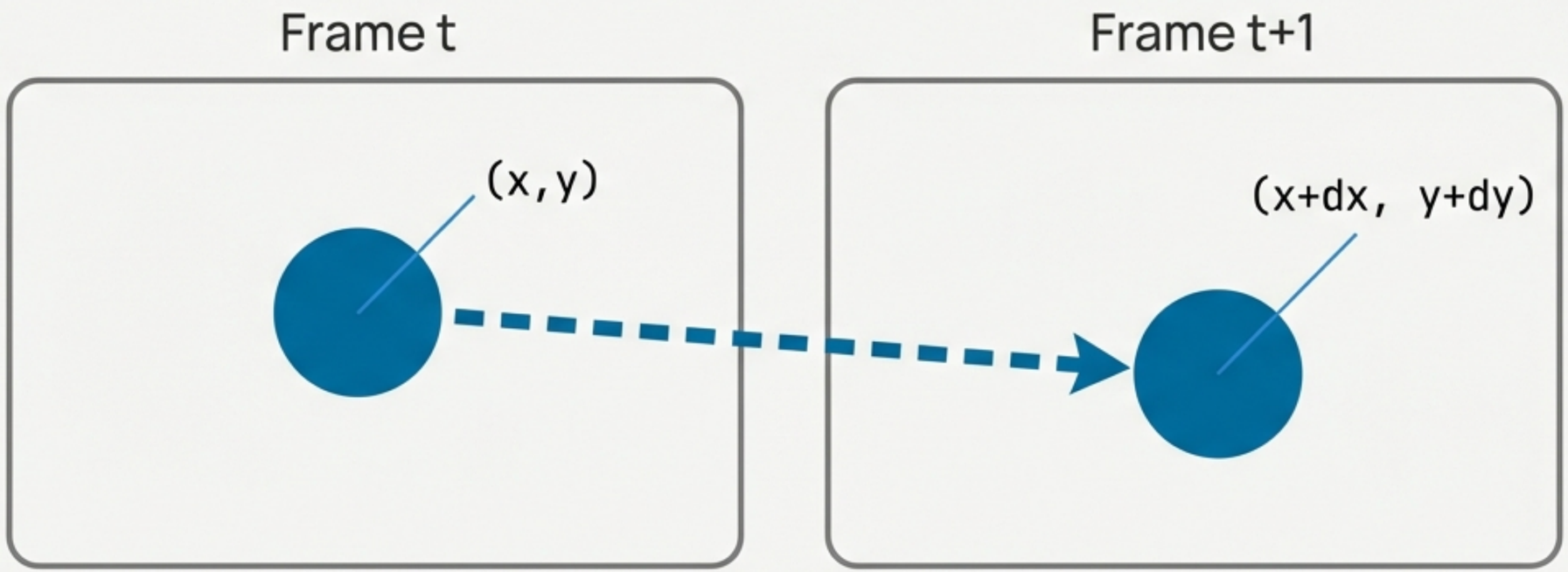
Module 6: Translating Motion into Data

A Deep Dive into Video Analysis with Optical Flow and Background Modeling

- Optical Flow: Lucas-Kanade (Sparse) & Farneback (Dense)
- Background Subtraction: MOG2 & KNN
- Building a Motion Detection Pipeline

How Can a Machine ‘See’ Motion?

The first step is to track how pixels move from one frame to the next. This is called **Optical Flow**.



The goal of optical flow is to find the **motin vector** `(dx, dy)` for each pixel or feature between consecutive frames.

The Foundational Principle: Brightness Constancy

We assume that the brightness of an object's pixel remains the same between frames, even as it moves.

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

I = "Pixel Intensity" (x, y) = "Position" t = "Time"

Frame t Frame t+1

From Physics to a Mathematical Constraint

Using a Taylor Expansion on the Brightness Constancy equation, we arrive at the **OpticFlow Constraint Equation**.

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

$$\nabla I \cdot \mathbf{v} + I_t = 0$$

I_x, I_y: Spatial derivatives (how intensity changes in x and y).
I_t: Temporal derivative (how intensity changes over time).
u, v: The flow velocities we want to find (`dx/dt`, `dy/dt`).

This gives us **one equation with two unknowns** (u and v). We cannot solve this for a single pixel alone. We need more constraints.

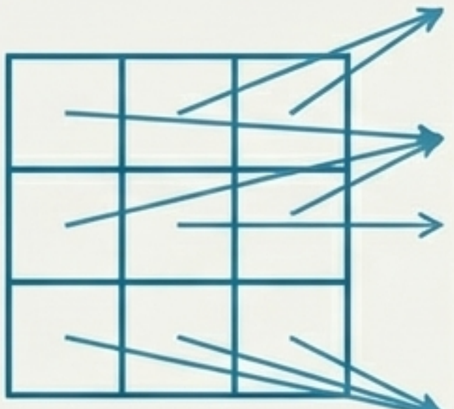
Solution 1: Lucas-Kanade's Local Neighborhood Assumption

Assume that the **flow vector** (u, v) is **constant** for all pixels within a small window (e.g., 3x3 or 5x5).

How it Works

1. By taking a neighborhood of n pixels, we now have n equations.

2. This transforms our unsolvable problem into an overdetermined system, which we can solve.



$$\begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \vdots \\ I_{xn} & I_{yn} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_{t1} \\ I_{t2} \\ \vdots \\ I_{tn} \end{bmatrix}$$

$$A \quad v = b$$

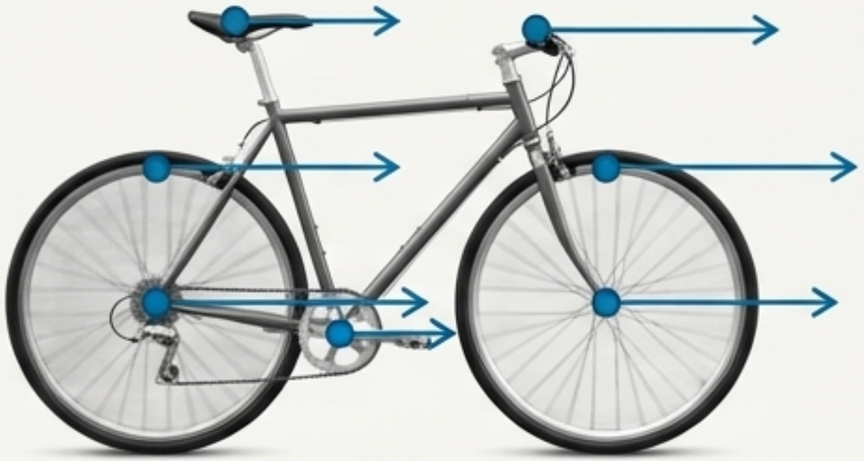
Lucas-Kanade in Practice: Sparse Feature Tracking

The Solution

We solve the system using a **Least Squares** method.

$$\mathbf{v} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$$

What it produces: "**Sparse Optical Flow**. It doesn't calculate flow for every pixel, only for specific feature points (like corners) that are provided as input."



OpenCV Implementation

```
# prev_pts contains points to track from the previous frame
next_pts, status, _ = cv2.calcOpticalFlowPyrLK(
    prev_gray, next_gray, prev_pts, None,
    winSize=(15, 15),
    maxLevel=2
)
```

The local neighborhood window

Use of image pyramids (explained next)

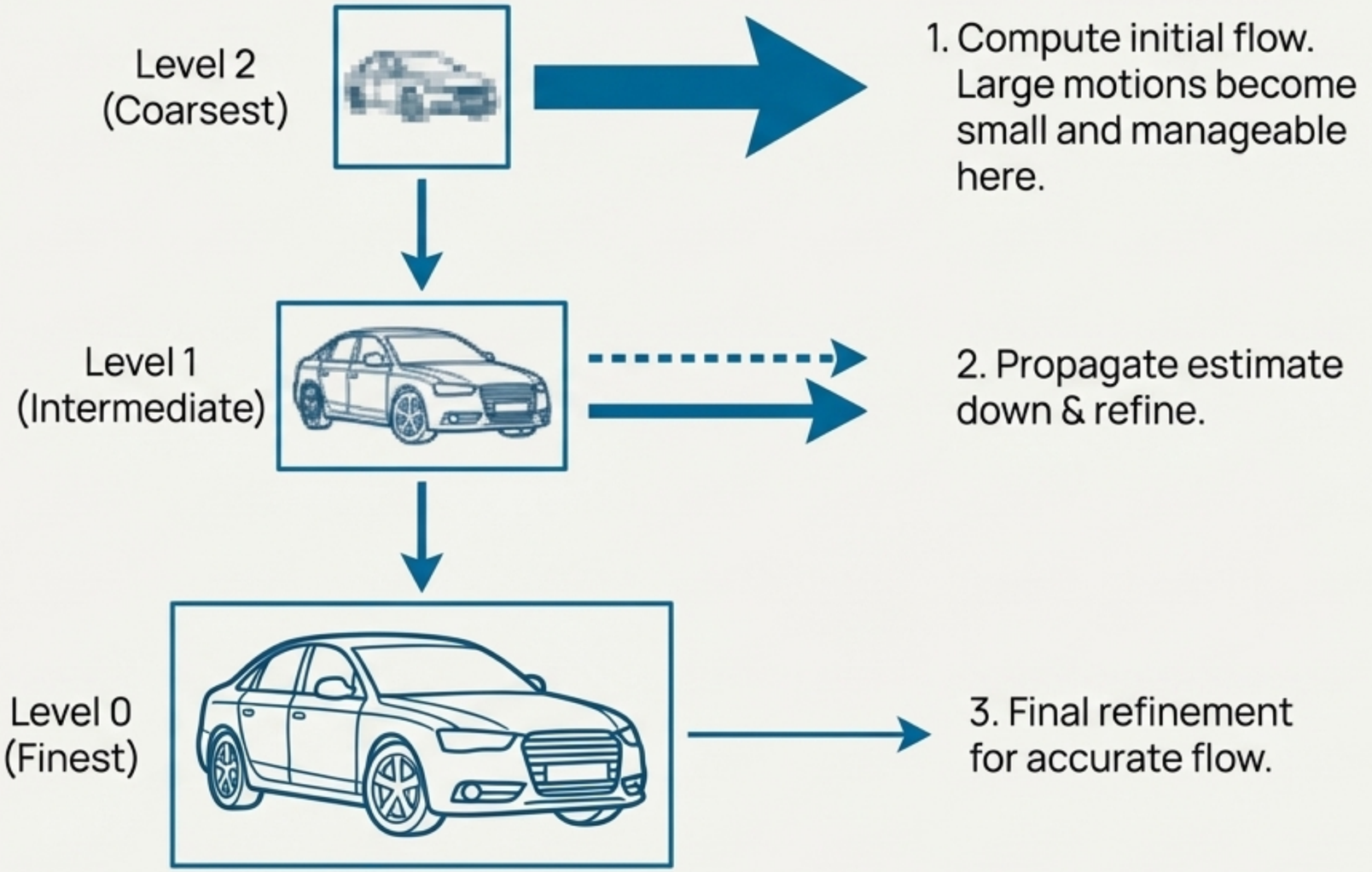
Handling Large Motions with Image Pyramids

The Challenge

The “small motion” assumption fails if an object moves farther than the window size between frames.

The Solution

Use a multi-scale approach with **Image Pyramids**.



Solution 2: Farneback's Dense Optical Flow

What if we need a motion vector for *every single pixel*? This is **Dense Optical Flow**.

The Method

Instead of assuming constant motion, Farneback approximates the pixel neighborhood in each frame with a quadratic polynomial: $f(x) \approx x^T A x + b^T x + c$.

Displacement Estimation

By analyzing how the polynomial's coefficients change between two frames, it can estimate the displacement d.

$$d = -(A_1 + A_2)^{-1} \times \frac{(b_2 - b_1)}{2}$$

Sparse Flow (Lucas-Kanade)

Sparse Flow (Lucas-Kanade)

Dense Flow (Farneback)

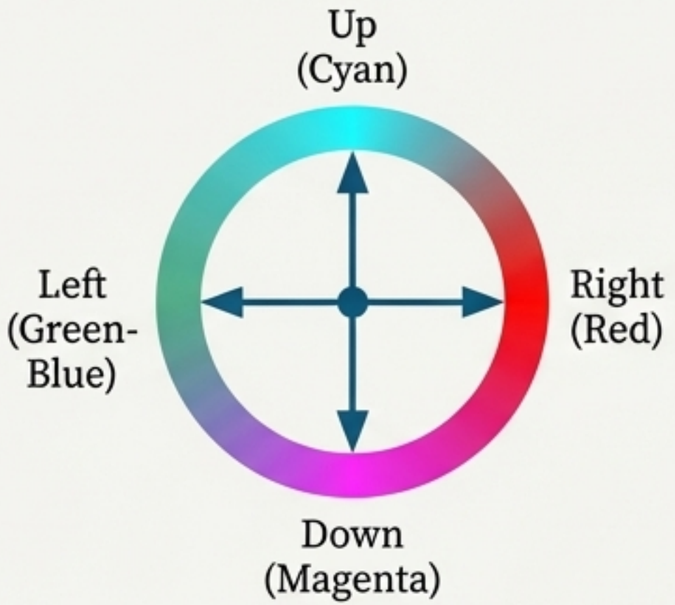
Dense Flow (Farneback)

Visualizing Dense Motion with Color

Challenge: How do you visualize two values (dx, dy) for every pixel?

Solution: We map the motion vectors to the HSV (Hue, Saturation, Value) color space.

- **Hue** (Color) represents the **direction** of motion.
- **Value** (Brightness) represents the **magnitude** (speed) of motion.



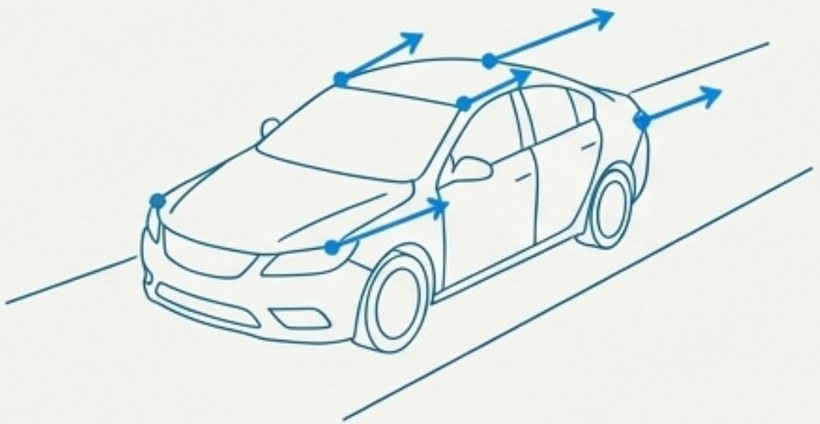
```
flow = cv2.calcOpticalFlowFarnback(prev_gray, next_gray, ...)

# Convert cartesian flow (dx, dy) to polar (magnitude, angle)
magnitude, angle = cv2.cartToPolar(flow[..., 0], flow[..., 1])

# Map to HSV
hsv[..., 0] = angle * 180 / np.pi / 2
hsv[..., 2] = cv2.normalize(magnitude, ...)
```

Hue <- Direction

Value <- Magnitude

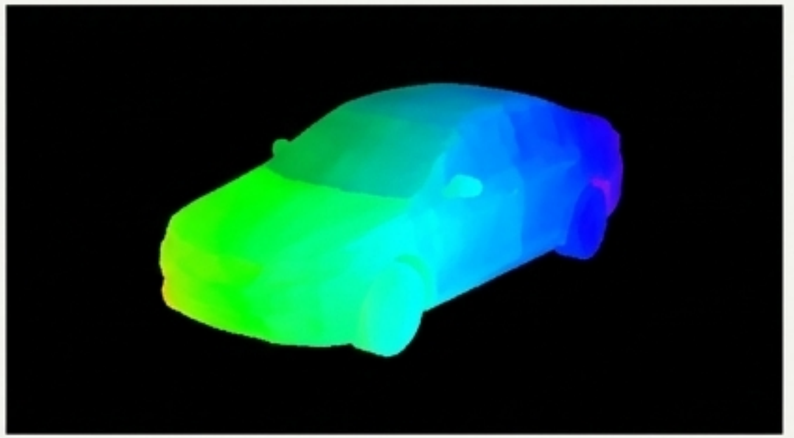
Choosing Your Tool: Sparse vs. Dense Optical Flow

SPARSE (Lucas-Kanade)



- Tracks a small number of selected points.
- **Speed:** Fast.
- **Use Case:** Object tracking, camera stabilization, feature matching.

DENSE (Farneback)



- Computes flow for all pixels in the frame.
- **Speed:** Slower, computationally intensive.
- **Use Case:** Motion segmentation, video compression, generating detailed motion visualizations.

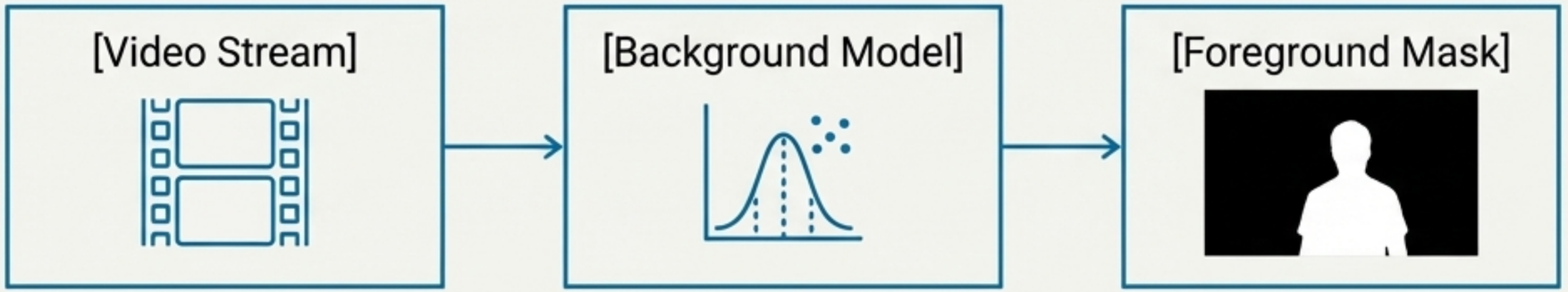
A New Task: Isolating Motion from a Static Background

The Problem: Often, we don't care about the motion of every pixel, just the objects moving in the foreground (e.g., cars, people).

The Technique: **Background Subtraction** (the goal is to build a statistical model of the statiotic background and then identify pixels that deviate significantly from that model.

The Process:

1. **Model the Background:** Learn what 'normal' looks like over several frames.
2. **Compare:** For each new frame, compare it to the background model.
3. **Classify:** Pixels that don't fit the model are labeled as 'foreground'.



Two Approaches to Background Modeling

MOG2 (Mixture of Gaussians)

- **Concept:** Models each pixel's color as a mixture of K Gaussian distributions. A pixel is background if it fits one of the high-weight Gaussians.
- **Key Feature:** Continuously adaptive model. Robust to lighting changes.

```
mog2 = cv2.createBackgroundSubtractorMOG2(
    history=500,
    varThreshold=16,
    detectShadows=True
)
fg_mask = mog2.apply(frame)
```

KNN (K-Nearest Neighbors)

- **Concept:** Models the background using a history of recent pixel values. A pixel is background if it's close to K of its recent neighbors in color space.
- **Key Feature:** Effective for complex, multi-modal backgrounds (e.g., waving trees).

```
knn = cv2.createBackgroundSubtractorKNN(
    history=500,
    dist2Threshold=400,
    detectShadows=True
)
fg_mask = knn.apply(frame)
```

Note: Both methods can detect shadows, outputting a mask where: **0**=background, **127**=shadow, **255**=foreground.

Building a Complete Motion Detection Pipeline

1. Input Frame

2. Background Subtraction

Apply `MOG2` or `KNN` to get a raw foreground mask.

3. Noise Reduction

Use **Morphological Opening** to remove small, spurious detections.

4. Hole Filling

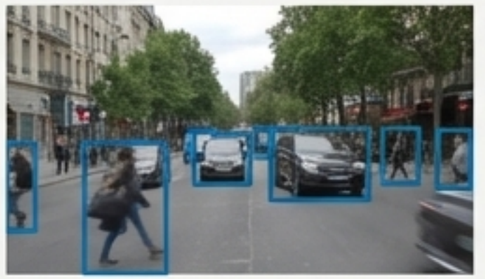
Use **Morphological Closing** to fill gaps in detected objects.

5. Contour Detection

Find the continuous boundaries of the clean foreground objects.

6. Filtering & Action

Filter contours by area to ignore tiny objects, then draw bounding boxes or track the remaining valid objects.



The Video Analysis Toolkit: A Quick Reference

1. Method Comparison

Method	Type	Speed	Use Case
Lucas-Kanade	Sparse	Fast	Feature tracking
Farneback	Dense	Medium	Full motion field
MOG2	Background	Fast	Surveillance
KNN	Background	Fast	Complex backgrounds

2. Key OpenCV Functions

Function Name	Description
`cv2.calcOpticalFlowPyrLK()`	Calculates Lucas-Kanade sparse optical flow.
`cv2.calcOpticalFlowFarneback()`	Calculates Farneback dense optical flow.
`cv2.createBackgroundSubtractorMOG2()`	Creates the MOG2 background subtractor.
`cv2.createBackgroundSubtractorKNN()`	Creates the KNN background subtractor.

Continuing the Exploration

The techniques covered in this module form the foundation of modern video analysis. To deepen your understanding and explore further applications, consult the official documentation and original research papers.

OpenCV Tutorials

- Optical Flow
- Background Subtraction

Foundational Research

- The Lucas-Kanade Paper: An Iterative Image Registration Technique with an Application to Stereo Vision