Lab 4

Stereo Vision. Camera calibration. Depth estimation

Task 1: Camera Calibration

Objective: Calibrate stereo cameras and extract parameters.

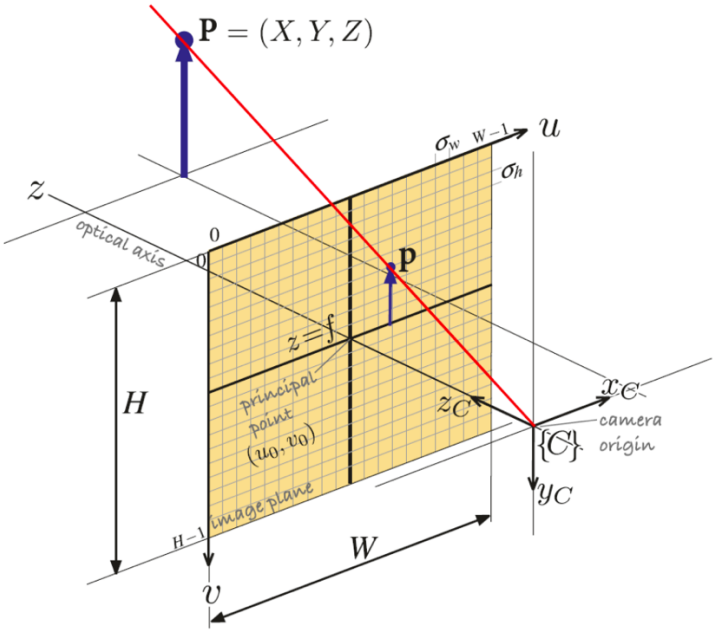
Instructions:

- Use the provided calibration dataset.
- Perform camera calibration and store parameters.
- Report intrinsic and extrinsic matrices for use in Task 2.

Expected Outcomes:

- Calibration script and saved parameters.
- Report with matrix values and brief explanation of their roles.

Camera calibration is the process of determining the unknown parameters in the equation that transforms a point from 3D space into 2D image (pixel) coordinates.



First, we will check what data we have and which parameters we will estimate during the calibration. The dataset includes the checkerboard parameters stored in a separate `.yaml` file, where we can find information about how many corners we need to detect and the distance between them. This allows us to determine the scale and the 3D object point coordinates P(X,Y,Z).

According to this information, The calibration target has a 7×6 square pattern, corresponding to 6×5 inner corners used for calibration. The size of each square between the corners is 6 by 6 centimeters. These coordinates allow us to construct the object points on the XY plane (because Z = 0 on a flat surface), for example:

(0,0,0),(0.06,0,0),(0.12,0,0),…

(0,0.06,0),(0.06,0.06,0),…

When the camera observes these corners from different angles, we can find the corresponding image points in the 2D pixel images. Now that we have both the 3D object points on the chessboard and their 2D image projections, we can use Zhang's calibration method to determine the missing parameters:

$$\tilde{\boldsymbol{p}} = \underbrace{\begin{pmatrix} f/\rho_w & 0 & u_0 \\ 0 & f/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{intrinsic}} \underbrace{\left({}^{0}\boldsymbol{T}_{C}\right)^{-1}}_{\text{extrinsic}} \tilde{\boldsymbol{P}}$$

$$s \cdot \begin{pmatrix} \dot{u}_{i,j} \\ \dot{v}_{i,j} \\ 1 \end{pmatrix} = \mathbf{A} \cdot \left(\mathbf{R}_i \mid \boldsymbol{t}_i \right) \cdot \begin{pmatrix} X_j \\ Y_j \\ Z_j \\ 1 \end{pmatrix}$$

Intrinsic parameters – focal length, principal point, and distortion coefficients - describe how the camera projects 3D rays through its lens onto the image sensor.

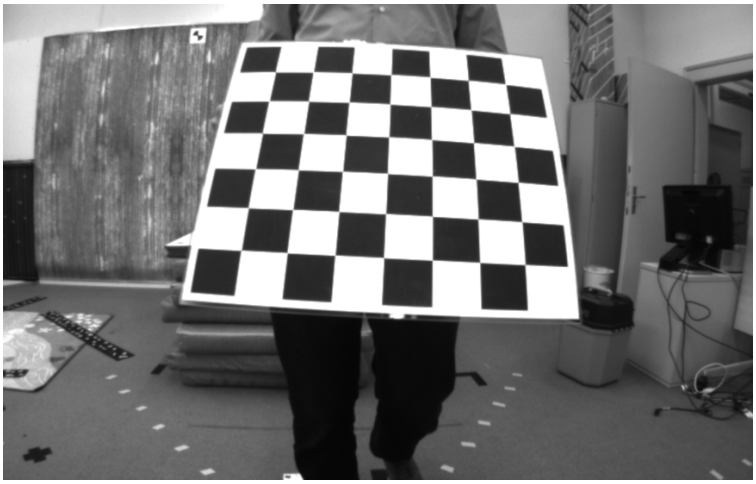
Extrinsic parameters – the camera's position and orientation relative to the chessboard in each frame - describe how the camera is rotated and shifted relative to the world.

$$s \cdot \begin{pmatrix} u_{i,j} \\ v_{i,j} \\ 1 \end{pmatrix} = \underbrace{A}_{\text{Intrinsic parameters: focal length and principal point}} \cdot \underbrace{\begin{pmatrix} R_i & t_i \end{pmatrix}}_{\text{Extrinsic parameters: camera orientation (R) and shift (t)}} \cdot \begin{pmatrix} X_j \\ Y_j \\ Z_j \\ 1 \end{pmatrix}$$

The next part of the data contains two datasets from two cameras. Each folder includes a set of `.png` images (as shown below), a `.csv` file, and a `.yaml` file with the camera parameters.

Each `.csv` file contains:

- **timestamp [ns]** : the time when the image was captured, in nanoseconds**.**
- **filename** : the name of the image file that corresponds to that timestamp.



We can see that both cameras exhibit the same type of distortion, barrel distortion, but they are placed in different positions in the room, which is visible from the different arrangements of the floor dots in their images.

Each camera folder also contains a `.yaml` file with the sensor and calibration information. This file describes what kind of sensor it is, how it is positioned relative to the robot or stereo rig, and its basic camera settings, and that the images resolution is [752, 480].

$$T_{B\to S} = \begin{bmatrix} R_{BS} & t_{BS} \\ 0\,0\,0 & 1 \end{bmatrix} \quad \text{or} \quad T = \begin{bmatrix} R_{3\times 3} & t_{3\times 1} \\ 0\,0\,0 & 1 \end{bmatrix}$$

So, our dataset already provides the **extrinsic parameters from the robot's body frame** to each camera, that is, the rotation and translation (shift) from the body to the sensor. In this task, we will estimate the rotation and translation between the two cameras using calibration images. This will allow us to obtain both the intrinsic (camera's internal) and extrinsic (camera-to-camera) matrices needed for Task 2.

A total of 2,376 image pairs were checked, and 1,649 valid pairs with visible chessboard patterns were used for calibration, because some images did not have the chessboard on the image at all. Determination **of intrinsic parameters** by a random sample 100 paired images gave the results fast, but it was interesting how long time it could take for the whole cleaned subset 1,649 paired images.

The process took 6 hours, because it keeps projecting all 3D points, comparing errors, and adjusting parameters thousands of times. Each iteration involves solving big systems of equations. The main performance bottleneck is the nonlinear least-squares optimization inside the `cv.calibrateCamera()` function, which uses the Levenberg–Marquardt algorithm. Since OpenCV performs this optimization entirely on the CPU and does not use GPU acceleration, the computation.

It took 220 minutes for the first camera and 140 minutes for the second camera to determine their intrinsic parameters. The difference in processing time is likely due to variations in how many corners were detected and how the optimizer converged, since calibration time depends nonlinearly on the number of valid detections and their spatial distribution.

Next step is to determine the **extrinsic parameters.** Even though we have already calibrated each camera individually using `cv.calibrateCamera`, OpenCV's `cv.stereoCalibrate` still needs those parameters, because it must know how each camera maps 3D points to 2D image coordinates in order to correctly estimate their relative pose (R and T).

As output, we will obtain the following matrices:

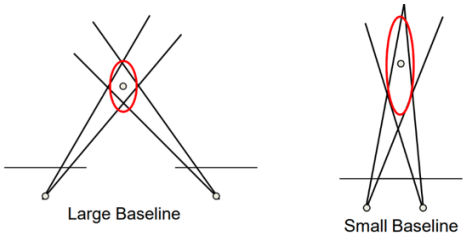
Symbol	Name	Represents	Coordinate space
R	Rotation matrix	Orientation difference between cameras	3D world
T	Translation vector	Shift (baseline) between cameras	3D world
E	Essential matrix	Combined R, T (geometry, no intrinsics)	Normalized camera coordinates
F	Fundamental matrix	Combined R, T + intrinsics	Pixel coordinates

The process took 3,66 hours and we obtained such results:

R (rotation matrix) - describes the rotation of the second camera relative to the first. It is almost an identity matrix (values around 1.0 and 0.00x), meaning the cameras are nearly parallel, exactly what is expected in a stereo setup.

T (translation vector) — shows how much the cameras are shifted relative to each other. The vector [−0.109, 0.0007, −0.0009] indicates that the cameras are separated by about 11 cm horizontally, which is typical for a stereo pair.

E (essential) and F (fundamental) matrices, define the 3D geometric relationship between both cameras. E encodes the relative rotation and translation between the two cameras (in normalized coordinates), while F maps points between the two image planes. Both matrices contain valid numeric values (no NaNs), which confirms that the stereo calibration was stable and mathematically consistent.



Baseline — the physical distance between the optical centers of the two cameras. It equals 0.109 m, confirming the same 11 cm separation as derived from T.

RMS (Root Mean Square**)** is the average re-projection error in pixels**.**

$$\text{RMS} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \| p_i^{measured} - p_i^{projected} \|^2}$$

RMS measures how well the camera model can explain where the real checkerboard corners appear in the images. It is a geometric consistency check:

- Low RMS → the model accurately predicts the corner positions (0–1 px is excellent).
- High RMS → the model's predicted corners deviate more from the actual ones.

After calibration, the RMS ≈ 5.2 px, this is relatively high but still within the "acceptable" range. A 5-pixel average reprojection error corresponds to roughly 1% of the image width, which is acceptable for small-baseline stereo systems. The high RMS value is likely due to the large number of images and imperfect lighting conditions during corner detection.

For this lab, achieving a perfect RMS is not required. However, if this were a more detailed research project, we could repeat the process from the beginning, removing outliers (images without a visible chessboard), selecting the top 200 sharpest pairs using a Laplacian-based sharpness filter, and then recalculating the intrinsic and extrinsic parameters for improved accuracy. But we stop at this point and will implement the calibration results in Task 2.

Task 2: Depth Estimation from Stereo Video

Objective: Estimate depth using stereo correspondence and triangulation.

Instructions:

- Use calibration parameters from Task 1.
- Detect and match features (justify feature choice).
- Calculate depth for matched points.
- Visualize depth values on selected image pairs.
- Generate disparity and depth maps.
- Reflect on code functionality and results.

Expected Outcomes:

- Feature matching visualizations with depth annotations.
- Disparity and depth maps.
- Commentary on feature selection, depth result, and code understanding.

If we move the same stereo rig (two cameras rigidly mounted on one platform) to a different room, nothing between the cameras has changed:

- The distance between the cameras (baseline) remains the same.
- The rotation and tilt of one camera relative to the other stay the same.
- The lenses and optical properties are identical.

Therefore, the matrices K, dist, R, and T from Task 1 remain valid. What changes is only the scene being captured, not the way the cameras project the 3D world onto the image plane. That's why we can reuse the calibration from Task 1 for Task 2 without recalibrating.

```
Loaded calibration:
K0=
[[461.27295302   0.         361.31911058]
[  0.         459.75826361 247.61789372]
[  0.           0.           1.        ]]
K1=
[[459.12670346   0.         373.08564114]
[  0.         457.74340931 253.99528538]
[  0.           0.           1.        ]]
dist0= [-3.11914860e-01  1.28378549e-01  4.36021106e-05  9.41519938e-05
-2.79416919e-02]
dist1= [-3.11642114e-01  1.31979842e-01 -2.19971616e-04  4.06962852e-04
-3.14582706e-02]
R=
[[ 0.99999369  0.0027334   0.00226879]
[-0.00276694  0.99988494  0.01491455]
[-0.00222776 -0.01492074  0.9998862 ]]
T= [-0.10944116  0.00071261 -0.00093779]
Baseline (m) = 0.10944749874784203

Image size (w,h) = (752, 480)
```

$$\underbrace{\begin{pmatrix} 1/\rho_w & 0 & u_0 \\ 0 & 1/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix}}_{K} \quad \text{or} \quad \underbrace{\begin{pmatrix} f/\rho_w & 0 & u_0 \\ 0 & f/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{intrinsic}}$$

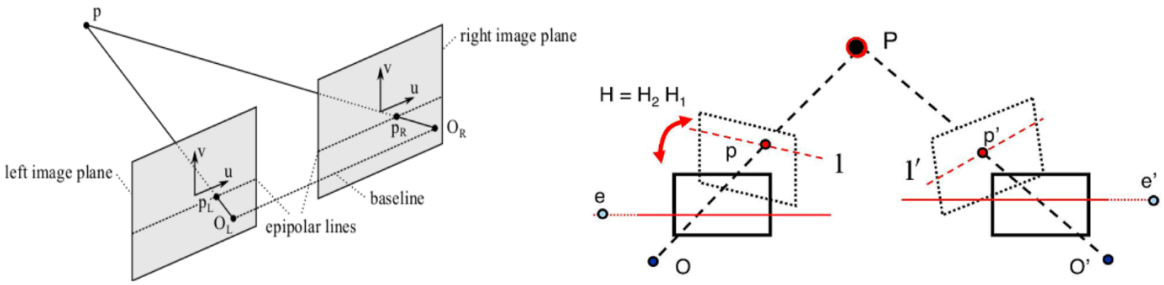
K0 and K1 are the intrinsic matrices of the cameras. They define how 3D points are projected onto the image plane. In our case, the focal length is about 460 pixels, and the principal point (optical center) is near the image center (around [361, 247]), which is excellent.

The coefficients dist0 and dist1 are the distortion coefficients. The negative value of the first coefficient indicates barrel distortion, which is typical for wide-angle lenses.

R and T are the extrinsic parameters between the two cameras. They show that the cameras are almost perfectly parallel (R ≈ identity matrix) and shifted by about 11 cm horizontally (T ≈ [−0.109, 0.0007, −0.0009]).

This configuration is ideal for a stereo camera setup, which means our stereo rig geometry (parallel axes, ~10 cm baseline, correct calibration) is well-suited for precise stereo depth estimation, according to both stereo geometry theory and OpenCV calibration best practices. Now we can take a next process, Rectification, as a step to Depth Estimation.

Rectification warps the left/right images so that corresponding scene points lie on the same image row in both views. After rectification, epipolar lines become horizontal; that makes disparity search 1-D (left↔right along rows) and greatly simplifies stereo matching.



The top image pair shows how the cameras actually "see" the scene. The bottom pair shows the same scene, geometrically corrected, now perfectly aligned for stereo depth computation.

ORIGINAL (unrectified) pair: 1403636582263555584.png

Left (original)

Right (original)

RECTIFIED pair: 1403636582263555584.png (epipolar lines are horizontal)

Left rectified

Right rectified

On the Original (Unrectified) Images, the two camera views look very similar but not perfectly aligned: same horizontal edges (like the planks) appear slightly tilted or vertically shifted between the left and right images. This means that points corresponding to the same 3D object (for example, a screw hole or the edge of a board) don't lie on the same image row — they are on different y-coordinates in each image.

On the Rectified Images, after applying the rectification: the same features are now horizontally aligned between the left and right images, for example, the edges of the wooden planks and the white net. The vertical misalignment (y-offset) is gone, meaning that the epipolar lines are now horizontal. This makes stereo matching (comparing left–right pixels) a simple 1D search along rows, which is essential for depth estimation and disparity map computation.

In other words, Task 1 camera calibration (K, dist, R, T) is consistent. Rectification maps were correctly generated and applied. The system is ready for stereo matching (ORB features, disparity, depth map). If the images had still looked tilted, shifted vertically, or with mismatched edges, it would mean calibration errors or wrong rectification parameters.

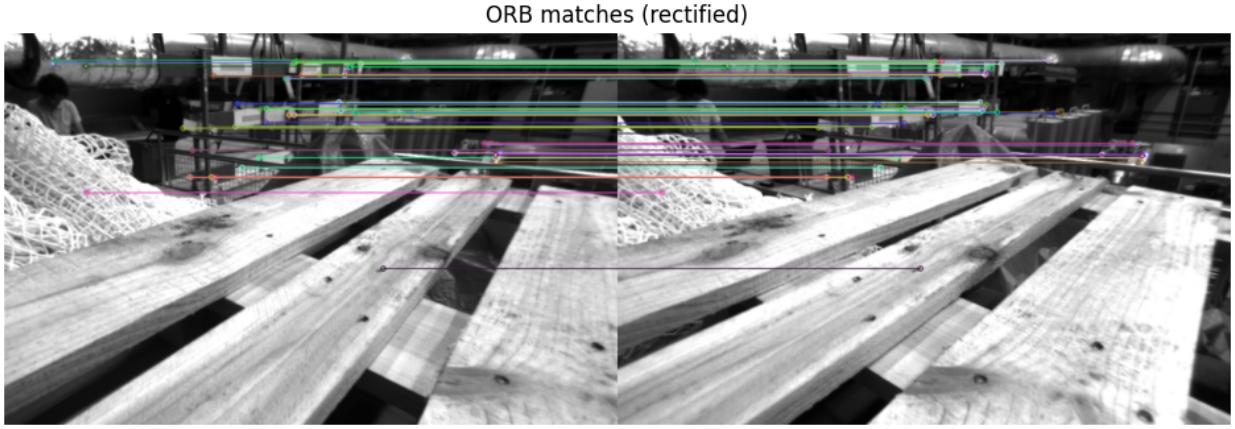
Next step is **Stereo Correspondence**. We used **ORB** (Oriented FAST and Rotated BRIEF) for detecting and matching keypoints between the rectified left and right images. We used ORB among other methods, because we only needed a handful of accurate, reliable points to compute approximate depth statistics. Dense metrics would have been too slow and unnecessary for that.

Type	Description	Examples
Sparse (feature-based)	Detect distinctive features (corners, blobs, edges) → match them between left/right images.	ORB, SIFT, SURF, AKAZE
Dense (area-based / correlation-based)	Compare every small image patch (block) between left and right → find best match using similarity metrics.	NCC, SAD, SSD, ZNCC, Census, etc.

Then we filtered matches with:

- Lowe's ratio test (0.7) --> keeps only matches that are clearly better than their second-best candidate.
- Epipolar constraint ($|yL - yR| < 2$ px) --> ensures matches are horizontally aligned after rectification.

This gave us 185 good matches that obey the stereo geometry.



After matching the points, we compute:

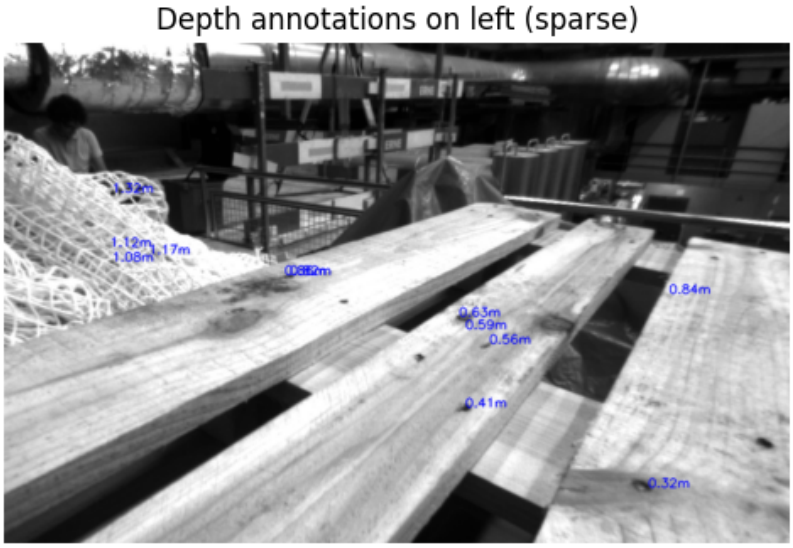
$$\text{disparity} = x_L - x_R$$

Since the images are rectified, disparity occurs only along x-axis (horizontal). Then we estimate depth for each matched point:

$$Z = \frac{f \cdot B}{\text{disparity}}$$

where:
f = focal length (≈ 460 px)
B = baseline (≈ 0.109 m)
disparity = pixel difference between matched points.

Closer objects → large disparity → smaller Z (depth).
Farther objects → small disparity → larger Z




```
Depth stats (m) on matched points:
count=12, min=0.317, median=0.834, mean=0.811, max=1.320
```

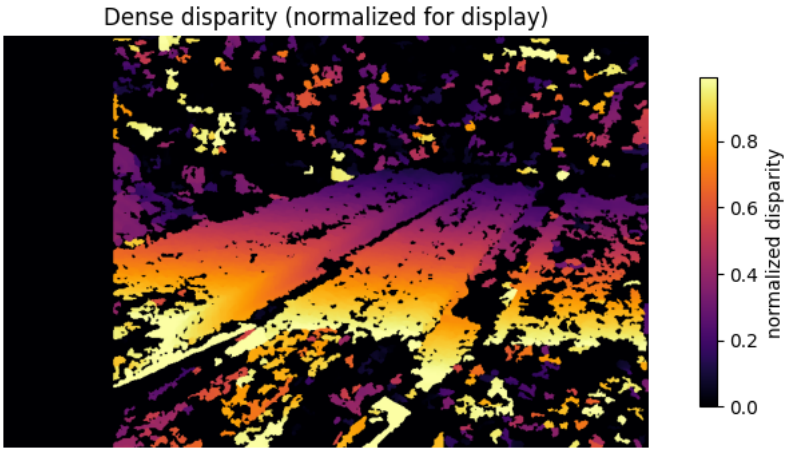
To avoid invalid or unstable results, the filter disp > 0.5 was applied. This skips matches with almost zero or negative disparity, which would otherwise produce infinite or physically meaningless depth values. Only points that have a clear horizontal parallax difference between the two images remain, typically those belonging to objects in the foreground. Only 12 valid depth points had reliable disparities with clear parallax and passed all geometric constraints, which is typical for sparse, feature-based depth estimation.

We can see that the nearest parts of the board (bottom) have the smallest depth (≈0.3-0.6 m), while background objects have larger values (≈1.2 m).

And we go to the final **Dense stereo reconstruction** step. This part computes depth for every pixel in the image, not just for a few matched feature points (like with ORB before), and it's the core of 3D reconstruction from two cameras. When two cameras look at the same scene from slightly different viewpoints, the horizontal shift is called disparity**.** And the whole goal here is to compute disparity for every pixel, and from that, infer depth everywhere.

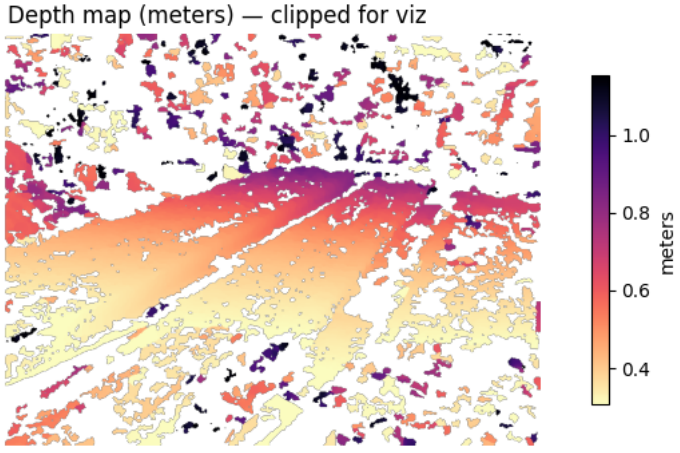
We used **StereoSGBM** (Semi-Global Block Matching), the algorithm that builds disparity maps by comparing small image blocks between left and right images:

- Divide the image into small blocks (5×5).
- Shift one image horizontally and compare each block at multiple disparity values.
- SGBM uses the absolute intensity difference cost (similar to SAD), then aggregates costs along multiple paths for semi-global optimization.
- Smooth the result globally - SGBM uses a semi-global optimization that balances accuracy and speed.
- Output: disparity map (how much each pixel moved between images).



Dense disparity map:

- Bright colors = large disparity = objects are closer to the cameras.
- Dark colors = small disparity = objects are further away.



Depth map:

- The disparity map was converted into metric depth values using the Q matrix (computed from calibration).
- Black/empty areas mean invalid or very far pixels (or where the algorithm couldn't find a reliable match).
- Bright/yellow areas are closer; darker/purple are farther.

Conclusion:

This lab demonstrates the complete stereo vision pipeline – from geometric camera calibration to dense 3D reconstruction – showing how accurate calibration directly influences the quality of depth estimation.

- Calibration parameters from Task 1 worked correctly in a new scene of Task 2.
- Rectification aligned both views accurately.
- ORB matching produced realistic depth values.
- The SGBM depth map clearly separated near and far surfaces, despite minor noise in textureless regions.