Prathamesh Arvind Jadhav

Moving Object Detection and Tracking

Introduction to Moving Object Detection and Tracking

1. What is Moving Object Detection and Tracking?

Moving object detection and tracking is a crucial component in the field of computer vision, where the goal is to detect, identify, and follow moving objects within video sequences or real-time camera feeds. The objects can be anything—people, vehicles, animals, drones, or other entities—depending on the application.

It involves two main tasks:

- **Detection**: Identifying whether and where objects are present in each video frame.
- **Tracking**: Following these objects as they move through subsequent frames.

---

2. Importance and Applications

Object detection and tracking are widely used in:

- **Surveillance Systems** (e.g., intruder detection, behavior analysis)
- **Autonomous Vehicles** (e.g., identifying and tracking pedestrians or cars)
- **Human-Computer Interaction** (e.g., gesture tracking)
- **Sports Analytics** (e.g., player and ball tracking)
- **Robotics** (e.g., obstacle detection and avoidance)
- **Augmented Reality** (e.g., overlaying graphics based on moving subjects)

---

3. Components Involved

A. Object Detection

This step identifies moving objects in each frame. Techniques include:

- **Background Subtraction**: Differentiates moving objects from the static background.
- **Optical Flow**: Estimates pixel motion between consecutive frames.
- **Deep Learning-based Detection**: Models like YOLO, SSD, and Faster R-CNN can detect objects in real-time video frames.

B. Object Tracking

Once detected, the object is tracked over time using:

- **Point Tracking** (e.g., KLT tracker): Tracks feature points across frames.
- **Kernel Tracking** (e.g., Mean Shift, CamShift): Uses object appearance for tracking.
- **Silhouette Tracking** (e.g., Contour tracking): Tracks object shapes.
- **Deep Learning Trackers**: Like SORT (Simple Online Realtime Tracking), Deep SORT, or ByteTrack.

□ 4. Challenges in Object Detection and Tracking

- **Occlusion**: Objects may be hidden partially or fully by other objects.
- **Illumination Changes**: Lighting changes may affect object visibility.
- **Camera Motion**: Moving cameras complicate background subtraction.
- **Object Appearance Variability**: Objects may change appearance due to rotation, deformation, or scaling.

- **Multiple Object Interactions**: Objects may merge or split, causing identity switches.

□ 5. Workflow Overview

1. **Video Frame Acquisition** – Get frames from a video or camera.
2. **Preprocessing** – Noise reduction, contrast enhancement.
3. **Detection** – Identify moving objects.
4. **Tracking Initialization** – Assign IDs and create tracking models.
5. **Tracking Update** – Update position and appearance over time.
6. **Result Visualization** – Display bounding boxes or trajectories.

□ 6. Metrics for Evaluation

- **Precision and Recall**: How accurately the system detects objects.
- **Multiple Object Tracking Accuracy (MOTA)**: Combines errors from false positives, missed targets, and identity switches.
- **ID Switches**: Number of times object identities are wrongly changed.

□ 7. Modern Approaches

Modern techniques use **deep learning**:

- **YOLO + SORT/Deep SORT**: Combines fast object detection with tracking.
- **Transformer-based Trackers**: Use attention mechanisms for high accuracy (e.g., TrackFormer).
- **Re-identification Models (ReID)**: Help track objects even after long occlusions or reappearances.

□ 8. Tools and Libraries

- **OpenCV**: Offers tracking algorithms like KCF, CSRT, and more.
- **TensorFlow / PyTorch**: For deep learning-based detection and tracking.
- **DeepStream (NVIDIA)**: Optimized for real-time tracking on edge devices.
- **Detectron2 / MMDetection**: Advanced object detection libraries.

□ Background Modeling in Computer Vision

□ 1. What is Background Modeling?

Background modeling is the process of creating a representation of the static or slowly-changing parts of a scene in video frames. It serves as a reference for identifying **moving foreground objects**—i.e., changes that deviate from this background model are interpreted as motion or activity.

This is one of the most fundamental techniques in **moving object detection**, especially in **video surveillance** and **real-time monitoring systems**.

□ 2. Purpose of Background Modeling

The goal is to **distinguish moving objects (foreground)** from the **non-moving parts (background)** in each frame of a video stream.

Why it matters:

- It's the **first step** in object detection pipelines.
- Reduces computational load by focusing only on regions of interest.
- Helps in noise filtering, shadow detection, and further processing like tracking.

□ 3. How Background Modeling Works

The process typically involves:

1. **Frame Capture**: Read video stream as a sequence of frames.
2. **Model Initialization**: Create a model of the static background from initial frames.
3. **Foreground Detection**: Subtract the current frame from the background model to get the moving regions.
4. **Model Update**: Dynamically update the background model to adapt to gradual scene changes.

□ 4. Background Modeling Techniques

A. Basic Techniques

1. **Frame Differencing**
    - Subtract the current frame from the previous one.
    - Good for detecting fast motion.
    - **Limitation**: Poor with slow-moving objects or static foregrounds.
2. **Static Background Frame**

- Take a clean frame with no moving objects.
- Subtract incoming frames from this fixed frame.
- **Limitation**: Doesn't adapt to environmental changes.

B. Statistical Techniques

3. **Running Average / Exponential Averaging**
   - Uses weighted average over time:

     $$\text{Background}(x,y,t) = \alpha \times \text{CurrentFrame}(x,y) + (1-\alpha) \times \text{Background}(x,y,t-1)$$

   - α is a learning rate.
   - Handles slow changes like lighting or waving trees.
4. **Gaussian Mixture Models (GMM)**
   - Every pixel is modeled as a mixture of Gaussians.
   - Detects multimodal backgrounds (e.g., trees moving in wind).
   - Adaptable, but computationally expensive.
   - OpenCV implements it in cv2.createBackgroundSubtractorMOG2().

C. Advanced Techniques

5. **Kernel Density Estimation (KDE)**
   - A non-parametric way to model the probability distribution of pixel intensities.
   - Suitable for complex backgrounds, but slow.
6. **Codebook Model**
   - Each pixel has a codebook of values seen over time.
   - Helps in handling dynamic backgrounds.
   - Efficient in memory and processing.

D. Deep Learning-based Models (Recent Advancements)

7. **CNN-based Background Subtraction**
    - Deep neural networks learn to classify each pixel as background or foreground.
    - Example: BGSNet, FgSegNet.
    - Requires labeled training data, but more robust to complex environments.
8. **Self-Supervised Models**
    - No manual labels needed.
    - Use consistency over time to learn background representation.

□ 5. Evaluation Metrics

To evaluate background modeling performance:

- **Precision & Recall**: For foreground classification accuracy.
- **F-Measure**: Harmonic mean of precision and recall.
- **False Positive Rate (FPR)**: Important in surveillance.
- **Adaptability**: How well the model adjusts to changes.

□ 6. Challenges in Background Modeling

Challenge	Description
Illumination Changes	Sudden lighting changes affect pixel values.
Dynamic Background	Moving trees, fountains, waves cause false detections.
Shadows &	Mistakenly identified as moving objects.

Challenge	Description
Reflections	
Camera Jitter	Small movements cause background instability.
Intermittent Object Motion	Objects stop for a while and are absorbed into the background.
Foreground Aperture	Uniform areas in moving objects are hard to detect with simple differencing.

---

7. Applications

- **Surveillance systems** (e.g., intruder detection)
- **Traffic monitoring** (e.g., counting vehicles)
- **Smart homes** (e.g., activity recognition)
- **Gesture recognition**
- **Human-computer interaction (HCI)**

Connected Component Labeling (CCL)

---

1. What is Connected Component Labeling?

Connected Component Labeling (CCL) is a fundamental technique in **image processing** and **computer vision** used to identify and label regions (or "blobs") of **connected pixels** in a **binary image**.

It assigns a unique label (ID) to each connected group of white (1-valued or foreground) pixels, allowing us to treat each component as a separate object or region.

---

2. Purpose and Applications

Purpose:
To **segment** the image into meaningful parts, enabling object counting, shape analysis, and further processing.

Applications:

- Object detection and counting
- Shape and size measurement
- OCR (Optical Character Recognition)
- Blob detection (e.g., detecting coins, cells, or vehicles)
- Foreground segmentation in video

□ 3. Basic Requirements

- A **binary image**: Typically black (0) for background and white (1) for foreground.
- A **connectivity rule**: Defines what "connected" means (4-connectivity or 8-connectivity).

□ 4. Types of Pixel Connectivity

1. **4-Connectivity**:
    - A pixel is connected to its **up, down, left, and right** neighbors.
    - Diagonal neighbors are **not** considered connected.
2. **8-Connectivity**:
    - A pixel is connected to **all 8 neighbors** (including diagonals).

□ 5. How Connected Component Labeling Works

Step-by-Step Overview:

1. **Input**: A binary image.
2. **Scan** the image pixel-by-pixel.
3. **Label** each foreground pixel by checking its neighbors.
4. **Assign a new label** if the pixel starts a new component.
5. **Record equivalence** if different labels are found for connected pixels.
6. **Resolve equivalences** to assign the same label to connected regions.
7. **Output**: A labeled image where each connected region has a unique ID.

□ 6. Algorithms for Connected Component Labeling

A. Two-Pass Algorithm (Most Common)

First Pass:

- Scan image left-to-right, top-to-bottom.
- Assign temporary labels.
- Record label equivalences.

Second Pass:

- Replace each label with the smallest equivalent label.

□ **Efficient and widely used.**

B. Union-Find Algorithm (Disjoint Set Forest)

- Efficiently manages label equivalences.
- Uses **path compression** and **union by rank** to optimize label merging.

 Faster for large and complex images.

C. One-Pass Algorithms

- Process the image in a single pass using advanced techniques like contour tracing.
- Memory-efficient but harder to implement.

 Useful in real-time systems.

---

7. Example

Binary Image (5x5):

0 0 1 1 0
0 1 1 0 0
0 0 0 1 1
0 1 0 1 1
0 1 0 0 0

After CCL with 4-connectivity:

0 0 1 1 0
0 1 1 0 0
0 0 0 2 2
0 3 0 2 2
0 3 0 0 0

Here, we have 3 connected components labeled 1, 2, and 3.

9. Evaluation Metrics

- **Number of components**: Used to count objects.

- **Size of each component**: Pixel count per label.
- **Bounding boxes / contours**: For shape analysis.
- **Centroids**: For positioning and tracking.

□ 10. Challenges

Challenge	Description
Noise in Image	May create small false components.
Touching Objects	Might be labeled as one large object.
Connectivity Choice	Affects how objects are segmented (4 vs 8).
Performance	Needs optimization for large images or video streams.

□ 11. Summary Table

Feature	Details
Goal	Label each connected region in a binary image.
Input	Binary image (0 = background, 1 = foreground).
Connectivity	4 or 8 neighbors.
Algorithms	Two-pass, Union-Find, One-pass.
Output	Labeled image with each component having a unique ID.
Applications	Object counting, OCR, segmentation, tracking.

□ Shadow Detection

□ 1. What is Shadow Detection?

Shadow Detection in computer vision is the process of identifying and separating **shadows** from actual **moving objects** in an image or video frame, especially in applications like **background subtraction**, **object tracking**, and **surveillance**.

Shadows, if not handled correctly, can cause:

- Incorrect object shape detection,
- Object size miscalculations,
- False motion detection,
- Errors in tracking multiple objects.

□ 2. Why is Shadow Detection Important?

Shadows can appear as **part of a moving object** during background subtraction, leading to:

- Overestimation of object size,
- False positives in motion detection,
- Merging of separate objects due to cast shadows.

Hence, detecting and **removing or separating shadows** helps in:

- Improving object segmentation,
- Enhancing tracking accuracy,
- Reducing noise and improving recognition.

□ 3. Types of Shadows

1. **Cast Shadows**:
    - Occur when an object blocks light and casts its shadow onto another surface (like the ground).
    - Usually *move* with the object.
2. **Self Shadows**:
    - Parts of the object itself that are not illuminated.
    - Usually *static* relative to the object.

 Cast shadows are more problematic for object detection and tracking.

---

4. Shadow Detection Techniques

A. Chromacity-Based Methods (Color Models)

Idea: Shadows change the **brightness** but not the **chromacity (color ratio)** of the background.

- Convert image to color spaces like **HSV**, **YCbCr**, or **Normalized RGB**.
- Shadows have **lower brightness (Value in HSV)** but similar **Hue and Saturation**.

Steps:

1. Convert RGB to HSV.
2. Compare Hue and Saturation between background and current frame.
3. If Value is lower, and Hue/Saturation difference is small → likely a shadow.

Formula (in HSV):

if (V_current / V_background < τ_v) and (|H_current - H_background| < τ_h):
    pixel = shadow

 Pros:

- Simple and effective for color images.
- Real-time performance.

 Cons:

- Fails under poor lighting.
- Sensitive to object color similar to background.

 B. Geometry-Based Methods

Idea: Shadows appear in predictable regions based on **light source direction** and **object geometry**.

- Track object's position and infer expected shadow area.
- Useful in **controlled environments**.

 Pros: Precise under known conditions.

 Cons: Requires **prior calibration**, not suitable for real-world dynamic scenes.

 C. Physical Models (Illumination Models)

Use **photometric models** of light reflection and surface properties.

- Shadows reduce the **illumination** but do not change surface reflectance.

- Model the scene lighting to distinguish between actual object pixels and shadow pixels.

 Pros: Accurate with known scene parameters.

 Cons: Computationally expensive; difficult to calibrate in real scenes.

 D. Statistical Learning / Machine Learning

Train models using features like:

- Color ratios (RGB/HSV),
- Texture (LBP, Gabor filters),
- Temporal changes (background evolution).

Techniques: SVM, k-NN, or Deep Learning (CNNs, RNNs).

 Pros:

- High accuracy with enough data.
- Can adapt to complex environments.

 Cons:

- Requires labeled datasets.
- Computationally intensive.

 E. Texture-Based Methods

Idea: Shadowed areas retain **similar texture** as background but are darker.

- Extract texture features (e.g., using **Local Binary Patterns (LBP)**).

- Compare background and current texture.
- If similar texture but lower brightness → shadow.

 Pros: Robust in textured backgrounds.

 Cons: Not effective in uniform regions.

---

5. Integration with Background Subtraction

Many motion detection pipelines include **shadow detection after background subtraction**.

Workflow:

1. **Background Subtraction** → Get foreground mask.
2. **Shadow Detection** → Refine the mask by removing shadows.
3. **Object Detection/Tracking** → Use refined mask.

This helps in **accurate object shape detection** and **trajectory tracking**.

---

7. Challenges in Shadow Detection

Challenge	**Description**
Varying Illumination	Changing lighting alters shadow appearance.
Similar Object/Background Colors	Hard to distinguish shadow from object.
Dynamic Background	Shadows and motion can mix in complex scenes.
Real-Time Constraints	High computational cost for accurate models.

□ 8. Evaluation Metrics

To assess performance:

- **True Positives (TP)**: Correctly identified shadow pixels.
- **False Positives (FP)**: Non-shadow pixels wrongly marked.
- **Shadow Detection Rate (SDR)**: TP / (TP + FN)
- **Shadow Discrimination Rate (SDisR)**: Non-shadow foreground correctly retained.

□ 9. Summary Table

Aspect	Details
Goal	Identify and remove shadows from moving object masks.
Main Types	Cast shadows (problematic), self-shadows (less critical).
Common Techniques	Chromacity, Geometry, Physical, Texture, Learning-based.
Useful Color Spaces	HSV, YCbCr, Normalized RGB.
Used In	Video surveillance, motion tracking, object detection.
Main Challenge	Distinguishing shadows from dark objects or textures.

□ Discrete Kalman Filtering

□ 1. What is Kalman Filtering?

The **Kalman Filter** is an **optimal recursive estimation algorithm** that estimates the internal state of a system (like position, velocity, etc.) from noisy measurements over time. It is widely used in:

- Object tracking,
- Robotics,
- Navigation (GPS),
- Financial forecasting,
- Control systems.

Discrete Kalman Filtering is the version of the Kalman Filter designed for **discrete-time** (i.e., time steps like t = 1, 2, 3...), which is suitable for digital systems such as cameras and sensors.

□ 2. Key Concepts

Let's understand it using a **target tracking** example (like a moving car):

Term	**Meaning**
State (x)	The variables you want to estimate (e.g., position, velocity).
Measurement (z)	The actual observations (e.g., sensor readings like GPS location).
Prediction	Estimating the next state using a model.
Correction	Updating the prediction using the new measurement.

□ 3. Kalman Filter Equations (Discrete-Time)

□ A. State Representation

Let:

- x_k → State vector at time step k
- u_k → Control input (if any)
- z_k → Measurement at time step k

✅ B. State Transition Model

1. Prediction Step:
   - Predicted State Estimate:

$$\hat{x}_{k|k-1} = A \cdot \hat{x}_{k-1|k-1} + B \cdot u_k$$

   - Predicted Covariance:

$$P_{k|k-1} = A \cdot P_{k-1|k-1} \cdot A^T + Q$$

2. Update (Correction) Step:
   - Kalman Gain:

$$K_k = P_{k|k-1} \cdot H^T \cdot (H \cdot P_{k|k-1} \cdot H^T + R)^{-1}$$

   - Updated State Estimate:

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \cdot (z_k - H \cdot \hat{x}_{k|k-1})$$

   - Updated Covariance:

$$P_{k|k} = (I - K_k \cdot H) \cdot P_{k|k-1}$$

□ Matrix Meanings:

Symbol	Description
A	State transition matrix
B	Control input matrix
H	Observation matrix

Symbol	Description
Q	Process noise covariance (model uncertainty)
R	Measurement noise covariance (sensor noise)
P	Estimate error covariance
K	Kalman Gain (how much trust in measurement)

□ 5. Kalman Filter Workflow

Step	Description
Initialization	Start with an initial guess for state and error covariance.
Prediction	Use model to predict the state at next time step.
Measurement	Get actual observation (e.g., sensor reading).
Correction	Combine prediction and measurement using Kalman Gain.
Repeat	For each new time step.

□ 6. Kalman Gain Intuition

- **High Kalman Gain (K → High)**:
  Measurement is trusted more than model → large correction.
- **Low Kalman Gain (K → Low)**:
  Prediction is trusted more → small correction.

Kalman Gain is automatically adjusted based on **uncertainties** in prediction (Q) and measurement (R).

□ 7. Advantages of Kalman Filtering

- Works in **real-time** and is **recursive** (no need to store history).
- Optimally handles **noisy data** and **uncertain models**.
- Simple and efficient for **linear systems** with Gaussian noise.

□ 8. Limitations

Issue	Description
Linear Assumption	Standard Kalman filter assumes linear models.
Gaussian Noise	Assumes noise is normally distributed.
Non-Linear Systems	Use **Extended Kalman Filter (EKF)** or **Unscented Kalman Filter (UKF)**.

□ 10. Applications of Discrete Kalman Filter

Domain	Use Case
Computer Vision	Object tracking, motion prediction
Robotics	Localization, path tracking
Aerospace	Missile guidance, aircraft navigation
Finance	Time series prediction, trend estimation
Medical Imaging	ECG/EEG filtering

□ 11. Summary Table

Component	Description
State (x)	What we want to estimate (e.g., position, velocity)
Measurement (z)	What we observe (e.g., GPS location)
Prediction	Use model to estimate the next state
Update	Adjust prediction based on actual measurement
Kalman Gain (K)	Balances trust between prediction and measurement
Assumption	Linear model + Gaussian noise

□ Mean-Shift Tracking

□ 1. Introduction to Mean-Shift Tracking

Mean-shift tracking is a **non-parametric**, **iterative** method used for **locating the mode** (i.e., the peak) of a probability distribution. In object tracking, it helps to **track an object's position** by maximizing the similarity between a target model and candidate region in each new frame.

Mean-shift doesn't require any motion model; instead, it uses **appearance-based tracking** – meaning it relies on how the object looks (color, texture, etc.).

□ 2. Basic Idea in Tracking

The main idea is:

- Build a model of the object (e.g., color histogram).
- In the next frame, search nearby regions.
- Use mean-shift to **move the search window** towards the region most similar to the object model.

Think of it like a **hill-climbing algorithm** where the algorithm shifts the search window to where similarity is highest.

□ 3. Key Concepts

Term	Description
Kernel	A function (usually Gaussian or Epanechnikov) used to weight nearby pixels
Histogram	Distribution of pixel features (e.g., color) in a region
Search Window	The rectangular region where the algorithm looks for the object
Bhattacharyya Coefficient	Measures similarity between two histograms

---

□ 4. Steps in Mean-Shift Tracking

1. **Initialization**:
    - Choose the object to track in the first frame (e.g., user selects it).
    - Extract features (e.g., color histogram) and define a **target model**.

2. **In New Frame**:
    - Place the search window at the predicted location.
    - Calculate the feature histogram of this region.
    - Compute similarity with the target model.
    - Calculate **mean shift vector** (direction to move the window).
    - Move the window towards higher similarity.
    - Repeat until convergence (shift is very small).
3. **Repeat for each frame** in the video.

📐 5. Mathematical Formulation

Let:

- x_i = pixel location
- $k(x)$ = kernel function
- $f(x)$ = feature vector (e.g., color at x)
- q = target model histogram
- $p(y)$ = histogram at location y in the new frame

Bhattacharyya Coefficient:

$$\rho[p(y), q] = \sum_{u=1}^{m} \sqrt{p_u(y) \cdot q_u}$$

This measures histogram similarity (higher = more similar).

Mean Shift Vector:

$$m(y) = \frac{\sum_{i=1}^{n} x_i \cdot w_i}{\sum_{i=1}^{n} w_i} - y$$

Where:

- w_i are weights based on histogram similarity
- y is the current location of the search window

The search window is shifted by $m(y)$ iteratively until convergence.

□ 6. Intuition With an Example

Let's say we're tracking a red ball:

- In the first frame, we record the red color histogram inside a selected rectangle.
- In the next frame, we search around the previous position.
- The red pixels that match are assigned **high weights**, others low.
- The mean of these weighted positions tells us where to shift the rectangle.
- Repeat until the position stabilizes → ball is tracked!

💡 8. Kernel Function

The **Epanechnikov kernel** is often used:

$$k(x) = \begin{cases} 1 - x^2 & \text{if } x^2 < 1 \\ 0 & \text{otherwise} \end{cases}$$

It gives higher weights to pixels near the center.

□ 10. Advantages of Mean-Shift Tracking

Advantage	Explanation
Simple and Efficient	No training, fast, and easy to implement.
Robust to Occlusion	Can handle partial occlusion of the target.
Non-parametric	Doesn't assume any specific model of the target.

□ 11. Limitations

Limitation	Description
Fixed Scale	Doesn't adapt to object size change unless enhanced (CamShift).
Sensitive to Appearance	Object color/appearance must stay relatively

Limitation	Description
Change	consistent.
Local Optima	May converge to incorrect position if distractors exist.

---

12. Extensions

- **CamShift (Continuously Adaptive Mean Shift)**:
  Enhances mean-shift by allowing dynamic window size (used in OpenCV for face/hand tracking).
- **Mean-shift + Kalman Filter**:
  Combines appearance and motion models for robust tracking.

---

13. Summary Table

Feature	Description
Method	Appearance-based tracking
Input	Target appearance (e.g., histogram)
Tracking Mechanism	Histogram comparison + mean shift iteration
Adaptability	Fixed scale unless modified
Speed	Fast (suitable for real-time)
Popular Usage	Human tracking, object tracking in video

Prathamesh Arvind Jadhav

□ Segmentation Tracking via Graph Cuts

□ 1. Introduction

Segmentation tracking via graph cuts is a powerful method that combines **object tracking** and **image segmentation**. It not only follows the object's motion through video frames but also **precisely delineates the object boundaries**.

This technique models the tracking problem as a **graph**, where the optimal segmentation is found by computing a **minimum cut** that separates the object (foreground) from the background.

□ 2. Key Objectives

- Accurately track **non-rigid** and **articulated objects**.
- Maintain **pixel-level precision** in separating object from background.
- Handle **complex motions** and **appearance changes**.

□ 3. Basic Concepts

Term	Meaning
Graph	An abstract structure made of nodes (pixels) and edges (connections between pixels)
Nodes	Each pixel is treated as a node in the graph
Edges	Represent similarity (or dissimilarity) between neighboring pixels
Source (S)	Virtual node representing the foreground (object)

Term	Meaning
Sink (T)	Virtual node representing the background
Graph Cut	Partitioning the graph into two disjoint sets such that one contains source, the other sink
Min-Cut	The cut with the **least total edge cost** that separates the source and sink

---

□ 4. Step-by-Step Procedure

1. **Initialize the Object**:
   - In the first frame, the user (or algorithm) selects the object (foreground).
   - Appearance models (e.g., color histograms) are created for **foreground** and **background**.
2. **Construct the Graph**:
   - Every pixel is a **node**.
   - Connect each node to:
     - **Source node (S)** with weight based on **likelihood of being foreground**.
     - **Sink node (T)** with weight based on **likelihood of being background**.
   - Add **edges between neighboring pixels** with weights based on similarity (smoothness).
3. **Graph Cut**:
   - Use a **min-cut/max-flow algorithm** (like Boykov-Kolmogorov algorithm) to partition the graph.
   - The result gives the **segmented object**, i.e., pixels connected to the source.
4. **Tracking in Subsequent Frames**:
   - Appearance models are updated based on the previous segmentation.
   - A new graph is constructed and segmented using updated data.

- The segmented object is tracked across frames.

5. Graph Energy Formulation

Let the energy function be:

$$E(L) = \sum_{p} D_p(L_p) + \sum_{(p,q)\subset\mathcal{N}} V_{p,q}(L_p, L_q)$$

Where:

- L: label assignment (foreground or background)
- $D_p(L_p)$: **Data term** – penalty for assigning label L_p to pixel p
- $V_{p,q}(L_p, L_q)$: **Smoothness term** – penalty for assigning different labels to neighboring pixels p and q

Term	Purpose
Data Term	Measures how likely a pixel belongs to the foreground or background using color models (e.g., Gaussian Mixture Models)
Smoothness Term	Encourages similar labels for similar neighboring pixels to avoid noisy segmentations

---

□ 7. Visual Intuition

Imagine a grid of pixels (like a checkerboard). Each pixel is:

- Connected to its neighbors with edges.
- Also connected to "source" and "sink" nodes.

If a pixel has red color (object), it has a strong edge to **source**. If a pixel is background-colored, it connects strongly to **sink**.

The **min-cut algorithm** slices the graph in such a way that the object pixels are separated from the background pixels, resulting in accurate segmentation.

□ 8. Algorithms Used

- **Boykov-Kolmogorov algorithm** – Efficient method for computing minimum cut.
- **GrabCut** – A variant of graph cuts used in static image segmentation, adapted for tracking.
- **Dynamic Graph Cuts** – Speed up the computation by reusing previous frame's flow information.

□ 9. Advantages

Feature	Benefit
Pixel-level accuracy	Tracks the object boundary very precisely
Handles complex shapes	Works well with non-rigid and articulated objects
Robustness	Effective in cluttered scenes with varying backgrounds

---

□ 10. Limitations

Limitation	Description
Computationally expensive	Requires solving min-cut for every frame
Sensitive to model drift	If the appearance model is not updated properly, tracking can fail
Manual initialization	Often needs user input in the first frame

---

□ 11. Example Use-Cases

- **Interactive Video Editing** – Object cutout from videos.
- **Medical Imaging** – Tracking organs in ultrasound or MRI sequences.
- **Surveillance** – Tracking humans in dynamic backgrounds.

□ 13. Summary Table

Feature	**Description**
Method	Graph-based segmentation and tracking
Core Technique	Min-cut on pixel graph
Input	Initial object region and frame
Output	Pixel-wise object mask in each frame
Accuracy	High (precise object boundary)
Suitable For	Non-rigid, deformable object tracking
Complexity	High (but optimizations exist)