Vehicle Speed Calculation Using Optical Flow

Nishant Pandey
University of Maryland
College park, Maryland
npandey2@umd.edu

Joseph Thomas
University of Maryland
College Park, Maryland
joseph10@umd.edu

Amogh Wyawahare
University of Maryland
College Park, Maryland
wamogh@umd.edu

Rishikesh Jadhav
University of Maryland
College Park, Maryland
rjadhav1@umd.edu

Jayasuriya Suresh
University of Maryland
College Park, Maryland
jsuriya@umd.edu

Abstract

In the automotive industry, optical flow analysis has become essential for various applications, including the determination of object and vehicle velocities. This paper aims to provide a thorough analysis of conventional optical flow algorithms such as Lucas-Kanade and Forneback Gunnar, as well as RAFT, a deep network architecture for optical flow. The focus is on the specific application of finding the speed of vehicles. The commonly used KITTI dataset is employed to train optical flow models and facilitate their usage in car (object) detection. Additionally, the paper presents a hardware implementation to demonstrate the practical implementation of the proposed method. This comprehensive analysis of optical flow algorithms and their application in the automotive industry is expected to provide valuable insights for researchers and practitioners in the field.

1. Introduction

The accurate estimation of vehicle speed is a crucial aspect of modern autonomous vehicles, enabling efficient driving and improving road safety. In recent years, optical flow algorithms have emerged as a promising approach to estimate vehicle speed using video data from onboard cameras. One popular optical flow algorithm is the Lucas-Kanade method, which tracks image features between consecutive frames and computes the apparent motion of objects in the scene. Furthermore, the Farneback algorithm used can estimate motion at every pixel, allowing for more accurate and detailed analysis of motion patterns in the scene.

However, to improve the accuracy and robustness of vehicle speed calculation, RAFT(Recurrent All-Pairs Field Transforms) can be used. This model is trained on the KITTI dataset[1]. To further have a better understanding of real time scenario hardware setup is used to verify the accuracy of the model.

The hardware setup consists of Raspberry Pi 4, Pi Camera, and a mobile robot on which these devices are mounted on.

The pipeline also involves use of Yolo for finding the region of interest in this case the car for which the speed has to be computed. The region of interest is found using Yolo and then the velocity for the same is obtained using RAFT.

2. Algorithm Overview - Lucas Kanade

Lucas-Kanade is a sparse optical flow algorithm that estimates the motion between two frames of an image sequence by tracking a small set of feature points. The algorithm assumes that the motion of the

features between the two frames is well approximated by a translational motion, and estimates the motion parameters by minimizing the sum of squared differences between the two frames at each feature point.

To find the velocity of the car using a video the following steps were executed:

1. Extract features of the first frame of the test video.
2. Calculate the optical flow for the extracted points using Lucas- Kanade algorithm.
3. Find the displacement of the points frame by frame.
4. Determine the velocity of the feature points.
5. Convert the value of velocity obtained from pixel per second to meter per second by calibrating the camera.

Input: Image frames I_1, I_2
Output: Velocity vector $v = (u, v)$

foreach *pixel* (x, y) *in* I_1 **do**
  $I_x(x,y) \leftarrow \frac{1}{2}(I_1(x+1,y) - I_1(x-1,y))$;
  $I_y(x,y) \leftarrow \frac{1}{2}(I_1(x,y+1) - I_1(x,y-1))$;
  $I_t(x,y) \leftarrow I_2(x,y) - I_1(x,y)$;
end
foreach *pixel* (x, y) *in a window around each feature point* (x_0, y_0) **do**
  $A \leftarrow I_x(x_0,y_0)I_y(x_0,y_0)$; $b \leftarrow -I_t(x_0,y_0)$; $v \leftarrow (u,v) \leftarrow (0,0)$;
  foreach *pixel* (x, y) *in the window* **do**
    $A \leftarrow A\ I_x(x,y)I_y(x,y)$; $b \leftarrow b\ - I_t(x,y)$;
  end
  $v \leftarrow (A^TA)^{-1}A^Tb$;
end
return v;

Algorithm 1: Lucas-Kanade Optical Flow

3. Algorithm Overview - Farneback Gunnar

Farneback Gunnar is a dense optical flow algorithm that estimates the optical flow for every pixel in the image. It can be computationally expensive and may not work well in situations where the motion is very complex or the images have a lot of noise.

To find the optical flow and optical shade of the car in the video:

1. Convert the input images to grayscale.
2. Apply Gaussian smoothing to the grayscale images to reduce noise.

3. Compute the derivatives of the images with respect to x and y using the Sobel operator.
4. Compute the products of the derivatives at each pixel to obtain the structure tensor.
5. Apply Gaussian smoothing to the structure tensor.
6. Compute the eigenvalues and eigenvectors of the smoothed structure tensor to obtain the principal directions of motion.
7. Compute the optical flow for each pixel by computing the flow vector that minimizes the difference between the principal directions of motion at the current pixel and the previous pixel in the pyramidal representation.
8. Upsample the optical flow to the original image size.
9. After getting the flow of the frame. It can be used to get the velocity of the car. This can be done by extracting the magnitude and direction of the motion vectors.

Input: Previous frame I_{t-1}, current frame I_t, number of pyramid layers n, size of window w, regularization parameter λ
Output: Optical flow u, v
I_{t-1}, $I_t \leftarrow$ grayscale(I_{t-1}), grayscale(I_t); G_x, $G_y \leftarrow$ compute image gradients of I_t; u,$v \leftarrow$ initialize flow vectors to 0; **for** $i = n$ *down to* 1 **do**
  I^i_{t-1}, $I^i_t \leftarrow$ downsample(I_{t-1},I_t) to level i; u^i, $v^i \leftarrow$ resize(u, v) to level i; G^i_x, $G^i_y \leftarrow$ downsample(G_x,G_y) to level i; A, B, C, D, E, F $\leftarrow$ compute Farneback polynomial terms using I^i_{t-1}, I^i_t, G^i_x, G^i_y, u^i, v^i, w, λ; $solve(Au^i + Bv^i + C, Du^i + Ev^i + F) \leftarrow$ solve for flow vectors u^i, v^i using polynomial terms; u, $v \leftarrow$ resize(u^i, v^i) to level 0;
end
return u, v

Algorithm 2: Farneback Optical Flow

4. RAFT

Recurrent All-Pairs Field Transforms (RAFT),is a new deep network architecture for optical flow. RAFT produces multi-scale 4D correlation volumes for every pair of pixels, extracts per-pixel characteristics, and repeatedly updates a flow field using a recurrent unit that searches the correlation volumes. In addition to being highly efficient in terms of inference time, training speed, and parameter count, it exhibits good cross-dataset generalization. Three crucial components make up RAFT: (1) a feature encoder that extracts a feature vector for each pixel; (2) a correlation layer that generates a 4D correlation volume for every pair of pixels, followed by pooling to generate lower resolution volumes; and (3) a recurrent GRU-based update operator that retrieves values from the correlation volumes and incrementally updates a flow field that is initially set to zero. The method can be divided into three stages. (1) feature extraction, (2) computing visual similarity, and (3) iterative updates, where all stages are differentiable into end-to-end architectures.

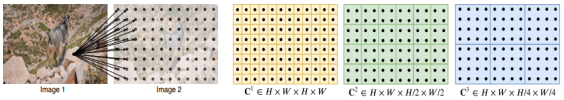

Fig 1: Building Correlation Volumes

2D slices of a full 4D volume are depicted above. For a feature vector in I1, we take the inner product with all pairs in I2, generating a 4D W ×H ×W ×H volume. The volume is then pooled using average pooling with kernel sizes {1, 2, 4, 8}.

a) Feature Extraction:
A convolutional network is used to extract the features. The feature encoder network converts the input images into dense feature maps with a reduced resolution for both I_1 and I_2. The encoder outputs features at a certain resolution and additionally a context network is used which extracts features from the first input image only.

b) Computing Visual Similarity:
A full correlation volume is calculated between all pairs to compute visual similarity. The correlation volume is formed by taking the dot product between all pairs of feature vectors. The correlation volume **C**, can be efficiently computed as a single matrix multiplication given as:

$$C_{ijkl} = \sum_h g_\theta(I_1)_{ijh} \cdot g_\theta(I_2)_{klh}$$

Now we construct a 4-layer pyramid $\{C^1, C^2, C^3, C^4\}$ by pooling the last 2 dimensions of the correlation volume with the above kernel sizes and an equivalent stride. By maintaining the first 2 dimensions, it is possible to maintain high resolution information, allowing this method to recover the motions of small fast-moving objects. A lookup operator L_C is used to generate a feature map by indexing from correlation pyramid. From the current estimate of optical flow it is possible to map each pixel in I_1 to estimated correspondence in I_2. A local grid is defined, and the local neighborhood of the corresponding points can be used to index from the correlation volume. Lookup is performed on the pyramid levels and the values from each level are appended into the feature map.

c) Iterative updates:
A sequence of flow estimates is estimated from an initial starting point and an update direction is produced with each iteration and applied to the current estimate. Flow, correlation, and a hidden state are taken as input and the updated Δf and hidden state are received as output. The flow field is first initialized to 0, then the current flow

estimate is used to obtain correlation features from the correlation pyramid and processed by 2 convolutional layers. 2 convolutional layers are also applied to the flow estimate to generate flow features. The input feature map is then taken as concatenation of the correlation, flow and context features, and is updated. The hidden state obtained by the GRU is then passed through 2 convolutional layers to predict the flow update. The predicted flow fields are sampled to match the resolution of the ground truth by taking full resolution flow at each pixel to be a convex combination of 3x3 grid of its coarse resolution neighbors. The final flow field is found by using a mask for a weighted combination over the neighborhood and then reshaping the flow field.

d) Our implementation of RAFT:

- The KITTI dataset was used to train and further generate the model for testing.
- This particular dataset was used as it was not very huge in size and aligned with our application.
- After evaluation, the error metric for the training was 1.096719 and the error metric for the validation was 3.894137.
- Testing dataset was made by taking videos of cars for which speed had to be computed.
- The idea was to obtain the velocity from the motion vectors of the car and then convert them to m/sec velocity.
- The flow of the image has two component matrices one is magnitude and other is angle.
- The magnitude value of the motion vectors of the car is obtained and are normalized.
- The mean of velocity in every frame is obtained to get an approximate idea of the average velocity and then the obtained velocity has to be converted to the desired unit.

5 YOLO

You only look once (YOLO) is a state-of-the-art, real-time object detection system. Classifiers or localizers are repurposed by other detection systems to carry out detection. They use the model to analyze an image at various scales and places. The image's high-scoring areas are referred to as detections. The model is trained using pre-processed and annotated images. These images are labeled using bounding boxes which are represented by 6 numbers – (pc, bx, by, bh, bw, c). Here, p_c is the confidence of an object being present in the bounding box. bx, by, bh and bw represent the center coordinates and the height and width of the bounding box. Finally, c represents the class of the object being detected and is an integer. A pre-trained model was used for the project as training a YOLO model takes a long time and high computation power.

a) Model details:

Inputs and outputs:

- The input is a batch of labeled images.
- The output is a list of bounding boxes along with the detected class.

Anchor Boxes:

- The anchor boxes are a set of predefined bounding boxes with a specific height and width chosen by exploring training data to choose height/width ratios that represent the different classes.

Encoding:

A grid cell responsible for detecting an object if the center/midpoint of the object falls in the grid cell. Each grid cell contains a specific number of anchor boxes. We flatten the last two dimensions for simplicity.

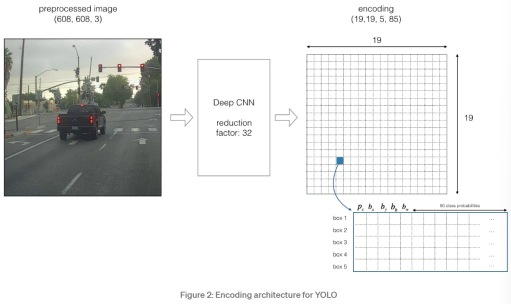

Fig 2: Encoding Image

Class Score:

For each anchor box, element-wise product is calculated and a probability is extracted that signifies that the box contains a certain class.
The class score is given by: $score_{ci} = p_c(c_i)$.

We use this score and assign the box to the object class with highest probability.

Visualizing Classes:

- For each grid cell, find the maximum probability scores
- Color that grid cell according to the object class with highest probability in that cell.
- Create a bounding box around the objects.

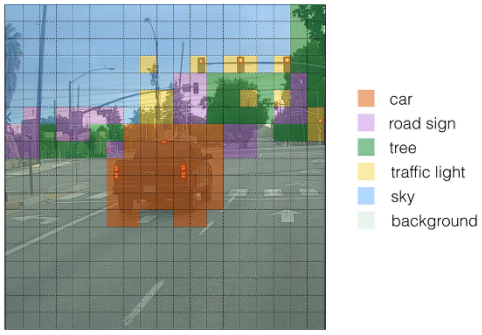

Fig 3: Bounding Box around Objects

Non-max suppression:

This is done to reduce the number of boxes from the steps followed above.

- Get rid of boxes with low probability scores
- Select only one box when more than one box overlaps with each other to detect the same object.

b) Filtering with threshold:

A filter is applied with a certain threshold to eliminate boxes with a score lower than the threshold. The dimensional tensor obtained is rearranged into the variables: 'box_confidence'(containing p_c), 'boxes'(containing box parameters), and 'box_class_probs'(containing class probabilities).

c) Non-max suppression:

This is a second filter for selecting the right bounding boxes. It uses a function called Intersection over Union (IoU)

- Box with the highest score is selected.
- Overlap of this box with other boxes is calculated and overlapping boxes are removed.
- The above steps are then iterated until boxes with a lower score are eliminated.

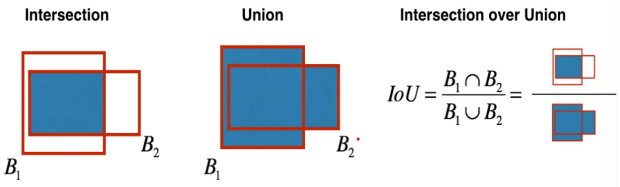

Fig 4: Intersection over Union (IoU)

5.2 Application of YOLO in the Pipeline

YOLO was used to focus on the car in our testing videos,It was used to get a bounding box after the car was identified by yolo.The dimensions of the bounding box was used to determine the ROI (Region of Interest) for which the optical flow was tracked and the relative changes in the motion of pixels was determined.The largest area of the bounding boxes found by the model was used as the ROI to focus and was assumed to be the car being tracked in the videos.Focusing on a ROI allowed for the velocity of the car only to be determined, otherwise it would have resulted in velocity being calculated for multiple targets within the video.

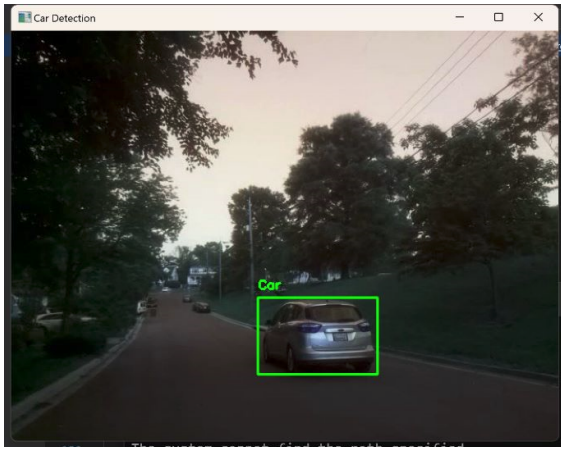

Fig 5: Max Area Bounding Box for ROI using YOLO

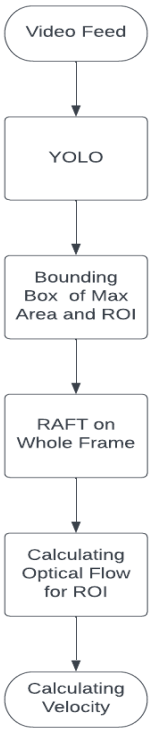

Fig 6: Pipeline for YOLO+RAFT for finding Velocity

6 Results - Using the test video (Lukas Canade and Farneback)

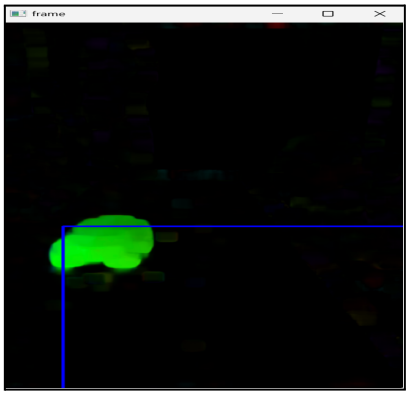

Fig 7: Optical Flow for Lucas

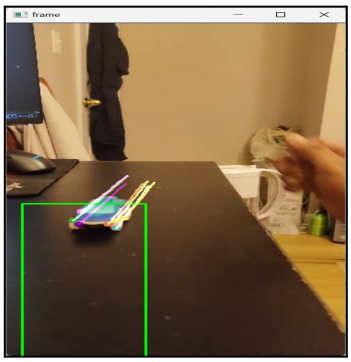

Fig 8: Feature points

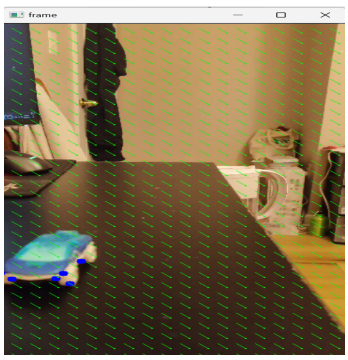

Fig 9: Optical Shade

6.1 Live Demo Results (Lukas-Farneback)

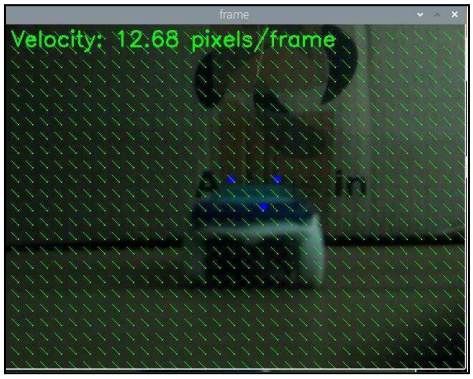

Fig 9: Optical shade

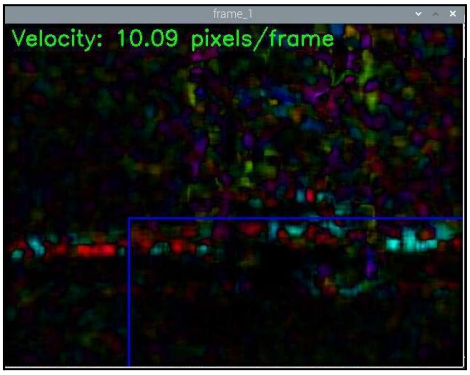

Fig 10: Optical flow

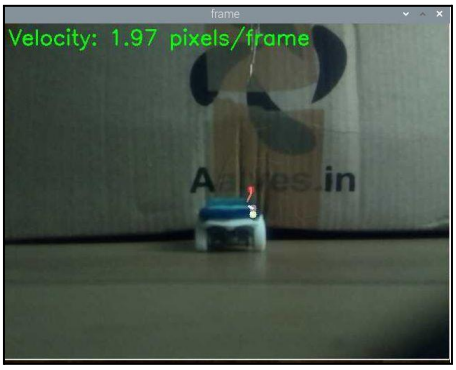

Fig 11: Velocity of the extracted points

6.2 Results - Using test video (RAFT)

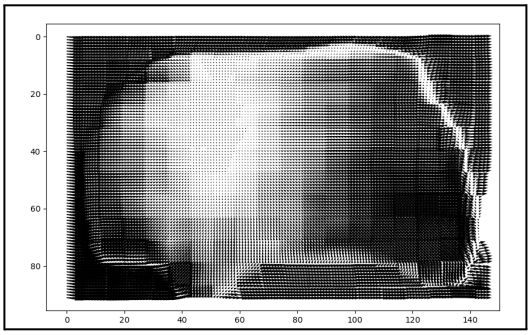

Fig 12: Motion vectors for just the car


Fig 13: Optical flow for test data

6.4 Live Demo (RAFT)

The live demo results are in the following links:

- https://drive.google.com/file/d/1vhBZ9c4gu8jtySNnI1wsDmVi3Eo-FCLR/view?usp=sharing
- https://drive.google.com/file/d/17wc-j3qBRNW92vrLlNJXw5YU8ts39wP5/view?usp=sharing

7. Hardware

A mobile robot was built to track targets and facilitate easy recording of videos.It is powered by DC motors that allow it to move in 4 directions. Encoders and a IMU are used to make the robot travel in a straight line or pivot left or right.The robot is controlled by a Raspberry 4.The camera used here is a Raspberry Pi Camera v2.1.The videos were recorded at a resolution of 640x480P at 30FPS.

The Raspberry Pi camera uses a 8MP Sony IMX219PQ CMOS sensor that is capable of 1080P at 30FPS. It has a focal length of 3.04 mm and a aperture of 2.0.Video was recorded at 480P for faster processing and to enable low latency video streaming over a local network.

The classical optical flow methods were tested on the Raspberry Pi 4 and the output is shown in the above sections. RAFT did not run on the Raspberry Pi 4 due to computational limitations and instead ran on a laptop.Video was streamed from the Raspberry Pi to the laptop to be fed into the model for processing.

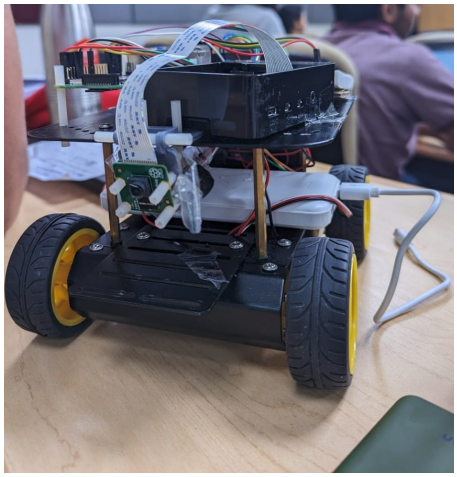
Fig 14: The Designed Robot

7.1 Video Streaming

Video streaming on the Raspberry Pi 4 was done using the python packages socketserver and server from https. These allow a network server to be set up on the Raspberry Pi that allows clients to directly stream video over a local network using the ip address of the server on the network.

The python package Picamera was used to easily access the camera and access the feed. OpenCV's videocapture object allows for directly grabbing video from a remote URL. The latency is barely visible to the human eye. Video streaming is done in the MJPEG (Motion JPEG) format. This format allows for low latency streaming of video with decent quality. The picamera python module has built-in support for this.

8. Future Work

We have been able to achieve our main goals. However there is scope for future work in a few areas.

We had a design for a robot to track and follow a moving target similar to leader-follower robots. However due to time constraints, we did not implement it. It was used to record videos only. This would require tracking of a moving object like a person and converting the velocity into scaled velocity for the robot to follow.

The velocity calculations to cover pixels per second into miles per hour is not very accurate. It can be made better by the use of depth data from the image and proper calibration of the camera to get the camera matrix.

Currently, velocity of the car tracked in videos is calculated after the output from RAFT is obtained. A modification in RAFT could be done to include velocity calculations after the optical flow is found by the model.

For our implementation, the velocity of a single car is determined from a video. Determining velocity of multiple cars from a single video would be better suited for real-world use. This would require labeling each target and tracking each target separately. Afterwards, velocity

for each tracked object has to be calculated in a similar manner done for getting the velocity of the car earlier.

9. Challenges

This was our first time working on optical flow so we faced many challenges:

- Understanding RAFT took a lot of time as it is a very complicated algorithm.
- We were unable to get the depth values needed to convert the pixel per sec velocity because we did not have a good camera. So the obtained values are not accurate.
- We got a lot of lag for live streaming and video analysis.
- The camera calibration did not have accurate values.
- The Raspberry Pi Camera struggled to give decent quality videos when lighting was not perfect.

10. Github link of the code

https://github.com/nishantpandey4/ENPM673_finalProject.git

References

[1] Andreas Geiger and Philip Lenz and Raquel Urtasun, Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite, {Conference on Computer Vision and Pattern Recognition (CVPR), 2012, https://www.kaggle.com/datasets/klemenko/kitti-dataset

[2] N. Sharmin and R. Brad, "Optimal Filter Estimation for Lucas-Kanade Optical Flow," Sensors, vol. 12, no. 9, pp. 12694–12709, Sep. 2012, doi: https://doi.org/10.3390/s120912694.

[3] Wu, Zhao, Gan, and Ma, "Measuring Surface Velocity of Water Flow by Dense Optical Flow Method," Water, vol. 11, no. 11, p. 2320, Nov. 2019, doi: https://doi.org/10.3390/w11112320.

[4] A. Dosovitskiy *et al.*, "FlowNet: Learning Optical Flow with Convolutional Networks." Accessed: May 11, 2023. [Online]. Available: https://openaccess.thecvf.com/content_iccv_2015/papers/Dosovitskiy_FlowNet_Learning_Optical_ICCV_2015_paper.pdf

[5] A. M. Mathew and T. Khalid, "Ego Vehicle Speed Estimation using 3D Convolution with Masked Attention," *arXiv:2212.05432 [cs]*, Dec. 2022, Accessed: May 11, 2023. [Online]. Available: https://arxiv.org/abs/2212.05432

[6] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras," Robotics: Science and Systems XIV, Jun. 2018, doi: https://doi.org/10.15607/RSS.2018.XIV.062.

[7] Raspberry Pi Camera Specifications ,https://elinux.org/Rpi_Camera_Module#Technical_Parameters_.28v.1_board.29

[8] MJPEG File Codec, https://docs.fileformat.com/video/mjpeg/

[9] Liu1, K. *et al.* (2018) *IOPscience*, *Journal of Physics: Conference Series*. Available at: https://iopscience.iop.org/article/10.1088/1742-6596/1004/1/012003 (Accessed: 18 May 2023).

[10] Teed, Z. and Deng, J. (2020) *Raft: Recurrent all-pairs field transforms for optical flow*, *arXiv.org*. Available at: https://arxiv.org/abs/2003.12039 (Accessed: 18 May 2023).

[11] Guide to Car Detection using YOLO | by Bryan Tan | Towards Data Science

[12] YOLO(YOU ONLY LOOK ONCE). What is YOLO | by Rajan Sharma | Medium

.