1. Custom Grayscale Conversion:

What was used: I manually converted each frame to grayscale using a weighted sum of the RGB channels.

Why used: Grayscale images simplify the edge detection process by reducing the complexity from color to a single intensity channel.

Why it works: Grayscale conversion is a standard preprocessing step to prepare an image for edge detection. It reduces the input data to essential intensity information.

Limitations: Grayscale may lose some useful color information, which can be helpful in distinguishing objects, especially in complex environments.

2. Custom Edge Detection using Canny:

What was used: I implemented Canny edge detection after converting the frame to grayscale.

Why used: Canny edge detection is a popular method to identify the boundaries of objects in an image.

Why it works: The Canny algorithm uses a multi-step process to detect strong edges in the image, making it effective for object segmentation.

Limitations: The edge detection can be sensitive to noise and parameter selection (thresholds). Small or blurry objects might not be detected effectively.

3. Custom Clustering:

What was used: I manually clustered centroids using a distance threshold (epsilon) to group close points together.

Why used: Clustering helps group moving objects (detected by edge detection) into meaningful clusters that represent individual objects.

Why it works: Using distance-based clustering (e.g., centroid-based) is simple and intuitive for grouping points that are likely to belong to the same object.

Limitations: This method does not handle occlusions or changes in object size well. It assumes that objects are always detected as connected points.

4. Speed Calculation:

What was used: Speed is calculated based on the movement of centroids between frames using the Euclidean distance and frames-per-second (FPS).

Why used: The speed of an object can be estimated by tracking its centroid position across multiple frames. The speed is derived from the distance traveled between frames.

Why it works: This method assumes the centroid is representative of an object's position and uses simple Euclidean distance for speed calculation, which is computationally efficient.

Limitations: The method may not handle fast-moving objects well if they move too much between frames (resulting in lost tracking).

5. Difference from Other Methods:

Traditional Edge Detection: Typically, edge detection methods like Sobel or Laplacian are used, but I used the Canny operator for its better handling of noise and ability to detect fine edges. However, Canny's parameter selection (low and high thresholds) needs to be fine-tuned for different videos, and incorrect thresholds can result in poor edge detection.

Cluster-based Tracking: Instead of more advanced tracking algorithms like Kalman filters or optical flow, I used basic centroid-based clustering. This is computationally simpler but lacks robustness in cases of occlusions, changes in shape, or fast motion.

Other Speed Calculation Methods: I used a basic Euclidean distance between centroids for speed calculation. More advanced methods like optical flow or tracking-by-detection can provide more accurate results, especially in dynamic or occluded scenes.

6. Why the Implementation Does Not Work 100% Correctly:

Edge Detection Issues: The Canny edge detection can struggle with noisy or blurry frames. Objects with subtle boundaries might not be detected effectively. Additionally, static edges (non-moving objects) are included in the speed calculation, even though they shouldn't contribute.

Clustering Issues: The clustering technique may fail when objects overlap or have irregular movement. It relies heavily on a fixed distance threshold (epsilon), which doesn't adapt to varying object sizes or densities.

Speed Calculation Issues: Speed calculations are based on centroid movement. If objects move too quickly or if there's significant displacement between frames (due to low frame rates), the speed estimate may be inaccurate. Also, if objects are missed or incorrectly clustered, speed values may not be reliable.

7. Final Thoughts:

The methods used here are basic and efficient for simple videos with clear object boundaries and slow movement. However, they do not scale well with complex scenes (e.g., multiple moving objects, occlusions, fast motion). More advanced techniques like optical flow, object tracking algorithms, or machine learning-based object detection could improve the performance in such cases.