**Comparitive Evaluation of Vehicle Direction and Motion Detection Methods for Multi-layer Contiguous Virtual Layer (MCVL)**

\*Manipriya Sankaranarayanan1,P. Rupesh1 and S.V.S Apparao1  
1Department of Computer Science and Engineering, Indian Institute of Information Technology Sri City, Chittoor, Andhra Pradesh, India  
[\*](mailto:*rupesh.p21@iiits.in)manipriya.s@iiits.in

\*Corresponding Author

**Abstract**

The efficiency of a traffic management application depends significantly on its ability to detect vehicles effectively from a traffic video. Hence vehicle detection techniques have to have an optimal balance on the computational time and space. The proposed Multi-layer Contiguous Virtual Layer (MCVL) framework aims to achieve this balance by integrating critical preprocessing steps, such as determining vehicle direction and motion within the traffic scenes or videos. While existing research predominantly relies on deep learning for motion boundary detection, these approaches often entail extensive training on large datasets, leading to high computational costs and increased complexity. Therefore, this paper gives a detailed analysis on the effectiveness of three distinct methods: Adaptive optical flow, Adaptive blob tracking, and Adaptive YOLO sort, modified and made adaptive for detecting vehicle movements in MCVL framework. Based on the result analysis from the benchmark dataset, it is evident that adaptive Blob Tracking in MCVL exhibits superior computational efficiency of processing time of 0.005 seconds per frame and consistent memory usage of 6.3 kb irrespective of the complexity of the scene. It also shows high accuracy for detecting motion direction and boundaries for MCVL with MSE of 1221.734 and RMSE 34.953. However, adaptive Optical Flow and YOLO Sort for MCVL do not consistently demonstrate superiority over the alternatives.  
**Keywords:** Vehicle detection**,** Multi-layer Contiguous Virtual Layer**,** Direction of motion, Adaptive Techniques.

1. **Introduction**

Detecting and tracking vehicle movements are essential across numerous applications including traffic management, autonomous driving, and urban planning. Precise vehicle tracking enables real-time monitoring and control of traffic, thereby bolstering road safety measures. Urban planners leverage detailed vehicle movement data to strategically design and enhance road networks, aiming to alleviate congestion and optimize urban mobility. Moreover, robust vehicle tracking supports the evolution of intelligent transportation systems, which integrate diverse technologies to innovate and enhance traffic management solutions. The critical role of these applications underscores the ongoing need for advanced methodologies in this field to ensure both efficacy and efficiency in addressing evolving transportation challenges.

Conventional methods for detecting vehicles often rely on deep learning techniques, renowned for their high precision. However, these approaches necessitate substantial computational resources and powerful hardware to process intricate algorithms. Moreover, they require extensive training datasets for effective learning, involving the collection and processing of vast amounts of data. Managing these large datasets and resultant models also demands significant storage capacity. These demands present formidable challenges, particularly in real-time applications where rapid processing is crucial and in environments with limited computational power and storage capabilities. Consequently, implementing deep learning-based vehicle movement detection in such contexts becomes challenging, prompting the exploration of more efficient alternatives. Given the complexities inherent in traffic scenarios, robust vehicle detection frameworks encounter significant hurdles. Addressing these limitations is pivotal for effective traffic management. Thus, this research work introduces an innovative approach, the Multi-layer Contiguous Virtual Layer (MCVL), to handle the diverse challenges observed in traffic video analysis. Expanding on prior research **[1], [2],** this paper enhances the MCVL framework to notably elevate detection accuracy and overall application performance. MCVL's adaptability and effectiveness are dependent by its ability to dynamically adjust based on the configuration and context of traffic video frameworks, highlighting its versatility.

This dynamicity of MCVL is achieved by obtaining the co-ordinates of the various motions predominantly seen in the traffic scene. Direction and motion detection of vehicles are pivotal aspects for the Multi-layer Contiguous Virtual Layer (MCVL) framework due to several critical reasons. Firstly, accurate determination of vehicle direction allows MCVL to predict and manage traffic flow effectively in real-time. By understanding which direction vehicles are moving, the system can optimize traffic signal timings, coordinate intersections, and implement lane management strategies that minimize congestion and enhance overall traffic efficiency. Secondly, detecting vehicle motion within traffic scenes enables MCVL to provide crucial insights into traffic dynamics and behavior. This includes identifying patterns such as lane changes, accelerations, and decelerations, which are essential for predicting potential traffic incidents and optimizing traffic management strategies accordingly. Moreover, direction and motion detection support the development of intelligent transportation systems (ITS) within the MCVL framework. Overall, the ability to accurately detect direction and motion of vehicles for MCVL not only enhances its effectiveness in real-time traffic management but also supports broader initiatives aimed a dynamic facility in detecting vehicles in any type of traffic video. To further enhance understanding and practical application, this work rigorously evaluates three distinct algorithms Optical Flow, Blob Tracking, and YOLO **[26]—**implemented using the Open CV library for detecting motion **[23].** These all are pre-existing algorithms and are chosen due to the advantages they provide in solving the problem statement.

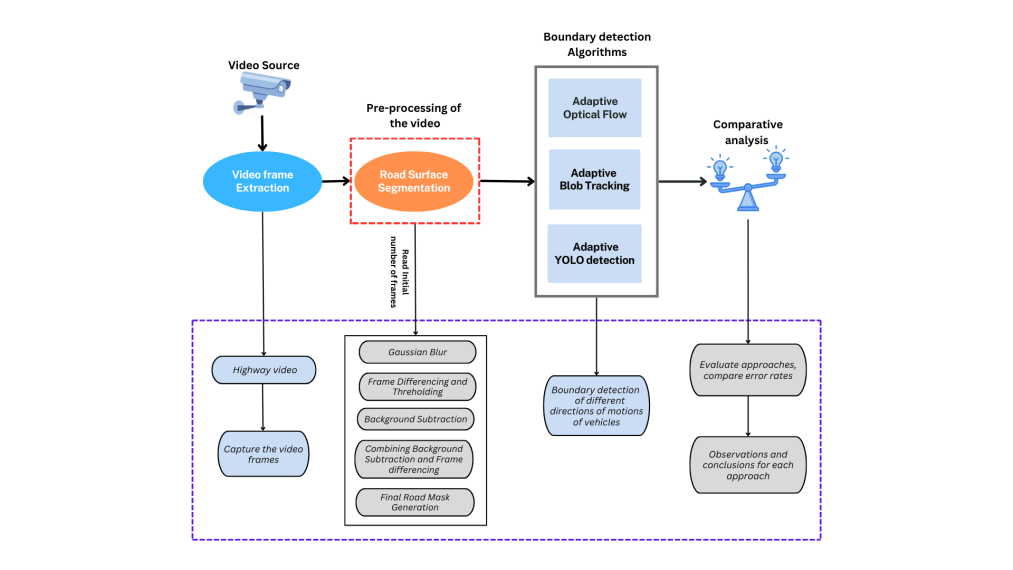
Optical flow **[7]** is widely used method for motion estimation. This method is conceptually straightforward and easy to implement, which makes it accessible for many applications. Also, it is computationally less expensive and more robust to local variations compared with other algorithms like Horn-Schunck Method **[18],** Farneback Method (Dense optical flow) **[19]**. Blob tracking **[14]** can be less sensitive to noise in the image compared to feature-based methods and require less computational power compared to more sophisticated tracking methods like deep learning-based trackers **[20]**. YOLO **[9]** is extremely fast because it predicts bounding boxes and class probabilities directly from full images in a single evaluation, unlike methods like R-CNN **[21][26]** which require multiple evaluations. SORT **[11]** is designed to be fast and efficient, making it suitable for real-time applications. Despite its simplicity, SORT provides a strong baseline performance for multi-object tracking tasks. All these algorithms don’t directly solve the problem statement. So, adaptive approaches are implemented in a heuristic way by using these existing methods to approach the solution of the problem.  
These algorithms specifically address challenges in vehicle movement and direction detection, offering insights into their comparative effectiveness within the MCVL framework. Unlike traditional deep learning approaches, these algorithms do not require extensive training phases, thereby significantly reducing computational complexity and memory usage. This innovative approach aims to provide efficient and practical solutions that can be readily applied across various domains within intelligent transportation systems.

This analysis is crucial for different research problems like estimation of macroscopic road traffic parameters, employing MCVL. By conducting a comparative analysis of the three methods, their performances under different scenarios, providing their respective strengths and weaknesses are seen. This comparison is essential for solving further problems in intelligent transportation systems (ITS) using MCVL, particularly for accurately estimating traffic parameters and vehicles motion boundaries. This work indicates that while deep learning techniques offer high accuracy, the proposed algorithms have the ability to provide a balanced trade-off between accuracy and computational efficiency.

The subsequent sections discuss the following: Section 2 discusses the detail of the task done for vehicle direction and motion detection. The subsections also include the details of Multi-layer Contiguous Virtual Layer (MCVL), pre-processing modules and the three different techniques used for MCVL. Section 3 presents the comparative analysis on benchmark datasets followed by conclusion.

1. **Vehicle Direction and Motion Detection Methods for MCVL**

This section describes the methods and heuristic techniques used for implementing and analyzing approaches to vehicle movement and direction detection for Multi-layer Continguous Virtual Layer (MCVL). The detail of the entire work flow for the analysis is presented in Fig.1. As in Fig.1, individual frames are extracted from the traffic video frame. Next, the road surface area is extracted using road surface segmentation by evaluating the initial number of frames. A binary mask of the road surface area is generated, which helps remove the static background and focus on the foreground of the road region, reducing computational complexity. Three different approaches are implemented using the binary road mask to detect the road boundaries for different directions of motions of vehicles. Finally, the predictions are evaluated by manually calculating the actual boundaries and generating error rates to compare the approaches, which is discussed the Results section.

 **Fig. 1.** Overall Workflow for Comparative Evaluation of Vehicle Direction and Motion Detection

The study focuses on developing a system for detecting vehicle direction and motion, designed for use in the Multi-layer Contiguous Virtual Layer (MCVL). Road traffic videos undergoes processing using an innovative framework known as Multiple Contiguous Virtual Layer (MCVL). This framework integrates advanced image processing methods and techniques to achieve a balance between accuracy and computational load, ensuring reliability.

MCVL comprises multiple layers of contiguous segments organized in a grid-like structure. Each segment is defined by a bounding box, forming a continuous loop of interconnected boxes. Pixel-level processing is carried out across all individual grids within the MCVL framework in paralles. Given the diverse coverage of traffic surveillance cameras, videos typically encompass multiple lanes and diverse traffic flow directions. By employing MCVL, the complexity of the traffic scene is reduced, thereby optimizing processing efficiency and potentially minimizing false positives **[22].** Figure 2 illustrates a detailed diagram of the MCVL design and its operational framework. Further elaboration on the operational details of MCVL are constrained due to the length limitations of the paper.

The first step in implementing MCVL involves defining the boundaries of the region of interest, crucial for solving the detection challenge. This paper's comparative analysis aims to dynamically pinpoint the optimal approach for establishing these boundaries, thereby advancing the problem-solving methodology. Detailed insights into this process will be elaborated upon in subsequent sections.Top of FormBottom of Form

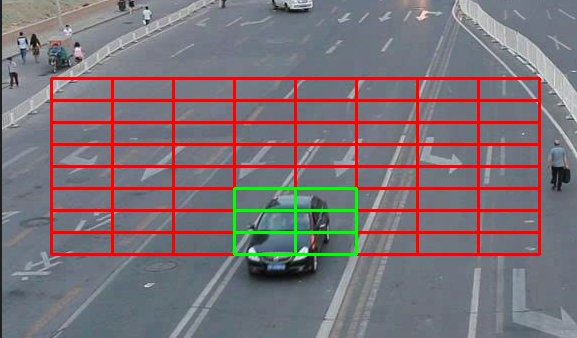


Fig. 2 Multi-Layer Contiguous Virtual Layer

* 1. **Video Frames Extraction**

To implement MCVL, the first step is to extract the frames from the given highway or traffic video. The processing of the video files are done using OpenCV library for computer vision based processing. The video is first captured using the function VideoCapture(),which is an inbuilt function is OpenCV library. Later the frames are extracted using the read() function and used for further processing

* 1. **Road Surface Segmentation - Pre-processing Modules**

To optimize the vehicle detection process and eliminate unnecessary computations, it is essential to focus exclusively on the road surface within the given video. By excluding background elements such as buildings, grass, and moving pedestrians, the efficiency and accuracy of the proposed algorithms are significantly enhanced. Fig. 3 illustrates the detailed flow of road surface segmentation. This section outlines the steps to obtain a precise mask of the road boundary, which will serve as the foundation for subsequent appraoches

* + 1. **Gaussian Blur**

Especially in the context of Intelligent transportation systems (ITS), Gaussian blur **[6]** is a commonly utilised technique in video processing. The reduction of noise in video frames is one of the main applications of Gaussian blur. Real-world traffic camera video feeds frequently include a variety of noises because of the noise due to environmental conditions, etc. By averaging out the pixel values, Gaussian blur helps to reduce these problems by smoothing the image and reducing noise. This is especially helpful for pre-processing stages, since a clearer image can greatly enhance the efficiency of the algorithm that follows.

* + 1. **Frame Differencing and Thresholding**

Among several existing methods, the detecting vehicle movement within video sequences is done using frame differencing **[5]** which is in conjunction with thresholding. This method effectively highlights changes between consecutive frames, allowing us to identify moving objects against a static background. The aim of this process is the calculation of the absolute difference between the blurred previous frame and the blurred current frame. This operation results in a new image, where pixel values represent the magnitude of change between the two frames. Regions with substantial changes appear brighter, while static areas remain dark. Thresholding is applied to convert the difference image into a binary image, facilitating easier detection of moving objects.A binary threshold is applied to the grayscale difference image. Pixels with intensity values above a specified threshold are set to white (255), indicating significant changes, while all other pixels are set to black (0). This method is computationally efficient and effective in real-time applications, as it reduces the complexity of detecting moving vehicles by focusing solely on significant changes in the scene. The resulting binary images serve as a foundation for further processing steps.

* + 1. **Adaptive Background Subraction**

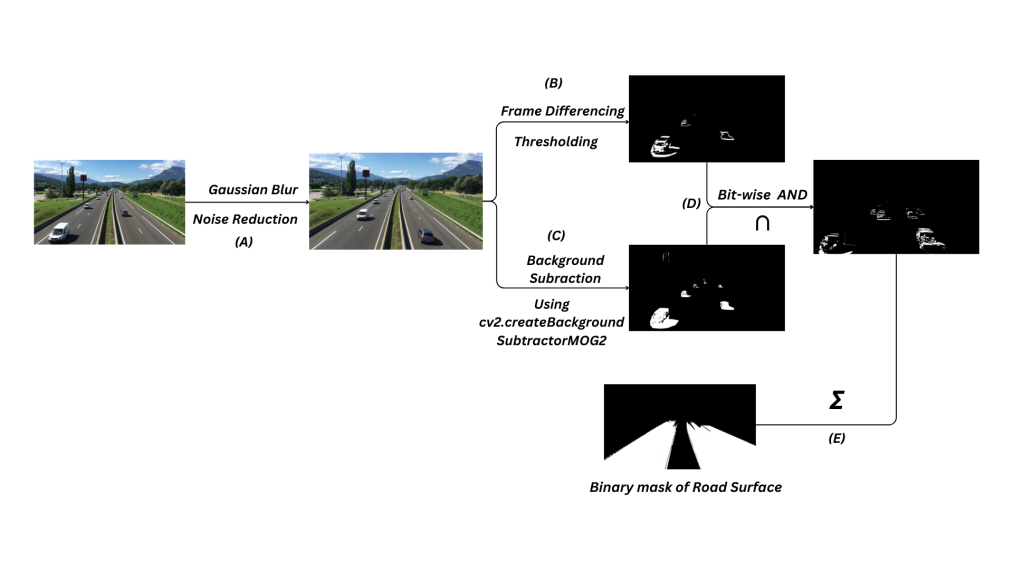
The detection of vehicles are also done using the Background Subtractor MOG2 **[3][4]** (Mixture of Gaussians) method provided by the OpenCV library to efficiently separate moving objects from the static background in video sequences.The Background Subtractor MOG2 algorithm is an advanced background subtraction method that models each pixel as a mixture of Gaussians. The adaptive background model is built and updated iteratively by feeding consecutive frames from the video to the background subtractor. As each frame is processed, the algorithm updates its understanding of the background and identifies foreground objects. The apply() method updates the background model and produces a binary mask where moving objects are highlighted.

* + 1. **Combining Frame Differencing and MOG2 Background Subtraction**

After completion of the frame differencing and the Background Subtraction the resultant binary images/masks are combined to generate binary masks highlighting moving vehicles. The innovative approach of this work is to enhance the accuracy and robustness of our vehicle detection by performing a bitwise-AND operation on these masks, resulting in a final binary mask that combines the strengths of both methods.

* + 1. **Final Road Surface Mask Generation**

Following the generation of individual binary masks for each frame through the combined approach of frame differencing and MOG2 background subtraction, a comprehensive representation of the road surface is created **[1] [2].** This is achieved by aggregating these masks using a bitwise OR operation, resulting in a final binary mask that delineates the entirety of the road surface while filtering out non-road elements.



**Fig. 3.** Workflow of Road Surface Extraction

* 1. **Boundary Detection Algorithms for Vehicle Directions**

Three different adaptive algorithms including optical flow, blob tracking and YOLO sort are discussed in this section for identifying the boundaries of different directions of motion of traffic video for MCVL.

* + 1. **Adaptive Optical Flow Alogrithm for MCVL**

Optical flow [8] is a powerful method for estimating motion by analyzing the apparent movement of objects between consecutive frames. This section outlines modified algorithm of classic Lucas Kanade optical flow for tracking vehicles using optical flow, highlighting key steps such as key point estimation, consistency checks, trajectory formation, and direction vector analysis.

After the initial Road Surface Segmentation **[1][2]** process the corner points are detected in the initial frame, serving as keypoints. These keypoints are selected based on their quality as features for tracking in subsequent frames. The location keypoints in the current frame are estimatied using the previous frame with the Lucas-Kanade optical flow *cv2.calcOpticalFlowPyrLK()* method. The gray-scale versions of the previous and current frames are used respectively, along with the key points from the previous frame, and the lk\_params object are parameters for the Lucas-Kanade method. The consistency of the estimated keypoints are verified. Only points with consistent estimates, determined by their small error values and stable status, are added to the trajectory. The trajectories of the direction of motion are formed by adding points that are at a consistent distance from the previous feature points in the trajectory. Furthermore, these trajectories are utilized to find the boundaries of different directions of motions of vehicles by generating direction vectors. These direction vectors utilize the last two points of each trajectory that are estimated **[7]**. The direction vector indicates the movement direction of the tracked vehicle.   
  
Using the direction vectors, the direction of vehicle are obtained and based on that, the boundar boundaries in that particular direction are enumerated. If the vehicles are moving towards or away from the camera, the coordinates of the boundaries are found using the x-coordinates of the feature points of vehicles. Similarly, if the vehicles are moving in horizantal direction then, boundaries are detected using y-coordinates. This additional feature is appended to the exisiting optical flow algorithm to utilize the benefits it produces.The pseudocode of the adaptive optical flow algorithm is presentedin Algorithm 1.

**Algorithm 1: Adaptive Optical Flow**

**Input** : Video frames  
 **Output** : Boundaries of different directions of motions  
 **Begin**  
 **Initialisation()**  
 **RSS(**video**)**  
 **while** *True* **do**:  
 FC()  
 Frame\_G  
**if** *not* Tj **then** *(compute optical flow)* Implement\_OF()  
 *Update* Tj:  
 Utj()**for**T in Tj **do** *(calculation of direction vector)  
 DV(T)* **end for** *(Finding the limits based on coordinates)* **for each**T*: (For sample, considering one direction)* **if** x<LL**then** x=LL**if** x>LL**then**x=RL  
 **end for**

*(Iterate all tracjectories to get the final limits of the frame   
 for a particular direction)*

TF() *(***At** *specified intervals, detect new features to track)* VD()PG= Frame\_G  
 **end while** **End** where,  
RSS() = Takes video input and performs Road Surface SegmentationIntialisation() = Function which initialises optical flow parameters, feature   
 detection parameters and varaibles for storage and processing  
 FC() = Captures the individual frames from given video  
 T = Single Trajectory  
 Tj = Trajectories (An Array of T’s)  
 Frame\_G = Gray scale frame is obtained using Opencv library  
 Implement\_OF() *=* Implements the *calcOpticalFlowPyrLK()*in Opencv Library  
 Utj() = Updates the Tj, by appending only trajectories if displacement is significant   
 and remove intial values of the T, if the length exceeds beyond certain value.  
 DV() = Takes the T as input and calculates the direction vector using last 2 points  
 which helps in finding the limits of road for particular direction of motion  
 LL,RL = reprents the left limit and right limit for a paritcular direction  
 TF()= Tracks new features using the function cv2.goodFeaturesToTrack() inopencv  
 library  
 VD = Visualise the detections using inshow() function in Opencv library  
 PG = Previous gray scale frame

* + 1. **Adaptive Blob Tracking and SORT for MCVL**  
       In this algorithm, a combination of Blob tracking**[14] [24]** and the SORT**[11]** (Simple Online and Realtime Tracking) algorithm to detect the boundaries of different direction of motions of vehicles are employed. The algorithm provides a comprehensive framework for vehicle movement analysis.

**Algorithm 2 : Adaptive Blob tracking**

**Input** : Video frames  
 **Output** : Boundaries of different directions of motions  
 **Begin**  
 **Initialize()  
 RSS(video)**

**while** *True* **do**:  
FC()FG()

**for**C in Cs **then**

*x, y, w, h = cv2.boundingRect(contour)*

***Append*** *[x, y, x+w, y+h, 1] to detections*

**if**not Dts **then**

*tracked\_objects = tracker.update(detections)  
 current\_ids = set()* **for** *obj in tracked\_objects:*

*obj\_id = int(obj[4])*

*bbox = obj[:4]*

*current\_ids.add(obj\_id)*

**if** *obj\_id in vehicle\_dict:*

*Track the trajectory points of the vehicle*

*Update* ***TO*** *with the new coordinates and ID*

**else:**

*TO[obj\_id] = {'id': obj\_id, 'bbox': bbox, 'direction': None*}  
 *obsolete\_ids = set(vehicle\_dict.keys()) - current\_ids*

**for** *obj\_id in obsolete\_ids:*

***Remove*** *obj\_id from vehicle\_dict*

***Record*** *frame boundaries*

*min\_left, max\_left, min\_right, max\_right = get\_min\_max\_coordinates()*

***Append*** *[frame\_count, min\_left, max\_left, min\_right, max\_right]   
 to frame\_boundaries*

**VD()**

*Visualize the detections using imshow() function in OpenCV library.*

**DT()**

*Draw the trajectories using OpenCV Library.*

**End while  
 End**

where, Initialize () = Imports the necessary libraries and initialize the required variables  
 RSS() = Takes the video input and performs the road surface segmentation  
 FC() = Captures the frames from given input video  
   
  
 FG() = This function performs the operations on frame like dilating, eroding using   
 dilate(),erode() from Opencv library  
 Cs = An array of Contours  
 C = A single contour in Cs  
 Dt’s = Detections Array which contains the detections made

**TO** = An Object which stores the IDs and corresponding coordinates.

**VD()** = Visualize the detections using imshow() function in OpenCV library   
 DT() = Draw the trajectories using OpenCV Library .

The algorithm uses MOG2 (Mixture of Gaussians) for dynamic background subtraction **[1],[2]**, which adapts to changes in the background, making it effective for environments with varying lighting and moving shadows. Apply blob tracking techniques to detect vehicles within the segmented road surface. Blob tracking **[14]** identifies connected pixel regions in binary images that correspond to moving vehicles. By using the OpenCV’s SimpleBlobDetector, blobs in each frame are detected. The SORT **[11]** tracker to manage multiple objects across frames are initialized. SORT(Simple Online and Realtime Tracking) **[11]** uses a combination of Kalman filtering **[13]** and the Hungarian algorithm for data association to track multiple objects in real-time. For each frame, detect vehicles using blob tracking and update the SORT tracker with the detected vehicle positions. The tracker assigns a unique ID to each vehicle and updates their positions across frames. Based on the current and previous positions of each tracked vehicle, determine the direction of movement. The algorithm uses the SORT **[11]** algorithm to maintain and update the identities of vehicles across frames, ensuring consistent tracking even with occlusions and temporary losses.  
  
The direction is calculated by comparing the vehicle’s coordinates in consecutive frames. Assign different colors to vehicles based on their direction of movement. This visual distinction helps in analyzing traffic patterns and directions.Using the coordinates of each vehicle obtained from the tracker, draw left and right trajectories on the video frames. This visualization aids in understanding vehicle paths and traffic flow. The pseudocode of the algorithm is shown in Algorithm 2.

* + 1. **Adaptive YOLO and SORT Algorithm for MCVL**

In this algorithm, YOLO **[9] [27]** (You Only Look Once) object detection algorithm combined with the SORT **[11]** (Simple Online and Real time Tracking) algorithm is used to detect and track vehicles. This method allows us to analyze vehicle movements and determine their direction, which subsequently helps in establishing boundaries in different directions

**Algorithm 3: Adaptive YOLO and SORT**

**Input** : Video frames  
 **Output** : Boundaries of different directions of motions  
 **Begin**  
 TO()RSS(video) *def* ***func():* while** fc**<** k **do**  
 *FC()*  
 fc+=1 results = YM(F*)  
 Using results get the detections array  
 (Use* SORT *tracker to get the ID’s using the detections array)* **for**R in RT **do** *Get the Tracking ID and coordinates of each vehicle* **if** *ID in TO* **then** *PP= TO[ID]  
 Get* CP *from* R**if** *CP[1] >= PP[1]* **then** *(Top to bottom movement of vehicles)  
 Store the coordinates of that vechile in a variable and   
 compare with next vehicle coordinates* **else** *(Bottom to top movement of vehicles)  
 Store the coordinates of that vechile in a variable and   
 compare with next vehicle coordinates* **end for****end while****Return** *ID’s*   
 import  ***func ()*** initialize()***-------(1)***  
 **while** *True* **do**:  
 FC()  
 results = YM(F*)  
 Get the boxes from the obtaines results* **for** *B in BX* **do** *Get the detections using coordinates obtained from detstructuring box  
 Using them get the detections* **end for** *(Use SORT Tracker to get the ID’s using detections array)* **for**R *in* RT**do** *Get the Tracking ID and coordinates of each vehicle  
 If those ID’s fectched in* ***(1)*** *are present in these ID’s:  
 Track the trajectory points of vechicle  
 It becomes the boundary in that direction* **end for**VD() DT() **end while  
 End** where, fc = frame count  
 F = Single Frame  
 k = fixed number of frames (e.g. 300)  
 FC() = Captures the video frames using read() in Opencv Library  
 YM() = Takes the gray frame as input and implements YOLO model and   
 returns the results  
 RT = ResultsTracker obtained from inplemention of SORT TrackerR = Single Result from RTTO() = Initialize empty tracking ObjectTO = An Object which stores the ID’s and corresponding coordinates PP,CP = Previous point and Current point variable representationsID = ID generated by SORT tracker ID’s =ID’s of vehicles which act as boundaries of road in   
 that particular direction of motion of vehicles initialize() **=** Loads the YOLO model and SORT tracker and get the vehicle  
 ID’s from func()   
 RSS() = Takes the video input and performs the road surface segmentation  
 FC() = Reads individual frames from given input video.  
 BX = Represents the boxes obtained from YOLO model  
 B=Represents the object which contained individual detections (box)   
 information   
 VD() = Visualise the detections using inshow() function in Opencv library  
 DT() = Draws the trajectories by using Opencv Library  
  
  
YOLO is applied to detect vehicles in each frame. YOLO provides bounding boxes around detected vehicles along with confidence scores. A detections array that includes the coordinates of the bounding boxes and the confidence scores are obtained. The SORT tracker is initialized to manage and update the positions of multiple vehicles detected by YOLO **[10].**  
  
SORT tracker with the YOLO detections for each frame is updated. For each tracked vehicle, determine its direction based on its current and previous positions. This involves comparing the coordinates of the vehicle in consecutive frames. If current position y-coordinate is greater than the previous position y-coordinate then it is moving from top to bottom of the frame, else it is moving in opposite to mentioned direction. Similarly x-direction is also checked. The detail of the algorithm is presented in Algorithm 1.

1. **Results and Discussion**

A demanding, real-world benchmark for multi-object tracking and detection is called UA-DETRAC. Images of typical and interstate road scenes are included in this dataset. The screenshot of few videos are shown in Fig.4.

**Fig. 4.** Screenshots from the **DETRAC** dataset

In this research work, the implemented methodologies focus on predicting motion directions and detecting boundaries. To evaluate their effectiveness, we employ standard metrics including computational time, space utilization, and error rates. The computational complexities are analyzed using an 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz processor with a base clock speed of 2.42 GHz and 8.00 GB of RAM (7.74 GB usable). Error rates for boundary coordinate detection in traffic frames are assessed using Mean Square Error (MSE) and Root Mean Square Error (RMSE).  
  
 **Mean Square Error (MSE)** is a commonly used metric for evaluating the accuracy of predictions as in Equation 1. It measures the average squared difference between the predicted values and the actual values

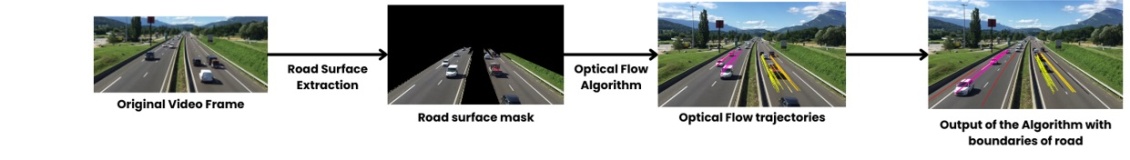
*- (1)*  
  
**Root Mean Square Error (RMSE)** is the square root of the MSE and provides a measure of the prediction error in the same units as the original values as in Equation 2. RMSE **[25]** particularly useful for understanding the magnitude of prediction errors

*- (2)*

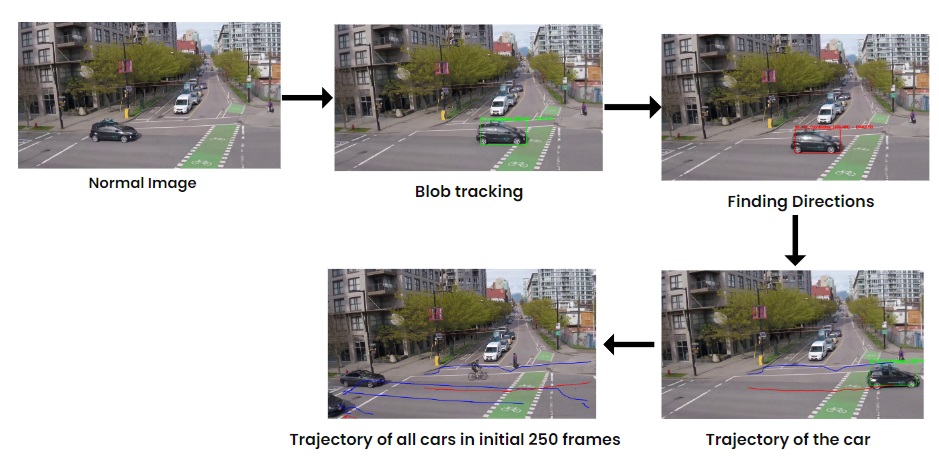


**Fig. 5.** Screenshots of GUI of Comparitive Analysis

The screenshot of the GUI of the analysis is shown in Fig 5. The workflow of the boundary detection using adaptive Optical flow, adaptive blob tracking and adaptive YOLO Sort is shown in **Fig.6, Fig.7, and Fig.8** respectively**.** It involves a series of steps which involves different actions performed from video frame extraction to boundary detection. On the extracted video frames the road surface segmentation algorithm is applied, which produces a binary mask of the road. After the road mask is generated the adaptive algorithms are implemented to get the trajectory of the vehicles which in turn used to fetch the boundaries of different directions of motions of vehicles.



**Fig.6.** Boundary detection using Adaptive Optical Flow Algorithm



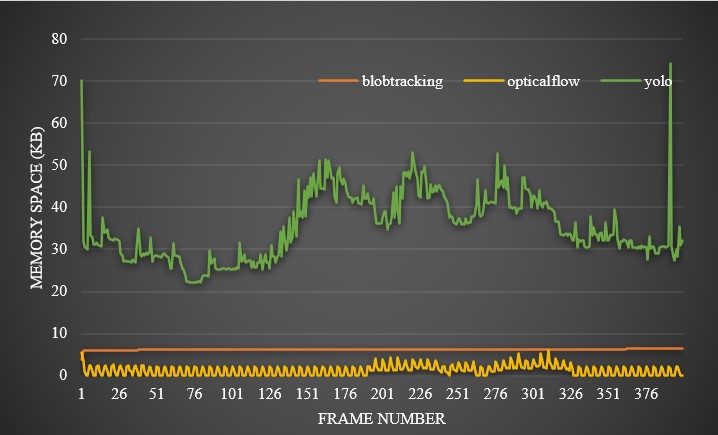
**Fig.7.** Boundary detection using Adaptive BLOB Tracking and SORT Algorithm

 **Fig.8.** Boundary detection using Adaptive YOLO and SORT Algorithm

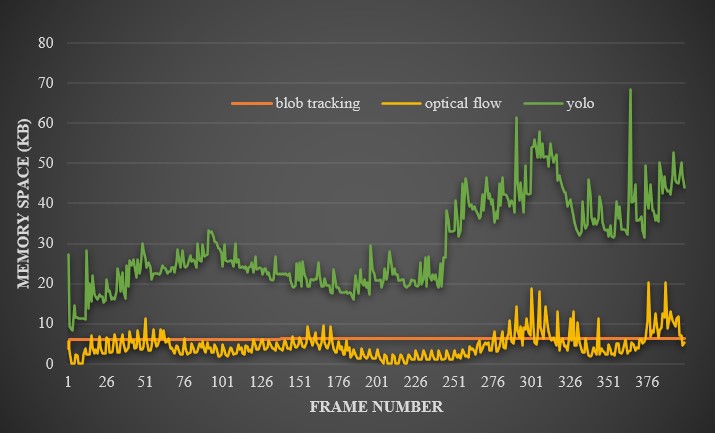
The results are analyzed based on the contents of the traffic scene in the datasets. The videos are categorized into well illuminated and moderate traffic and night time traffic videos. The results of computational time and space are shown in Fig. 9 and Fig. 10.

* 1. **Comparative Analysis by Memory Usage**

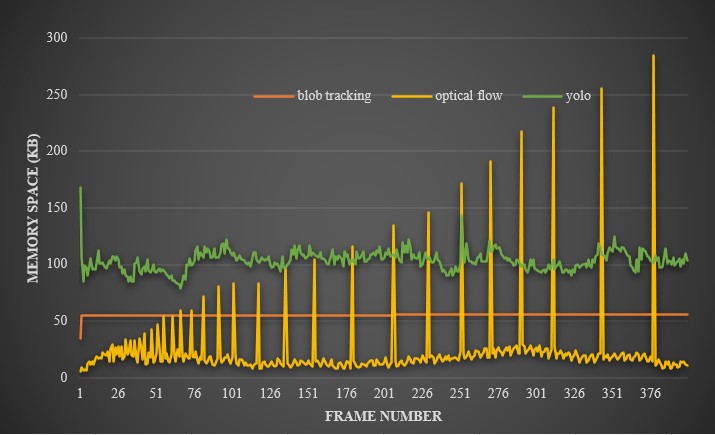
In comparing the space complexities of adaptive blob tracking, optical flow, and YOLO algorithms in well-illuminated traffic videos, distinct patterns emerge in memory usage. Blob tracking demonstrates the most efficient memory usage among the three, peaking at 6.3 KB. This efficiency stems from its focus on individual object segmentation and frame-based tracking, requiring less memory compared to broader analysis methods. Optical flow follows closely with a maximum of 5.35 KB, emphasizing temporal coherence in pixel movements across frames without excessive memory overhead. Conversely, YOLO, known for real-time object detection, exhibits significantly higher memory consumption, peaking at 52.96 KB. This increase is due to YOLO's deep neural network architecture, which processes entire images for accurate multi-object detection simultaneously. Therefore, while blob tracking and optical flow excel in memory efficiency suitable for well-illuminated traffic videos, YOLO's superior object detection performance comes with higher memory usage, illustrating trade-offs between functionality and resource consumption in computer vision applications.



**Fig.9.** Comparative Analysis of Adaptive Algorithms with Memory Space allocation for well   
illuminated Traffic Videos



**Fig.10.** Comparative Analysis of Adaptive Algorithms with Memory Space allocation for moderately   
illuminated Traffic Videos



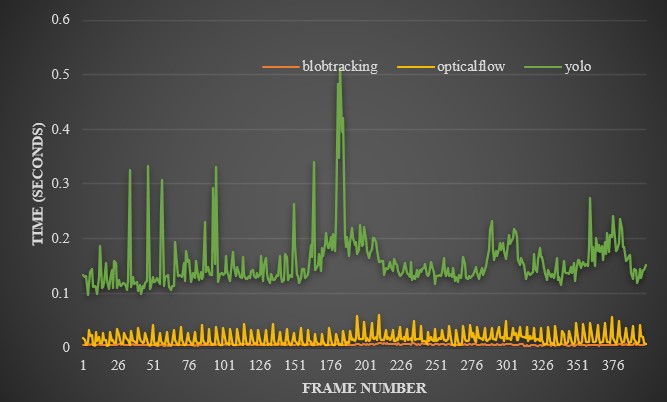
**Fig.11.** Comparative Analysis of Adaptive Algorithms with Memory Space allocation for Night Traffic Videos

The space complexity for the moderate traffic video in Fig. 10 shows that adaptive blob tracking shows the lowest space complexity, with measurements ranging mostly between 3.87 and 6.25 KB. Optical flow demonstrates moderate space usage, generally between 0 and 11.63 units, occasionally peaking up to 40.69 KB. In contrast, YOLO consistently exhibits the highest space complexity, ranging widely from 8.33 to 61.42 KB, indicating its significantly larger memory requirements compared to blob tracking and optical flow algorithms.

The night time video as in Fig. 11 with poor illumination shows that blob tracking generally requires the least memory, with measurements ranging from approximately 34.84 KB to 55.61 KB. Optical flow's memory usage varies more widely, from around 5.55 KB to 256.04 KB. YOLO consistently demands the most memory, with values ranging from 78.80 KB to 145.84 KB. These variations highlight the differing memory demands of each algorithm, potentially influenced by factors such as image resolution, processing complexity, and specific implementation details during night time.

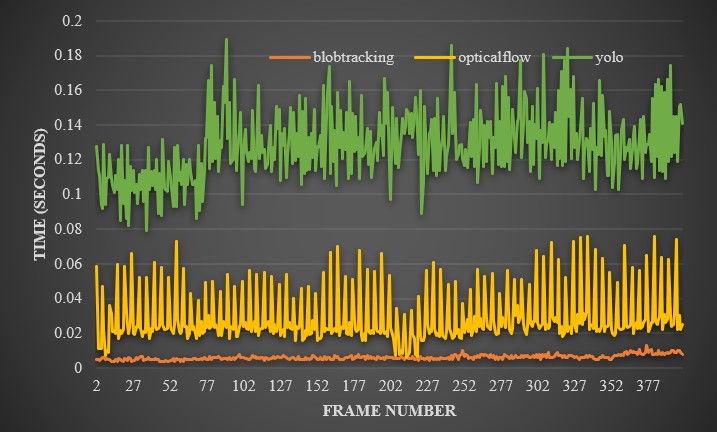
* 1. **Comparative Analysis by Computational Time**

The computational time is for the three adaptive algorithms are calculated by the processing time in seconds for each frame of the traffic video. While comparing the time complexities of the proposed method in well illuminated traffic video as in Fig. 12 —Adaptive Blob Tracking, Adaptive Optical Flow, and Adaptive YOLO—measured in seconds per frame across a dataset, several insights emerge. Adaptive Blob Tracking consistently exhibits the lowest time complexity, with an average runtime of approximately 0.005 seconds per frame. This efficiency suggests its suitability for real-time applications where computational resources are constrained. Optical Flow, with an average runtime of 0.016 seconds per frame, demonstrates moderate performance, making it viable for scenarios where accuracy in motion detection is crucial but with a higher tolerance for processing time. Conversely, YOLO, despite its significantly higher average runtime of 0.145 seconds per frame, offers unparalleled accuracy and robustness in object detection tasks. This high computational demand aligns with its capability to handle complex scenes and a wide variety of objects, albeit at the cost of slower processing speeds compared to the other two algorithms.



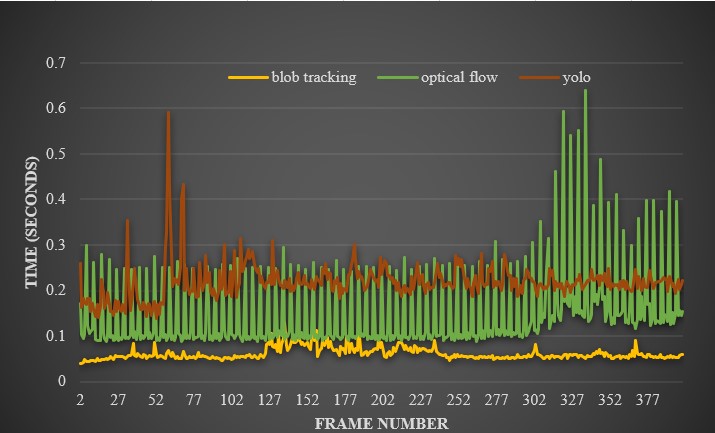
**Fig.12.** Comparative Analysis of Adaptive Algorithms by processing time for well illuminated Traffic Videos

For a moderate illuminated traffic video as in Fig.13 adaptive Blob tracking consistently shows the lowest execution times across the dataset, with an average of approximately 0.005 seconds per frame. This suggests it is the most computationally efficient among the three methods. Adaptive Optical flow, with an average of about 0.025 seconds per frame, demonstrates higher computational demands compared to blob tracking but remains relatively efficient. In contrast, Adaptive YOLO exhibits significantly higher execution times, averaging around 0.125 seconds per frame. This indicates YOLO is the most computationally intensive of the three, likely due to its comprehensive object detection capabilities that involve complex deep learning models.



**Fig.13.** Comparative Analysis of Adaptive Algorithms by processing time for moderately illuminated Traffic Videos

In night video as in Fig.14, blob tracking shows the lowest average time complexity of approximately 0.055 seconds, optical flow estimation averages around 0.102 seconds, and YOLO object detection being the most computationally intensive with an average of 0.221 seconds. These measurements highlight YOLO's higher computational demands compared to blob tracking and optical flow, which are more lightweight tasks in terms of processing time. Understanding these time complexities is crucial for optimizing performance of MCVL.



**Fig.11.** Comparative Analysis of Adaptive Algorithms by processing time for Night Traffic Videos

* 1. **Comparative Analysis by Error Rate**

For calculating the error rates the actual boundaries for the videos in the DETRAC dataset are calculated manually. Later, the error is computed by comparing these manually established boundaries with the predicted boundaries for vehicles motion directions. Table 1 represents the comparison of the results on different videos using the three approaches mentioning their MSE and RMSE values

**Table 1.** Comparison of Adaptive Approaches on different videos

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Videos** | **Adaptive  Optical Flow** | | **Adaptive   Blob tracking** | | **Adaptive   YOLO** | |
|  | **MSE** | **RMSE** | **MSE** | **RMSE** | **MSE** | **RMSE** |
| Well Illuminated Highway Video | 1384.38 | 37.21 | 648.25 | 25.46 | 594.75 | 24.39 |
| City Traffic video | 2467.25 | 49.671 | 742.5 | 27.25 | 3407.75 | 58.36 |
| Moderatly illuminated Highway video | 1133.75 | 33.67 | 2260.05 | 47.54 | 18889.25 | 43.46 |

It is observed that the blob tracking approach performs well among the three methods evaluated. However, there are instances where the optical flow method outperforms the others. Each approach has its unique advantages and disadvantages. Additionally, variations in the error values were noted when different videos were analyzed.   
   
One reason the blob tracking approach records minimal error is that the bounding boxes, or blobs, adjust as the vehicle moves farther from the camera. This adjustment allows the predicted boundary to closely match the actual value. In the optical flow approach, the boundary is detected using trajectory points derived from feature detection. As the car moves away from the camera, it becomes smaller, and fewer features are detected. In turn, the road boundary is not accurately detected, affecting the error.   
 **Table 2.** Average MSE of adaptive Optical flow, Block tracking and YOLO Approaches

|  |  |  |
| --- | --- | --- |
| **Adaptive Optical Flow** | **Adaptive Blob tracking** | **Adaptive   YOLO** |
| 1673.7234 | 1221.734 | 1969.167 |

In the YOLO approach, the size of the bounding box remains almost fixed unless the vehicle size changes significantly. This rigidity causes the bounding box to sometimes exceed the vehicle's actual size, impacting the boundary detection and increasing the error.The average RMSE and MSE values are compiled into Table 2 and Table 3, respectively.

**Table 3.** Average RMSE of adaptive Optical flow, Block tracking and YOLO Approaches

|  |  |  |
| --- | --- | --- |
| **Optical Flow** | **Blob tracking** | **YOLO** |
| 40.91 | 34.953 | 44.38 |

All these approaches involve analyzing some initial number of frames and extracting the required information from those frames to solve the problem statement. So some intial time is consumed to fulfil this step for all three approaches.  
  
In summary, the three adaptive algorithms—Adaptive Optical Flow, Adaptive Blob Tracking, and Adaptive YOLO—were evaluated on the UA-DETRAC dataset for multi-object tracking and detection in various traffic conditions. Adaptive Blob Tracking emerged as the most computationally efficient, averaging 0.005 seconds per frame, ideal for real-time applications. Adaptive Optical Flow followed with moderate performance at 0.016 seconds per frame, balancing accuracy and speed. Adaptive YOLO, known for robust object detection, required the most computational resources at 0.145 seconds per frame but excelled in complex scene analysis. Blob Tracking also demonstrated superior memory efficiency (peak of 6.3 KB), whereas YOLO required significantly more memory (peak of 52.96 KB). Error rates favored Blob Tracking with lower MSE (1221.734) and RMSE (34.953) compared to Optical Flow and YOLO, highlighting its effectiveness in boundary detection. These results underscored trade-offs between computational efficiency, memory utilization, and accuracy across different adaptive algorithms in challenging traffic scenarios.

1. **Conclusion**

A traffic management system's effectiveness depends on how well it detects vehicles in video footage while balancing computational efficiency and accuracy. This research emphasizes the critical importance of efficient vehicle detection techniques in enhancing the efficacy of traffic management applications. The Multi-layer Contiguous Virtual Layer (MCVL) framework, integrating Adaptive Optical Flow, Adaptive Blob Tracking, and Adaptive YOLO Sort methods, aims to optimize computational resources while maintaining accurate vehicle motion detection. From the analysis, Adaptive Blob Tracking emerges as a complimenting technique for MCVL, achieving a processing time of 0.005 seconds per frame and consistent memory usage of 6.3 KB. It demonstrates superior accuracy in detecting motion direction and boundaries, as evidenced by MSE of 1221.734 and RMSE of 34.953. However, Adaptive Optical Flow and YOLO Sort, while effective, do not consistently surpass alternative methods in computational efficiency and accuracy.

1. **Future work**

In future research, extending the capabilities of the Multi-layer Contiguous Virtual Layer (MCVL) framework could involve investigating the framework's dynamicity, scalability and automation across diverse traffic videos with varying environments and traffic conditions. Furthermore, exploring real-time adaptation mechanisms within MCVL to dynamically adjust to changing traffic dynamics and environmental conditions could further optimize its performance in practical traffic management scenarios.

##### **Acknowledgment**

This publication is an outcome of the R&D work undertaken in the project under TiHAN Faculty Fellowship of NMICPS Technology innovation Hub on Autonomous Navigation Foundation being implemented by Department of Science & Technology National Mission on Interdisciplinary Cyber-Physical Systems (DST NMICPS) at IIT Hyderabad.

**References**

1. Hadi Ghahremannezhad, Hang Shi, Chengjun Liu, “Automatic Road Detection in Traffic Videos”, *IEEE 4th International Conference on Image Processing, Applications and Systems (IPAS) pp., 2021.*
2. Huansheng Song, Haoxiang Liang\* , Huaiyu Li, Zhe Dai and Xu Yun (2019) “Vision-based vehicle detection and counting system using deep learning in highway scenes”. *European Transport Research Review*
3. Salman Qasim, Kaleem Nawaz Khan, Miao Yu; Muhammad Salman Khan "Performance Evaluation of Background Subtraction Techniques for Video Frames". *2021 International Conference on Artificial Intelligence (ICAI)*
4. Radhakrishnan, M (2013).Video object extraction by using background subtraction techniques for sports applications. *Digital Image Processing, 5(9),91–97.*
5. “Moving Object Detection and Segmentation using Frame Differencing and Summing Technique” Gopal Thapa, Kalpana Sharma, M.K.Ghose, *September 2014 International Journal of Computer Applications 102(7):20-25*
6. Estevao Gedraite, M.Hadad January 2011 “Investigation on the effect of a Gaussian Blur in image filtering and segmentation”. *ELMAR, 2011 Proceedings*
7. H.Yedjour, January 2021 Optical Flow Based on Lucas-Kanade Method for Motion Estimation. In : *Artificial Intelligence and Renewables Towards an Energy Transition (pp.937-945)*
8. Dhara Patel, Saurabh Upadhyay, Volume 61– No.10, January 2013 Optical Flow Measurement using Lucas kanade Method. *International Journal of Computer Applications (0975 – 8887)*
9. Real-Time Vehicle Detection Based on Improved YOLO v5 Yu Zhang, Zhongyin Guo, Jianqing Wu. *September 2022 Sustainability 14(19):12274*
10. “Vehicle Detection and Tracking using YOLO and Deep SORT” Muhammad Azhad Bin Zuraimi; Fadhlan Hafizhelmi Kamaru Zaman *2021 IEEE 11th IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*
11. Mahmoud Abouelyazid (2023) “Comparative Evaluation of SORT, DeepSORT, and ByteTrack for Multiple Object Tracking in Highway Videos”. *Vol. 8 No. 11 (2023): IJSICS-November*
12. Karl Leyven Leonida, Karla Veronica Sevilla, Cyrel O. Manlises "A Motion-Based Tracking System Using the Lucas-Kanade Optical Flow Method". *2022 14th International Conference on Computer and Automation Engineering (ICCAE)*
13. Y. Ng, Y. Latif, T.-J. Chin, and R. Mahony, “Asynchronous kalman filter for event-based star tracking,” in Computer Vision–ECCV 2022 Workshops: Tel Aviv, Israel*, October 23–27, 2022, Proceedings, Part I. Springer, 2023, pp. 66–79*
14. A. Chaturvedi and A. Shukla, "Automatic detection of satellite images using blob detection and boundary tracking techniques", *2020.*
15. S. Li, J. Chen, W. Peng, X. Shi and W. Bu, "A vehicle detection method based on disparity segmentation", *Multimedia Tools Appl.*, vol. 82, no. 13, pp. 19643-19655, May 2023.
16. A. A. Kumar et al., "A Comparative Study of Various Filtering Techniques", *ICOEI, 2021.*
17. N. M. Al-Shakarji, F. Bunyak, G. Seetharaman and K. Palaniappan, "Multi-object tracking cascade with multi-step data association and occlusion handling", 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), pp. 1-6, 2018.
18. Enric Meinhardt-Llopis1 , Javier S´anchez2 , Daniel Kondermann3 “Horn–Schunck Optical Flow with a Multi-Scale Strategy”, *Image Processing On Line on 2013–07–19*
19. K.C Hari; Sushil Shrestha; Manish Pokharel,"Video Object Motion Tracking using Dense Optical Flow Techniques", *2023 International Conference on Informatics and Computing (ICIC)*
20. Shuman Guo,Shichang Wang"A Review of Deep Learning-Based Visual Multi-Object Tracking Algorithms for Autonomous Driving", *Applied Sciences 12(21):10741,October 2022*
21. Chakradhara Panda,"Object Detection and Tracking using Faster R-CNN", September 2019,International Journal of Recent Technology and Engineering (IJRTE)
22. Sankaranarayanan, M., Mala, C., Mathew, S. “Efficient vehicle detection for traffic video-based intelligent transportation systems applications using recurrent architecture”. Multimed Tools Appl 82, 39015–39033, 2023.
23. Adithya Urs, Nagaraju C, "Object Motion Direction Detection and Tracking for Automatic Video Surveillance", International Journal of Education and Management Engineering (IJEME), Vol.11, No.2, pp. 32-39, 2021. DOI: 10.5815/ijeme.2021.02.04
24. Iqra Nosheen,Aysha Naseer,Ahmad Jalal, "Efficient Vehicle Detection and Tracking using Blob Detection and Kernelized Filter",*2024* *5th International Conference on Advancements in Computational Sciences (ICACS)*
25. Hodson, T. O.: Root-mean-square error (RMSE) or mean absolute error (MAE): when to use them or not, Geosci. *Model Dev., 15, 5481–5487, https://doi.org/10.5194/gmd-15-5481-2022, 2022.*
26. Madhusri Maity, Sriparna Banerjee, “Faster R-CNN and YOLO based Vehicle detection: A Survey”, 2021 5th International Conference on Computing Methodologies and Communication (ICCMC)
27. Meenu Gupta, Rakesh Kumar, Muskaan Gupta, "YOLO-Based Vehicle Detection and Counting for Traffic Control on Highway", *2024 2nd International Conference on Advancement in Computation & Computer Technologies (InCACCT)*