III UNIT

3D VISION AND MOTION ANALYSIS

Stereo vision is a method that uses two cameras placed a little apart to capture two different views of the same scene. By comparing these views, it calculates the depth and creates a 3D understanding of the environment.

Example:
When you look at an object with both your eyes, each eye sees it from a slightly different angle. Your brain uses these differences to judge how far away the object is. Similarly, a stereo camera system captures two images from different angles and uses the differences to find out how far objects are from the cameras.

Example of Stereo Vision:

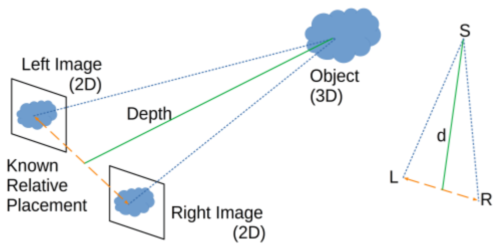
Imagine you have a stereo camera setup and you're capturing a picture of a chair in a room.

- **Camera 1** captures an image from the left side.
- **Camera 2** captures an image from the right side.

When you compare the two images, the chair appears at slightly different positions in each image. By analyzing the disparity (the shift between the two positions), you can calculate the chair's position in 3D space — how far it is from the cameras, and in which direction it is located (left, right, up, down).

How Stereo Vision Works:

1. **Two Cameras**: Two cameras (or a stereo camera setup) are placed at different horizontal positions, similar to how human eyes are positioned.
2. **Capture Images**: Each camera captures an image of the scene.
3. **Disparity Calculation**: Since the cameras are positioned at different viewpoints, the objects in the scene will appear at slightly different locations in the two images. This difference is called **disparity**.
4. **Depth Estimation**: Using the disparity between corresponding points in the two images, the depth (or distance) of those points can be calculated using triangulation. The greater the disparity, the closer the object is to the cameras.



Applications of Stereo Vision:

1. **Autonomous Vehicles**: Stereo vision helps self-driving cars understand the 3D layout of the world around them, enabling them to navigate roads and avoid obstacles.
2. **3D Mapping**: Stereo vision can be used to generate 3D models of environments for applications in robotics, archaeology, or video games.
3. **Augmented Reality (AR)**: It allows AR devices to accurately overlay virtual objects in the real world by understanding depth and spatial positioning.
4. **Robotics**: Stereo vision helps robots perform tasks like object manipulation or navigation in a 3D space.

Techniques Used in Stereovision

- Block Matching
- Semi-Global Matching
- Graph Cut Optimization
- Dynamic Programming
- Local and Global Stereo Matching Algorithms

Advantages of Stereovision

- No need for active sensors.
- Relatively low-cost with commercial stereo cameras.
- Useful in both indoor and outdoor environments.
- Good depth resolution for nearby objects.
- Can be used in real-time systems.

Applications of Stereovision

- Autonomous Vehicles for obstacle detection and navigation.
- Robotics for grasping and interaction with environments.
- Medical Imaging.
- Industrial Inspection

COMPONENTS

1. Stereo Camera Setup

- **Two Cameras (or Lenses)** – Placed at a fixed distance apart (baseline), capturing left and right images simultaneously.
- **Baseline** – The physical distance between the two cameras; a larger baseline usually improves depth accuracy for distant objects.

2. Image Acquisition

- Capturing synchronized images from both cameras.
- Cameras must be **calibrated** to ensure they are aligned and have known intrinsic parameters (focal length, distortion, etc.).

3. Camera Calibration

- **Intrinsic Parameters** – Focal length, optical center, distortion coefficients.
- **Extrinsic Parameters** – Relative position and orientation of the two cameras.
- Calibration ensures accurate mapping between 2D image points and 3D world points.

4. Rectification

- Process of aligning images so that **corresponding points lie on the same scanline** (epipolar lines become horizontal).
- Simplifies the search for matching points between left and right images.

5. Feature Matching / Correspondence

- Finding the same point (feature) in both images.
- Techniques:
  - **Block Matching** (using correlation windows)
  - **Feature-based Matching** (SIFT, SURF, ORB)
- The difference in position (horizontal shift) between corresponding points is called **disparity**.

6. Disparity Map Generation

- Creating a 2D map that shows disparity (pixel shift) for every point.
- Bright areas = closer objects (larger disparity), dark areas = farther objects (smaller disparity).

7. Depth Estimation

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

Where:

- f = focal length of camera
- B = baseline (distance between cameras)
- d = disparity

This converts 2D pixel disparity into real-world depth.

Disparity:

$$d = x_{\text{left}} - x_{\text{right}}$$

8. 3D Reconstruction / Output

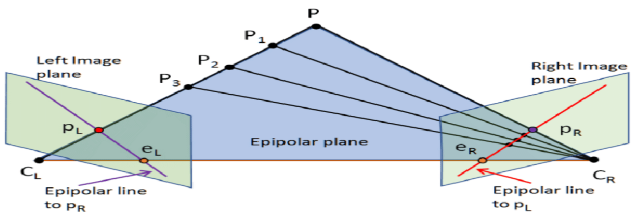
- Using depth information to build a 3D model of the scene.
- Output can be a **point cloud**, **3D mesh**, or used for applications like robot navigation, AR/VR, or object detection.

EPIPOLAR GEOMETRY

Epipolar geometry in computer visiondescribes the intrinsic geometric relationship between two images of a 3D scene taken from different viewpoints, which constrains how corresponding points in the two images can be found, Key elements include the epipolar plane, defined by the two camera centers and a 3D point; the epipoles , the image projections of the other camera's optical center; and the epipolar lines , where the epipolar plane intersects each image plane. This geometric constraint significantly reduces the search space for matching points across images

Why It's Importants

- **Reduces Search Space:** Instead of searching everywhere in the second image, we just search along the epipolar line.
- **Helps in Depth Estimation:** Epipolar geometry is the foundation for computing disparity and 3D reconstruction.
- **Used in Stereo Matching, Motion Estimation, and Camera Calibration.**



Epipolar geometry is expressed by the **Fundamental Matrix (F)**:

$$p'^T F p = 0$$

Where:

- p = coordinates of a point in the first image (in homogeneous form)
- p' = corresponding point in the second image
- F = Fundamental matrix encoding the epipolar constraint

This equation means that for every point p in the first image, its corresponding point p' in the second image must lie on the epipolar line defined by F.

Main Components

1. **Camera Centers (C and C')** – The positions of the two cameras in 3D space.
2. **Baseline** – The straight line joining the two camera centers.
3. **Epipoles (e and e')** – The projection of one camera center onto the other camera's image plane.
4. **Epipolar Plane** – A plane passing through:
   - The two camera centers
   - The 3D point being observed
5. **Epipolar Lines** – Intersection of the epipolar plane with each image plane.

Advantages of Epipolar Geometry

1. **Reduces Search Space**
   - Instead of searching the whole 2D image for a matching point, we only search along a **1D epipolar line**.
   - This makes stereo matching **faster and more accurate**.
2. **Foundation for Depth Estimation**
   - Helps compute **disparity** and then calculate **depth** for 3D reconstruction.
3. **Works with Any Scene**
   - It is purely **geometric** and does not depend on the type of objects in the scene.
4. **Enables Image Rectification**
   - Allows aligning images so corresponding points lie on the same row (simplifies stereo vision tasks).
5. **Essential for Multi-View Geometry**
   - Used in camera calibration, structure-from-motion (SfM), SLAM, and AR/VR systems.

Disadvantages / Limitations of Epipolar Geometry

1. **Sensitive to Calibration Errors**
   - Requires accurate knowledge of **camera parameters** (intrinsic & extrinsic).
   - Small errors in calibration can cause incorrect epipolar lines and bad depth estimates.
2. **Fails on Textureless Regions**
   - If an image area has no unique features (e.g., plain wall), finding correspondences along epipolar lines becomes difficult.
3. **Occlusion Problem**
   - Some points visible in one image may not be visible in the other (hidden objects), so they cannot be matched.
4. **Computational Effort for Setup**
   - Computing the fundamental/essential matrix and rectifying images requires preprocessing.
5. **Assumes Rigid Scene**
   - Works best when objects are not moving. Moving objects can break the epipolar constraint.

DISPARITY MAPPING

Disparity means the **difference in position** of the same point when viewed by two cameras (or two eyes). Disparity mapping is the process of creating a **map of these differences** for every pixel in a pair of stereo images.

Disparity mapping is a technique in 3D computer vision that uses a stereo image pair (two images taken from slightly different viewpoints) to create a depth map, where each pixel's value represents the object's horizontal displacement between the two images. This displacement, or disparity, is inversely proportional to an object's distance from the camera.

1. **Capture Stereo Images**
   - Two cameras (left & right) capture the same scene from slightly different angles.
2. **Rectify Images**
   - Align images so that epipolar lines are horizontal (makes disparity calculation easier).
3. **Correspondence Matching**
   - For each pixel in the left image, find the matching pixel in the right image along the epipolar line.
   - Methods:
     - **Block Matching**
     - **Semi-Global Matching**

- **Feature Matching (SIFT, SURF, ORB)**

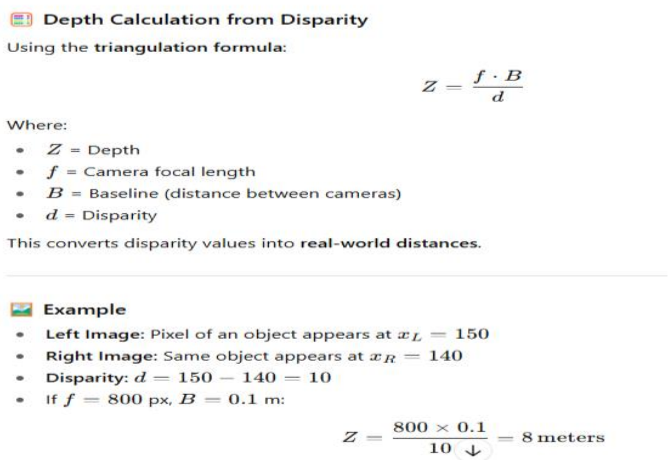
4. **Compute Disparity**

$$d = x_L - x_R$$

5. **Generate Disparity Map**
   - A 2D image where intensity represents disparity.
   - Brighter regions → closer objects (large disparity)
   - Darker regions → farther objects (small disparity)

DEPTH ESTIMATION TECHNIQUES

Depth estimation techniques in 3D involved using active sensors like LiDAR or passive sensors like cameras to recover a scene's three-dimensional geometry, Key methods include Structure from Motion (SFM) and stereo vision (requiring multiple views), depth from defocus/focus (using blur from a single image), motion-based methods like visual odometry from video, and advanced deep learning techniques that use neural networks to learn depth from single images. Active sensors like Time-of-Flight (ToF) and LiDAR provide direct depth measurements,



Depth Calculation from Disparity

Using the **triangulation formula:**

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

Where:

- Z = Depth
- f = Camera focal length
- B = Baseline (distance between cameras)
- d = Disparity

This converts disparity values into **real-world distances.**

Example

- **Left Image:** Pixel of an object appears at $x_L = 150$
- **Right Image:** Same object appears at $x_R = 140$
- **Disparity:** $d = 150 - 140 = 10$
- If $f = 800$ px, $B = 0.1$ m:

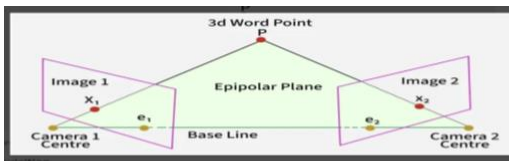
$$Z = \frac{800 \times 0.1}{10} = 8 \text{ meters}$$

STRUCTURE FROM MOTION(SfM)

Structure from Motion is a technique in computer vision used to reconstruct 3D structure of a scene and simultaneously estimate camera motion using multiple images taken from different viewpoints.

Structure from Motion (SfM) is a computer vision technique that reconstructs the three dimensional (3D) structure of a scene from a set of two-dimensional (2D) images taken from different viewpoints. It simultaneously estimates both the 3D coordinates of scene points ("structure") and the positions and orientations of the cameras ("motion") that captured the images.

Workflow of SfM



The SfM pipeline consists of steps:

1. Image Acquisition

- Multiple overlapping images of the scene are captured from different angles.
- The more overlap and variety in viewpoints, the better the reconstruction.

2. Feature Detection

- Distinctive points (features) in each image are identified using algorithms like SIFT, SURF or ORB.
- These features are robust to scale, rotation and illumination changes.

3. Feature Matching

- Features are matched across different images to find correspondences i.e., the same physical point seen in multiple images.
- Reliable matching is crucial for accurate 3D reconstruction.

4. Camera Pose Estimation

- The relative positions and orientations (extrinsic parameters) of the cameras are estimated using the matched features.
- This step often involves solving for the essential or fundamental matrix between image pairs.

5. Triangulation

- The 3D coordinates of matched features are computed by triangulating their positions from multiple views, using the known camera poses.

6. Bundle Adjustment

- Both the 3D point positions and camera parameters are refined simultaneously to minimize the reprojection error (the difference between observed and predicted feature positions in the images).
- This is a non-linear optimization problem and is key to achieving high accuracy.

7. Dense Reconstruction (Optional)

- After sparse point clouds are obtained, further algorithms can interpolate and densify the reconstruction, producing a detailed 3D model.

8. Texturing and Model Generation

- The final step may involve generating a mesh and applying textures from the original images for realistic visualization.

Applications of Structure from Motion (SfM)

1. **3D Mapping and Surveying :** Rapid creation of accurate 3D maps for land surveying, urban planning, construction monitoring and disaster assessment using drone images.

   Reconstructs accurate **3D models of real-world objects or environments** from multiple images.

   Example: Drone images of a construction site used to create a 3D site model for monitoring progress.

2. **Robotics and Navigation :** Enables autonomous vehicles and drones to understand and navigate environments by building 3D maps in real-time.

   Helps robots or drones understand their **3D environment** for navigation and obstacle avoidance.

   Example: Drones mapping a forested area to plan safe flight paths.

3. **Virtual and Augmented Reality :** Creates realistic 3D models for immersive VR experiences and accurate AR overlays in fields like architecture and cultural heritage.

   Provides realistic 3D models of real environments for immersive VR/AR experiences.

   Example: Scanning a room to create an AR scene for interior design apps.

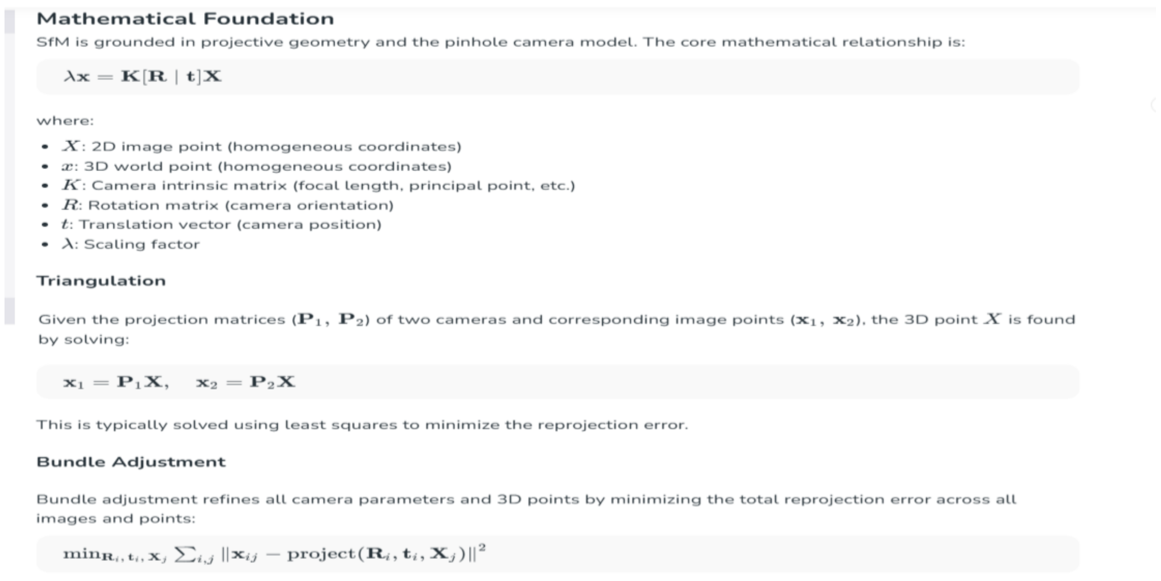
4. **Archaeology and Cultural Heritage :** Documents and digitally preserves archaeological sites and artifacts, aiding restoration and public education.

5. **Environmental Monitoring :** Tracks changes in ecosystems, supports precision agriculture, models natural disasters and assists climate research with detailed 3D data.

6. **Forensics and Crime Scene Reconstruction:**

   Reconstructs crime or accident scenes in 3D for investigation or legal analysis.

   Example: Creating a 3D model of a car accident scene from photos for court evidence.



Mathematical Foundation

SfM is grounded in projective geometry and the pinhole camera model. The core mathematical relationship is:

$$\lambda \mathbf{x} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]\mathbf{X}$$

where:

- X: 2D image point (homogeneous coordinates)
- x: 3D world point (homogeneous coordinates)
- K: Camera intrinsic matrix (focal length, principal point, etc.)
- R: Rotation matrix (camera orientation)
- t: Translation vector (camera position)
- λ: Scaling factor

Triangulation

Given the projection matrices $(\mathbf{P}_1, \mathbf{P}_2)$ of two cameras and corresponding image points $(\mathbf{x}_1, \mathbf{x}_2)$, the 3D point X is found by solving:

$$\mathbf{x}_1 = \mathbf{P}_1\mathbf{X}, \quad \mathbf{x}_2 = \mathbf{P}_2\mathbf{X}$$

This is typically solved using least squares to minimize the reprojection error.

Bundle Adjustment

Bundle adjustment refines all camera parameters and 3D points by minimizing the total reprojection error across all images and points:

$$\min_{\mathbf{R}_i, \mathbf{t}_i, \mathbf{X}_j} \sum_{i,j} \|\mathbf{x}_{ij} - \text{project}(\mathbf{R}_i, \mathbf{t}_i, \mathbf{X}_j)\|^2$$

Features across frames

It involves detecting an object in a sequence of images (or point clouds) and then predicting its location in subsequent frames. The goal is to continuously estimate the position and orientation of the object, even in the presence of occlusions, camera motion, and changing lighting

Feature tracking is the process of detecting distinctive points in images and following them across multiple frames to estimate camera motion and 3D structure.

Feature Detection

- Detect distinctive points like corners or edges in an image.
- **Algorithms:** SIFT, SURF, ORB.
- **Example:**
    - A drone captures the first image of a building. The corners of windows, roof edges, and doors are detected as **feature points**.

Feature Description

- Assign a **descriptor** to each feature to make it identifiable across frames.
- **Example:**
    - Each detected window corner gets a unique vector describing its local appearance, so it can be matched in other frames even if the camera moves.

Feature Matching

- Match features between consecutive frames based on descriptors.
- **Example:**
    - The corner of a window detected in frame 1 is found in frame 2, even though the camera has moved slightly.

Outlier Removal

- Use **RANSAC** to remove incorrect matches.
- **Example:**
    - A reflection on a glass window might be mistakenly matched; RANSAC removes this outlier to avoid errors in 3D reconstruction.

Track Features Across Multiple Frames

- Maintain the **trajectory of each feature point** across all frames.

- **Example:**
    - A drone captures 50 images while flying around a building.
    - Each tracked window corner, roof edge, or statue point is followed across all frames.
    - Triangulation converts these tracked points into 3D coordinates, forming a **3D point cloud of the building**.

3D RECONSTRUCTION FROM MOTION

3D reconstruction from motion is the process of building a 3D model of a scene using multiple 2D images captured by a moving camera. SfM simultaneously estimates camera motion and 3D structure from feature correspondences across images.

Step-by-Step Workflow

1. **Capture Images**
    - Take multiple overlapping images of a scene from different viewpoints.
2. **Feature Detection & Tracking**
    - Detect key points (corners, edges) in each image.
    - Track the same features across images.
3. **Camera Pose Estimation**
    - Estimate the **position and orientation** of the camera for each frame relative to the scene using matched features.
4. **Triangulation**
    - For each feature tracked across frames, compute its **3D coordinates** using triangulation.
5. **Bundle Adjustment**
    - Refine 3D points and camera poses simultaneously to minimize **reprojection error**.
6. **3D Reconstruction Output**
    - Result is a **sparse or dense 3D point cloud**.
    - Optional: Apply **surface reconstruction** to create a **mesh** or textured 3D model.

Advantages

- Works with a **single moving camera**.
- Can reconstruct **large-scale environments** like buildings, landscapes, or archaeological sites.
- Non-contact and **non-invasive**, ideal for cultural heritage.

Limitations

- Requires **good image overlap** and **distinctive features**.
- Textureless surfaces or motion blur can reduce accuracy.
- Sparse reconstruction initially; dense reconstruction may require extra computation.

Example:

- **Drone mapping a building:**
    1. Drone captures 50 images while flying around a building.
    2. SfM detects roof corners, window edges, and doors in each image.
    3. Features are tracked across frames to compute camera motion.
    4. Triangulation gives 3D coordinates of all features.
    5. Bundle adjustment refines the 3D points.

Applications of SfM in Scene Understanding

Scene understanding is about interpreting the 3D structure, layout, and objects in an environment from images. SfM helps by reconstructing 3D information from 2D images, which is crucial for many applications.

1.3D Environment Reconstruction:

SfM reconstructs the 3D geometry of a scene from multiple images.

- **Example: A drone captures images of a forest; SfM creates a 3D model of the trees, terrain, and pathways for analysis.**

2.Object Localization and Mapping:

By reconstructing the scene in 3D, objects can be accurately located and mapped within the environment.

- **Example: Robots navigating indoors use SfM to map furniture and obstacles for path planning.**

3.Autonomous Navigation:

SfM provides 3D spatial understanding for drones, cars, and robots to navigate safely.

- **Example: Self-driving cars reconstruct the road scene to detect lanes, vehicles, and pedestrians in 3D.**

4.Urban Planning and Architecture:

3D models of streets, buildings, and public spaces help in planning and analysis.

- **Example: SfM reconstructs a city block from aerial images to simulate building shadows or optimize layouts.**

5.Forensics and Crime Scene Analysis:

3D reconstruction of a scene allows better spatial analysis.

- **Example: SfM reconstructs an accident scene from photos to analyze distances, angles, and object positions.**

Optical Flow and Motion Analysis

Definition:

- Optical flow is the apparent motion of objects (or pixels) in an image sequence caused by the movement of the camera or objects in the scene.
- Motion analysis uses optical flow to estimate camera motion and the 3D structure of the scene.

In SfM, optical flow is often used to track features across frames, which is essential for camera pose estimation and 3D reconstruction.

Workflow in SfM Using Optical Flow

1. **Capture Image Sequence**
    - Single moving camera or video frames.
2. **Feature Detection**
    - Detect distinctive points (corners, edges).
3. **Optical Flow Computation**
    - Track movement of these points between consecutive frames.
4. **Outlier Rejection**
    - Remove incorrect matches using RANSAC or geometric constraints.
5. **Camera Pose Estimation**
    - Compute relative rotation and translation of the camera between frames.
6. **Triangulation**
    - Compute 3D coordinates of tracked features using multiple views.
7. **3D Reconstruction**

- **Build a sparse or dense 3D point cloud of the scene.**

 Optical Flow

- **Describes the motion vector of each pixel between two consecutive frames.**
- **Denoted as (u,v)(u, v)(u,v) for horizontal and vertical pixel displacement.**
- **Algorithms:**
  - **Lucas-Kanade Method – Tracks sparse points.**
  - **Horn-Schunck Method – Computes dense flow for the whole image.**

 Motion Analysis

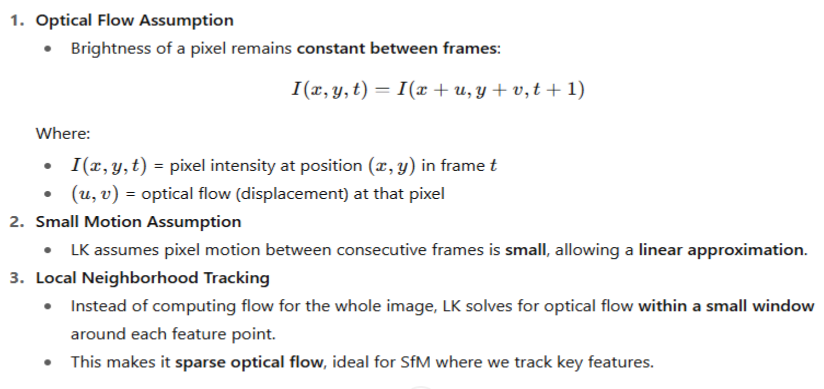
- **Uses the tracked points (via optical flow) to determine:**
  - **Camera translation and rotation between frames.**
  - **Depth estimation of scene points using triangulation.**

 Relation to SfM

- **Optical flow provides the 2D correspondences needed in SfM.**
- **SfM uses these correspondences to reconstruct 3D structure and camera trajectory.**

LUCAS-KANADE METHOD

The Lucas-Kanade (LK) method is a classical algorithm to compute optical flow — the apparent motion of pixels between consecutive frames. In SfM, it is widely used for tracking feature points across frames, which is essential for camera motion estimation and 3D reconstruction.

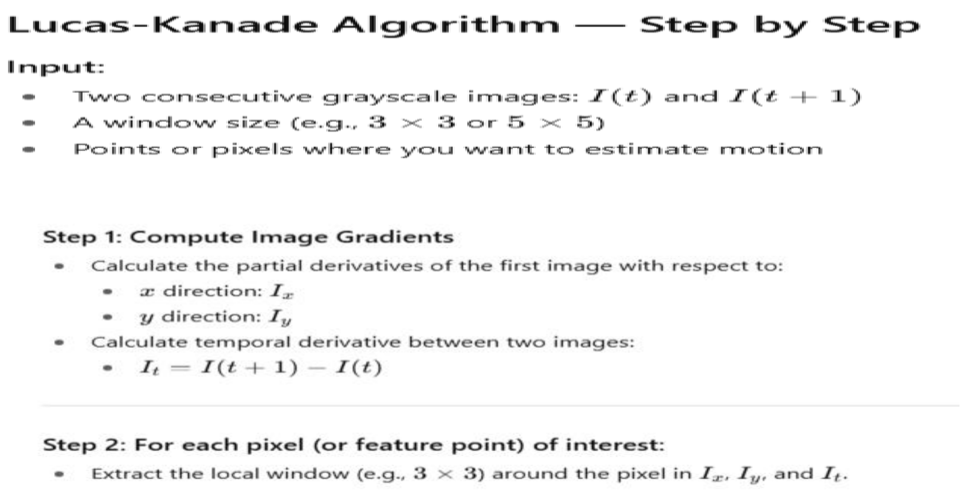


1. **Optical Flow Assumption**
   - Brightness of a pixel remains **constant between frames**:

$$I(x,y,t) = I(x+u, y+v, t+1)$$

   Where:
   - $I(x,y,t)$ = pixel intensity at position (x,y) in frame t
   - (u,v) = optical flow (displacement) at that pixel
2. **Small Motion Assumption**
   - LK assumes pixel motion between consecutive frames is **small**, allowing a **linear approximation**.
3. **Local Neighborhood Tracking**
   - Instead of computing flow for the whole image, LK solves for optical flow **within a small window** around each feature point.
   - This makes it **sparse optical flow**, ideal for SfM where we track key features.

Lucas-Kanade Algorithm:



Lucas-Kanade Algorithm — Step by Step

Input:

- Two consecutive grayscale images: $I(t)$ and $I(t+1)$
- A window size (e.g., 3×3 or 5×5)
- Points or pixels where you want to estimate motion

Step 1: Compute Image Gradients

- Calculate the partial derivatives of the first image with respect to:
  - x direction: I_x
  - y direction: I_y
- Calculate temporal derivative between two images:
  - $I_t = I(t+1) - I(t)$

Step 2: For each pixel (or feature point) of interest:

- Extract the local window (e.g., 3×3) around the pixel in I_x, I_y, and I_t.

Step 3: Formulate system of linear equations

- For every pixel i in the window:

$$I_x(i)u + I_y(i)v = -I_t(i)$$

- Stack all these equations into matrix form:

$$A = \begin{bmatrix} I_x(1) & I_y(1) \\ I_x(2) & I_y(2) \\ \vdots & \vdots \\ I_x(n) & I_y(n) \end{bmatrix}, \quad b = \begin{bmatrix} -I_t(1) \\ -I_t(2) \\ \vdots \\ -I_t(n) \end{bmatrix}$$

Step 4: Solve for (u, v) using Least Squares

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b$$

Step 5: Repeat for all pixels/points

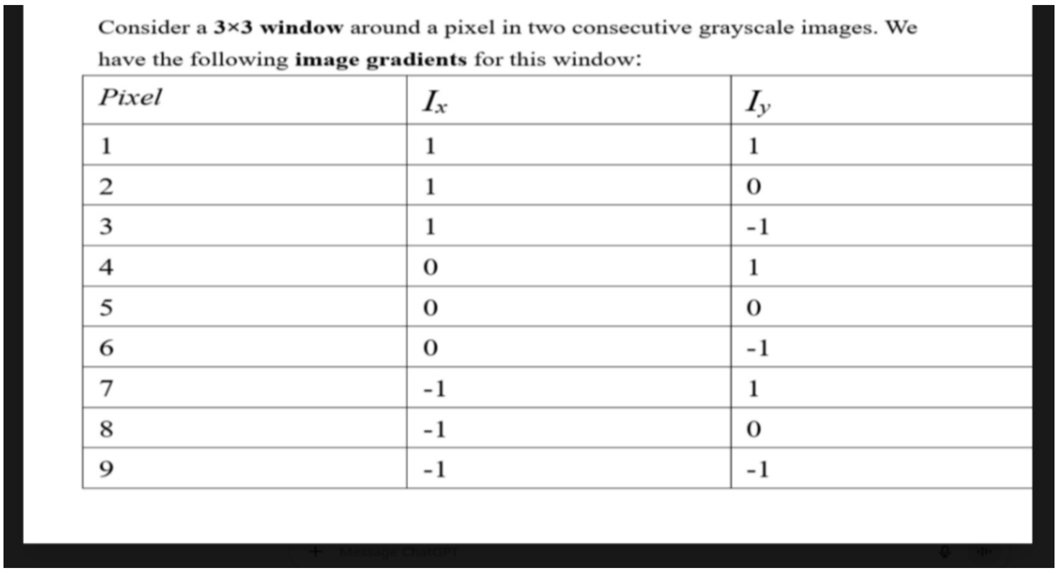
- Calculate the flow vector (u, v) for each pixel or feature point.

Step 6 (Optional): Use Image Pyramid

- If motion between frames is large, use Gaussian pyramids to estimate flow at coarse levels and refine at finer levels.

Lucas-Kanade Algorithm Step-by-Step with Example

Consider a **3×3 window** around a pixel in two consecutive grayscale images. We have the following **image gradients** for this window:



Consider a **3×3 window** around a pixel in two consecutive grayscale images. We have the following **image gradients** for this window:

Pixel	I_x	I_y
1	1	1
2	1	0
3	1	-1
4	0	1
5	0	0
6	0	-1
7	-1	1
8	-1	0
9	-1	-1

Step 1: Build matrices A and b

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & -1 \\ 0 & 1 \\ 0 & 0 \\ 0 & -1 \\ -1 & 1 \\ -1 & 0 \\ -1 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

(Note: $b = -I_t$ so signs flipped)

Step 2: Calculate $A^T A$

$$A^T A = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Calculate:

- $\sum I_x^2 = 1^2 + 1^2 + 1^2 + 0 + 0 + 0 + (-1)^2 + (-1)^2 + (-1)^2 = 6$
- $\sum I_y^2 = 1^2 + 0 + (-1)^2 + 1^2 + 0 + (-1)^2 + 1^2 + 0 + (-1)^2 = 6$
- $\sum I_x I_y = (1)(1) + (1)(0) + (1)(-1) + (0)(1) + (0)(0) + (0)(-1) + (-1)(1) + (-1)(0) + (-1)(-1) = 0$

So,

$$A^T A = \begin{bmatrix} 6 & 0 \\ 0 & 6 \end{bmatrix}$$

Step 3: Calculate $A^T b$

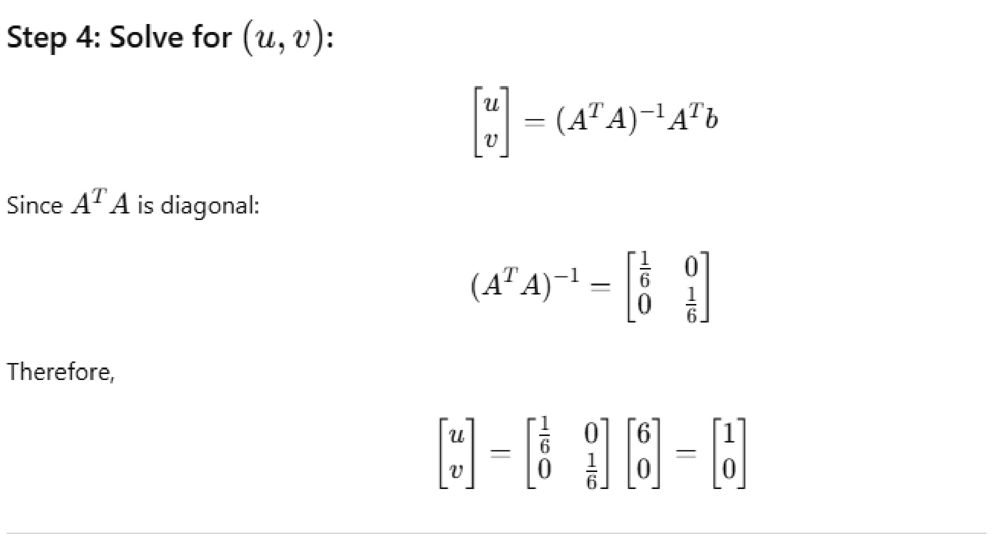
$$A^T b = \begin{bmatrix} \sum I_x b \\ \sum I_y b \end{bmatrix}$$

Calculate:

- $\sum I_x b = 1*1 + 1*1 + 1*1 + 0*0 + 0*0 + 0*0 + (-1)*(-1) + (-1)*(-1) + (-1)*(-1) = 6$
- $\sum I_y b = 1*1 + 0*1 + (-1)*1 + 1*0 + 0*0 + (-1)*0 + 1*(-1) + 0*(-1) + (-1)*(-1) = 0$

So,

$$A^T b = \begin{bmatrix} 6 \\ 0 \end{bmatrix}$$



Step 4: Solve for (u, v):

$$\begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T b$$

Since $A^T A$ is diagonal:

$$(A^T A)^{-1} = \begin{bmatrix} \frac{1}{6} & 0 \\ 0 & \frac{1}{6} \end{bmatrix}$$

Therefore,

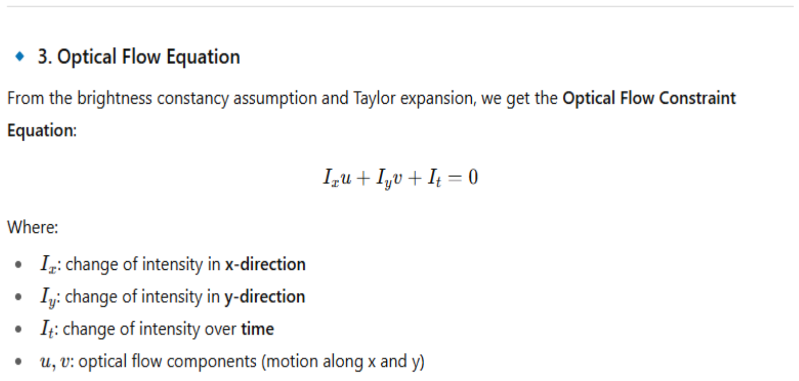
$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{1}{6} & 0 \\ 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} 6 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

RESULT: The optical flow vector at the center pixel of the window is:**(u,v)=(1,0)**

This means the pixel moves **1 unit in the x-direction** and **0 units in the y-direction** between the two frames.

HORN_SCHUNK METHOD

The Horn–Schunck method is a global optical flow estimation technique used in motion analysis and 3D reconstruction in Computer Vision and Image Processing (CVIP). It estimates the apparent motion of brightness patterns between two consecutive frames of a video or image sequence.



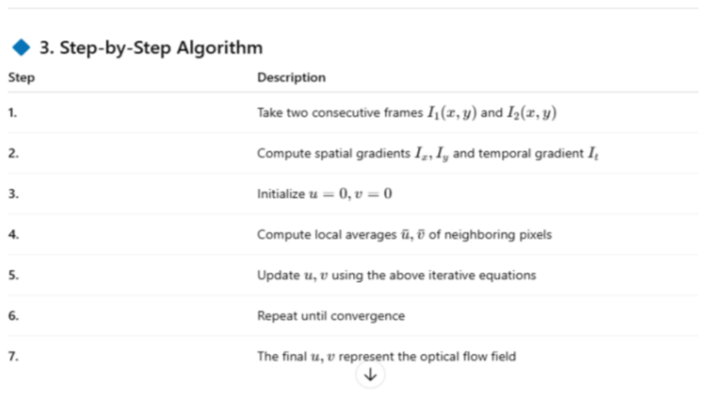
◆ 3. Optical Flow Equation

From the brightness constancy assumption and Taylor expansion, we get the **Optical Flow Constraint Equation:**

$$I_x u + I_y v + I_t = 0$$

Where:

- I_x: change of intensity in **x-direction**
- I_y: change of intensity in **y-direction**
- I_t: change of intensity over **time**
- u, v: optical flow components (motion along x and y)



◆ **3. Step-by-Step Algorithm**

Step	Description
1.	Take two consecutive frames $I_1(x, y)$ and $I_2(x, y)$
2.	Compute spatial gradients I_x, I_y and temporal gradient I_t
3.	Initialize $u = 0, v = 0$
4.	Compute local averages $\bar{u}, \bar{v}$ of neighboring pixels
5.	Update u, v using the above iterative equations
6.	Repeat until convergence
7.	The final u, v represent the optical flow field

Frame 1

```
0 0 1
0 1 1
0 1 1
```

Frame 2

```
0 1 1
1 1 1
1 1 1
```

---

7. Advantages & Limitations

Advantages	Limitations
Produces dense optical flow	Sensitive to noise
Smooth and continuous motion field	Assumes global smoothness
Suitable for small motion	Poor for large or fast movements

Meaning of Each Term

Symbol	Meaning	Example Value
I_x	Change in brightness in **x** direction	1
I_y	Change in brightness in **y** direction	0.5
I_t	Change in brightness over **time**	-2
$u_{\text{avg}}, v_{\text{avg}}$	Average motion of neighboring pixels	0.2, 0.1
α	Smoothness constant (e.g. 1)	1

Simplified Steps (Easy Version)

1. Find how brightness changes → I_x, I_y, I_t
2. Start with $u = 0, v = 0$
3. Find average flow of nearby pixels → $u_{\text{avg}}, v_{\text{avg}}$
4. Update using:

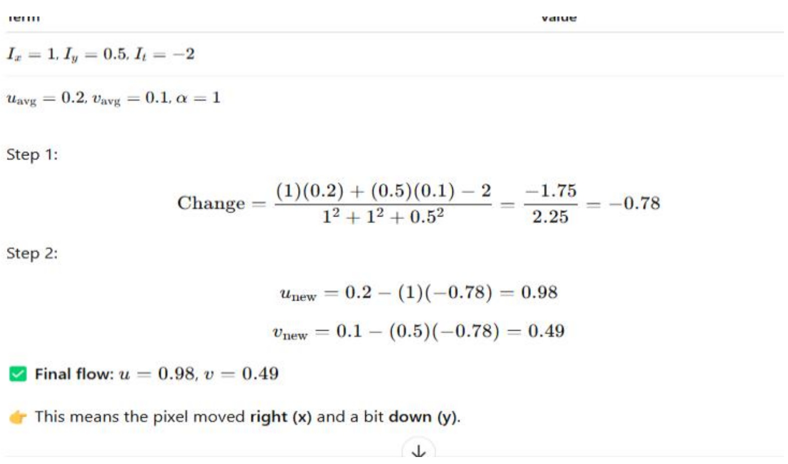
$$\text{Change} = \frac{I_x u_{\text{avg}} + I_y v_{\text{avg}} + I_t}{\alpha^2 + I_x^2 + I_y^2}$$

5. Then:

$$u_{\text{new}} = u_{\text{avg}} - I_x \times \text{Change}$$

$$v_{\text{new}} = v_{\text{avg}} - I_y \times \text{Change}$$

6. Repeat until values stop changing.



Term	Value
$I_x = 1, I_y = 0.5, I_t = -2$	
$u_{\text{avg}} = 0.2, v_{\text{avg}} = 0.1, \alpha = 1$	

Step 1:

$$\text{Change} = \frac{(1)(0.2) + (0.5)(0.1) - 2}{1^2 + 1^2 + 0.5^2} = \frac{-1.75}{2.25} = -0.78$$

Step 2:

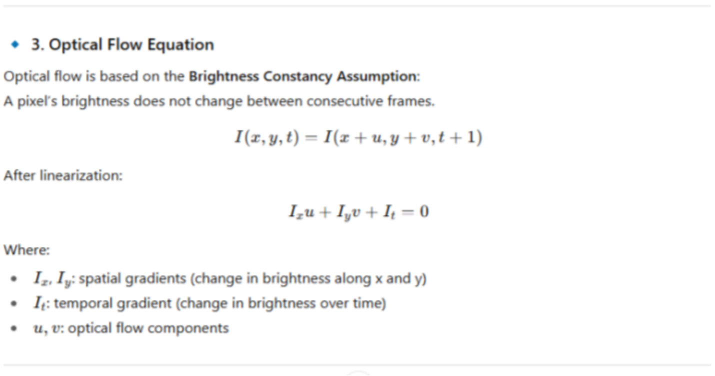
$$u_{\text{new}} = 0.2 - (1)(-0.78) = 0.98$$
$$v_{\text{new}} = 0.1 - (0.5)(-0.78) = 0.49$$

✅ **Final flow:** $u = 0.98, v = 0.49$

👉 This means the pixel moved **right (x)** and a bit **down (y)**.

OPTICAL FLOW AND MOTION

Optical flow refers to the apparent motion of brightness patterns (pixels) between consecutive image frames in a video or image sequence**.**



3. Optical Flow Equation

Optical flow is based on the **Brightness Constancy Assumption**:
A pixel's brightness does not change between consecutive frames.

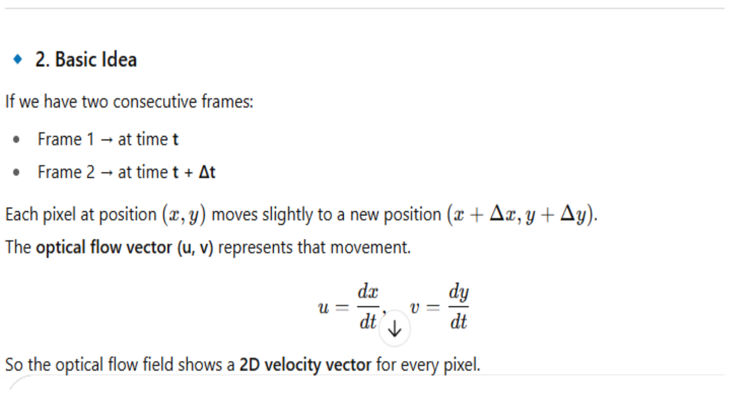
$$I(x, y, t) = I(x + u, y + v, t + 1)$$

After linearization:

$$I_x u + I_y v + I_t = 0$$

Where:

- I_x, I_y: spatial gradients (change in brightness along x and y)
- I_t: temporal gradient (change in brightness over time)
- u, v: optical flow components



2. Basic Idea

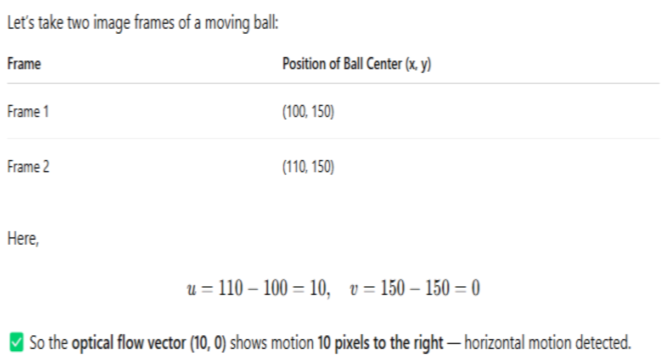
If we have two consecutive frames:

- Frame 1 → at time **t**
- Frame 2 → at time **t + Δt**

Each pixel at position (x, y) moves slightly to a new position $(x + \Delta x, y + \Delta y)$.
The **optical flow vector (u, v)** represents that movement.

$$u = \frac{dx}{dt}, \quad v = \frac{dy}{dt}$$

So the optical flow field shows a **2D velocity vector** for every pixel.



Let's take two image frames of a moving ball:

Frame	Position of Ball Center (x, y)
Frame 1	(100, 150)
Frame 2	(110, 150)

Here,

$$u = 110 - 100 = 10, \quad v = 150 - 150 = 0$$

✅ So the **optical flow vector (10, 0)** shows motion **10 pixels to the right** — horizontal motion detected.

MOTION AND CAMERA SEGMENTATION

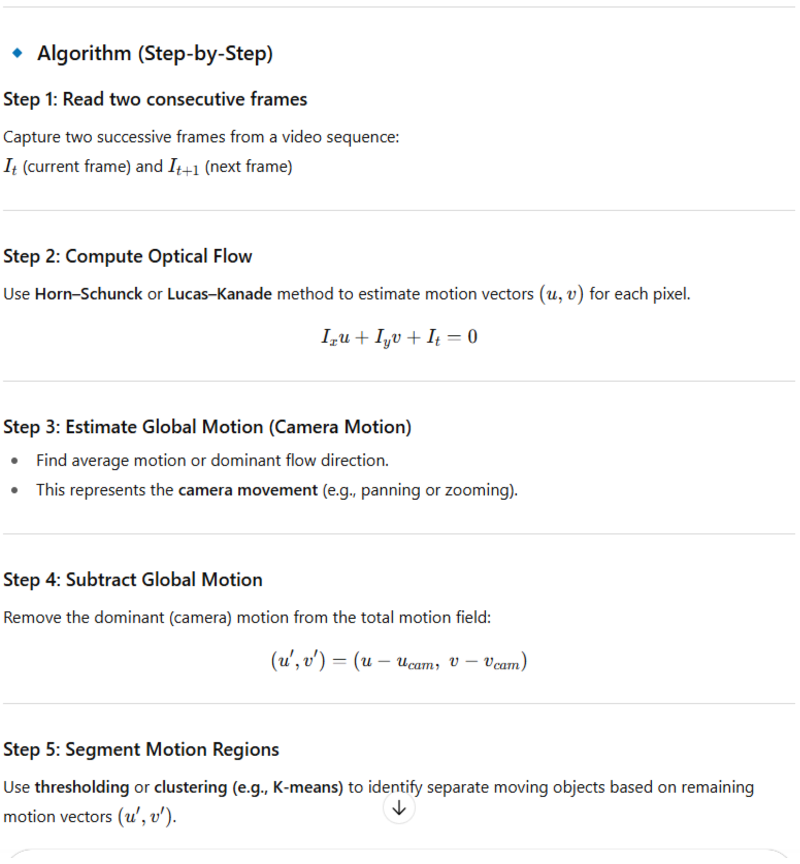
motion segmentation and camera segmentation are important techniques used to separate moving objects and analyze camera motion in a video sequence. To perform motion and camera segmentation on video frames to separate object motion from camera-induced motion using Computer Vision and Image Processing (CVIP) techniques.

In video sequences, motion can come from two sources:

1. Object Motion — when objects in the scene move.
2. Camera Motion — when the camera itself moves (panning, tilting, zooming, etc.).

Motion Segmentation aims to separate regions that correspond to different motions.

Camera Segmentation removes global motion (due to camera) so only the object's true motion remains.



◆ Algorithm (Step-by-Step)

Step 1: Read two consecutive frames

Capture two successive frames from a video sequence:
I_t (current frame) and I_{t+1} (next frame)

Step 2: Compute Optical Flow

Use **Horn–Schunck** or **Lucas–Kanade** method to estimate motion vectors (u, v) for each pixel.

$$I_x u + I_y v + I_t = 0$$

Step 3: Estimate Global Motion (Camera Motion)

- Find average motion or dominant flow direction.
- This represents the **camera movement** (e.g., panning or zooming).

Step 4: Subtract Global Motion

Remove the dominant (camera) motion from the total motion field:

$$(u', v') = (u - u_{cam},\ v - v_{cam})$$

Step 5: Segment Motion Regions

Use **thresholding** or **clustering (e.g., K-means)** to identify separate moving objects based on remaining motion vectors (u', v').

Step 6: Visualize Results

- Display background (static) and moving regions (foreground).
- Use color coding or bounding boxes to show segmented objects.

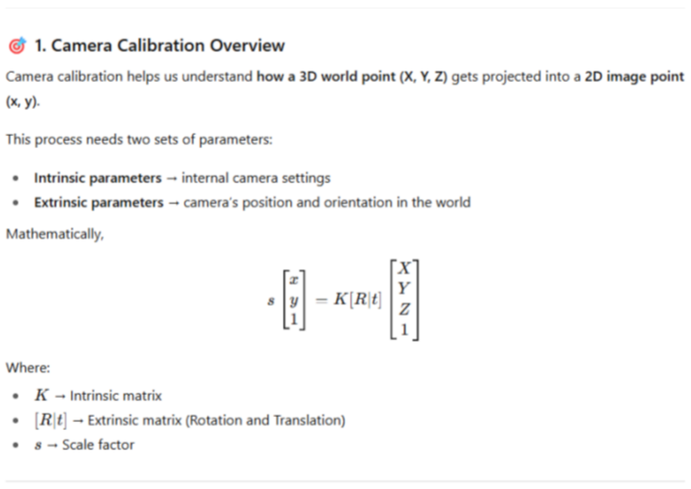
CAMERA CALIBRATION

Camera Calibration

Camera calibration means finding the mathematical relationship between
👉 **3D world points (X, Y, Z) and**
👉 **2D image points (x, y).**

This relation uses two kinds of parameters:

1. **Intrinsic parameters → inside the camera**
2. **Extrinsic parameters → outside the camera (position and orientation)**



🎯 1. Camera Calibration Overview

Camera calibration helps us understand **how a 3D world point (X, Y, Z)** gets projected into a **2D image point (x, y)**.

This process needs two sets of parameters:

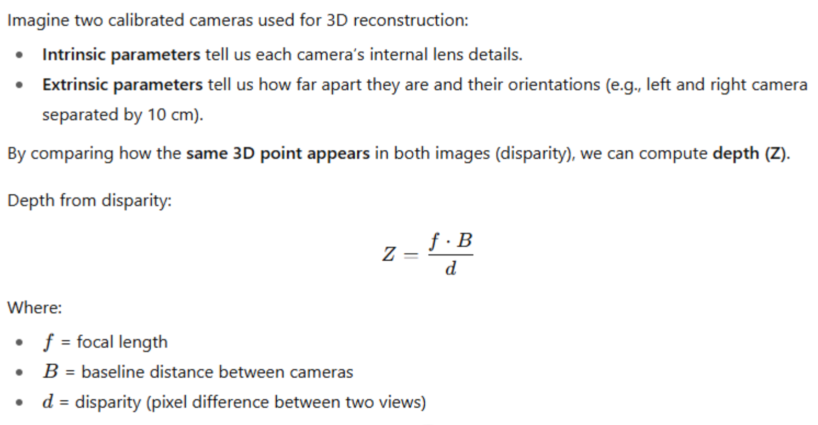
- **Intrinsic parameters** → internal camera settings
- **Extrinsic parameters** → camera's position and orientation in the world

Mathematically,

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K[R|t]\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Where:

- K → Intrinsic matrix
- $[R|t]$ → Extrinsic matrix (Rotation and Translation)
- s → Scale factor



Imagine two calibrated cameras used for 3D reconstruction:

- **Intrinsic parameters** tell us each camera's internal lens details.
- **Extrinsic parameters** tell us how far apart they are and their orientations (e.g., left and right camera separated by 10 cm).

By comparing how the **same 3D point appears** in both images (disparity), we can compute **depth (Z)**.

Depth from disparity:

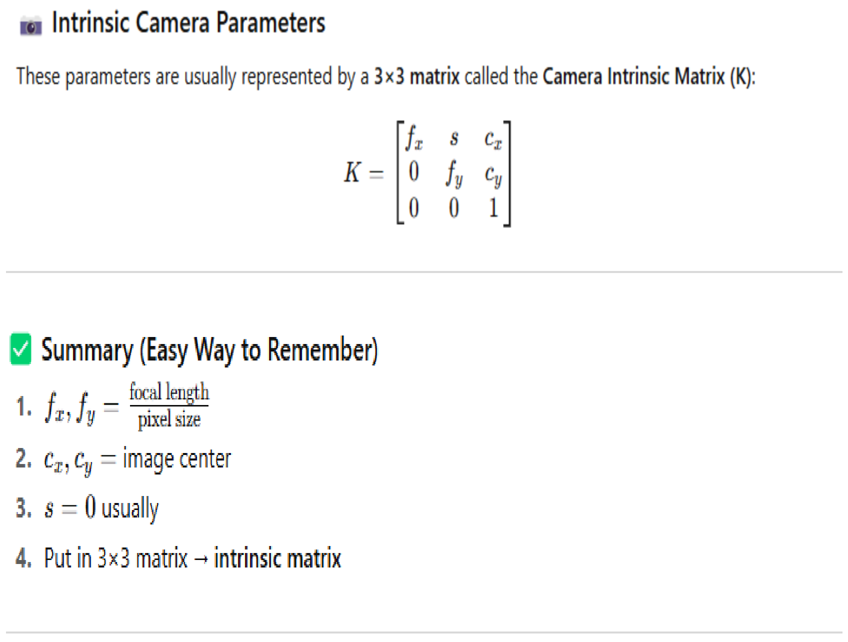
$$Z = \frac{f \cdot B}{d}$$

Where:

- f = focal length
- B = baseline distance between cameras
- d = disparity (pixel difference between two views)

What are Intrinsic Parameters?

Intrinsic parameters are the internal characteristics of a camera that describe how 3D points in the camera's coordinate system are projected onto the 2D image plane.



📷 Intrinsic Camera Parameters

These parameters are usually represented by a **3×3 matrix** called the **Camera Intrinsic Matrix (K)**:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

✅ Summary (Easy Way to Remember)

1. $f_x, f_y = \frac{\text{focal length}}{\text{pixel size}}$
2. $c_x, c_y = \text{image center}$
3. $s = 0$ usually
4. Put in 3×3 matrix → **intrinsic matrix**

Problem

A camera has the following properties:

- Focal length $f = 6\,mm$
- Pixel size $0.01\,mm$
- Image resolution: 640×480
- Principal point at the image center

Question: Find the **intrinsic matrix** K of the camera.

Step 1: Compute focal lengths in pixels

$$f_x = \frac{f}{\text{pixel size}} = \frac{6}{0.01} = 600$$
$$f_y = \frac{f}{\text{pixel size}} = \frac{6}{0.01} = 600$$

Step 2: Find principal point coordinates

Image center:

$$c_x = \frac{\text{image width}}{2} = \frac{640}{2} = 320$$
$$c_y = \frac{\text{image height}}{2} = \frac{480}{2} = 240$$

Step 3: Assume skew is 0

Most cameras have rectangular pixels → $s = 0$

Step 4: Form the intrinsic matrix

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 600 & 0 & 320 \\ 0 & 600 & 240 \\ 0 & 0 & 1 \end{bmatrix}$$

🎯 **Extrinsic Parameters Problem (Simple Example)**

A camera is placed **2 meters to the right** of the world origin and is **not rotated** (it faces straight ahead).

We have a **3D point** in the world at

$$P_w = \begin{bmatrix} 4 \\ 3 \\ 10 \end{bmatrix} \text{ meters.}$$

Find:
Where this point is **in the camera coordinate system** (P_c).

🧩 **Given:**

- **Rotation (R)** = identity (no rotation):

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- **Translation (t)** = camera moved **2 meters right** →

$$t = \begin{bmatrix} -2 \\ 0 \\ 0 \end{bmatrix}$$

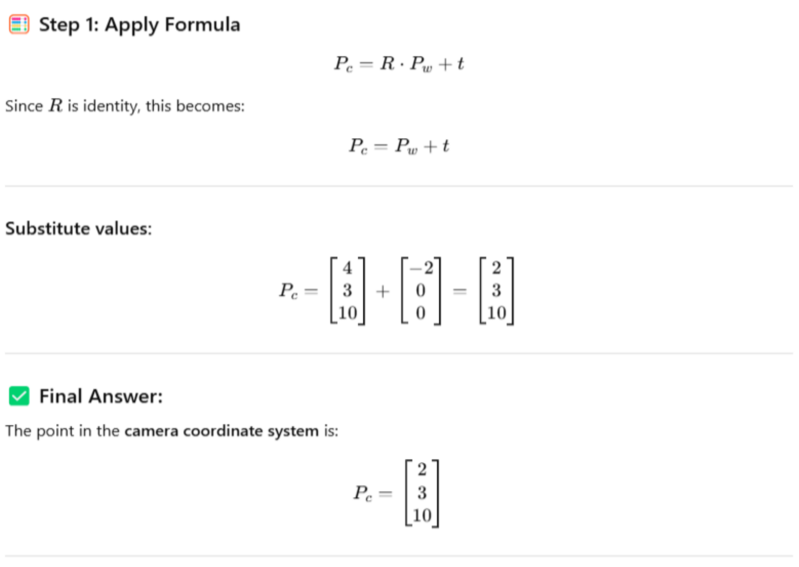
(It's negative because we move the world coordinates opposite to camera direction.)

s

Summary Formula

$$P_c = RP_w + t$$

where R = rotation matrix, t = translation vector.



Step 1: Apply Formula

$$P_c = R \cdot P_w + t$$

Since R is identity, this becomes:

$$P_c = P_w + t$$

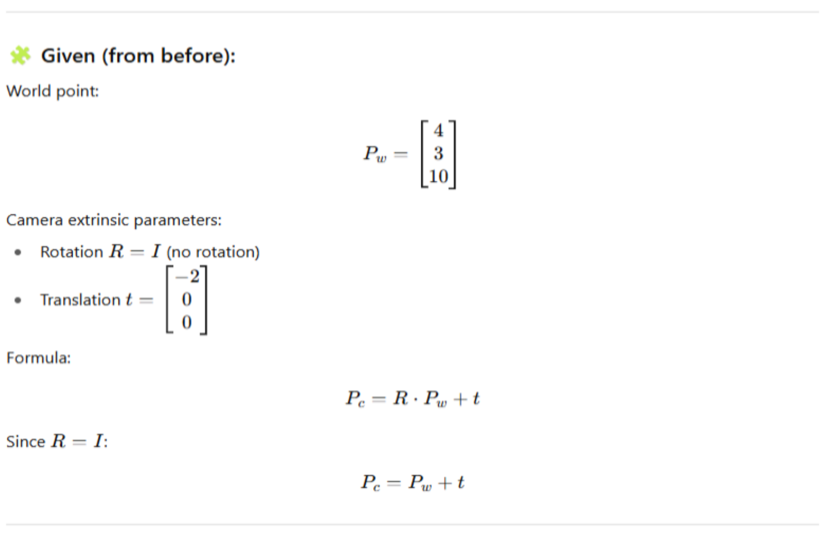
Substitute values:

$$P_c = \begin{bmatrix} 4 \\ 3 \\ 10 \end{bmatrix} + \begin{bmatrix} -2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 10 \end{bmatrix}$$

Final Answer:

The point in the **camera coordinate system** is:

$$P_c = \begin{bmatrix} 2 \\ 3 \\ 10 \end{bmatrix}$$



Given (from before):

World point:

$$P_w = \begin{bmatrix} 4 \\ 3 \\ 10 \end{bmatrix}$$

Camera extrinsic parameters:

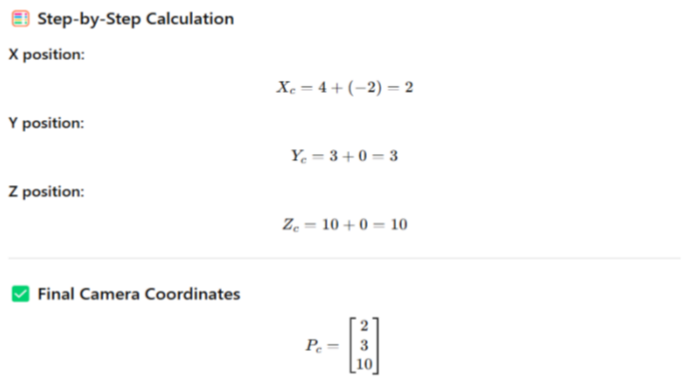
- Rotation $R = I$ (no rotation)
- Translation $t = \begin{bmatrix} -2 \\ 0 \\ 0 \end{bmatrix}$

Formula:

$$P_c = R \cdot P_w + t$$

Since $R = I$:

$$P_c = P_w + t$$



Step-by-Step Calculation

X position:

$$X_c = 4 + (-2) = 2$$

Y position:

$$Y_c = 3 + 0 = 3$$

Z position:

$$Z_c = 10 + 0 = 10$$

Final Camera Coordinates

$$P_c = \begin{bmatrix} 2 \\ 3 \\ 10 \end{bmatrix}$$

3D POINT CLOUD GENERATION

A **3D point cloud** is a collection of **points in 3D space**, each having coordinates **(X, Y, Z)** — representing the **shape or surface** of an object or scene.

Here's a generalized pipeline for 3D point cloud generation:

1. **Image Acquisition / Depth Acquisition**
   - Collect stereo images, multiple images from different views, RGB-D frames, scan data, etc.
2. **Camera Calibration**
   - Determine intrinsic parameters (focal length, optical center, distortion) and extrinsics (position & orientation of camera) if using multiple views. Without calibration, depth / triangulation is inaccurate.
3. **Depth Estimation or Depth Measurement**
   - From stereo disparities, depth sensors, or via ML models.
4. **Back-Projection to 3D**
   - For each pixel, convert (u, v, depth) using intrinsic calibration to world-coordinate (X, Y, Z).

$$X = (u - c_x) * Z / f_x, \quad Y = (v - c_y) * Z / f_y, \quad Z = \text{depth}$$

   - Where (c_x, c_y) are principal point offsets, f_x, f_y focal lengths, (u,v) image coordinate.
5. **Merging Multiple Views (if available)**
   - Transform point clouds from different views into a common coordinate frame (extrinsic calibration), align & merge.
6. **Post-processing**
   - Filtering (remove noise), downsampling, estimating normals, outlier removal.
   - Surface reconstruction (mesh) if needed: e.g. Poisson reconstruction, Delaunay, etc.

🔷 Example 1: Stereo Vision

Suppose you have two calibrated cameras. You take two images:

- Compute disparity map: at each pixel, find where the same feature appears in left vs right image → disparity d.
- Depth $Z = \frac{f \cdot B}{d}$ where f = focal length, B = baseline between cameras.
- For each pixel (u, v), compute X, Y, Z as above → you get a point cloud.

You might then merge left and right images, filter noise, maybe color each point using RGB from one image.

🔷 Example 2: RGB-D Sensor

You have an RGB-D camera (e.g. Kinect, RealSense) which directly gives a depth map for each pixel along with RGB color.

- Use intrinsic calibration of camera.
- For each pixel u,v you retrieve depth Z from sensor.
- Compute X,Y,Z as above.
- Create a colored point cloud: each point has (X,Y,Z) + (R,G,B) from RGB image.
- Optionally, for multiple frames, you transform them into world coordinate using camera poses and merge.

🔷 Example 3: Deep Learning Single Image → Point Cloud

Some recent works try to generate 3D point clouds from just one RGB image. For example:

- **3D-FEGNet** generates point clouds from a single image by extracting features from the image (and edges), then via fully connected layers generate a point cloud pyramid of multiple resolutions.

🔷 Example Walkthrough (Simplified)

Let me sketch a small example you might do in a lab:

- Use a calibrated RGB-D camera.
- Take one frame: RGB image + corresponding depth image.
- Intrinsics known: focal length 600 px, principal point at (320,240), depth in meters.
- Suppose pixel at (u=400, v=200) has depth Z = 2.5 m, RGB color (R,G,B) = (255,100,50).
- Compute:

$$\begin{gathered} X = (400 - 320) * 2.5/600 = 0.333\,\text{m} \\ Y = (200 - 240) * 2.5/600 = -0.167\,\text{m} \\ Z = 2.5\,\text{m} \end{gathered}$$

- So one 3D point: (0.333, -0.167, 2.5) with color (255,100,50).
- Do similarly for all pixels → point cloud.
- If you take another frame from a different position (with known camera pose), transform the new point cloud and merge.
- After merging, you might remove points with very large depth changes (noise), or downsample to reduce size, and maybe estimate normals.