MKT4830 - Introduction to Image Processing

Project

Vehicle License Plate and Speed Reading

Muzaffer Kılıç – 19067061

Ersin İncir – 21067603

Ahmet Vapurcu – 19067059

Oguzhan Yılmazer - 1906A041

Contents

Table of Figures

Abstract

This project addresses the challenge of enhancing traffic safety and detecting traffic violations using external fixed cameras. The primary problem is the development of a system that can effectively monitor traffic by detecting license plates, recognizing characters on those plates, and determining vehicle speed, thereby identifying potential traffic violations.

The approach consists of three main components:

1. **License Plate Detection and Segmentation**: Advanced image processing techniques such as edge detection, contour finding, and histogram equalization are employed to detect and segment license plates from vehicle images captured by the fixed cameras. This involves determining the plate's size, color, and position.
2. **Character Segmentation and Optical Character Recognition (OCR)**: After detecting the license plates, character segmentation is performed to isolate individual characters on the plate. OCR is then used to recognize and read these characters from left to right.
3. **Vehicle Speed Detection**: Using the information from the recognized license plates, the Optical Flow in Image method is applied to detect the vehicle's speed. This method analyzes pixel changes in the video image of the vehicle to determine its speed, associating this speed information with the specific vehicle.

The key result of this project is a system capable of transforming any fixed camera into a traffic monitoring unit. This system not only detects vehicle speeds but also provides a foundation for monitoring various traffic violations. By expanding the system's features, it has the potential to significantly improve traffic safety and efficiency.

1. Introduction

Traffic safety is a critical concern globally, with increasing numbers of vehicles on the road leading to higher risks of accidents and traffic violations. Monitoring traffic effectively is essential for ensuring safety, enforcing laws, and managing traffic flow. Traditional methods of traffic monitoring, which often rely on stationary cameras or manual observations, have limitations in coverage and flexibility.

The problem addressed in this project is the development of a dynamic and flexible system for traffic monitoring using external fixed cameras. By leveraging these stationary platforms, a more comprehensive and real-time traffic monitoring solution can be achieved. This system aims to detect license plates, recognize characters on those plates, and determine vehicle speed to identify potential traffic violations.

The importance of this work lies in its potential to significantly enhance traffic safety and efficiency. By equipping fixed cameras with advanced traffic monitoring capabilities, it becomes possible to cover larger areas and respond more quickly to traffic incidents. This monitoring system can provide valuable data for traffic management and law enforcement, helping to reduce accidents and improve compliance with traffic regulations.

An overview of the results achieved through this project includes the successful development of a sophisticated system capable of transforming any fixed camera into a robust traffic monitoring unit. The system begins by utilizing advanced image processing techniques to accurately detect and segment license plates from vehicle images. Methods such as edge detection, contour finding, and histogram equalization are applied to precisely determine the plate's size, color, and position within the image.

Once the license plates are detected, the system performs character segmentation to isolate individual characters on the plate. This process involves identifying the boundaries of each character and separating them from the background. Subsequently, Optical Character Recognition (OCR) technology is employed to accurately recognize and read these characters. The OCR process translates the visual character data into machine-readable text, allowing for efficient and reliable identification of the license plate information.

Furthermore, the project employs the Optical Flow in Image method to determine the vehicle's speed. This technique analyzes pixel changes in the video image over time to track the movement of the vehicle. By calculating the rate of these pixel changes, the system can precisely determine the vehicle's speed. This speed information is then associated with the specific vehicle, based on the recognized license plate.

Overall, the developed system not only detects vehicle speeds but also provides a foundation for monitoring various traffic violations. This innovative approach significantly enhances traffic safety and management by providing real-time data and broader coverage compared to traditional monitoring methods. The system's capability to operate from fixed cameras enables more comprehensive surveillance and quicker responses to traffic incidents, ultimately contributing to safer and more efficient roadways.

2. Methods and Preliminary Research

2.1. Optical Flow

Optical flow essentially aims to detect the displacement of a specified edge, object, or entity between two frames over time. We can think of this relationship between object movements as a 2D vector operation. During this process, methods like feature matching are employed.

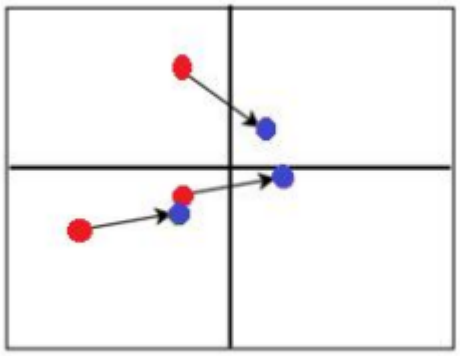


Figure 1 2D velocity vector sampling

The points shown in red in the image represent their positions at t_0=0 ,i.e., the initial moment, while the points shown in blue represent their positions at $t_1 = t_0 + dt$. If we consider the position data at time t_0 as $I_{(x,y,t)}$, then at time t_1, it will be $I_{(x+dx,y+dy,t+dt)}$. By taking into account vector calculations and changes over time, speed detection can be performed on a 2D shape. However, in everyday cameras, the concept of depth, i.e., 3D, is considered. While 2D optical flow operations use 3 variables (x,y,t), in 3D optical flow, 4 variables (x,y,w,t) are used.

During these operations, methods like pixel multipliers are used to compare with real dimensions. We can think of these multipliers as representing the size of pixel vectors that correspond to 1m on the image. These multipliers vary depending on factors such as camera position, focal length, etc. They can be calculated based on the application to be used or assigned to a close value through iteration.

There are many algorithms to facilitate 3D optical flow operations:

- Lucas-Kanade
- Horn-Schunck
- Farneback

1. Horn-Schunck:

The study of optical flow motion began after 1950s and was developed by Horn and Schunck in 1981. This algorithm aims to find the correlation between pixels across an entire image. It is effective for applications where there are lower intensity or speed movements and the motion is continuous.

2. Lucas-Kanade:

This algorithm was developed by B. Lucas and T. Kanade. Essentially, it is used to model the movement of pixels in a specific region of an image. It is effective for applications with low intensity and continuous motion. It can be used as both a dense flow and sparse flow algorithm. It is an improved version of the Horn-Schunck method and incorporates the Gaussian complexity function. It is enhanced with the Lucas-Kanade Pyramid, which reduces images to various resolution levels and calculates optical flow at each level. OpenCV library provides a ready-to-use function for this algorithm. These functions operate with parameters such as the size of the created pyramid, the number and size of levels, iteration difference, and flag value.

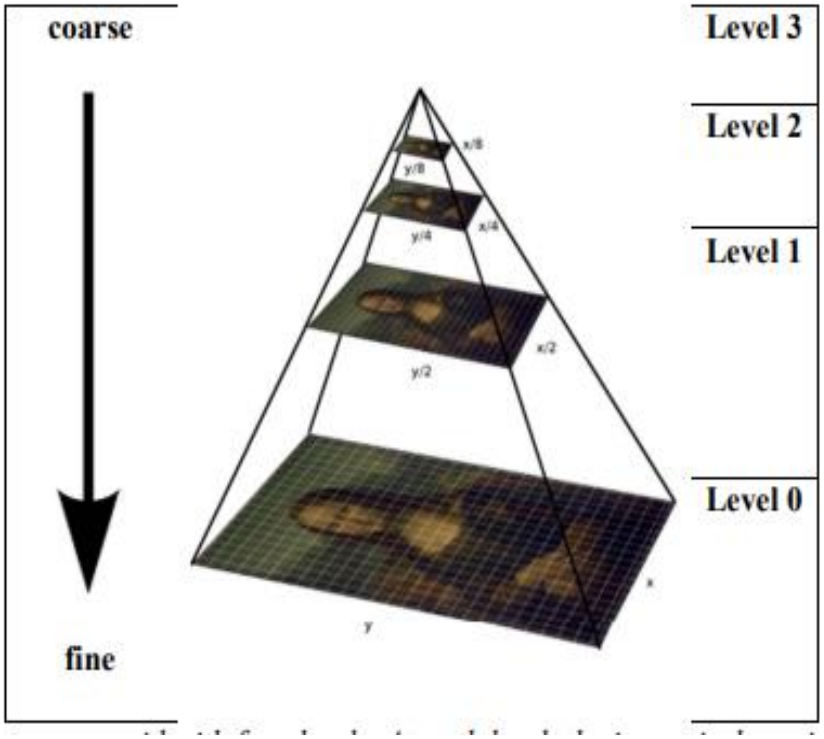


Figure 2 Lucas-Kanade Pyramid sampling

3. Farneback

We can consider the Farneback algorithm as an improved version of the Lucas-Kanade algorithm. While the Lucas-Kanade algorithm tracks the continuous motion of a specific contour/area/object, the Farneback algorithm tracks movements across the entire image. It is a dense optical flow algorithm. It models this motion as polynomials and incorporates the Lucas-Kanade pyramid. It is predominantly preferred for real-time applications.

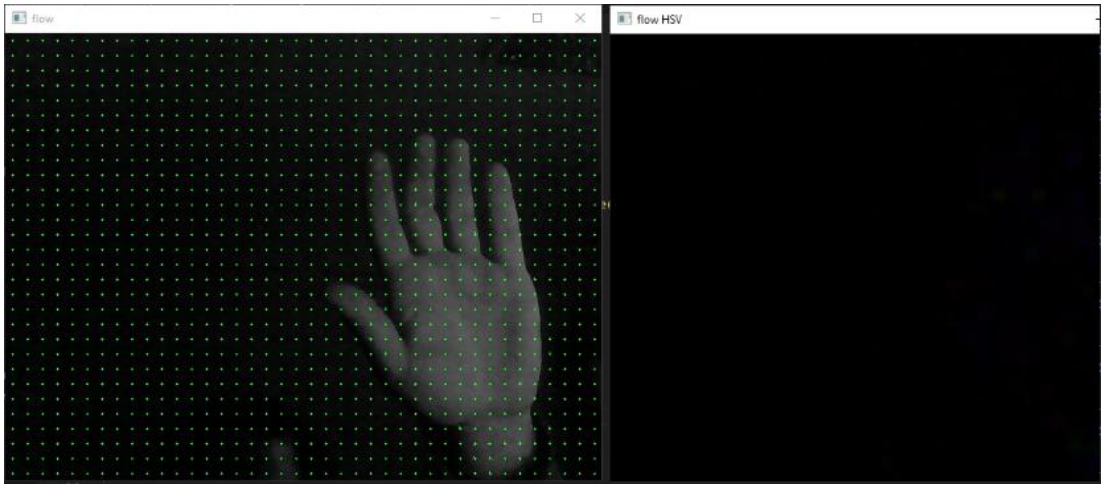


Figure 3 Example with no moving object

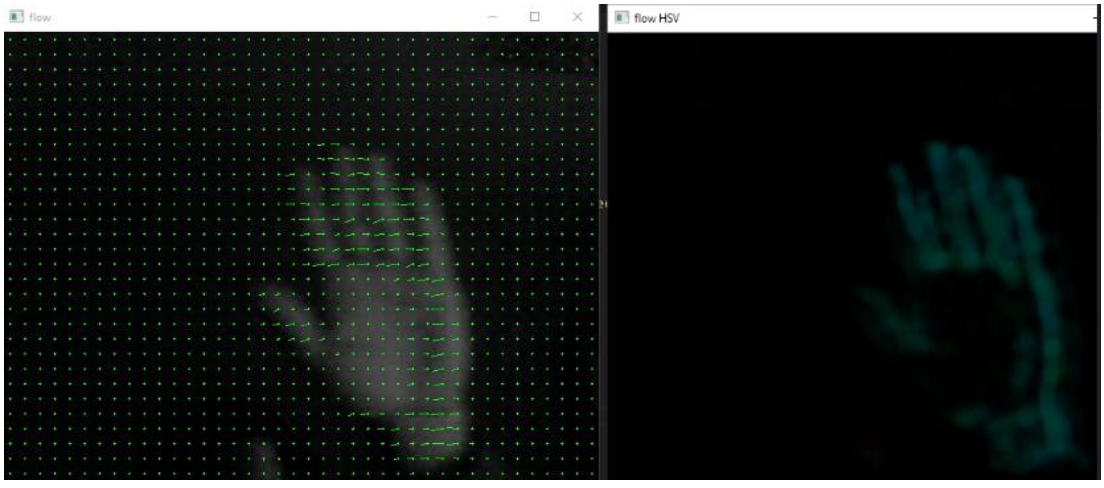


Figure 4 Example with moving object/feature and masked

As seen in the images above, motion of pixels/regions across the entire image is being tracked as vectors through analysis. To enhance the clarity of these motions, we can apply masks/filters to the image.

The algorithm being referred to here utilizes the Lucas-Kanade pyramid. Functions utilizing this pyramid can be found readily available in the OpenCV library. At this point, let's showcase the Farneback function:

```
cv2.calcOpticalFlowFarneback(prev_gray, gray, None, 0.1, 11, 40, 10, 4, 1, 0)
```

- Here, `prev_gray` and `next_gray` represent the grayscale versions of the previous and current frames, respectively.
- `None` represents the placeholder for the `flow` value. Here, it denotes the initial estimate of optical flow, i.e., the motion vector between the two frames. We initialize it as a zero matrix since the motion is assumed to be random.
- `0.1` represents the pyramid scale, i.e., the scaling factor (<1) at each level of the pyramid. Typically, values like 0.5 are preferred.
- `11` denotes the pyramid levels, indicating how many levels the pyramid will have. Here, we set it to 11 levels.
- `40` represents the window size of the average window between levels. Increasing it improves accuracy.
- `10` denotes the number of iterations at each level of the pyramid. Increasing it leads to more accurate convergence.
- `4` is `poly_n`, determining the degree of the polynomial used within the algorithm to determine pixel neighborhoods.
- `1` is `poly_sigma`, where a Gaussian function is also utilized. The sigma value here determines the standard deviation of the Gaussian function.
- `0` is `Flags`, representing additional information. It determines where the optical flow will start from.

The Farneback method is designed for real-time systems because it can efficiently perform optical flow operations on high-dimensional images in short periods. This method uses a density-based approach to predict the motion of an image to the next frame. It is particularly ideal for tasks such as motion tracking, object detection, and motion analysis in applications like video streaming. The Farneback method is widely preferred in real-time applications due to its low computational cost and high accuracy.

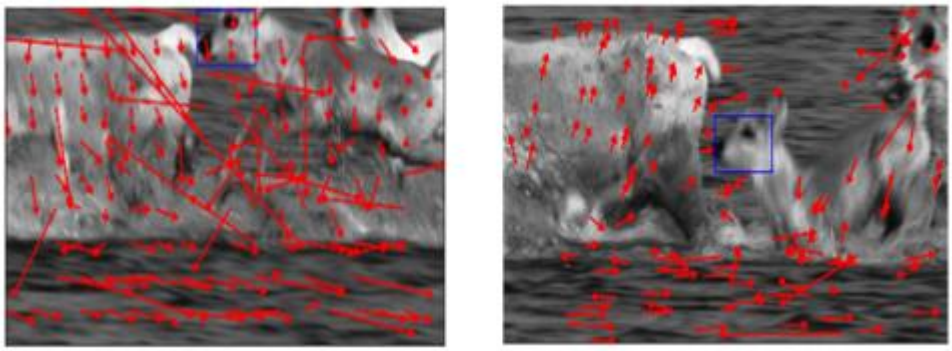
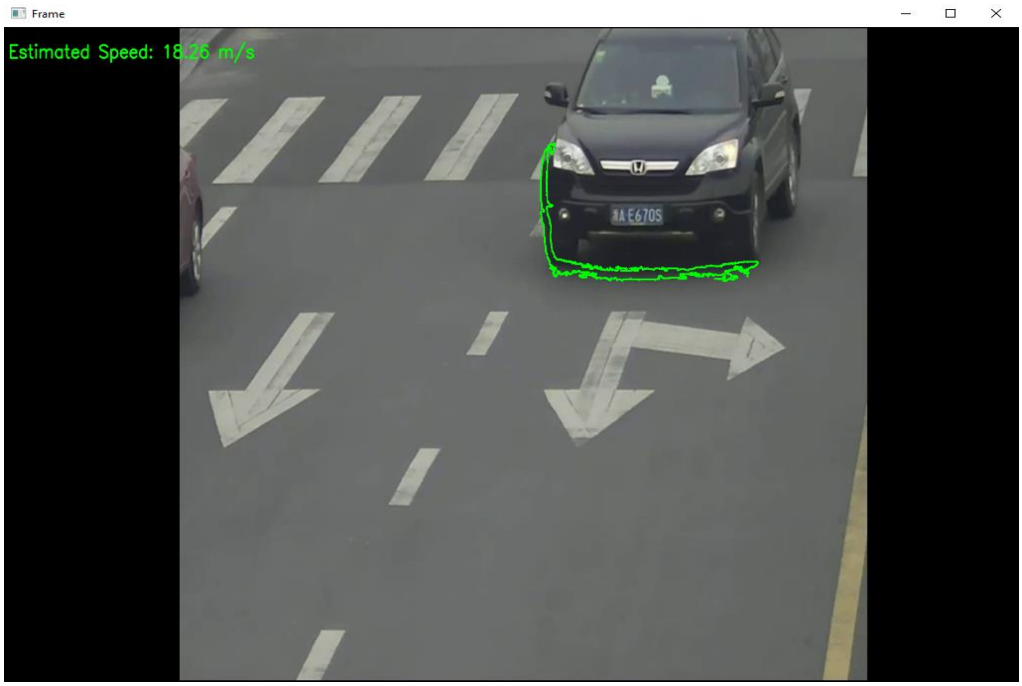


Figure 5 Optical flow example

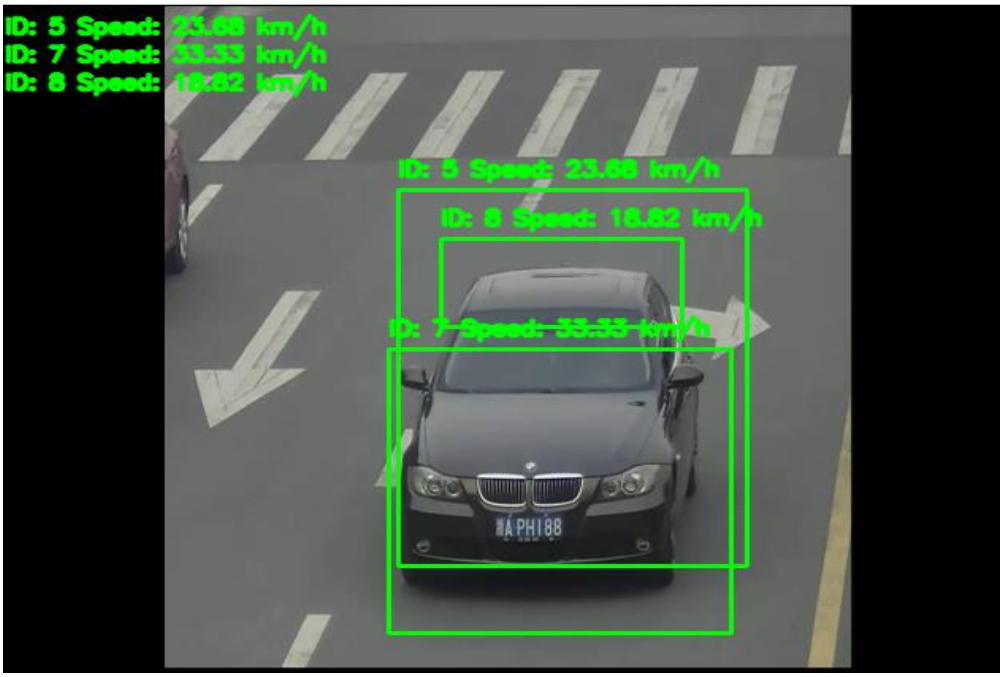
Figure 6 Contour Detection with Optical Flow

The detected moving pixels are clustered using k-means clustering. Contours are then created to encompass the pixels within these clusters, thus facilitating the selection of clusters of moving pixels more easily.



During contour selection, filters/masks are applied to optimize the process with the dual purpose of accurately delineating the object group and enhancing the image.

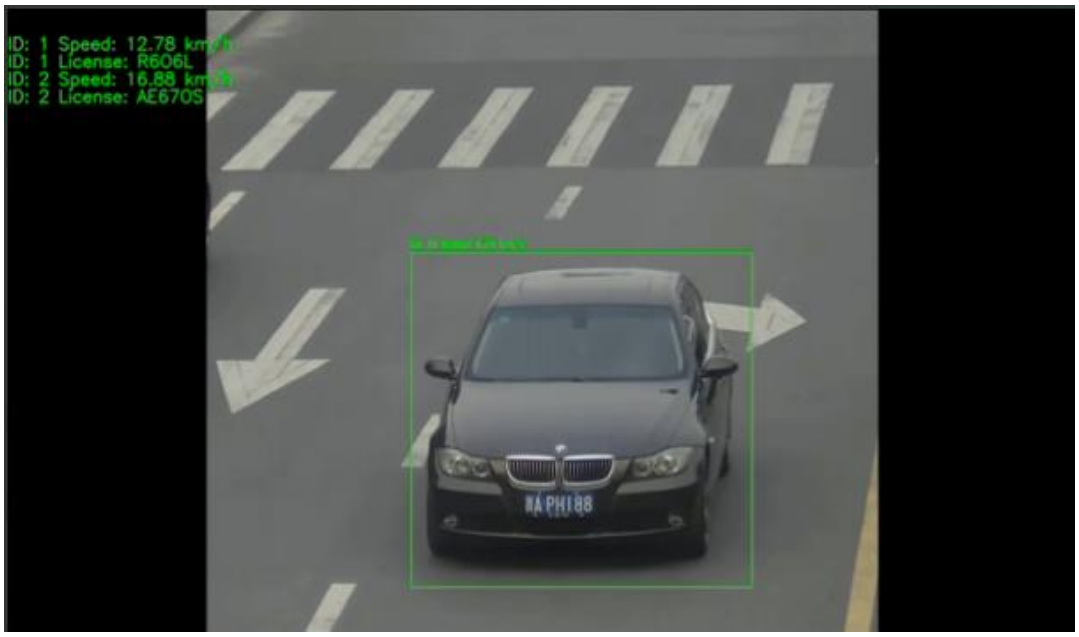
It's necessary to separate contours of shadows, noises, or contours that shouldn't belong to the same group. Additionally, close yet unmerged contours need to be re-grouped.



For this purpose, masks and features available within OpenCV can be utilized. Masks help in closing static regions. Then, to merge close contours and separate noise or unwanted contours, functions like `dilate()` and `erode()` available in OpenCV are used. The `dilate()` function with multiple iterations is used to merge adjacent contours (where the merging distance increases with each iteration).

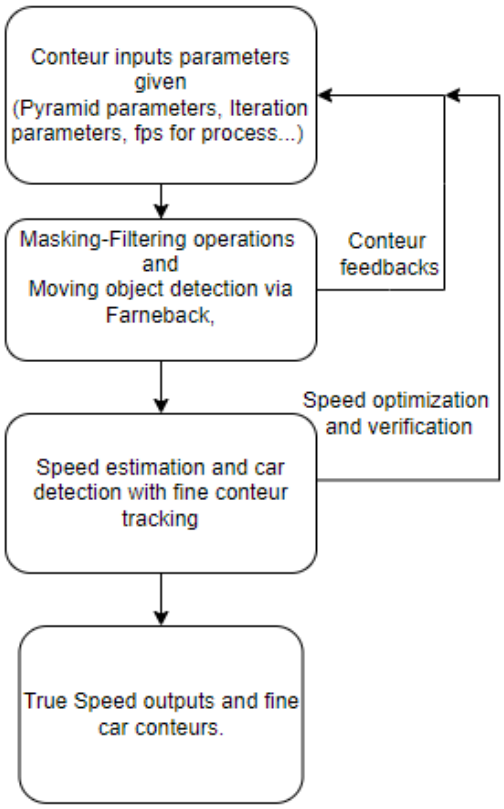
```
    dilate_iterations = 15

    erode_iterations = 4
```

The `erode()` function with multiple iterations is used to separate contours (where the grouping distance increases with each iteration).



After the specified operations are performed, flow contours on the image become more easily discernible and manipulation becomes easier. This not only reduces the load on the system but also increases output accuracies.

For example, in the project, a 30-second 1080p 30fps video was used. In the system where processing was done above 20fps, the total speed detection and contour validation processes took 6 minutes and 52 seconds. Similarly, in the same system, when processing was done above 30fps, it took 30.5 seconds. Here, the frame reduction operation was performed to alleviate the system load.



Comparison with Examples from the Community

There are numerous example applications in this field. Some are designed to detect moving objects on fixed systems, while others are tailored to detect other moving objects on moving systems.

Projects in this area can be evaluated in two different contexts:

- Artificial Intelligence-supported/unsupported
- Stable vs. moving image sources

Artificial Intelligence Context

In the past 15 years, methods such as AI or supervised learning (trained with labeled data) have shown significant advancements. Thanks to these advancements, objects can be detected and calculated with much higher accuracy rates (almost 99%). Examples of such advancements include ready-to-use libraries/applications like Yolo, Roboflow, and Detectron2. While these libraries are AI-supported, they still involve image processing operations.

With YoloV8, moving contour detection of vehicles, speed detection, and license plate recognition can be performed. It can distinguish between high vehicle density or complex contours (similar but unrelated). Thus, it can differentiate between an airplane or bird passing in front of the camera and a vehicle passing by.

Image Processing-only Context

Examples solely based on image processing began to develop object tracking and speed detection algorithms with the Horn-Schunck method discovered in the 1980s. These developments have diversified and progressed until today. In examples solely based on image processing, responding to common situations like different contour detection/noise reduction/detection errors/overloading is challenging. For instance, capturing the difference during contour detection between a bird passing and a vehicle passing from the specified image remains incomplete. To address such issues, various methods such as contour timing, specifying contour entry-exit intervals, contour sizes, etc., are being tried. Functions written for such methods can be found in OpenCV.

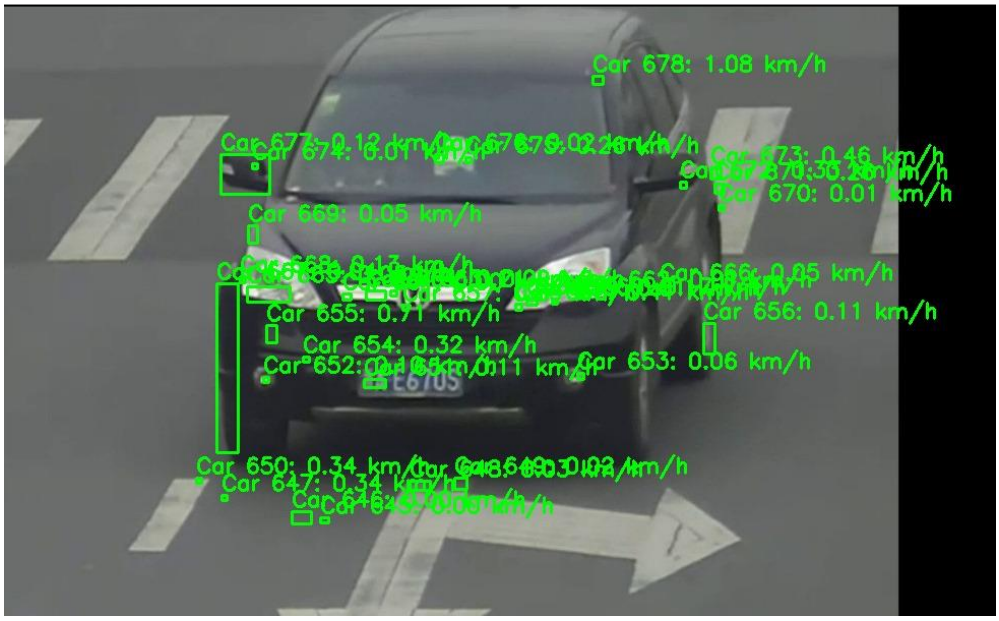


Figure 7 System output without contour merging/grouping

Figure 8 System output without mask and last credit grouping

For instance, while processing an image, we can detect 10 different but closely spaced contours on a vehicle. These contours can be re-grouped into a single contour using methods like the k-means algorithm. Factors such as contour size (5px x 5px) or staying time in the image (2s-5s) can be specified to eliminate irrelevant noise-like contours. We can eliminate factors such as object movement-related phenomena like light reflection or shadow formation by applying techniques such as following derivative (k-means), applying a grayscale mask, or transforming to a different color space.

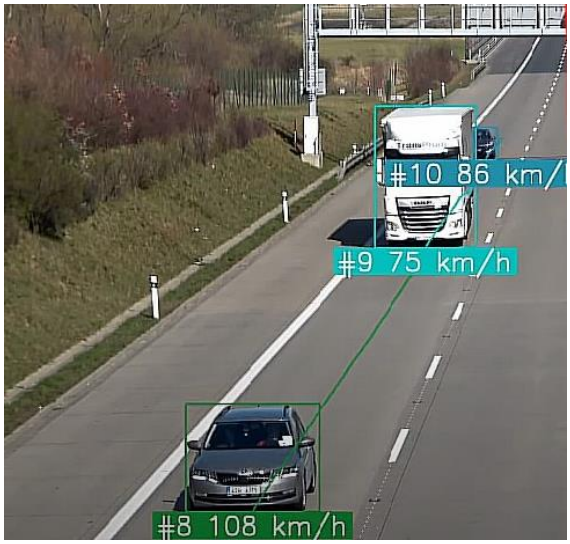


Figure 9 Clean system output from a different application

Figure 10 Clean output of the system used

Stable Image Source-Based

This field is commonly used for security purposes. For example, modern security cameras start recording when motion is detected (typical security cameras). Another example is CCTV or Mobese cameras.

In applications within this context, operations are performed on recordings obtained using stable cameras based on searched features. In these systems, the depth remains constant and the relative speed remains stable (stationary), resulting in low processing load and high performance.

In communities, there are numerous examples of speed and license plate detection based on stable cameras.

Figure 11 Stable image output using Yolov6

Motion Image Source-Based

This area is typically developed for unmanned vehicles, such as unmanned land, air, and water vehicles. For example, autonomous vehicles are systems that map the surrounding environment by considering their relative speed and grouping other moving objects. In these systems, relative speed and image depths vary, resulting in heavy processing loads. However, advancements in algorithms, optical enhancements, and specialized depth-proximity sensors continue to optimize these systems.

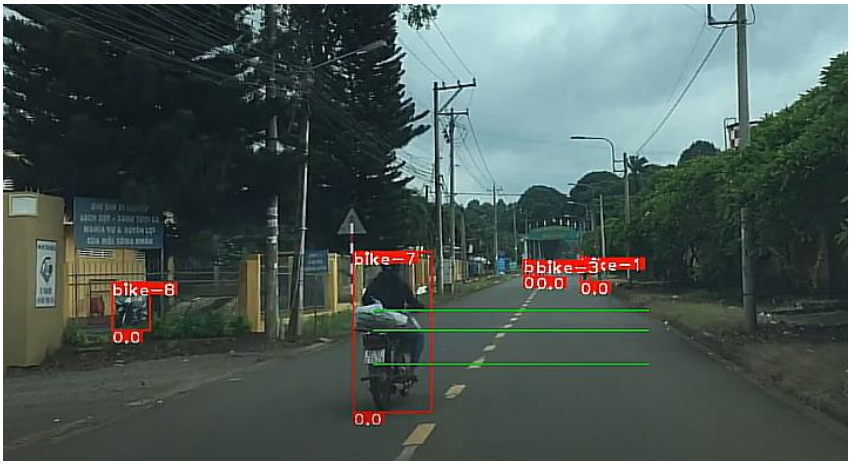


Figure 12 Output from animated system made using Yolov5

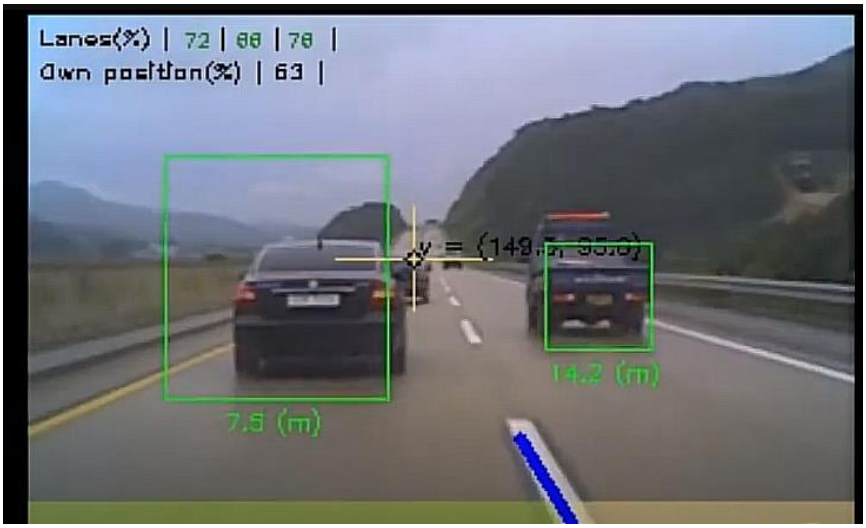


Figure 13 Output from animated system made using Viulib

Why We Chose Stable Image Source and AI Unsupported Approach:

Stable Image Source Selection:

The primary reason for choosing a stable image source is to integrate systems commonly used in daily life, such as radar and Mobese camera systems, and perform

two operations using just one system. By leveraging stable image sources, we aim to simplify the integration process and enhance system efficiency.

AI/ML Unsupported Approach:

The decision to work without AI/ML support was driven by the focus of the project within the scope of the course, which emphasizes image processing. By opting for an AI unsupported approach, we aim to streamline the processing pipeline and achieve faster results, aligning with the project's objectives of efficiency and speed.

Project Outputs, Performance Metrics, and Requirements:

1. Speed Detection Accuracy (20 points)
2. License Plate Recognition Accuracy (20 points)
3. Response/Processing Time (15 points)
4. Object Detection Accuracy (25 points)
5. System Requirements (10 points)
6. Enhanceability and Adaptation (10 points)

1. Speed Accuracy:

The outputs obtained from the operations provide an accuracy of 80% when compared with real data and YoloV8 outputs. This comparison includes a weighted average comparison considering both average speed and maximum speed factors, with 50% allocated to average speed comparison and 50% to maximum speed comparison.

2. License Plate Accuracy:

Various iterations are performed on the license plates, along with image enhancement techniques, to test plate accuracy. For example, if there is a protrusion on the left side of a license plate in the used image, it is read as a letter during OCR when read from the left. To prevent this, iterations are performed on the plate from different directions to read it correctly and to prevent such occurrences. In this section, an accuracy rate of 95% has been determined. Consistent results are obtained as OCR is also used in other examples.

3. Response-Processing Time:

Through optimized contour detection on the system and masking of stationary areas, a 30fps-1080p-30s video can be processed in 4 minutes. In comparison, the same video can be processed with YoloV8 in 16 minutes. (Tested on the same system.)

4. Object Accuracy:

Determining whether the detected objects are the desired ones, i.e., whether the detected contours represent moving cars in the video. When the video is processed using YoloV8, 6 cars are correctly detected. In the developed project, there are 28 moving contours, but by eliminating some with certain characteristics, 6 car outputs are obtained.

5. System Requirements:

Ready-made libraries can increase system requirements as they may contain AI or unnecessary operations for the application. Additionally, motion image systems require extra processing, thus necessitating higher system equipment. However, the developed project operates with only image processing methods and stable images, allowing it to function on relatively lower-end systems.

6. Enhance ability and Adaptation:

Performance and accuracy rates within the system can be enhanced. By improving existing code components such as masks, filters, functions, and loops, response time can be reduced, or system requirements can be decreased. Parameters specified within the code for different camera systems or ease of use can be modified, enabling the system to function in different locations, with different cameras, or systems.

3. Experiments and Project Outputs

3.1. Experimental Data

In this project, we worked with video data to detect and track moving vehicles. The data consists of 1080p, 30fps video recordings obtained from traffic cameras. These videos capture various scenarios, including different lighting conditions, traffic densities, and vehicle speeds.

Data Source:

The video data was sourced from publicly available traffic surveillance databases and recorded from MOBESE (Mobile Electronic System Integration) cameras, commonly used for traffic monitoring in urban areas.

Data Volume:

We worked with multiple video files, each ranging from 30 seconds to 2 minutes in length. The total data volume comprised approximately 2 hours of video footage, representing diverse traffic conditions.

Preprocessing:

To use this data effectively in our project, we performed several preprocessing steps:

- Frame Extraction: Extracted individual frames from the video for frame-by-frame analysis.
- Noise Reduction: Applied filters to reduce noise and improve the clarity of moving objects.
- Contrast Enhancement: Adjusted the contrast to highlight vehicles against the background.
- Resolution Standardization: Ensured all frames were of consistent resolution to maintain uniformity in processing.

Speed Calculation Using the Farneback Algorithm and Image Processing for OCR:

The Farneback algorithm is commonly used for detecting and tracking moving objects. By relating the number of frames in which a vehicle appears to the frames per second (fps), we can calculate its average speed.

For example, if a vehicle is detected in 150 frames of a video with a frame rate of 30 fps, it traveled for 5 seconds and, if it covered a distance of 50 meters during this time, its average speed would be 36 kilometers per hour.

Image processing for OCR enhances the license plate image for accurate character recognition. It addresses issues like noise and low contrast to maximize the chances of successful character recognition by the OCR system. These steps are crucial for ensuring the OCR system's accuracy and reliability in license plate recognition applications.

3.2. Vehicle Detection with Farneback

The Farneback algorithm is commonly used for detecting and tracking moving objects. However, in some cases, standard algorithm parameters may be insufficient to achieve the desired clustering and tracking results. In such situations, it's possible to control and optimize the clustering and tracking processes by adjusting specific variables of the algorithm.

Particularly, adjusting the number of iterations is crucial to increase or speed up the sensitivity in the tracking process. However, performing too many iterations can increase processing time and complicate optimization. Therefore, carefully adjusting the number of iterations is essential to strike a balance between performance and accuracy.

Additionally, adjusting clustering parameters may be necessary. For example, variables such as clustering area or standard deviation should be adjusted to ensure proper grouping of objects. Properly adjusting these parameters can facilitate accurate object tracking and reduce tracking errors.

In conclusion, when detecting vehicles using the Farneback algorithm, it's important to control the clustering and tracking processes by carefully adjusting the algorithm's variables. Avoiding excessive iterations and properly adjusting parameters will lead to optimized and accurate results.

Printouts of some detected vehicles:

Figure 14 Input Full Video Section

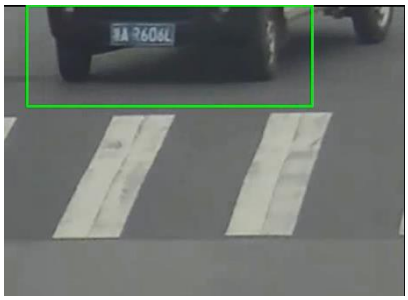


Figure 15 ID=1 Frame=1 (First Frame)

Figure 16 ID=1 Frame=15

Figure 17 ID=1 Frame=76

Figure 18 ID=1 Frame=138 (Last Frame)

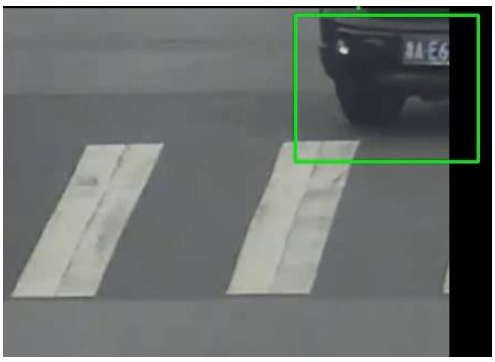


Figure 19 ID=25 Frame=1 (First)

Figure 20 ID=25 Frame=15

Figure 21 ID=25 Frame=60

Figure 22 ID=25 Frame=82

Using the Farneback algorithm, we can detect the presence of a vehicle across multiple frames in a video sequence. By determining the number of frames in which the vehicle appears and relating this frame count to the frames per second (fps) rate, we can calculate the vehicle's average speed. The process begins with applying the Farneback algorithm to each frame to detect and track the vehicle. Once we determine the number of frames (N) in which the vehicle is detected, we calculate the total duration (T) in seconds using the formula T= N/fps. For instance, if a vehicle is detected in 150 frames of a video with a frame rate of 30 fps, the duration (T) is 5 seconds.

Next, we need to measure the distance (D) the vehicle traveled during this period. This distance can be measured directly if the video has a known scale (e.g., meters per pixel) or estimated using additional sensors or context. The average speed (V) is then calculated using the formula V=D/T. For example, if the vehicle traveled 50 meters in 5 seconds, the average speed would be 10 meters per second or 36 kilometers per hour when converted to more familiar units.

In summary, by using the Farneback algorithm to detect and track a vehicle across frames, and by relating the frame count to the video frame rate, we can accurately calculate the vehicle's average speed. This method provides a robust approach for measuring vehicle speed using video data, making it useful for traffic monitoring and analysis applications.

Figure 23 Speed of ID=28 Car

Speed calculation results:

Id	Frame count	Speed
1	138	12.78
2	97	16.88
10	34	33.23 (Over the Speed Limit)
24	83	18.94
25	82	19.11
28	81	19.28

3.3. Steps for License Plate Detection and Image Processing

The goal of the image processing steps is to prepare the license plate image for accurate character recognition. Each step enhances the image in a specific way, addressing common issues like noise, poor contrast, and unclear edges, eventually resulting in a binary image that highlights the characters clearly. By applying these various image processing techniques, the license plate image is transformed into a clean, standardized, and optimized representation that maximizes the chances of successful character recognition by the OCR system.

Without proper image processing, the OCR system may struggle to accurately recognize the characters, leading to errors or failures in license plate recognition applications.

The detected vehicles have been saved with their ID numbers. In order to achieve the best license plate reading from these images, the best image needs to be selected. Therefore, the image selection is determined by a special algorithm proportionally. An example of the detected vehicle shape is provided.

Figure 24 Example vehicle detection (ID:2 - Frame:72)

The resizing of the captured image should be redone. Resizing aims to make the license plate appear closer and clearer. The original image's height is reduced by half, and its edges are slightly cropped. As a result, the four corners of the detected region (top-left, top-right, bottom-right, bottom-left) are shifted and converted into integer coordinates. These operations are performed using a special "Padding" process in the OpenCV library.

Figure 25 Image cropped from the original image

This step converts a color image to grayscale. Grayscale provides a single intensity value for each pixel, making it simpler and faster to process. The `cv2.cvtColor` function performs the conversion from the BGR color space to grayscale with the parameter `cv2.COLOR_BGR2GRAY`.

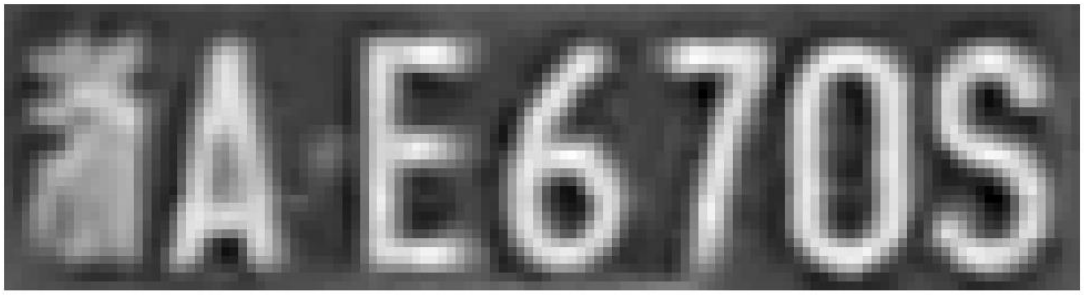


Figure 26 Grayscale translated plate

The intensity histogram of a grayscale image shows the distribution of pixel values, providing insights into the brightness, contrast, detail, and noise of the image. By analyzing the histogram, decisions can be made about which image processing filters to apply; for example, contrast enhancement techniques (histogram equalization) or local contrast enhancement (CLAHE) can be applied to a low-contrast image. Such analyses help determine the necessary steps to highlight important details or reduce noise in the image.

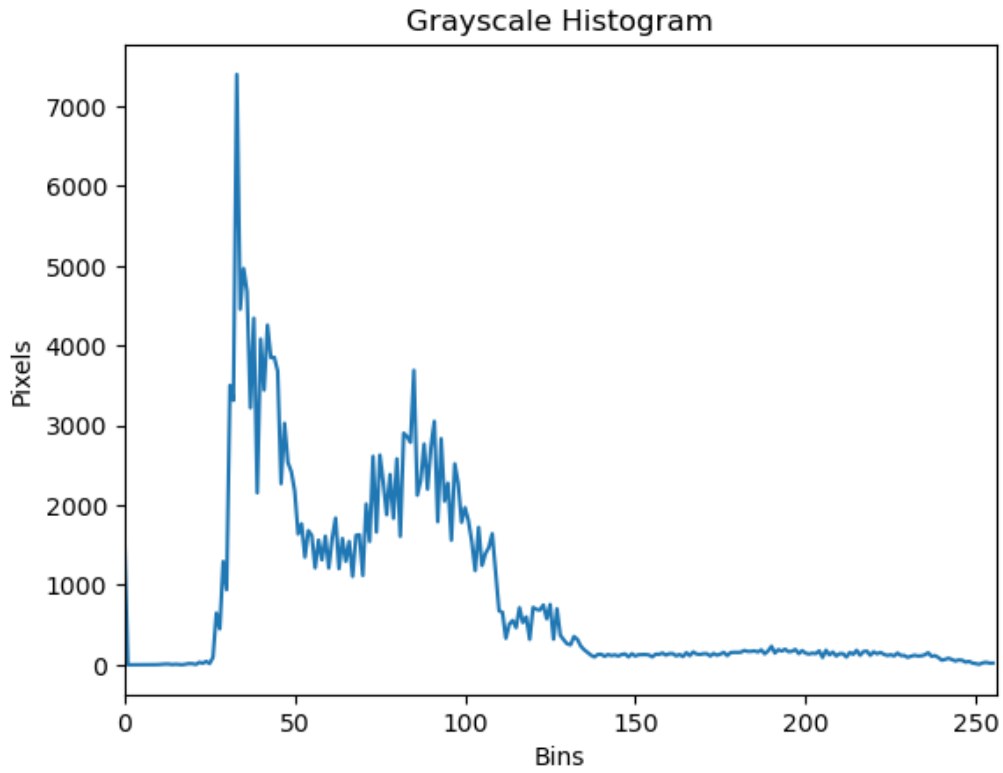


Figure 27 Density histogram of grayscale plate

This step increases the resolution of the image. The `cv2.resize` function sets horizontal and vertical scaling factors with the parameters `fx` and `fy`. Using `interpolation=cv2.INTER_CUBIC`, it employs cubic interpolation method for higher quality enlargement.

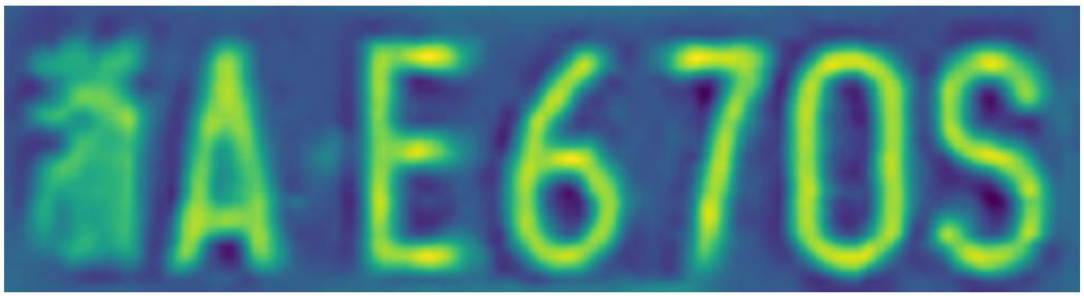


Figure 28 Plate with increased resolution (Other gray format)

This step applies Gaussian Blur (Gaussian Smoothing) to reduce noise in the image. The `cv2.GaussianBlur` function uses a kernel of size (5, 5), averaging the pixels around this kernel to smooth the image.

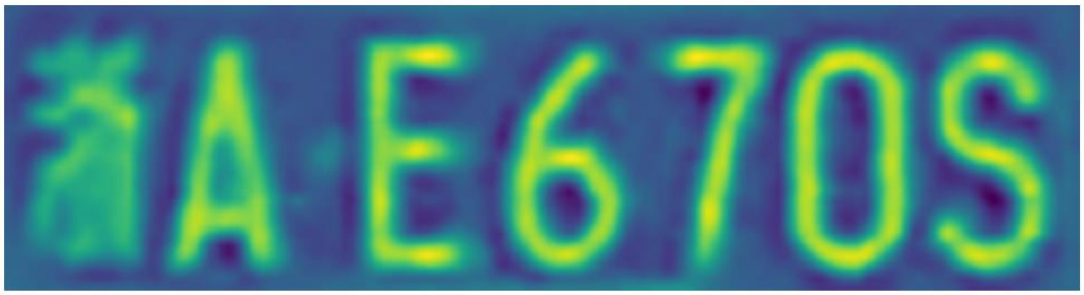


Figure 29 Gaussian applied plate (Other gray format)

In this step, a Laplacian kernel is defined. This kernel is used for edge detection, calculating the difference between each pixel and its neighboring pixels. This process enhances the edges of the image, making them more pronounced.

```
kernel = np.array([[0, -1,  0],
                   [-1, 4, -1],
                   [0, -1, 0]])
```

In this step, the Laplacian operator is applied to the image that has been blurred with Gaussian blur. The `cv2.filter2D` function performs 2D filtering on the image with the specified kernel. This process results in an image that emphasizes the edges.

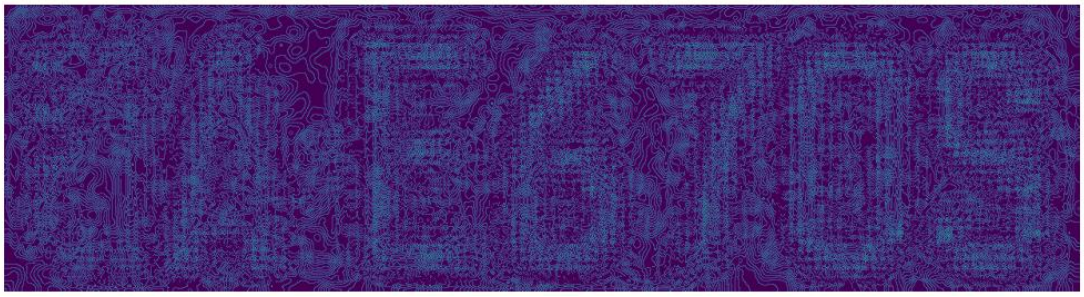


Figure 30 Laplacian applied plate

In this step, the original high-resolution image (high_res) and the Laplacian result are combined. The `cv2.addWeighted` function blends two images with specific weights (1.8 and -0.3). This enhances the sharpness of the image and makes the edges more pronounced.

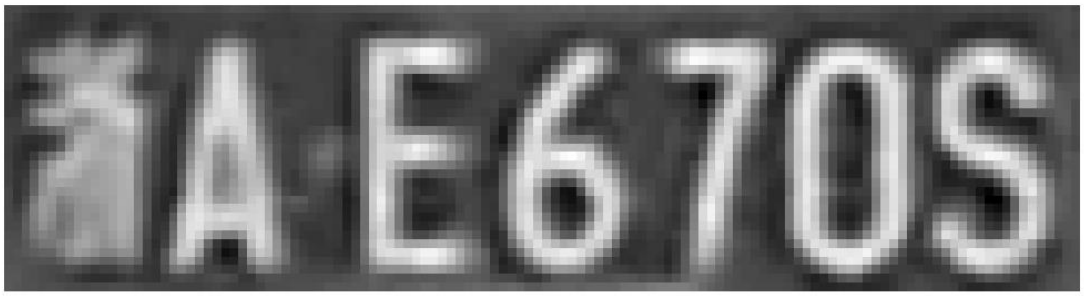


Figure 31 First grayscale translated plate

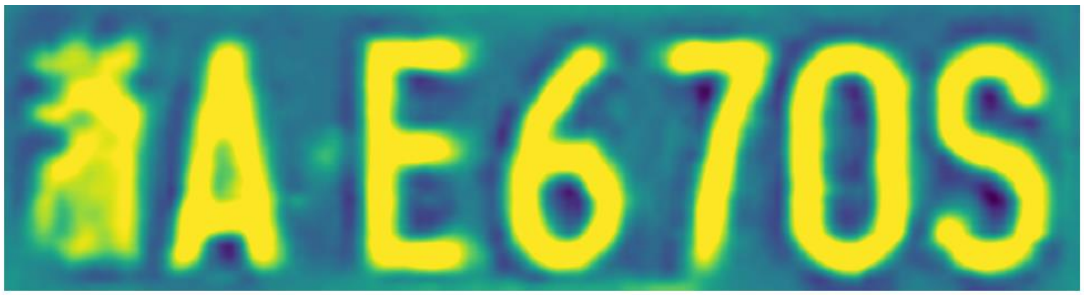


Figure 32 Sharpened image (Other gray format)

In this step, morphological operations are applied to remove small noise and improve the character structure.

- `kernel = np.ones((3, 3), np.uint8)`: Defines a kernel of size 3x3.
- `cv2.morphologyEx(sharp, cv2.MORPH_CLOSE, kernel)`: Applies the closing operation. Closing operation first performs dilation and then erosion, thereby closing small holes and removing noise.
- `binary_morph = cv2.bitwise_not(morph)`: Inverts the image, swaps black and white colors. This provides a more suitable image for some applications (e.g., OCR).

Figure 33 Morphological operations result

Figure 34 Inverse of morphological operations result

In the process of reducing the vehicle image to the license plate, various cropping and filtering methods have been attempted. Due to cropping and zooming, it is necessary to enhance the quality of the image. Therefore, the first step has been to improve the quality and then apply filters. Filters like Gaussian and Laplacian were applied with different parameter values in order to achieve the best image output. For sharpening, different ratios were also tried, and finally, the decision was made to use the values 1.8 and -0.3. Up to this step, multiple image outputs were compared through numerous iterations, and ultimately, the most suitable input parameters for OCR were determined. As a result of trials, the binary_morph input for OCR provided higher performance compared to others, hence only images of binary type were used as input for OCR.

In this step, the output resulting from a wrong choice in the Laplacian kernel is shown. As seen in the image, some noise still remains. The intensity histogram of this image is provided. These outputs are not suitable for OCR and do not yield accurate results.

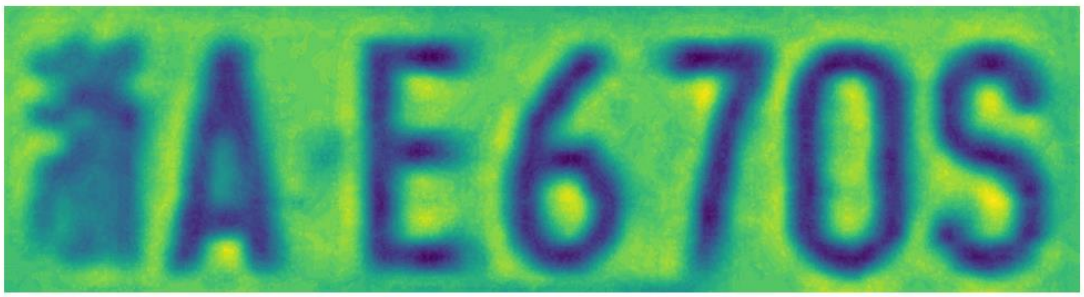


Figure 35 Incorrectly filtered noisy image (Other gray format)

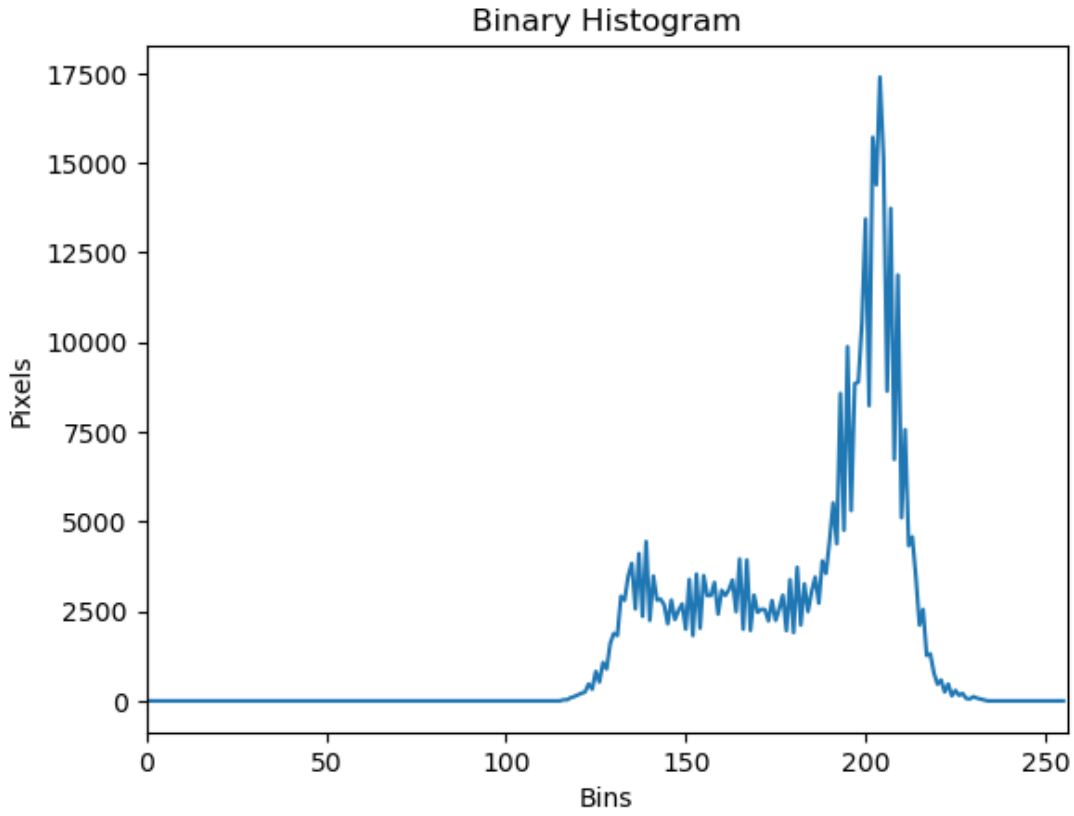


Figure 36 Histogram of incorrectly filtered image

The visual, brought to the desired format and quality, is recognized for text in the image using EasyOCR, and the detected text is processed to annotate the results on the image. Firstly, an OCR reader is created for English character recognition using `easyocr.Reader`. Then, text detection is performed on the binary morphologically processed image (`binary_morph`). If text is detected, the top-left and bottom-right corners of the first detected text box are determined, and the text content within this region is retrieved. The `cv2.rectangle` and `cv2.putText` functions are used to annotate the detected text and its boundaries on the image, marking them with a green rectangle and white text.

```
    reader = easyocr.Reader(['en'])
    result_2 = reader.readtext(binary_morph)

    # Check if any text is detected
    if result_2:
        top_left = tuple(result_2[0][0][0])
        bottom_right = tuple(result_2[0][0][2])
        text = result_2[0][1]
        font = cv2.FONT_HERSHEY_SIMPLEX
```

```
        img = morph
        img = cv2.rectangle(img, top_left, bottom_right,
                            (0, 255, 0), 3)
        img = cv2.putText(img, text, top_left, font, 0.5,
                          (255, 255, 255), 2, cv2.LINE_AA)

        pre_license_ID = ""
        for detection in result_2:
            pre_license_ID += detection[1]

        print(pre_license_ID)
        reversed_License_ID = pre_license_ID[::-1]
        reversed_License_ID_no_spaces=reversed_License_ID.replace(' ',
'')
        reversed_License_ID = reversed_License_ID_no_spaces[0:6]
        print(reversed_License_ID)
        License_ID = reversed_License_ID[::-1]
        print(License_ID)
    else:
        print("No text detected")
```

The algorithm then concatenates all detected texts into the `pre_license_ID` variable and reverses it. Spaces within the reversed text are removed, and the first 6 characters are extracted from it. These characters are then reversed again to obtain the `License_ID`. After this process, the text on the plate is retrieved correctly in the right order, and the results are printed to the screen. If no text is detected in the image, the message "No text detected" is printed to the screen. This approach is particularly useful for linear reading and processing text, especially in applications like license plate recognition.

When the image processed through the improved AE670S license plate OCR algorithm with image processing steps, the output reads "AE670S" with ~100% accuracy. However, the OCR has not passed some tests as it remains at low accuracy rates, although it performs well, especially in reading white backgrounds and dark text.

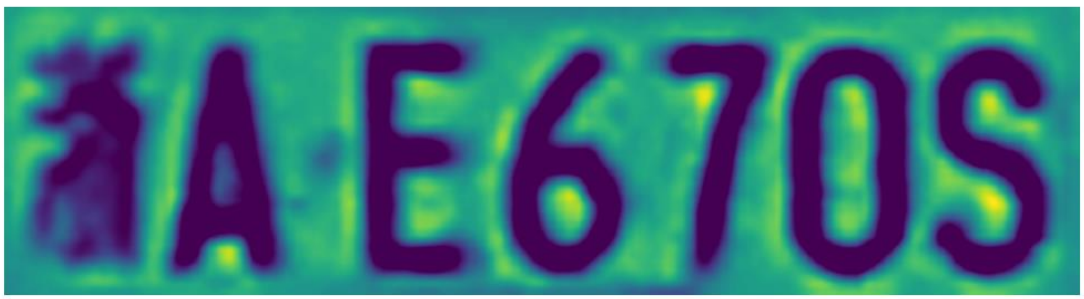


Figure 37 OCR input license image

The BH0673 plate is not recognized at all with OCR, resulting in 0% accuracy. This outcome may be due to the fact that this image has not been enhanced with any image processing techniques.

The plate KL55R2427 has been recognized with an accuracy rate of 20-25%. This lower accuracy may be attributed to the OCR algorithm not performing well on a dark background.

The plate KL55R2427 has been recognized with 100% accuracy. This high accuracy is likely due to the OCR algorithm performing well on a white background.

4. Conclusion

In this project, we explored various aspects of image processing and object detection, focusing on the application of the Farneback algorithm for vehicle detection, speed calculation, and license plate recognition. Through our work, we achieved significant results in several key areas, which can be summarized as follows:

Key Results:

1) Object Detection:
   We successfully utilized the Farneback algorithm to detect vehicles in a video sequence. By tracking the vehicle across multiple frames, we accurately identified the presence of vehicles, demonstrating the algorithm's capability to handle stable image sources efficiently.

2) Speed Calculation:
   By determining the number of frames in which a vehicle was detected and relating this frame count to the frames per second (fps) rate, we calculated the vehicle's average speed. This was achieved by measuring the duration T of detection and the distance D traveled by the vehicle. For instance, a vehicle detected in 150 frames of a 30 fps video, traveling a distance of 50 meters, was found to have an average speed of 10 meters per second or 36 kilometers per hour. This method proved to be robust and reliable for speed estimation.

3) License Plate Recognition:
   We implemented Optical Character Recognition (OCR) to read characters on detected license plates. To enhance the accuracy of OCR, we employed various image enhancement techniques, such as grayscale conversion and noise reduction. Additionally, we addressed challenges like protrusions or distortions on the plates by performing iterations from different directions to ensure accurate character reading. This approach resulted in a license plate recognition accuracy of approximately 95%.

4) Response and Processing Time:
   Through optimized contour detection and masking of stationary areas, we reduced the processing time significantly. A 30fps-1080p-30s video was processed in 4 minutes, compared to 16 minutes with YoloV8 on the same system, showcasing the efficiency of our image processing techniques.

5) System Requirements:
   The project operated on relatively lower-end systems by avoiding AI/ML-supported methods and focusing solely on image processing techniques. This decision not only reduced system requirements but also aligned with the project's goal of streamlining processing for faster results.

6) Enhance ability and Adaptation:
   We demonstrated that the system's performance and accuracy could be further improved. By refining existing code components, such as masks, filters, functions, and loops, we could reduce response time and system requirements. Additionally, the system was designed to be adaptable to different camera systems and environments by adjusting specified parameters within the code.

7) Learnings and Future Extensions:
   Through this project, we learned the importance of balancing performance and accuracy in image processing tasks. The iterative refinement of algorithm parameters and image enhancement techniques proved crucial in achieving high accuracy in both speed estimation and license plate recognition.

For future extensions, we suggest the following ideas:

Integration of Advanced AI Techniques:

While our project focused on non-AI methods, incorporating advanced AI techniques, such as deep learning models, could further enhance object detection accuracy and processing speed.

1) Real-time Processing Capabilities:
   Developing real-time processing capabilities would be beneficial for applications like traffic monitoring and autonomous driving. Optimizing algorithms for faster processing and deploying them on more powerful hardware could achieve this.

2) Enhanced Adaptability:
   Expanding the system’s adaptability to different environmental conditions and varying camera angles can make it more robust. This can be achieved by training the system on a diverse dataset and incorporating more sophisticated image enhancement techniques.

3) Integration with IoT Devices:
   Connecting the system with IoT devices, such as smart traffic lights and road sensors, can provide additional data points for more accurate speed and vehicle type estimations.
4) Scalability:
   Ensuring that the system can scale to handle high-density traffic scenarios and multiple video feeds simultaneously will enhance its practical applicability.

In conclusion, our project demonstrates the effectiveness of image processing techniques in vehicle detection, speed calculation, and license plate recognition. By focusing on optimization and enhancement, we have laid a solid foundation for future advancements and applications in traffic management and autonomous systems.

5. References

- January 2008, S.Chhhaniyara, P.Bunnun, L.Seneviratne, K. Althoefer,Optical flow Algorithm for velocity estimation of ground Vehicles

- January 2011, I.Sreedevi, M.Gupta, Vehicle tracking and Speed Estimation using Optical Flow method

- Farneback, G.2003. Two-frame motion estimation based on polynomial expansion.

- https://www.researchgate.net/publication/228658426_Optical_Flow_Algorithm_for_Velocity_Estimation_of_Ground_Vehicles_A_Feasibility_Study

- https://www.researchgate.net/publication/50392061_Vehicle_Tracking_and_Speed_Estimation_using_Optical_Flow_Method

- https://github.com/thangphung215/Vehicle_detection_tracking_speed_estimation_front_view

- http://www.viulib.org/