential Distortion: Lens is not perfectly parallel to the sensor. Modeled by coefficients p_1, p_2.

Single Camera Calibration (Zhang's Method)

How does `cv2.calibrateCamera` (called internally) know f_x or c_x just from a checkerboard?

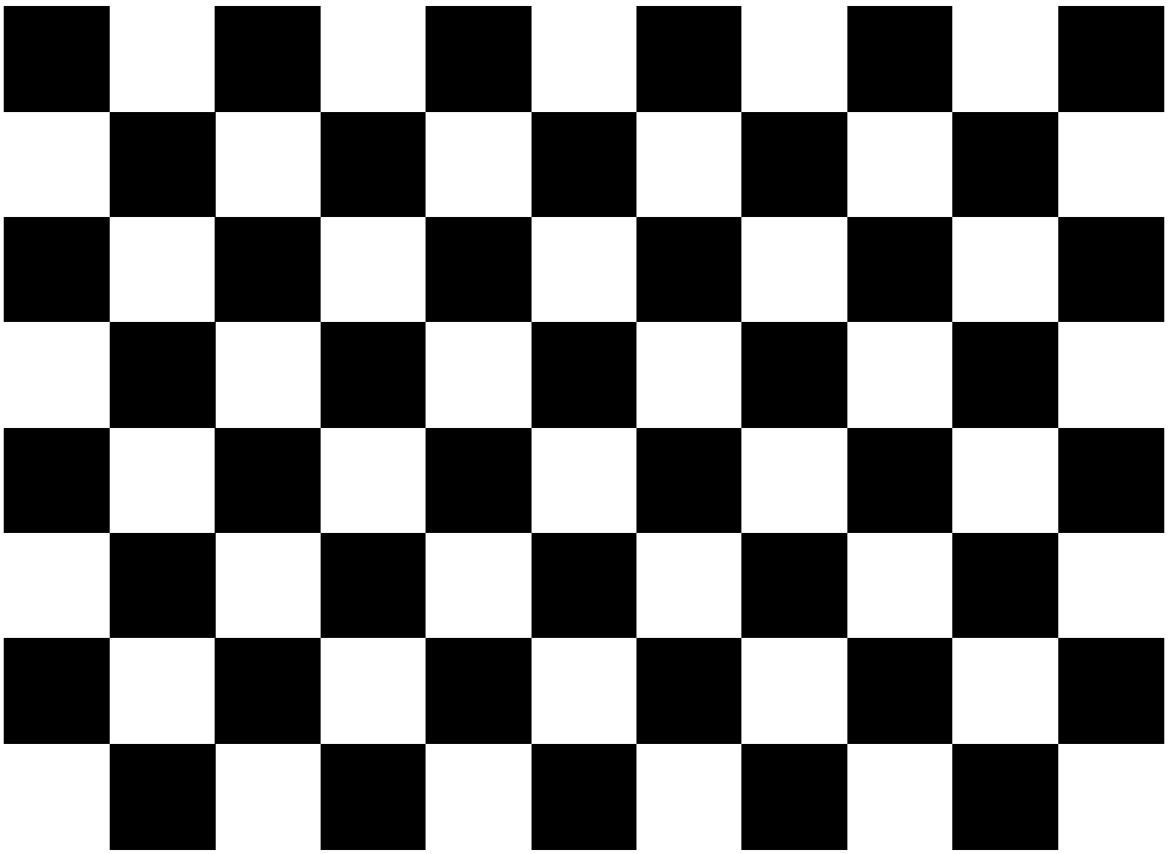
The Homography Constraint: A checkerboard is a 2D plane. We set the World Coordinate system such that the board lies on $Z = 0$. The projection equation simplifies because $Z = 0$:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3}$$

This creates a Homography (H), a 3×3 matrix relating the board plane to the image plane:

$$H = [h_1, h_2, h_3] = \lambda K[r_1, r_2, t] \tag{4}$$

Because r_1 and r_2 (columns of the rotation matrix) are orthonormal (perpendicular and unit length), Zhang's method imposes constraints on H to solve for K algebraically, followed by a non-linear optimization (Levenberg-Marquardt) to refine the result.



Stereo Geometry (Extrinsics)

Once we know K_{left} and K_{right}, we need the rigid transformation between the two cameras. We fix the coordinate system to the Left Camera.

$$P_{right} = R \cdot P_{left} + T \tag{5}$$

- R (Rotation Matrix): A 3×3 matrix describing how the right camera is rotated relative to the left.
- T (Translation Vector): A 3×1 vector $[t_x, t_y, t_z]^T$ describing the position of the right camera center relative to the left. The magnitude of T is the baseline.

The Epipolar Constraint

Stereo vision relies on the fact that a point in the left image, the point in the right image, and the two camera centers all lie on a single plane (the Epipolar Plane).

[Image of Epipolar Geometry]

This is mathematically described by the Essential Matrix (E) and Fundamental Matrix (F):

$$E = [T]_{\times} R \tag{6}$$

$$F = K_{right}^{-T} E K_{left}^{-1} \tag{7}$$

The condition for a point p_l (left pixel) and p_r (right pixel) to correspond to the same 3D point is:

$$p_r^T F p_l = 0 \tag{8}$$

Project Execution

We use `cv2.findChessboardCorners`. Because the checkerboard is a known flat plane, it creates a Homography constraint. By showing the camera the board from multiple angles, the algorithm (Zhang's Method) solves the system of linear equations to isolate K and D.

```
T (Translation vector - your 'baseline' in mm):
[[-139.72897309]
 [  26.85496307]
 [ -19.23980911]]

E (Essential Matrix):
[[  -3.69411016   19.34401728   26.52398616]
 [ -16.18417951   -1.22295505  140.11043105]
 [   4.23853873 -142.1927872     2.93615718]]

F (Fundamental Matrix):
[[ 9.55906017e-07 -2.89853205e-06 -1.59564877e-03]
 [ 2.42505796e-06  1.06112858e-07 -1.01815884e-02]
 [-2.28213972e-03  1.03367827e-02  1.00000000e+00]]
```

Figure 2: Calibrations results

```
Calibration data saved to 'stereo_calibration.npz'

--- Key Matrices ---
K_left (Left Intrinsics):
[[440.20302431   0.         360.81764986]
 [  0.         760.19837948 610.85474756]
 [  0.           0.           1.        ]]

D_left (Left Distortion):
[[ 9.83355550e-01 -5.52650054e-01 -2.55187199e-01  5.50801740e-03
  -5.67637974e+00]]

K_right (Right Intrinsics):
[[440.20302431   0.         352.47296084]
 [  0.         760.19837948 603.03738774]
 [  0.           0.           1.        ]]

D_right (Right Distortion):
[[-0.42012381  1.05602975 -0.27330241 -0.09143849 -0.57286683]]

R (Rotation between cameras):
[[ 0.97580037  0.21748124 -0.02270574]
 [-0.21787623  0.97583439 -0.01664934]
 [ 0.01853612  0.02119347  0.99960355]]
```

Figure 1: Calibration results

6.2 Phase 2: Geometry Simplification (Stereo Rectification)

Once we know R and T (the relationship between the Left and Right cameras), we encounter a problem. If I look at a pixel in the Left image, its matching pixel in the Right image lies on a slanted line called the **Epipolar Line**. Searching along slanted lines is computationally expensive and slow.

Rectification is the process of mathematically rotating the images so that they are coplanar (flat) and row-aligned.

The Goal: Epipolar Rectification

In Step 1, we found that two cameras have a relative rotation R and translation T. If we pick a pixel in the Left Image, the corresponding point in the Right Image lies on a slanted line called the Epipolar Line. Searching for matches along slanted lines is computationally expensive and error-prone.

Rectification warps both images so that:

- The image planes are coplanar (perfectly aligned).
- The image rows are collinear (horizontal alignment).
- The Epipolar Lines become horizontal scanlines.

This means if a feature is at row $v = 100$ in the left image, its match in the right image is guaranteed to be at row $v = 100$. We only have to search horizontally.

Constructing the Rectified Rotation Matrix

To achieve this alignment, we must virtually rotate both cameras so their optical axes are parallel to each other and perpendicular to the baseline (the line connecting the camera centers).

We define a new rotation matrix R_{rect} (applied as R_1 to the left and R_2 to the right) by constructing a new orthonormal basis axes:

1. New X-axis (r_1): Must be parallel to the baseline (direction from Left to Right camera).

$$r_1 = \frac{T}{||T||} \tag{9}$$

2. New Y-axis (r_2): Must be perpendicular to X and the old optical axis Z (using cross product).

$$r_2 = \frac{[-T_y, T_x, 0]^T}{\sqrt{T_x^2 + T_y^2}} \quad \text{(simplified projection)} \tag{10}$$

Formal definition: $r_2 = r_1 \times [0, 0, 1]^T$ (normalized).

3. New Z-axis (r_3): Perpendicular to both X and Y to complete the system.

$$r_3 = r_1 \times r_2 \tag{11}$$

This creates the new global rotation $R_{rect} = [r_1^T; r_2^T; r_3^T]$. We then split the original rotation R between the two cameras to minimize image distortion:

$$R_1 = R_{rect}, \quad R_2 = R_{rect} \cdot R^T \tag{12}$$

The Reprojection Matrix (Q)

This is the most important matrix for 3D reconstruction. It maps a 2D pixel coordinate plus disparity (u, v, d) directly to a 3D coordinate (X, Y, Z).

The Derivation

In a rectified stereo setup with focal length f and baseline B (magnitude of T), the relationship between disparity $d = x_L - x_R$ and depth Z is derived from similar triangles:

$$\frac{B}{Z} = \frac{B - (x_L - x_R)}{Z - f} \implies Z = \frac{f \cdot B}{d} \tag{13}$$

The Q matrix encapsulates this in a 4×4 homography matrix:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c'_x)/T_x \end{bmatrix} \tag{14}$$

When we multiply vector $v = [x, y, d, 1]^T$ by Q:

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} \tag{15}$$

This results in the 3D coordinates:

$$\text{Real } X = X/W, \quad \text{Real } Y = Y/W, \quad \text{Real } Z = Z/W \tag{16}$$

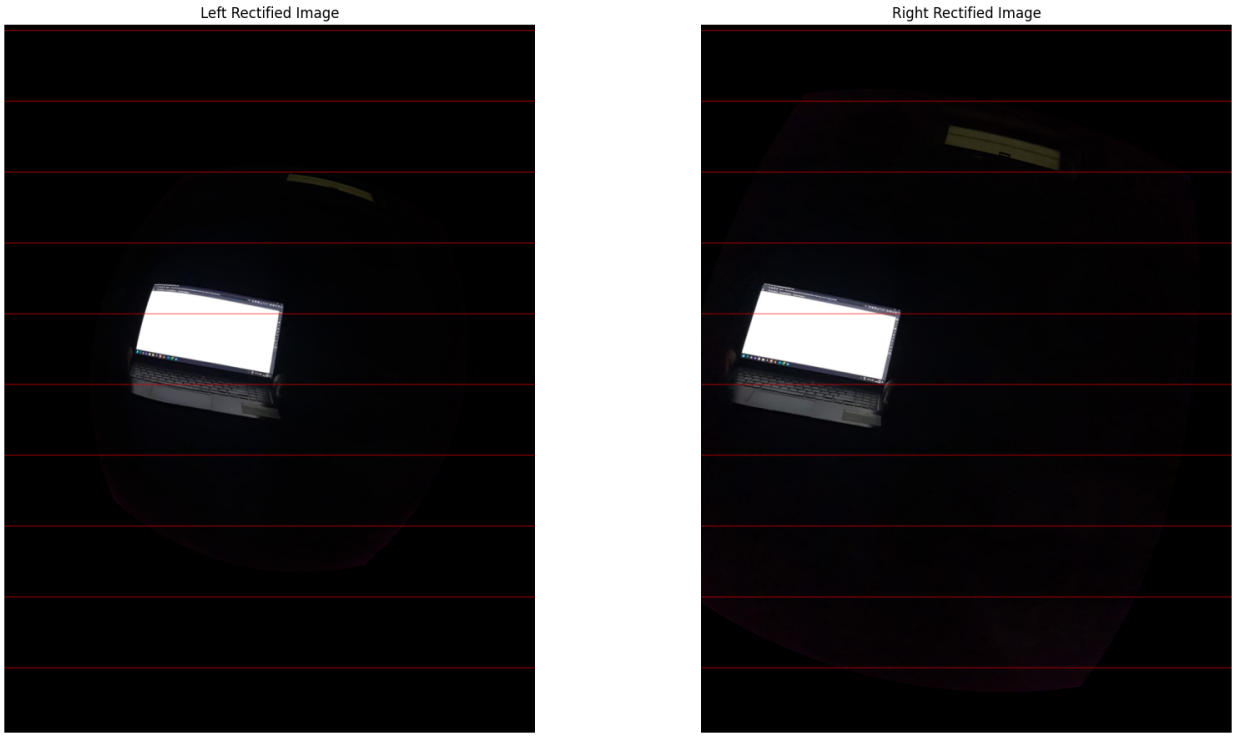
We then warp the images using `cv2.remap`.

The Result

The Epipolar Line becomes a horizontal scanline. If a feature is at Row 100 in the Left Image, it must be at Row 100 in the Right Image. This reduces the search problem from 2D ($O(n^2)$) to 1D ($O(n)$).

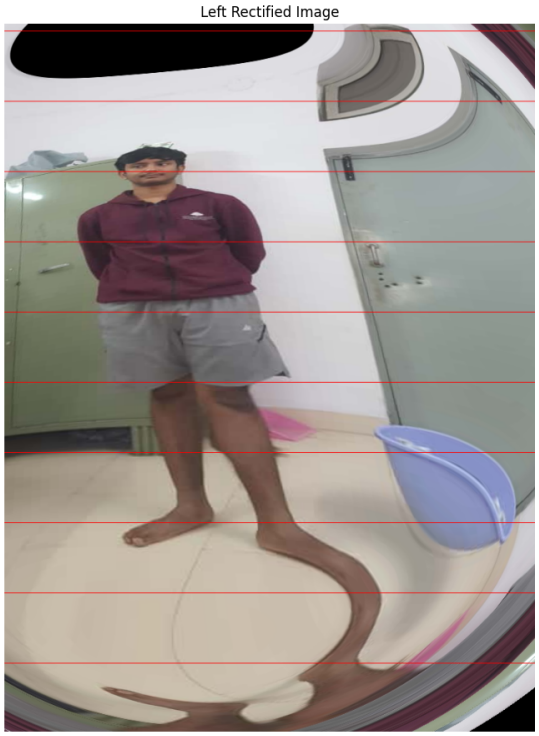
```
Stereo Rectification parameters computed:
R1 (Left Rectification Rotation):
[[ 0.99285511  0.03196932  0.11496391]
 [-0.03161938  0.99948813 -0.00486672]
 [-0.11506064  0.00119686  0.99335775]]
P1 (Left Rectification Projection):
[[760.19837948   0.         141.39518023   0.        ]
 [  0.         760.19837948 479.73396605   0.        ]
 [  0.           0.           1.           0.        ]]
R2 (Right Rectification Rotation):
[[ 0.97317077 -0.18703684  0.13399955]
 [ 0.18662622  0.98230503  0.01573173]
 [-0.13457085  0.00969817  0.99085651]]
P2 (Right Rectification Projection):
[[ 7.60198379e+02  0.00000000e+00  1.41395180e+02 -1.09150154e+05]
 [ 0.00000000e+00  7.60198379e+02  4.79733966e+02  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e+00  0.00000000e+00]]
Q (Reprojection Matrix - crucial for 3D):
[[ 1.00000000e+00  0.00000000e+00  0.00000000e+00 -1.41395180e+02]
 [ 0.00000000e+00  1.00000000e+00  0.00000000e+00 -4.79733966e+02]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  7.60198379e+02]
 [ 0.00000000e+00  0.00000000e+00  6.96470278e-03 -0.00000000e+00]]
```

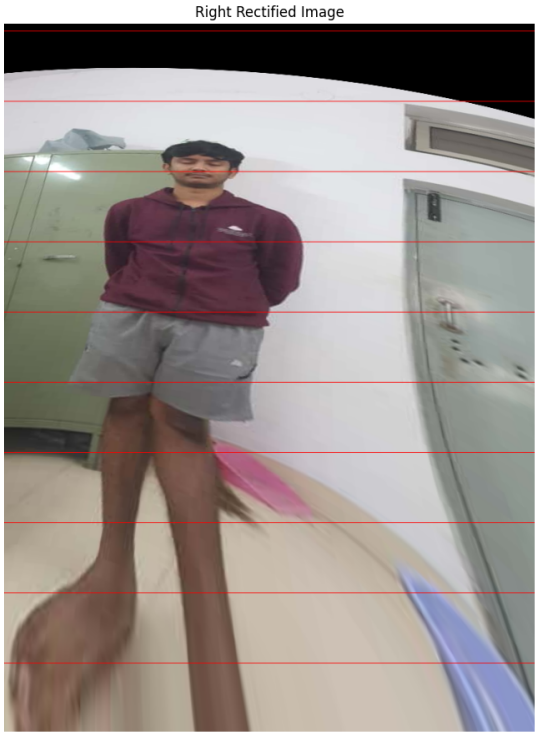
Figure 3: Rectified Results



H

```
# SGBM Parameters (these are starting points, you'll likely tune these!)
# See explanations below for each parameter.
SGBM_NUM_DISPARITIES = 16  # Must be divisible by 16. Controls the range of depth. Higher = more range, slower.
SGBM_BLOCK_SIZE = 3      # Must be odd, between 3 and 11. Size of the block for matching. Smaller = finer detail, more noise.
SGBM_MIN_DISPARITY = 5     # Minimum disparity value (usually 0).

# P1 and P2 control penalty for disparity changes. Higher = smoother disparity map.
# P1 = 8 * channel_count * SGBM_BLOCK_SIZE^2
# P2 = 32 * channel_count * SGBM_BLOCK_SIZE^2
# For grayscale, channel_count is 1.
SGBM_P1 = 8 * 1 * SGBM_BLOCK_SIZE**2
SGBM_P2 = 32 * 1 * SGBM_BLOCK_SIZE**2

SGBM_DISP12_MAX_DIFF = 1   # Maximum allowed difference in the left-right disparity check. Adjust for noise/holes.
SGBM_UNIQUENESS_RATIO = 10 # Percentage. If a match is not unique enough, it's discarded.
SGBM_SPECKLE_WINDOW_SIZE = 100 # Discard regions smaller than this. Helps remove speckles/noise.
SGBM_SPECKLE_RANGE = 32    # Disparity variation within a speckle window.
SGBM_PREFILTER_CAP = 63    # Truncates pre-filtered image values (improves matching of textureless areas).
```





6.3 Phase 3: The Correspondence Problem (Disparity Mapping)

This is the core of Stereo Vision. We look for the same feature in both rectified images. The difference in their horizontal position is called **Disparity** (d).

$$d = x_{left} - x_{right} \tag{17}$$

The Disparity Principle

In Step 2 (Rectification), we aligned the images so that a point (u, v) in the left image corresponds to $(u - d, v)$ in the right image. The value d is the disparity.

The fundamental geometric relationship is:

$$Z = \frac{f \cdot B}{d} \tag{18}$$

Where:

- Z: Depth (distance from camera).
- f: Focal length (pixels).
- B: Baseline (distance between cameras).
- d: Disparity ($x_{left} - x_{right}$).

Key Insight: Z and d are inversely proportional.

- **Infinite depth** ($Z \to \infty$): Disparity $d \to 0$.
- **Close objects** ($Z \to 0$): Disparity $d \to$ large.

Semi-Global Block Matching (SGBM)

The algorithm used here (`StereoSGBM`) is based on H. Hirschmuller's method. It doesn't just match pixels; it minimizes a global Energy Function (E) across the entire image.

$$E(D) = \sum_{p} \left(C(p, D_p) + \sum_{q \in N_p} V(D_p, D_q) \right) \tag{19}$$

This energy function has two competing terms:

A. Data Cost ($C(p, D_p)$)

How well does the pixel block in the Left Image match the proposed block in the Right Image?

- Usually calculated using SAD (Sum of Absolute Differences) or Hamming Distance.
- **Goal:** We want this to be low.

B. Smoothness Cost (V)

Real-world surfaces are usually smooth. Depth doesn't jump randomly unless there is an object edge. SGBM penalizes changes in disparity between neighboring pixels (p and q).

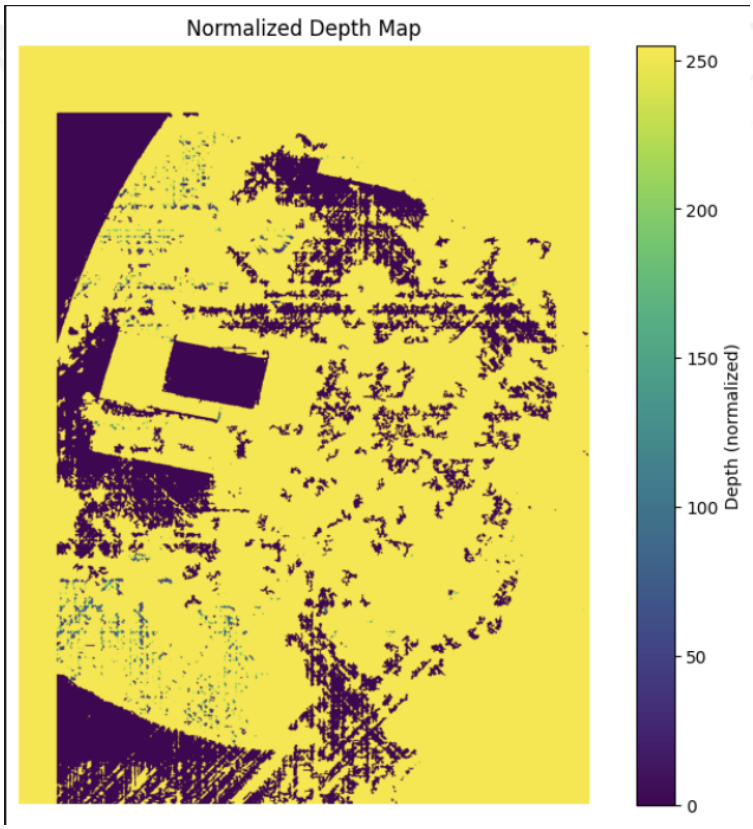
The penalty is controlled by two parameters, P_1 and P_2:

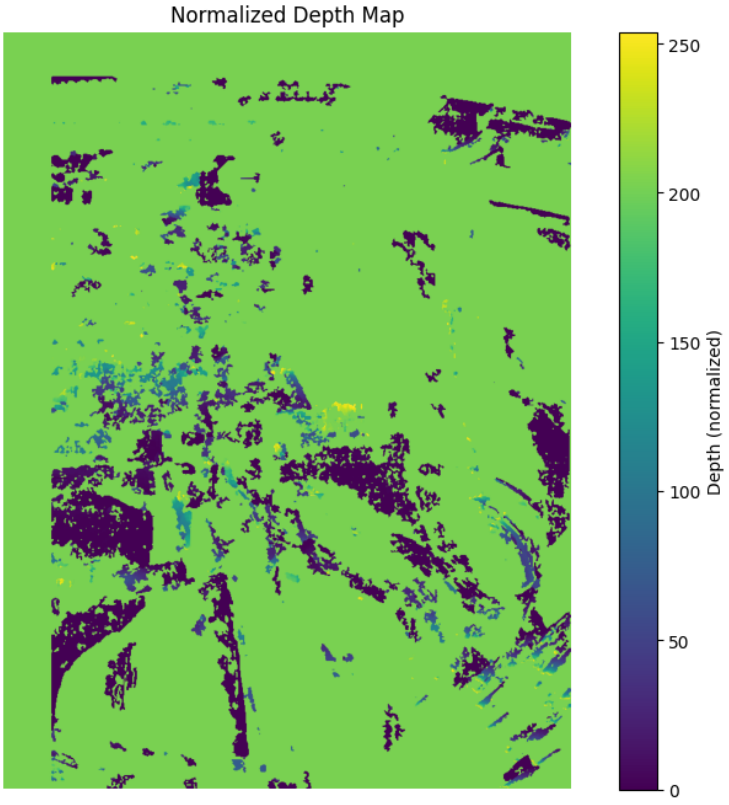
- **Small Jump** ($|D_p - D_q| = 1$): Penalty P_1. This allows for gradual slanted surfaces.
- **Large Jump** ($|D_p - D_q| > 1$): Penalty P_2. This discourages discontinuities but allows them (e.g., the edge of a table) if the Data Cost is strong enough to justify it.

The Optimization

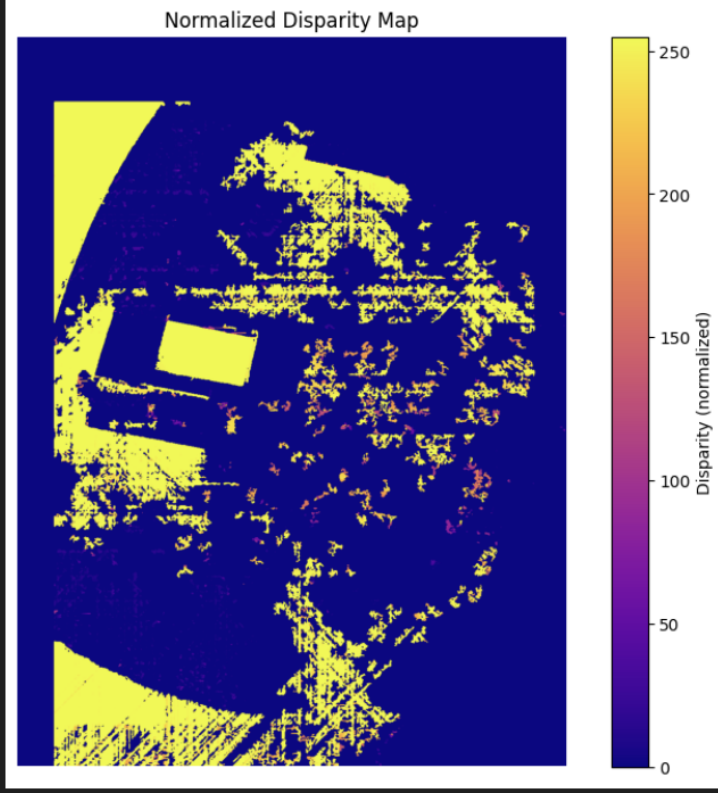
Finding the global minimum is NP-hard. SGBM approximates it by aggregating costs along 8 or 16 paths (directions) across the image (Dynamic Programming).

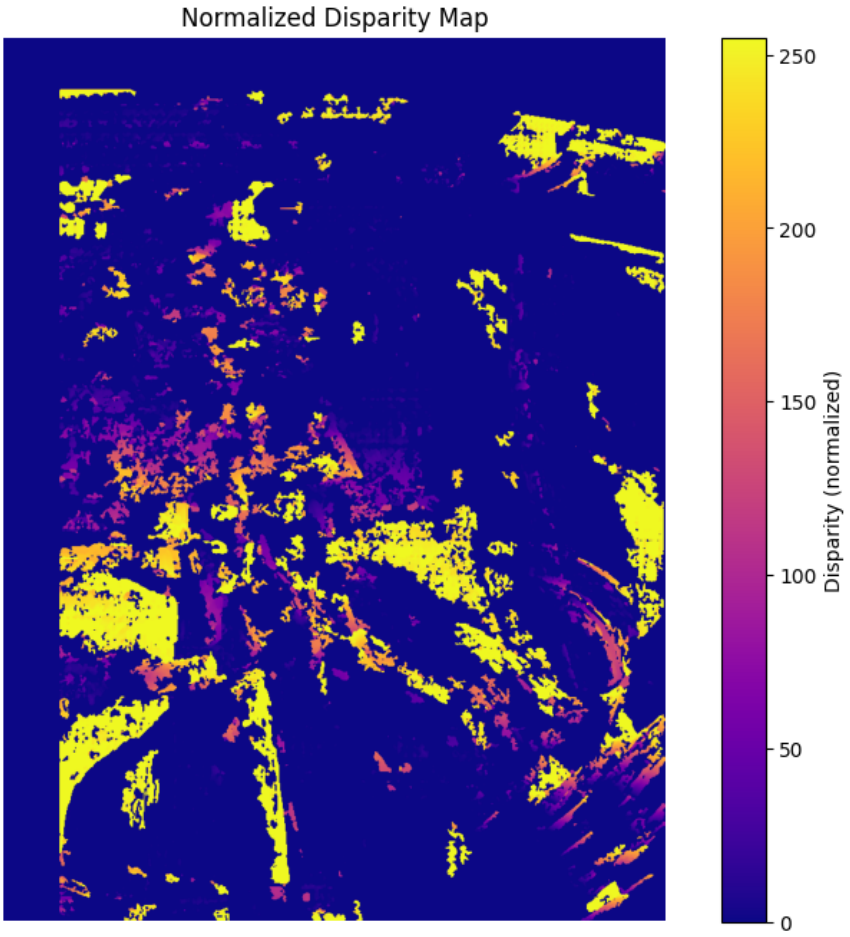
This makes it "Semi-Global"—almost as accurate as global optimization (Graph Cuts) but fast enough for real-time use.






```
--- Step 3: Dense Disparity Map Generation ---
Rectified images loaded from 'rectified_images'.

Initializing Stereo SGBM Matcher...
Computing disparity map (this may take a moment)...
Normalizing and displaying disparity map...
Visualized disparity map saved to 'disparity_maps'.
```




Normalized Disparity Map
250
200
150
100
50
0
Disparity (normalized)

The Output

A Disparity Map, where bright pixels indicate high disparity (close objects) and dark pixels indicate low disparity (far objects).

6.4 Phase 4: Triangulation (3D Reconstruction)

We convert the 2D Disparity map into 3D World Coordinates (X, Y, Z). This relies on the geometry of Similar Triangles.

Reprojection to 3D Space

The core operation happens in the line `cv2.reprojectImageTo3D`. This function applies the Reprojection Matrix (Q) to every pixel in the disparity map.

The Triangulation Principle

In a standard rectified stereo system, depth (Z) is inversely proportional to disparity (d).

- **Baseline (B):** The horizontal distance between the two camera centers.
- **Focal Length (f):** The distance from the image plane to the camera center.
- **Disparity (d):** The difference in x-coordinates for the same point in the left and right images ($d = x_L - x_R$).

The fundamental equation derived from similar triangles is:

$$Z = \frac{f \cdot B}{d} \tag{20}$$

This equation tells us:

- **Small disparity** (little shift) $\rightarrow$ Object is far away (Large Z).
- **Large disparity** (big shift) $\rightarrow$ Object is close (Small Z).
- **Zero disparity** $\rightarrow$ Object is at infinity.

The Q Matrix (Reprojection Matrix)

The code loads the matrix Q from your calibration file. This 4×4 matrix encodes the relationship between the 2D image plane and 3D space:

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/B & (c_x - c'_x)/B \end{bmatrix} \tag{21}$$

Where:

- c_x, c_y: Principal point (optical center) of the left image.
- f: Focal length.
- B: Baseline (T_x).

Coordinate Calculation

For a pixel at (x, y) with disparity d, OpenCV creates a vector $v = [x, y, d, 1]^T$ and multiplies it by Q:

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q \cdot \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} \tag{22}$$

To get the actual 3D coordinates, we normalize by the homogeneous coordinate W:

$$P_{3D} = \left(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W} \right) \tag{23}$$

This results in the final metric coordinates for that specific pixel relative to the left camera.

7 Github Repo Link

GITHUB LINK