CP1

Terenti Kakhniashvili

November 19, 2024

Abstract

This document presents a project focused on object tracking and speed detection using video processing techniques. The implementation utilizes OpenCV and machine learning algorithms to detect and track objects in real-time while calculating their speeds based on pixel movement.

Attention: It could take up to 2-3 minutes to run

1 Introduction

In recent years, video processing has become an essential tool in various fields such as surveillance, traffic monitoring, and autonomous vehicles. This project aims to develop an object tracking system that can detect moving objects in a video feed and calculate their speeds in real-time.

2 Methodology

The methodology consists of several key components:

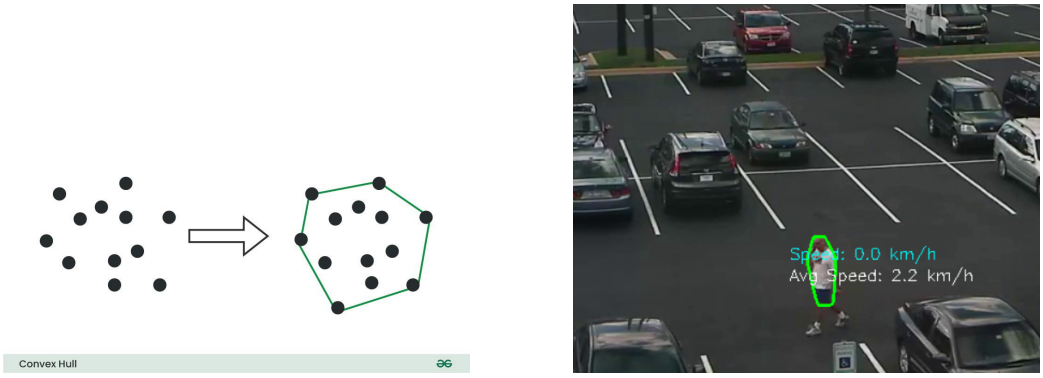
2.1 Video Input and Background Subtraction

The system captures video frames using OpenCV's `VideoCapture` class. Background subtraction is performed using the MOG2 algorithm to isolate moving objects from the static background.

2.2 Object Detection and Tracking

The detected objects are processed using contour detection and clustering techniques (DBSCAN) to identify individual objects. The tracking is managed through a custom `ObjectTracker` class that maintains the state of each detected object. also DBscan is useful when we do not know exact amount of objects, it gives the great result for the some epsilot and cluster numbers

i also use Convex hull, it contains all the points. In simpler terms, it is the smallest polygon



(a) general (b) Caption for Image 2

Figure 1: Overall Caption for Both Images

2.3 FIlters

```
    def apply_filters(self, frame: np.ndarray, fg_mask: np.ndarray)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        blurred = cv2.GaussianBlur(gray, (5, 5), 1.0)
        sobelxy = cv2.Sobel(blurred, cv2.CV_64F, 1, 1, ksize=5)
        edges = cv2.Canny(cv2.convertScaleAbs(sobelxy), 40, 80)
        filtered = cv2.bitwise_and(edges, edges, mask=fg_mask)
        return filtered
```

gray_scale : $Simplifiesthe image by reducing the dimensionality from three channels (BGR) to one ech$
$reduces noise and smoothens the image, which helps in improving the accuracy of edge detection. The C$
$Highlights edges by detecting abrupt changes in intensity. The kernel size of 5 ensures that the gradien$

The following code snippet illustrates the core logic for updating tracked objects:

2.4 Interpolation

i also use interpolation to fill the gaps in the contours and have better object detection, i use linear interpolation

```
def update(self, centroids):
    current_time = time.time()
    new_tracked_objects = {}

    for centroid in centroids:
        # Logic to match centroids with tracked objects...
        if matching_id is not None:
            # Update positions and timestamps...
            speed = (dist / self.pixels_per_meter) / time_diff
            speed_kph = speed * 3.6
            obj_data['speed'].append(speed_kph)
            # Calculate moving average speed...
```

2.5 Speed Calculation

The speed of each tracked object is calculated by measuring the distance traveled between frames and converting it into kilometers per hour (km/h). A moving average is employed to smooth out fluctuations in speed readings. in the video we can clearly see that pedestrian is walking bu 8-9 parking space in 20 seconds we, based on that average speed is 8 km/h, based on that speed calculations are right. at the end of the video we can see that pedestrians speed is somewhere 7.9-8.2 range. other speed's also tracked correctly,

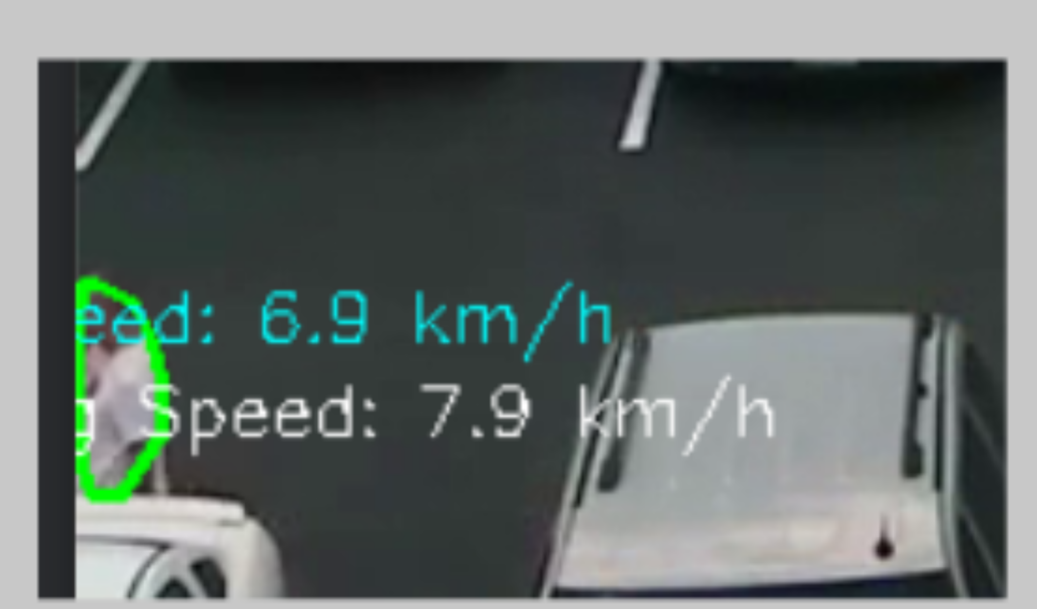
3 Results

The system was tested on a variety of video inputs, including parking lot surveillance footage. The results demonstrated accurate detection of moving vehicles with corresponding speed calculations displayed on the video feed.

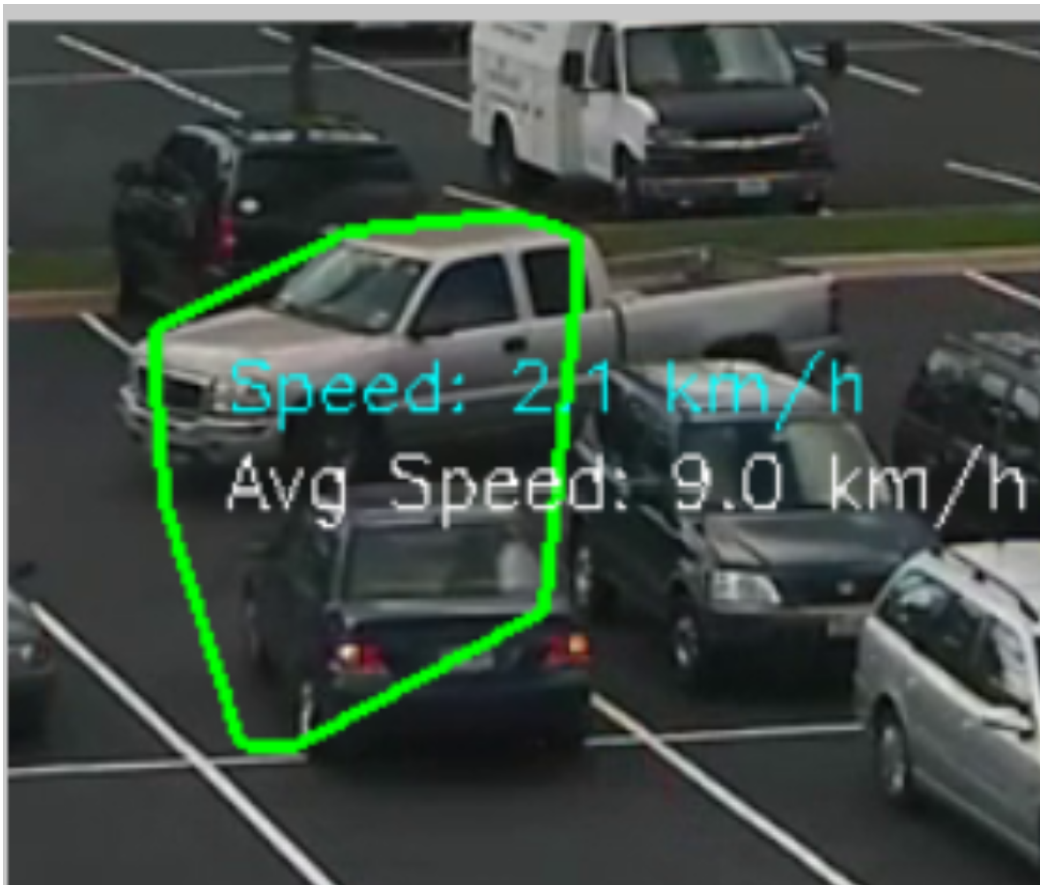
An example output frame is shown in Figure 5, illustrating detected objects with their calculated speeds.

4 Conclusion

This project successfully implemented a real-time object tracking and speed detection system using Python and OpenCV. The results indicate that the



system can effectively track multiple objects and provide accurate speed measurements, making it suitable for applications in traffic monitoring and surveillance. but there are some unadjustable errors, when 2 cars(FIgure 6) are close together their respective cluster collide because of the high eplsilon, but without this epsilon object detection would not be possible. tried schaar's filters but result was not satisfactory. therefore i use computational heavy techniques such as interpolation, background removal, convex hull to get the best result, also with with epsilon, i could increase points of interpolation but my laptop wont be able to do that, DBscan is also doesnt works with these many points


Speed: 2.1 km/h
Avg Speed: 9.0 km/h

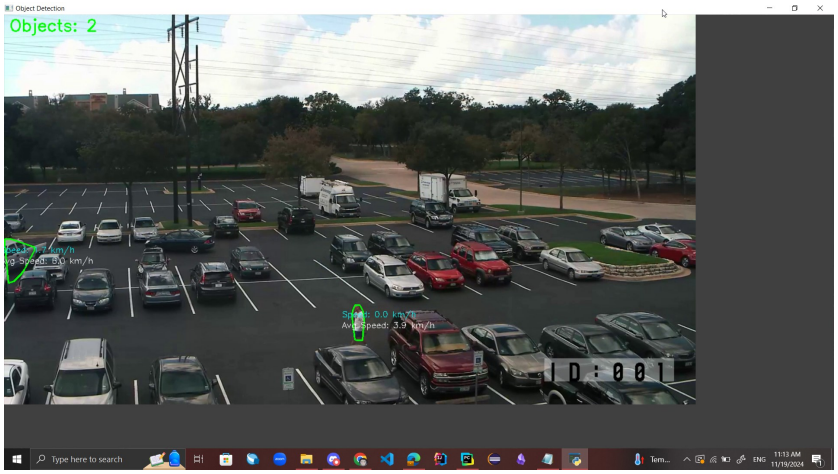


Figure 2: Example output frame showing detected objects with speed annotations.

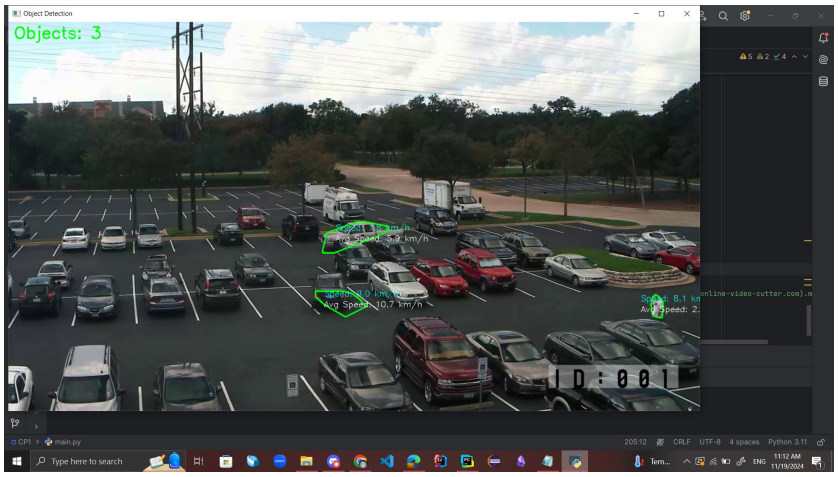


Figure 3: Example output frame showing detected objects with speed annotations.

Figure 4: Example output frame showing detected objects with speed annotations.