UNIVERSIDAD YACHAY TECH

UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO:
Vehicle Speed Estimation From Fixed Point Camera Using Projective Geometry and Object Tracking

Trabajo de integración curricular presentado como requisito para la obtención
del título de Ingeniero en Tecnologías de la Información

Autor:

Mejía Vallejo Héctor Andrés

Tutor:

Ph.D – Pineda Israel

Urcuquí, 05 de enero del 2022

Urcuquí, 1 de diciembre de 2021

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2021-00030-AD

A los 1 días del mes de diciembre de 2021, a las 14:00 horas, de manera virtual mediante videoconferencia, y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. MOROCHO CAYAMCELA, MANUEL EUGENIO , Ph.D.
Miembro No Tutor	Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.
Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.

El(la) señor(ita) estudiante **MEJIA VALLEJO, HECTOR ANDRES**, con cédula de identidad No. **0924656150**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, realiza a través de videoconferencia, la sustentación de su trabajo de titulación denominado: **Vehicle Speed Estimation From Fixed Point Camera Using Projective Geometry and Object Tracking.**, previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación a través de videoconferencia, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.	10,0
Presidente Tribunal De Defensa	Dr. MOROCHO CAYAMCELA, MANUEL EUGENIO , Ph.D.	10,0
Miembro Tribunal De Defensa	Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.	10,0

Lo que da un promedio de: **10 (Diez punto Cero)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

Firmado electrónicamente por:
HECTOR ANDRES MEJIA VALLEJO

MEJIA VALLEJO, HECTOR ANDRES
Estudiante

MANUEL EUGENIO MOROCHO CAYAMCELA
Digitally signed by MANUEL EUGENIO MOROCHO CAYAMCELA
Date: 2021.12.01 15:42:15 -05'00'

Dr. MOROCHO CAYAMCELA, MANUEL EUGENIO , Ph.D.
Presidente Tribunal de Defensa

Firmado electrónicamente por:
ISRAEL GUSTAVO PINEDA ARIAS

Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.
Tutor

ERICK EDUARDO CUENCA PAUTA
Digitally signed by ERICK EDUARDO CUENCA PAUTA
Date: 2021.12.01 15:46:41 -05'00'

Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.
Miembro No Tutor

DAYSY MARGARITA MEDINA BRITO

Firmado digitalmente por DAYSY MARGARITA MEDINA BRITO
Fecha: 2021.12.01 15:15:29 -05'00'

MEDINA BRITO, DAYSY MARGARITA
Secretario Ad-hoc

AUTORÍA

Yo, **Héctor Andrés Mejía Vallejo**, con cédula de identidad 0924656150, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autora (a) del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, 05 de enero del 2022.

Firmado electrónicamente por:
HECTOR ANDRES
MEJIA VALLEJO

Héctor Andrés Mejía Vallejo
CI: 0924656150

AUTORIZACIÓN DE PUBLICACIÓN

Yo, **Héctor Andrés Mejía Vallejo,** con cédula de identidad 0924656150, cedo a la Universidad de Investigación de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior

Urcuquí, 05 de enero del 2022.

Firmado electrónicamente por:
HECTOR ANDRES MEJIA VALLEJO

Héctor Andrés Mejía Vallejo
CI: 0924656150

Dedicatoria

"Para quienes estuvieron ahí para mí, con toda su paciencia, con todo su corazón: mis padres y mi hermana.

A los que me levantaron el ánimo cuando y me trajeron alegría tantas veces: mis mejores amigos Peter, Francisco, Matthew, Julio y mis monos Bryan, Luis, Demetrio, Fer, Cristhian e Isaac.

Para mis maestros más cercanos, que me dieron mucho a pesar de que teníamos tan poco. Para mi abuela, Doménica y Diana, las personas más cariñosas que he tenido. Finalmente, a los que vendrán mañana, trayendo nuevas oportunidades y experiencias. Un millón de gracias, este hermoso capítulo de mi vida siempre estará ahí dentro de mí. ¡Lo hice! Esto es para ustedes."

Héctor Andrés Mejía Vallejo

Resumen

Cada año, aproximadamente 1,35 millones de personas mueren en las carreteras del mundo debido a accidentes de tránsito. Muchas ciudades importantes han implementado sistemas de cámaras que pueden usarse para estimar la velocidad de vehículos.

Este trabajo propone un flujo de trabajo de tres componentes para tal propósito: Primero, la estimación de homografía se emplea para retroproyectar las coordenadas de la imagen a las coordenadas de longitud-latitud. Luego, se utiliza un detector de objetos para localizar vehículos en la escena. Finalmente, un rastreador de objetos recibe esas detecciones para dar seguimiento a los vehículos y estimar su velocidad. El proceso de estimación de velocidad se realiza para cada vehículo convirtiendo las ubicaciones del plano de la imagen en coordenadas del mundo real utilizando la matriz de homografía, luego calculando la distancia entre líneas y dividiéndola por un marco de tiempo de medio segundo. Además, se compararon tres implementaciones de estimación de homografía: un algoritmo básico de Transformación lineal directa (DLT), un DLT robusto que utiliza el consenso de muestra aleatoria (RANSAC) y una metodología propuesta que emplea algoritmos evolutivos para una búsqueda localizada de puntos óptimos, así como dos versiones de un rastreador de objetos. El primero usa la distancia euclidiana para asignar detecciones a las pistas, mientras que el segundo usa Intersection over Union (IoU), más un umbral para minimizar la asignación incorrecta de pistas.

Finalmente, los resultados muestran que el flujo que utiliza cualquiera de los tres métodos de homografía y el rastreador de IoU puede registrar con éxito distribuciones de velocidad para vehículos que están a la expectativa de acuerdo con el entorno del tráfico urbano y también puede manejar cambios en la velocidad del vehículo.

Palabras Clave:

Estimación de velocidad, Homografía, Detección de objetos, Rastreo de objetos.

Abstract

Each year, approximately 1.35 million people die on roadways around the world due to traffic accidents. Most major cities have implemented a grid of surveillance cameras in areas of interest that can be used for vehicle speed estimation using only image sequences as a source of input.

This work proposes a three-component workflow for such purpose: First, homography estimation is employed to backproject image coordinates to longitude-latitude coordinates. Then, an object detector is used to locate vehicles in the scene. Finally, an object tracker receives those detections to create vehicle tracks and estimate speed. The speed estimation process is done for each vehicle by converting the image plane locations to real-world coordinates using the homography matrix, then calculating haversine distance, and dividing by a time frame of half a second. Furthermore, three implementations of homography estimation were compared: a base Direct linear Transformation (DLT) algorithm, a robust DLT using Random Sample Consensus (RANSAC), and a proposed methodology that employs evolutionary algorithms for a localized search of optimal points, as well as two versions of an object tracker. The first uses euclidean distance to assign detections to tracks, while the second uses Intersection over Union (IoU), plus a threshold to minimize wrong track assignment.

Finally, results show that the workflow using any of the three homography methods and the IoU tracker can successfully register speed distributions for vehicles that are on expectation according to the urban traffic setting and can also handle changes in vehicle speed.

Key Words:

Speed Estimation, Homography, Object Tracking, Object Detection.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Currently, the standard for vehicle speed estimation on urban areas and highways is the use of radar or lidar speed signs. They can be costly to buy, install and maintain. Therefore, they have to be placed only on limited, strategic areas. With the rise of artificial intelligence, the internet of things, and scientific computing in general, backed by increasingly sophisticated frameworks and massive parallelization of computation, lies an opportunity in developing novel systems that can serve as an alternative to traditional methods. In addition, most major cities already implement networks of traffic surveillance cameras, placed on a fixed position, next to traffic lights or at the side of the road, that can be utilized for vehicle speed estimation using computer vision.

In this work, such system is studied, designed and implemented. Specifically, the proposed method is composed of three main components that work together to enable speed estimation using a camera. First, it uses projective geometry to estimate a homography matrix that can translate image coordinates from the camera into longitude-latitude coordinates. This is done through robust optimization methods on top of Direct Linear Transform, with image coordinates - world coordinates correspondences given beforehand. Then, after camera calibration, a deep learning based object detector is used to recognize and extract the location of vehicles on the video feed. In this work, YOLOv4 model is used as the detector. Furthermore, a distance based tracking algorithm takes those detections as input, and frame by frame establishes tracks for each unique object identified. The distance metrics to be tested are euclidean distance, and Intersection over Union (IoU). Moreover, the same tracker is designed to calculate the speed of vehicles by estimating the distance traveled over a fixed period of time, and continuously being updated using the same time frame until the object leaves the scene. Note that speed, not velocity, is the variable to estimate since only magnitudes are relevant, regardless of the direction of the vehicles.

Finally, a compilation of records taken from 14 different live feeds from the United States city of Seattle are used to test the proposed method. These live feeds are available to the public and provided by the Department of Transportation of that city. Finally, after all frames are processed, speed information is stored and aggregated for computing statistics and visualizations that will assess the performance of the system.

1.2 Problem Statement

The inspiration for the development of this work arose from the increasing accidents on the streets caused by irresponsible speeding drivers, becoming threats for other people, potential public or private property destruction, further traffic congestion and many other consequences. In addition, the solution proposed by this work can also help monitor and alleviate traffic congestion and other related problems when integrated with other technologies. Nowadays, major cities already have implemented infrastructure that enables novel solutions. Thus the question becomes: How can we leverage that?

According to the Centers for Disease Control and Prevention (CDC), each year, 1.35 million people die on roadways around the world. Almost 3700 people die in crashes involving cars, buses, motorcycles, bicycles, trucks or pedestrians every day. Moreover, it is also estimated that fatal and nonfatal crash injuries will cost the world economy approximately 1.8 trillion dollars. This places a special toll on low and middle-income countries, that lack resources to mitigate further consequences. In addition, a report on the analysis of driving speed as a risk factor from the Pan American Health Organization (PAHO) and the World Health Organization (WHO) [16] says that both the probability of fatal injuries for a pedestrian by car hits and stopping distance for emergency braking increase exponentially as driving speed increases. For instance, an individual hit by a vehicle, traveling at 40 km/h, has 60% of survival probability, while a hit by a vehicle that travels at 70 km/h is almost always fatal.

There are also economic and environmental consequences derived from traffic congestion and non-optimized traffic management. For instance, added fuel consumption, increase in delivery times and loss of productivity for industries that rely on transportation [17]. This, in particular, happens at rush hours and there are no widespread IoT solutions that can mitigate these problems in real time.

Finally, the development of intelligent systems that can measure speed while also having visual information are becoming increasingly important, as they enable public authorities to make informed decisions much faster, and could serve as a basic platform for new technologies that can tackle the aforementioned problems. For instance, with such solution, one can also produce intelligent traffic lights to better manage traffic, implement public displays that could advise drivers to take alternate routes in real-time. The benefits of implementing these intelligent systems are explained on Section 4.1

1.3 Objectives

1.3.1 General Objective

Implement a vehicle speed estimation solution using cameras in unconstrained environments. That is using video feeds, regardless of acquisition hardware, environmental conditions and without limiting scenes based on visual characteristics.

1.3.2 Specific Objectives

- Compare two different tracking implementations. One using Euclidean distance, and the other using Intersection over Union for track assignment.
- Compare three robust methods for homography estimation to transform image coordinates into real longitude-latitude coordinates: a base Direct Linear Transformation (DLT) algorithm, a robust DLT algorithm using Random Sample Consensus (RANSAC) and a proposed methodology using evolutionary algorithms.
- Extend a classic tracking algorithm to also store real world plane locations, as well as speed measures.

Chapter 2

Theoretical Framework

2.1 Deep Learning

2.1.1 Deep Learning Overview

Deep learning is a subfield of machine learning that employs a collection of computational models called deep neural networks used both for supervised or unsupervised tasks. Traditionally, the performance of machine learning models highly relied on feature engineering to describe data. Moreover, bad representations most likely result in a model achieving low performance. Deep learning is a shift in this paradigm as it tries to generate automatic, high-level abstractions from data as features. This has the cost of high computational and data demands, since these models need large volumes of samples, in the range from thousands to millions, to be able to generalize knowledge for a specific task. For instance, an image classification model generally needs thousands of image samples to achieve high performance [18]. In addition, these algorithms require computationally intensive matrix operations across all its layers. However, deep learning has become popular in recent years due to the birth of technologies that harness graphics cards to achieve massive parallelization of computations. One example of such technologies is CUDA, or Compute Unified Device Architecture, specifically built for Nvidia Graphics Processing Units (GPUs) that enables general-purpose programming on this kind of hardware [19]. Finally, neural networks has shown impressive performance in the fields machine vision, for multi object detection [20, 21, 22], and image classification [23, 24, 20]; in speech processing, for speech-to-text systems [25, 26]; in natural language processing for topic modeling, text classification, summarization [27], and many other fields.

2.1.2 Artificial Neural Networks Forward Pass

Neural networks are connectionist systems as they loosely mimic human brain connections. Their structure consists of multiple layers: input, hidden layers, and output. Each layer is composed of artificial-mathematical neurons as seen in Definition 1.

Definition 1. *Let $m-1$ be the number of input variables in a single neuron with signals $x_2, x_3, ..., x_m$ and a bias term $x_1 = 1$. Let the weight vectors be $\boldsymbol{w_1}, \boldsymbol{w_2}..., \boldsymbol{w_m}$ and an*

activation function σ. Then, the output of such neuron is defined by:

$$y_j = \sigma(\sum_{i=0}^{m} w_{i,j} x_i). \tag{2.1}$$

Furthermore, with $j = 0, 1, ..., k$, where k is the number of neurons on a neural network layer, the forward pass of that same layer can be written as:

$$\boldsymbol{y} = \sigma(\boldsymbol{x^T} \cdot \mathbf{W}). \tag{2.2}$$

From Equation 2.2, in the forward pass of a multi-layered fully connected neural network of l layers, the inputs of layer l, are the outputs from $l-1$:

$$\boldsymbol{h_1} = \sigma(\boldsymbol{x^T} \cdot \mathbf{W_1}),$$

$$\boldsymbol{h_2} = \sigma(\boldsymbol{h_1} \cdot \mathbf{W_2}),$$

$$\vdots$$

$$\boldsymbol{y} = \sigma(\boldsymbol{h_{l-1}} \cdot \mathbf{W_{l-1}}).$$

Finally, a diagram of these formulations is represented in Figure 2.1.

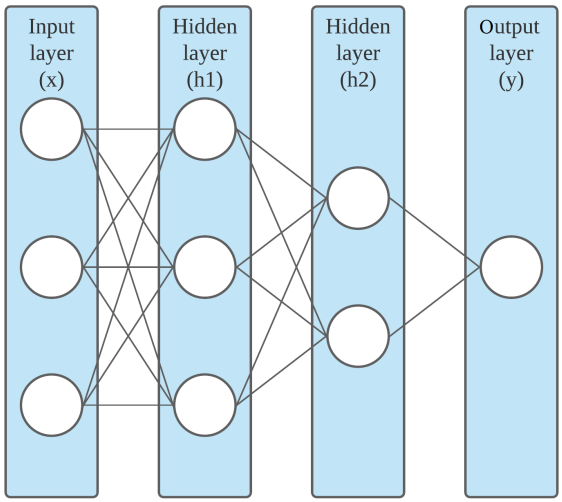


Figure 2.1: Neural network with input (x), hidden layers (h1,h2), and output (y).

2.1.3 Activation Functions

Activation functions are an essential component of deep learning. Without them, each neuron reduces to a linear model. Since a neural network is a composition of functions, then the overall model would only be able to learn linear patterns, as stated in Theorem 1.

Theorem 1. *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^n$ be two linear transformations. Then $g \circ f : \mathbb{R}^n \to \mathbb{R}^n$ is also a linear transformation.*

Proof. Let $x, y \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$. Then:

$$\begin{aligned}(f \circ g)(x+y) &= f(g(x+y)) \\ &= f(g(x) + g(y)) \\ &= f(g(x)) + f(g(y)) \\ &= (f \circ g)(x) + (f \circ g)(y)\end{aligned} \tag{2.3}$$

$$(f \circ g)(\alpha x) = f(g(\alpha x)) = f(\alpha g(x)) = \alpha f(g(x)) = \alpha (f \circ g)(x) \tag{2.4}$$

□

Activation functions let a deep learning model learn complex, non-linear patterns and specializes hidden neurons to learn different features from input data [28]. Furthermore, these functions aid in neural network convergence. The most popular activation functions are rectified linear unit (ReLU), exponential linear unit (ELU), leaky ReLU, sigmoid (logistic), softmax, hyperbolic tangent (tanh), and a recently developed function by Google called Swish [29].

All rectifier units, along with Swish, are used in hidden layers because of fast convergence and can model sparse, complex patterns easily [30]. On the other hand, sigmoid and hyperbolic tangent are mostly used in binary classification. In particular, a sigmoid function transforms an input into a probability. All these functions have the signature $f : \mathbb{R} \to \mathbb{R}$. Finally, softmax is a generalization of sigmoid function that enables multi-class classification, and has $f : \mathbb{R}^n \to \mathbb{R}^n$ signature. Table 2.1 defines all formulas for each activation function, along with their derivatives.

In addition, Figure 2.2 depicts all plots for each activation function.

2.1.4 Weights Initialization of Neural Networks

It is well known that proper weights initialization is crucial for network convergence since a bad strategy could lead to exploding gradients, vanishing gradients, and even leaving the network unable to update its weights. Some of the most popular initialization techniques are described Table 2.2, following the work of Boulila *et al.* [1].

These methods are applicable when neural networks are trained from scratch. However, there is active research [31] in reusing previously trained model weights for any general task, and fine-tuning for a more specific or similar version of the former. This is especially beneficial when there is a small number of training examples or when facing training time constraints.

2.1.5 Artificial Neural Networks Back-Propagation

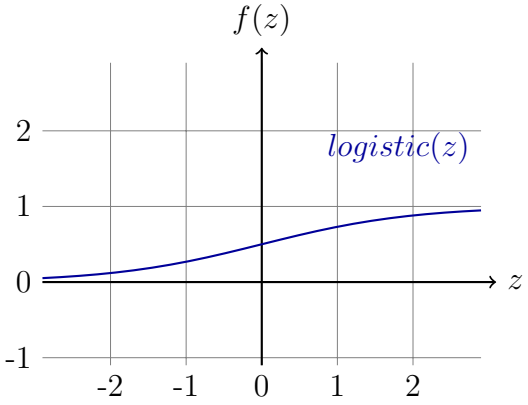
Machine Learning is an important shift as a programming paradigm. It lets a model learn from data without explicitly defining rules in advance. In general, most problems of this

Table 2.1: Activation functions formulas and their first order derivatives.

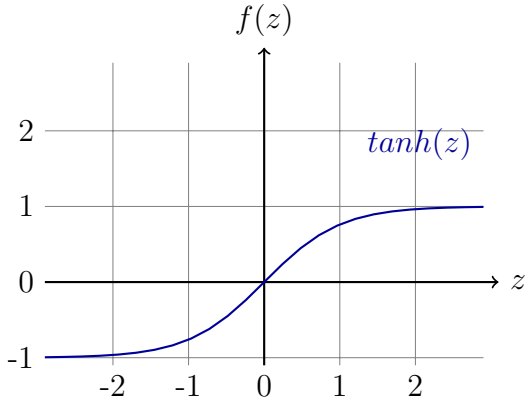
Function name	**Formula**	**Derivative formula**
Rectified Linear Unit (ReLU)	$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$	$f'(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$
Sigmoid-Logistic	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Hyperbolic Tangent (Tanh)	$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	$f'(x) = 1 - f(x)^2$
Leaky ReLU	$f(x) = \begin{cases} \alpha x & x \leq 0 \\ x & x > 0 \end{cases}$	$f(x) = \begin{cases} \alpha & x < 0 \\ 1 & x > 0 \end{cases}$
Exponential Linear Unit (ELU)	$f(x) = \begin{cases} \alpha(e^{x} - 1) & x \leq 0 \\ x & x > 0 \end{cases}$	$f(x) = \begin{cases} \alpha e^{x} & x < 0 \\ 1 & x > 0 \end{cases}$
Google Swish	$f(x) = \frac{x}{1+e^{-x}}$	$f(x) = \frac{1 + e^{-x} + xe^{-x}}{(1+e^{-x})^2}$

kind can be written as a finite sum:

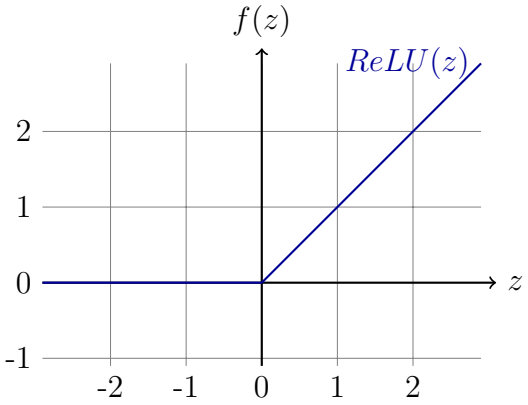
$$f(\Theta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\Theta), \tag{2.5}$$



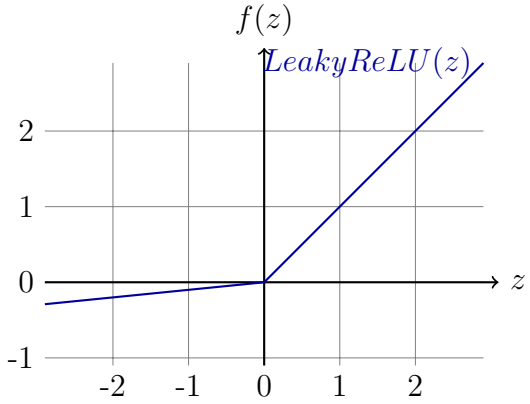
(a) Logistic or Sigmoid activation function.



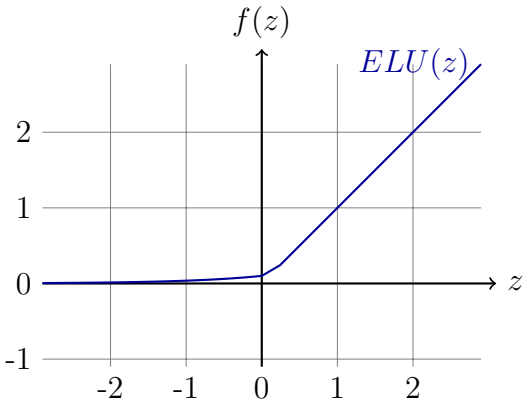
(b) Hyperbolic Tangent activation function.



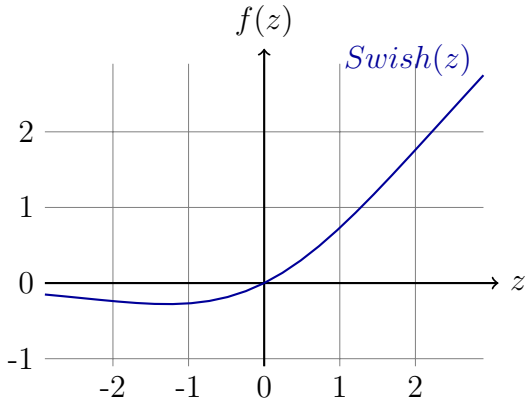
(c) Rectified Linear Unit activation function.



(d) Leaky ReLU activation function.



(e) Exponential Linear Unit activation function.



(f) Swish activation function.

Figure 2.2: Activation functions graphs

with Θ being the model parameters, f is a defined and differentiable loss function that measures how distant a model output is from a true label, and n, the total number of samples on a dataset. This formulation is convenient since the most popular optimization

Table 2.2: Neural network initialization approaches. Source [1]

Initialization Method	**Advantages**	**Disadvantages**
All-Zeros/Constants	Simplicity	Symmetry problem leading neurons to learn the same features
Uniformly at random	Improves the symmetry-breaking process of constant initialization	Could lead to vanishing gradients. The gradient is small, and could cause slow gradient descent convergence, or no convergence.
LeCun	Solves growing variance and gradient problems	Not useful in constant width networks, and takes into account the forward propagation of the input signal.
Xavier	Decreases the probability of experiencing vanishing or exploding gradient	Can result in dying neurons problem during training.
He	Solves dying neurons problem	Not useful for layers with differentiable activation functions like ReLU.

technique used in artificial intelligence currently is Gradient Descent, which uses formulations of this kind. The core concept is to find the minimum of f, traveling in the opposite direction of the gradient. Thus, the finite sum, described in Equation 2.6, becomes an optimization problem with the following form:

$$\min_{\Theta} f(\Theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\Theta). \tag{2.6}$$

In the domain of deep learning, this concept is also applicable. See Definition 2.

Definition 2. *Let f be a loss function, y_i, $i = 1, ..., n$ a true data label; $\mathbf{D}$ is a deep learning model that maps a data point $\boldsymbol{x} \in \mathbb{R}^n$ into a prediction $\hat{y} \in \mathbb{R}^n$ with weights Θ. The learning process on a deep neural network using gradient descent can be formulated as:*

$$\min_{\Theta} f(\Theta) = \frac{1}{n}\sum_{i=1}^{n} f(y_i, \mathbf{D}(x; \Theta_i)), \tag{2.7}$$

with $\mathbf{D}(x; \Theta_i) = \hat{y}_i$.

In this sense, the weights of a neural network can be fitted to output predictions as close to a true training label as possible, while being generalizable to unseen examples. These

weights, or neural network parameters are updated using a Gradient Descent iteration as seen in Definition 3.

Definition 3. *Let $\Theta = \{\theta^{(1)}, ..., \theta^{(l)}\}$ be the set of model parameters of a model with l layers, t the gradient descent iteration index, f the loss function, and $\eta \in (0, 1]$, the learning rate that is going to limit the computed gradients; then the parameters update is given by:*

$$\Theta_{t+1} \leftarrow \Theta_t - \eta \nabla f(\Theta), \tag{2.8}$$

or more explicitly:

$$\Theta_{t+1} \leftarrow \Theta_t - \eta \frac{\partial f(\Theta)}{\partial \Theta}. \tag{2.9}$$

2.1.6 Loss Functions

In each forward propagation, a neural network produces a numerical output. On the training phase of the model, these estimations are compared against the true labels of a dataset to produce a distance using a loss function. These loss values are then used to correct the weights in the backpropagation stage, and to monitor the optimization process. Generally, it is expected that these values drop dramatically on the first training iterations and to finally stagnate on later stages. However, vanishing or exploding gradients can disrupt this process making the loss value settle before reaching a minima or to unexpectedly increase.

Multiple loss functions are employed according to two general tasks: regression for continuous value predictions, or classification for categorical values. Moreover, the most common functions are defined in Table 2.3.

2.1.7 Deep Convolutional Neural Networks

In the past, most computer vision research revolved around feature engineering, since traditional machine learning models required tailored features for a single task. For instance, scale-invariant feature transform (SIFT) [32], histogram of oriented gradients (HOG) [33], and many other descriptors were needed for computer vision tasks like face recognition. However, the rise of convolutional neural networks revolutionized the field, focusing the research on building better deep learning architectures, rather than designing features. These new generation of models dramatically outperformed their traditional machine learning counterparts on computer vision benchmarks like COCO [34] and Imagenet [35].

The most basic convolutional neural networks are composed of convolutional, pooling, fully connected, and output layers.

Convolutional layers apply two dimensional correlations between a set of filters and inputs, that along with an activation function produces a feature map. Taking into account that image features are restricted to certain areas of the image, a fully connected layer would be impractical, since it produces a combination of all neurons. Moreover, introducing convolutions are a much more efficient approach, as it produces feature maps only on its receptive field. Definition 4 explains how a convolutional layer is forward propagated on a single input region.

Table 2.3: Most common loss functions used in deep learning

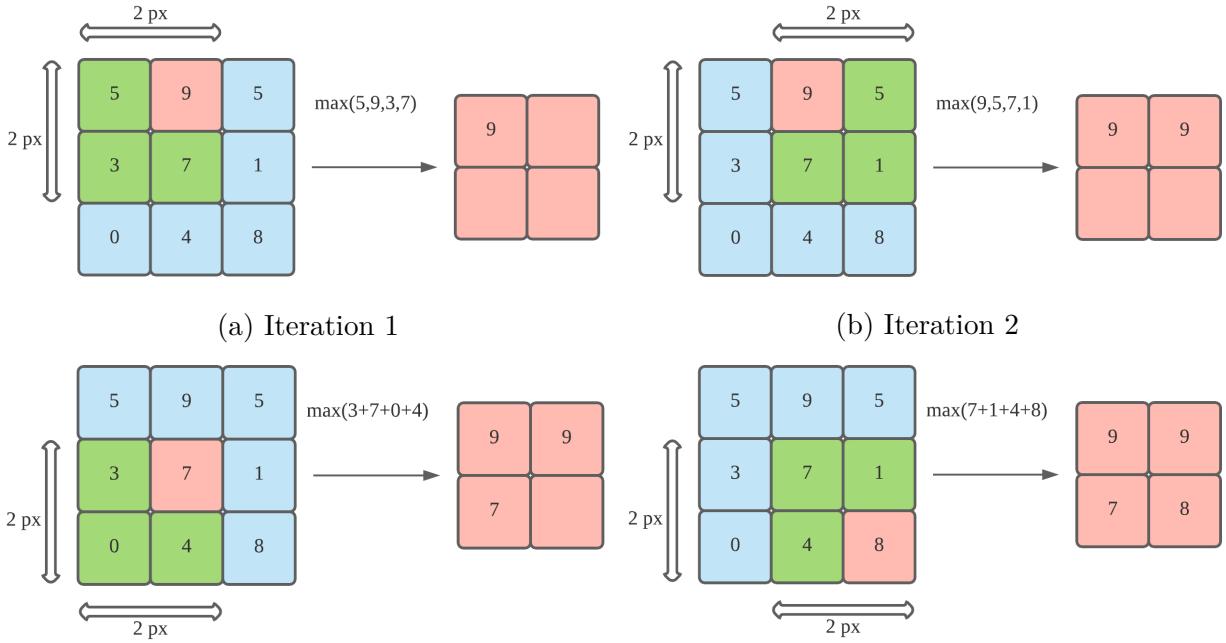
Function Name	**Task**	**Formula**
Mean Squared Error	Regression	Let $n \in \mathbb{N}$ the number of examples and $y, \hat{y} \in \mathbb{R}^n$ be the true labels and estimates, respectively. Then: $$L(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
Root Mean Squared Error	Regression	Let $n \in \mathbb{N}$ the number of examples and $y, \hat{y} \in \mathbb{R}^n$ be the true labels and estimates, respectively. Then: $$L(y, \hat{y}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$
Categorical Cross-Entropy	Classification	Let $n \in \mathbb{N}$ be the number of examples, $m \in \mathbb{N}$ the number of classes, and $\mathbf{Y}, \hat{\mathbf{Y}} \in M_{n\times m}(\mathbb{N})$ be the true labels and estimates, respectively. Then: $$L(\mathbf{Y}, \hat{\mathbf{Y}}) = -\sum_{i=0}^{n}\sum_{j=0}^{m}(y_{i,j} log(\hat{y}_{i,j}))$$

Definition 4. *Let $\mathbf{W_z} \in \mathbf{M_{m\times m\times c}}(\mathbb{R})$ be any convolutional filter with $\mathbf{W} = \{\mathbf{W_1}, ..., \mathbf{W_n}\}$, the set of all filters in the layer; $m \in \mathbb{N}$ the width and height of the filter, and $c \in \mathbb{N}$ the number of channels in the filter. Let $\mathbf{X} \in \mathbf{M_{s\times s\times c}}(\mathbb{R})$ be an input image with $s, c \in \mathbb{N}$, s is the width and height of the image, and c is also the number of channels of the image. Let $b_z \in \mathbb{R}$ be any bias term with $\mathbf{B} = \{b_1, ..., b_n\}$, the set of all biases in the layer. Finally, let σ be an activation function. Then a single output pixel of a convolutional layer with $\mathbf{X}$ as input, centered at $x, y \in \mathbb{N}$ is given by:*

$$C(\mathbf{X}, \mathbf{W_z}, b_z)_{x,y} = \sigma(b_z + \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{c}\mathbf{W_z}_{i,j,k}\mathbf{X}_{x+i-1,t+j-1,k}). \tag{2.10}$$

Pooling layers are non-trainable layers that perform a downsampling operation, spatially reducing the size of the image. The purpose is to introduce translation invariance

to the feature maps and reduce the number of computations on later layers of the network. Similar to a convolutional layer, it also needs filter size, stride, and padding hyperparameters. Moreover, there exist two kinds of pooling operations: max-pooling and average-pooling. When the kernel is over an image region, the maximum or the average is taken according to the defined operation. Figures 2.3 and 2.4 depict max-pooling and average-pooling respectively.



(a) Iteration 1 (b) Iteration 2

(c) Iteration 3 (d) Iteration 4

Figure 2.3: Max pooling operation

Finally, using max-pooling and convolutional layers, one can build convolutional neural networks like AlexNet [36] architecture.

2.1.8 Residual Neural Networks

As networks increase in depth, their performance increase as well since the feature maps can be more robust and specialized. Nevertheless, they present a problem at training time as gradients in earlier layers start to vanish. Resnet [37] was proposed to solve this problem, introducing skip connections or shortcuts between layers, as explained in Definition 5 to let the error propagate more easily and reduce accuracy saturation.

Definition 5. *Let f_i, f_j be two neural network layer functions with $\boldsymbol{x_i}$ the input to f_i, and $j > i$. Then the input $\boldsymbol{x_j}$ of f_j is given by:*

$$x_j = f_i(x_i) + x_i. \tag{2.11}$$

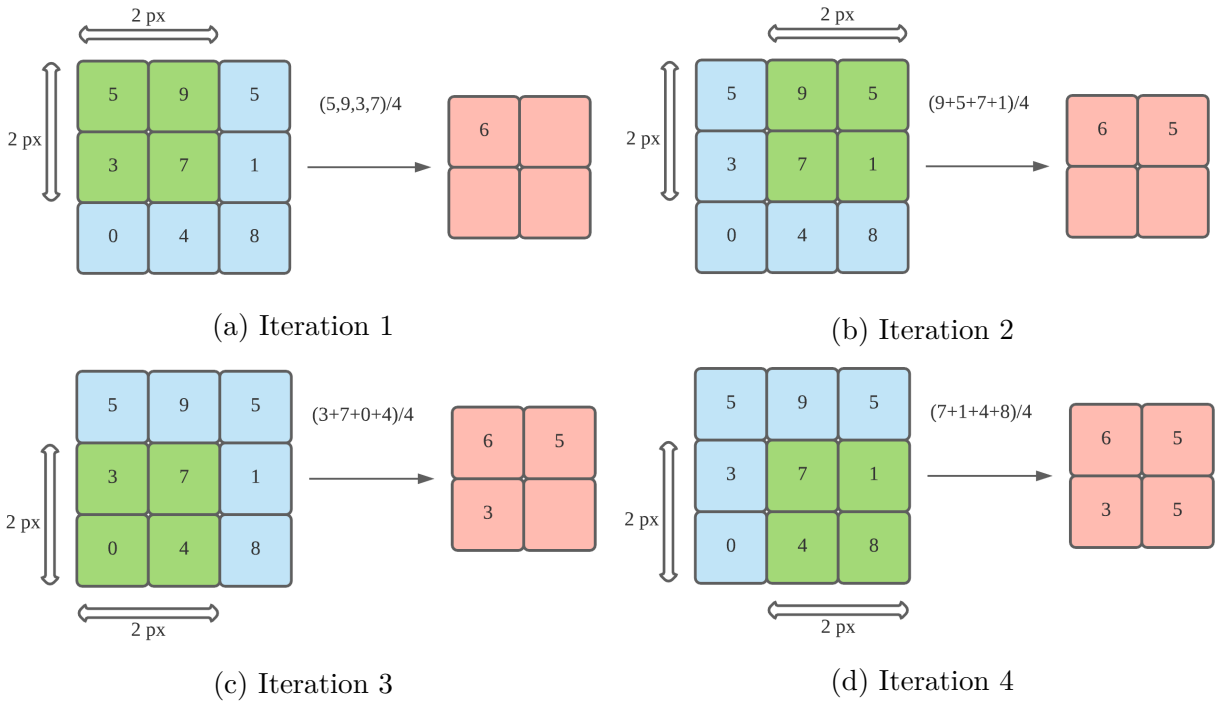


Figure 2.4: Average pooling operation

Note that, in the context of neural networks, some f can be a composition of layers and therefore, residual connections are not necessarily constrained to consecutive layers. They can be further apart, as depicted in Figure 2.5.

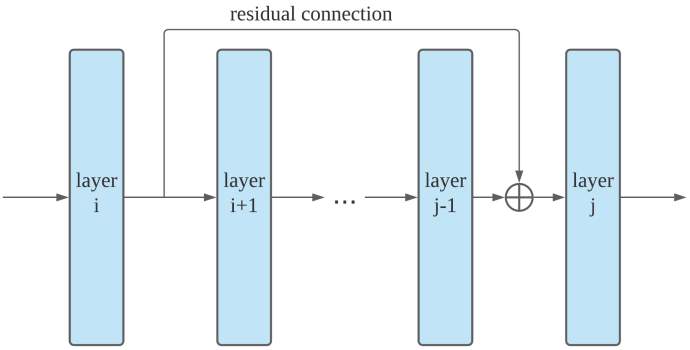


Figure 2.5: Residual connection.

2.1.9 Batch Normalization

Training deep learning models is complicated as the network has to continuously adapt to the changing distribution of the inputs on each layer. This phenomenon is known as the internal covariance shift. To solve this problem, batch normalization [38] was proposed. Described in Definition 6, this layer transforms each input dimension to have zero mean

and unit variance, resulting in a dramatic increase in training speed.

Definition 6. *Let $\boldsymbol{x} \in \mathbb{R}^n$ the input of a neural network layer. The normalization is performed in each dimension and is given by:*

$$\hat{\boldsymbol{x}}^{(k)} = \frac{\boldsymbol{x}^{(k)} - E[\boldsymbol{x}^{(k)}]}{\sqrt{Var[\boldsymbol{x}^{(k)}]}}, \tag{2.12}$$

where k is the layer index, $E[\boldsymbol{x}^{(k)}]$ is the expected value, and $Var[\boldsymbol{x}^{(k)}]$ is the variance of the input. Consequently, the normalized value is scaled and shifted:

$$y^{(k)} = \gamma^{(k)}\hat{\boldsymbol{x}}^{(k)} + \beta^{(k)}, \tag{2.13}$$

where $\beta \in \mathbb{R}$ and $\gamma \in \mathbb{R}$ are trainable parameters.

Note that the expectation and variance are computed over the training dataset only.

2.2 Projective Geometry

The first studies about the geometry of perspective relate to the Renaissance era, introducing the concepts of points at infinity, and how parallel lines when applied a perspective transformation converged to those points at infinity [39]. This early concept is very intuitive, as can be seen in Figure 2.6, a landscape with a straight railroad heading right to the horizon line. Such parallel lines meet at one vanishing point, belonging to the horizon.

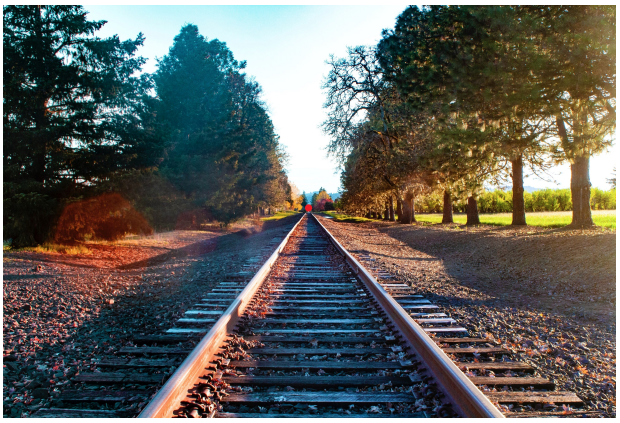


Figure 2.6: Railway image with a red dot indicating the vanishing point. Source `https://unsplash.com/photos/S5jD0E8D0C0?utm_source=unsplash&utm_medium=referral&utm_content=creditShareLink`

Nowadays this field is extensively being studied since it enables applications like GPS systems, radar, satellite, and civilian that require locations in real time [40].

2.2.1 Homography Transformation

A homography refers to the transformation between two planes. Under the image processing setting, it provides a means for camera calibration that describes a projection from world points into image points. Moreover, it is possible to explain this mapping in terms of matrix multiplications using homogeneous coordinates from the viewing point and the point on the image plane. See Definition 7.

Definition 7. *Let $s \in \mathbb{R}$ be a scale factor, $\boldsymbol{p'} = \begin{pmatrix} x' & y' & 1 \end{pmatrix}^T$ and $\boldsymbol{p} = \begin{pmatrix} x & y & z & 1 \end{pmatrix}^T$ be the homogeneous coordinates of a point in the image and world plane, respectively. Then a homography is expressed as:*

$$s\boldsymbol{p'} = \mathbf{MQ}\boldsymbol{p},$$

$$s\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}, \tag{2.14}$$

with $\mathbf{M}$ the matrix of intrinsic camera parameters: $f_x, f_y \in \mathbb{R}$, $(c_x, c_y) \in \mathbb{N}^2$ and $\mathbf{Q}$ is the extrinsic camera matrix with parameters r_{ij}; t_i for $i = 1, 2, 3; j = 1, 2, 3$ [41].

A unique solution for the reverse transformation would seem impossible since there are more equations than variables to map points from the image plane to the world plane. At most, an image point would represent a ray in real-world coordinates, as we lose depth information when the image is captured. However, the transformation is assumed to take place on a planar region of interest, and hence, a simplification where $Z = 0$ can be performed. Furthermore, Equation 2.14 can be rewritten as follows:

$$s\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \mathbf{H}\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{2.15}$$

$$s\boldsymbol{p'} = \mathbf{H}\boldsymbol{p}. \tag{2.16}$$

Now, the homography matrix $\mathbf{H}$ can be solved using Direct Linear Transformation (DLT) [42], by rearranging Equation 2.15:

$$sx' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}},$$

$$sy' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}},$$

then, setting $h_{33} = 1$ to enforce 8 degrees of freedom:

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1},$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1},$$

$$(h_{31}x + h_{32}y + 1)x' = h_{11}x + h_{12}y + h_{13},$$
$$(h_{31}x + h_{32}y + 1)y' = h_{21}x + h_{22}y + h_{23}.$$

Thus:

$$x' = h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx', \tag{2.17}$$
$$y' = h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy'. \tag{2.18}$$

From equations 2.17 and 2.18, it is clear that to find all parameters in $\mathbf{H}$, a minimum of four points on both planes are needed. Moreover:

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & x_4y_4' & -y_4y_4' \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_nx_n' & -y_nx_n' \\ 0 & 0 & 0 & x_n & y_n & 1 & x_ny_n' & -y_ny_n' \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \\ \vdots \\ x_n' \\ y_n' \end{pmatrix}. \tag{2.19}$$

The previous Equation 2.19 has the structure of a system:

$$\mathbf{A}\boldsymbol{h} = \boldsymbol{b},$$

then by solving $\boldsymbol{h}$, using the known point correspondences, all the parameters of $\mathbf{H}$ can be found.

2.3 Object Detection

Object detection refers to the computer vision technique that focuses on locating all objects of interest on an image frame. Those objects have semantic importance in the scene, such as people, buildings, vehicles, and traffic signs in urban areas. Currently, this technology already offers a wide range of applications, including video surveillance in static or moving cameras, face detection and recognition, commercial logo recognition, autonomous vehicles, and so forth. An object detector may also be a precursor to tracking by detection systems, becoming a two-phase technique that recognizes objects and also records their trajectories.

2.3.1 Object Detection Techniques

This computer vision task has been an active research field for many years before the widespread adoption of deep convolutional neural networks, using traditional features that leveraged the unique shape or color of an object, and positioning on the image. Currently, this workflow has been replaced by deep learning, which is more robust to changes in image quality due to varying acquisition hardware, external conditions, etc. The most popular object detection techniques are summarized in Table 2.4.

Table 2.4: Brief overview of common object detection techniques.

Methodology	**Description**
Viola-Jones Detection Framework	A framework that uses Haar cascade features computed from intermediate representations of images, called *"Integral Images"*. These representations contain the sum of the pixels above and to the left of a coordinate, for each coordinate. After the features are retrieved, an Adaboost ensemble model is trained [43].
Scale Invariant Feature Transform	This method requires a prior database of keypoints extracted from reference images. Then, an object is recognized on a new image by finding features that match the original object in the database using euclidean distances of the feature vectors [44].
Region Proposal and Classification	A two-stage methodology that uses the family of R-CNN deep learning models that first proposes regions for possible objects, then classification is performed [45].
End-to-End Deep Object Detection	This methodology is performed using deep learning models like You Only Look Once (YOLO) [6] that divide an image into a grid, then simultaneously estimates the bounding box of each object along with its confidence and object class.

2.3.2 Object Detection Challenges

An object detection model should be robust enough to detect targets in unconstrained scenarios. Some of the problems that can affect the quality of images or target visibility, and therefore affect the performance of the model are depicted in Table 2.5.

2.3.3 Object Detection Performance Metrics

Once the detection model is fitted, a test must be carried on unseen data, to determine performance. One of the most common performance metrics used in object detection is average precision (AP) [46], and to obtain this value, one must first perform a series of steps.

First, correct and incorrect detections should be defined, and intersection over union (IoU) metric, defined in Definition 8, is a common way to establish boundaries for such task.

Definition 8. *let* $\boldsymbol{b_p}, \boldsymbol{b_g} \in \mathbb{N}^4$ *be the prediction bounding box, and ground truth bounding box, respectively. The intersection over union (IoU) metric is given by:*

$$IOU(\boldsymbol{b_p}, \boldsymbol{b_g}) = \frac{area(\boldsymbol{b_p} \cap \boldsymbol{b_g})}{area(\boldsymbol{b_p} \cup \boldsymbol{b_g})}. \tag{2.20}$$

Table 2.5: Challenges in object detection. Source [2].

Challenge	**Description**
Changes in illumination	Shadows, absence or excess of light.
Background clutter	Background near target is similar in color or texture as the target.
Target occlusion	When target visibility is blocked by another object.
Low resolution	Pixel quantity inside target bounding box is low.
Scale variation	Caused by perspective on a scene.
Change in target position	Target may be rotated, deformed or moved.
Fast motion	Motion causes noise on image sequences.
Environment changes	Weather conditions like rain, can affect the image quality.
Objects agglomeration	Multiple objects on close area.
Hardware changes	Image quality variability due to capturing device.

By giving an IoU threshold, a detection can be classified as being correct or not. Then, performance is assessed in terms of precision and recall [47], respectively described in definitions 9 and 10.

Definition 9. *The precision value p is defined as:*

$$p = \frac{|TP|}{|TP| + |FP|}, \tag{2.21}$$

where TP and FP are the sets of true positive and false positive detections.

Definition 10. *The recall value r is defined as:*

$$r = \frac{|TP|}{|TP| + |FN|}, \tag{2.22}$$

where TP and FN are the sets of true positives and false negatives detections.

Moreover, a precision-recall curve can be generated using precision and recall values for different confidence scores associated with the predicted bounding boxes. Ideally, the precision of an object detector should stay high as recall increases, meaning that as false negatives decrease, false positives remain low as well. Finally, the average precision is defined as the area under the curve of such graph as seen in Definition 11.

Definition 11. *Given a precision-recall curve. The average precision (AP) is defined as:*

$$AP = \int_0^1 p(r)dr. \tag{2.23}$$

For practical purposes on computing average precision, n evenly spaced recall values are taken, instead of the whole domain, then the interpolated precision values are averaged:

$$AP_n = \frac{1}{n} \sum_{r \in R} p_{interp}(r). \tag{2.24}$$

In Equation 2.24, R is the set of n recall values evenly spaced and p_{interp} is given by:

$$p_{interp}(r) = \max_{\hat{r}:\hat{r} \geq r} p(\hat{r}).$$

Finally, for a multi-class object detection, AP values are retrieved for each class. These values are then averaged to compute the mean average precision (mAP), as stated in Definition 12.

Definition 12. *Let $i \in \mathbb{N}$ be the object class index. The mean average precision (mAP) formula is given by:*

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i, \tag{2.25}$$

where N is the number of object classes and AP_i is the average precision value for class index i.

2.4 Object Tracking

Object tracking is a step for an image processing workflow that aims at using multiple objects detections across image sequences, capturing individual features like appearance, position or motion to assign an ID to such objects. Currently, it is an active research field in computer vision, with numerous applications in augmented reality, traffic detection, vehicle navigation, autonomous driving, object identification, and so forth.

2.4.1 Tracking Techniques

An image sequence is rich in information for a tracking method to identify multiple objects. For instance, the spatial dimension provides features like color or shape, and temporal information can unveil features from motion. Some of the various approximations in object tracking are explained in Table 2.6.

Table 2.6: Tracking methodologies currently used.

Methodology	**Description**
Shape Based Representation	Edges and silhouettes from objects are used for tracking objects. Histogram of oriented gradients (HOG) is a useful tool in this method.
Appearance Based Representation	Color cues from objects are widely used in object tracking. Histograms from RGB, HSV, and other color spaces are employed as appearance descriptors.
Motion Based Tracking	Useful for articulating, or moving objects. Optical flow is employed on non-rigid objects, while residual flow is used on the rigid counterparts.
Tracking by Detection	Performing multi object detection on every frame of a sequence and trajectories are estimated across frames. Currently, deep learning models play a major role on object detection.
Composite Method	Robust tracking methods that employ a combination of some, or all the methods explained. Weights can be assigned to features to limit their contribution.

2.4.2 Tracking Challenges

Tracking is an active area of research, since real-world applications, demand methods that are robust enough to maintain high performance in unconstrained environments. For instance, autonomous vehicles and sensible military systems that depend on these technologies could cause major incidents, if all possible sources of error are not taken into account. Some of the challenges are explained in Table 2.7, extended from the work of Islam, Md. *et al.* [48].

2.4.3 Tracking Results Evaluation

A tracking algorithm should find all objects at all timestamps, and keep a record of their trajectories. Additionally, every object has to be assigned a unique ID that remains constant throughout its appearance duration on the video. According to Bernardin and Rainer [49], it is expected that to measure the performance of a tracker a metric should:

Table 2.7: Challenges in object tracking.

Challenge	**Description**	**Example**
Hijacking Problem	If two objects that share similar features used for a certain tracking methodology come close enough, the tracker of one object may wrongly associate to the other, leaving the original untracked.	
Centralization Problem	Occurs when two objects are in almost the same position, or if one is occluding the other. Furthermore, the tracker misses to identify each object individually	
Drifting Problem	For tracking methodologies that estimate future object positions, a sudden change in direction can difficult the correct estimation of trajectory.	

- Have as few adjustable parameters as possible to keep evaluations straightforward.
- Be clear, easily understandable, and behave according to human intuition.
- Be as general as possible to allow comparisons across many kinds of tracking methodologies. Some examples were explained in Table 2.6.
- Be practical for large evaluations and yet complete and expressive in meaning.

Based on these considerations, the two most common performance metrics are explained in definitions 13 and 14.

Definition 13. *Let $i, t \in \mathbb{N}$ be the target and frame index. The multiple object tracking precision (MOTP) [49] is defined as:*

$$P_{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t}, \tag{2.26}$$

where c_t denotes the number of matches at frame t, and d_t^i is the bounding box overlap of target i with its assigned ground truth object.

Definition 14. *Let $t \in \mathbb{N}$ be the frame index. Then the multiple object tracking accuracy (MOTA) [49] is defined by:*

$$P_{MOTA} = 1 - \frac{\sum_t (f_{n_t} + f_{p_t} + m_t)}{\sum_t g_t}, \tag{2.27}$$

where g_t is the number of ground truth objects at index t and $f_{n_t} + f_{p_t} + m_t$ are the number of false negatives, false positives, and misses at index t, respectively.

There are other qualitative markers for the performance of tracking algorithms. Each target trajectory can be labeled as mostly tracked (MT), mostly lost (ML) or partially tracked (PT). A target is said to be mostly tracked if its trajectory is successfully recovered for at least 80%. On the other hand, it is said to be mostly missed, if it is only tracked 20% of its true trajectory; otherwise, it is partially tracked. Moreover, it is desirable to maximize MT while minimizing ML for all targets [50].

Chapter 3

State of the Art

3.1 Deep Convolutional Neural Networks

The field of Computer Vision is constantly being revolutionized by novel convolutional neural network architectures, maximizing feature map quality and meaning, while making the networks more efficient. These networks are used in image classification by conecting a dense layer with softmax activation, and are also very important in object detection, since convolutional networks are the backbone of those models. They are the main feature extractors that will provide the detection heads with meaningful features to recognize objects on the image.

One example of such models is the work of Tan and Le [3], introducing a family of convolutional nets called EfficientNets. Based on the premise that scaling neural networks often increases performance, they developed a method that uniformly scales network depth d, width w and resolution r, as seen in Figure 3.1, using a compound coefficient ϕ. First, the authors defined a baseline architecture using neural architecture search that optimizes accuracy and FLOPS. This baseline is called EfficientNet-B0. Then, this model is scaled up using a two step framework: Considering $d = \alpha^\phi, w = \beta^\phi, r = \gamma^\phi$, fix $\phi = 1$ and do a grid search to find α, β and γ constants. Furthermore, with these known constants, the scaling is performed with varying values of ϕ to obtain EfficientNet-B1 to B7. The results are a significant improvement in image classification with imagenet dataset when compared to previous leading architectures, obtaining a top-1 accuracy of 84.3% and top-5 accuracy of 97% for Efficientnet-B7.

Other authors question the status-quo of current works looking for better ways to increase model performance. Such is the case of Brock *et al.* introducing another family of models called NFNets [20] that stands for Normalizer Free ResNets. The main problem is that recent computer vision models rely on batch normalization for better training performance, but batch normalization itself posses three disadvantages: it increases time required to evaluate gradient in some networks, introduces discrepancy between the behavior of the model during training and inference time and most importantly, batch normalization breaks independence between training examples on the minibatch. Furthermore, that work proposes adaptive gradient clipping, which clips gradients based on unit-wise ratio of gradient norms to parameter norms, enabling larger batch sizes and stronger data augmentations. As for the architecture, the authors started using SE-ResNeXt-D [51] as

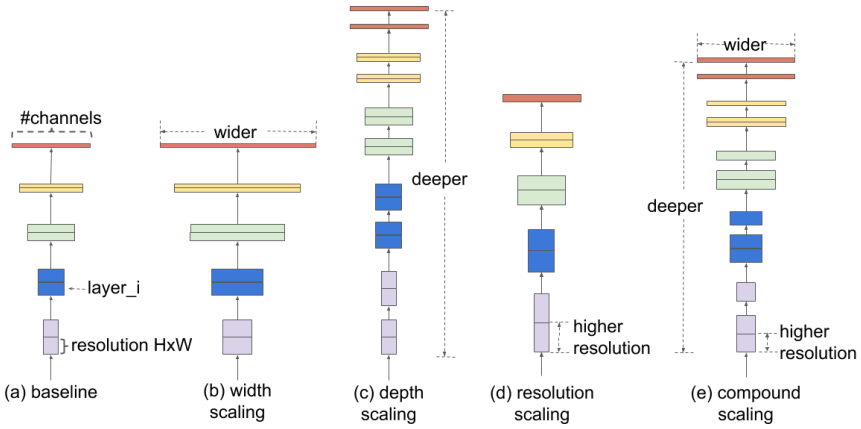


Figure 3.1: Methods for scaling neural networks. Extracted from [3]

base model, changing how many bottleneck blocks to allocate on each network stage, i.e residual blocks whose activations are the same width and resolution. Figure 3.2 shows a comparison between normal residual blocks, and the modified blocks used in NFNets. Moreover, the results obtained are even better than that of the EfficientNets, with top-1 accuracy of 86% and top-5 accuracy of 97.9% for the NFNet-F5, which is the biggest model, while maintaining similar training speeds.

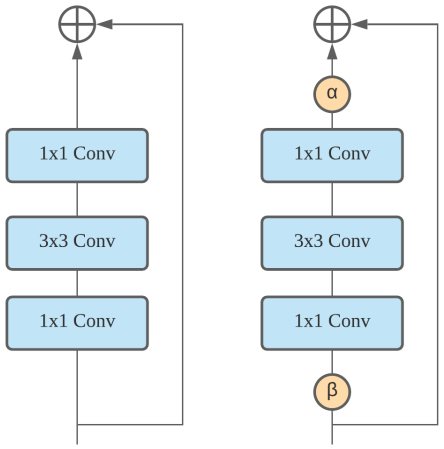


Figure 3.2: Normal ResNet block (a), and modified block used in NFNets (b).

There are other works that instead of advancing performance by creating new architectures, they present novel learning techniques to increase existing models' performances. Pham, H. et al. present a semi-supervised learning method called meta pseudo labels [23] that improves on self-training. The starting concept is that there exists two networks: one that acts as a teacher and the other as a student. The pre-trained teacher generates pseudo-labels on unlabeled images and are combined with labeled images to train the student. The objective is that the student becomes better than the teacher network. However, if pseudo-labels are inaccurate, this objective might not be reached. Moreover, this work

utilizes the feedback signals from the student to inform the teacher to generate better pseudo labels. These signals are the performance of the student on the labeled dataset. As for the results, using a network called EfficientNet-L2 with meta pseudo labels scheme, a top-1 accuracy of 90.2% and a top-5 accuracy of 98.8% were obtained on imagenet dataset for image classification, and currently holds the first place on the leaderboard.

It should be noticed that these improvements, although reported increases in performance on image classification tasks, they are not constrained on this domain. Since improving network performance leads to models being able to learn better features from images, these models can also be used as backbones for others tasks like image segmentation or object detection, which is highly beneficial for computer vision as a whole.

3.2 Object Detection

Object detection task on images taken from unconstrained environments is hard. The COCO dataset object detection benchmark has not even reached 60% on Box Average Precision. However, these models are still practical for use and are constantly improving.

The first major improvement that set object detection apart from traditional machine learning methods came from the work of Girshick *et al.* with R-CNN [4], which is a three module system, depicted in Figure 3.3. The first module consist in region proposal to determine objects locations on the image. The second module involves a pre-trained CNN to compute a fixed length feature vector from each proposed region. Finally, n support vector machines, one for each object class, takes these vectors and determines the object class. R-CNN reported mean average precision of 53.3%, a performance value 30% higher that the previous best result on PASCAL VOC dataset [52].

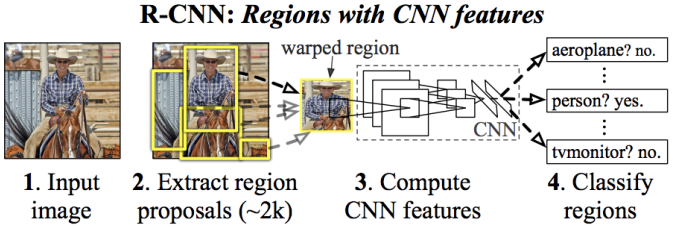


Figure 3.3: RCNN worflow. Source [4].

R-CNN poses many problems, since it takes a large amount of time to train the feature extractor CNN, making the model unsuitable for real time inference, and most importantly, the region proposal method is not a trainable model, but rather a fixed algorithm. This in turn may lead to bad object candidates. From these observations, fast R-CNN [5] was created by the same author as the original R-CNN. The difference is that instead of feeding patches of the image corresponding to the object proposal region, the entire image is sent to the network along with the proposals to compute feature maps that are then converted to fixed length vectors by means of a RoI pooling layer. The improved workflow is shown in Figure 3.4. This approach results in dramatic drops in training and inference time and higher detection quality with 65.7% mAP for PASCAL VOC 2012 dataset.

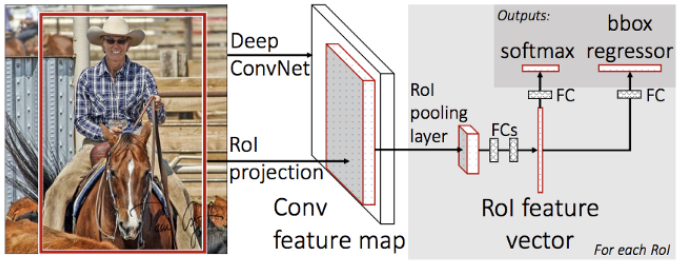


Figure 3.4: Improved Fast-RCNN workflow. Source [5]

Even after these improvements, these models were still composed of various stages before getting detections and were not suitable for real time video processing. For this reason, the work of Redmon *et al.*, with You Only Look Once (YOLO) [6] object detection framework, was a major change on the field. It proposed an end-to-end convolutional network that receives images and outputs bounding boxes, object classes and confidences for each detection on a single stage. This is done by splitting the image into an $S \times S$ grid. If an object falls into a grid cell, then that cell is responsible for detecting that object. The backbone for feature map extraction was inspired from GoogLeNet model for image classification [53], but instead of inception modules, a 1×1 reduction layers, followed by 3×3 convolutional layers were employed for a total of 24 convolutional layers. Moreover the final predictions are encoded as a $S \times S \times (B * 5 + C)$ tensor, where B is the maximum number of objects a grid cell can detect and C are the number of object classes. This procedure is depicted in Figure 3.5. Furthermore, the result is an object detector capable of processing 45 frames per second, instead of 0.5 fps achieved by fast R-CNN and an mAP value of 57.9% on PASCAL VOC 2012.

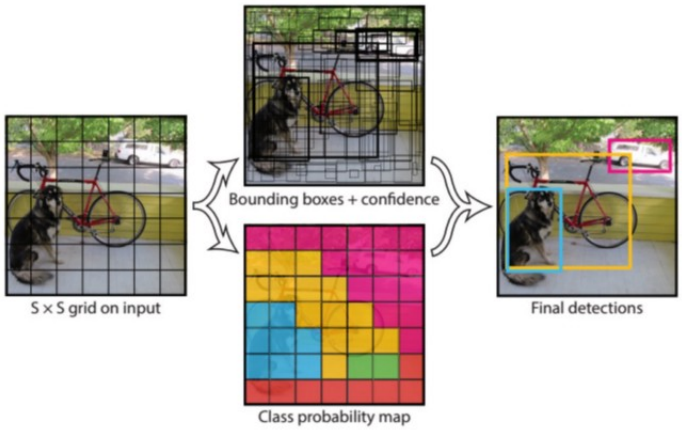


Figure 3.5: YOLO framework. Source [6].

The same author presented improved versions of YOLO to tackle the limitations of the first version. That is: the small number of objects that can be detected on each grid cell, difficulty to detect small objects and low recall. The second version, YOLOv2 [54] introduced batch normalization, higher image resolution and convolutions with anchor

boxes, instead of fully connected layers trying to predict every bounding box from scratch. The third iteration, YOLOv3 [14] introduced 53 convolutional layers instead of 19, residual blocks and up-sampling layers, making the model slower than previous version, but much more robust in terms of capability to detect small objects and the number of objects that can be detected per grid cell.

Further work on this model was carried out by different authors. Bochkovskiy *et al.* created YOLOv4 [55], an object detection model composed of a backbone convolutional net called CSPDarknet53, followed by a spatial pyramid pooling block [15] and finally the head of YOLOv3, composed of 53 additional convolutional layers. The backbone is derived from the same Darknet-53 network used on the previous YOLO version, but introducing the concept of a cross stage partial network (CSPNet) [56], which enhances the learning process by increasing gradient paths. Experiments show that this model is 10% better in terms of average precision and 12% faster than previous version in MS COCO object detection benchmark.

Finally, there are other works that diverge completely from aforementioned architectures, introducing novel frameworks for object detection. For instance, Liu *et al.* introduces Swin transformer [7]. This model, observed in Figure 3.6, takes the concepts of transformers used in natural language processing and adapts it to computer vision. Transformers are neural architectures that implement attention mechanisms to model dependencies in data. In addition, Swin Transformer constructs hierarchical feature maps, starting from small image patches and gradually merging neighboring patches on deeper layers. This model scales linearly against input due to the self-attention mechanism, which makes it very efficient for object detection. As for the performance, the variant Swin-L, the biggest model, achieved 58.0 mAP on COCO dataset.

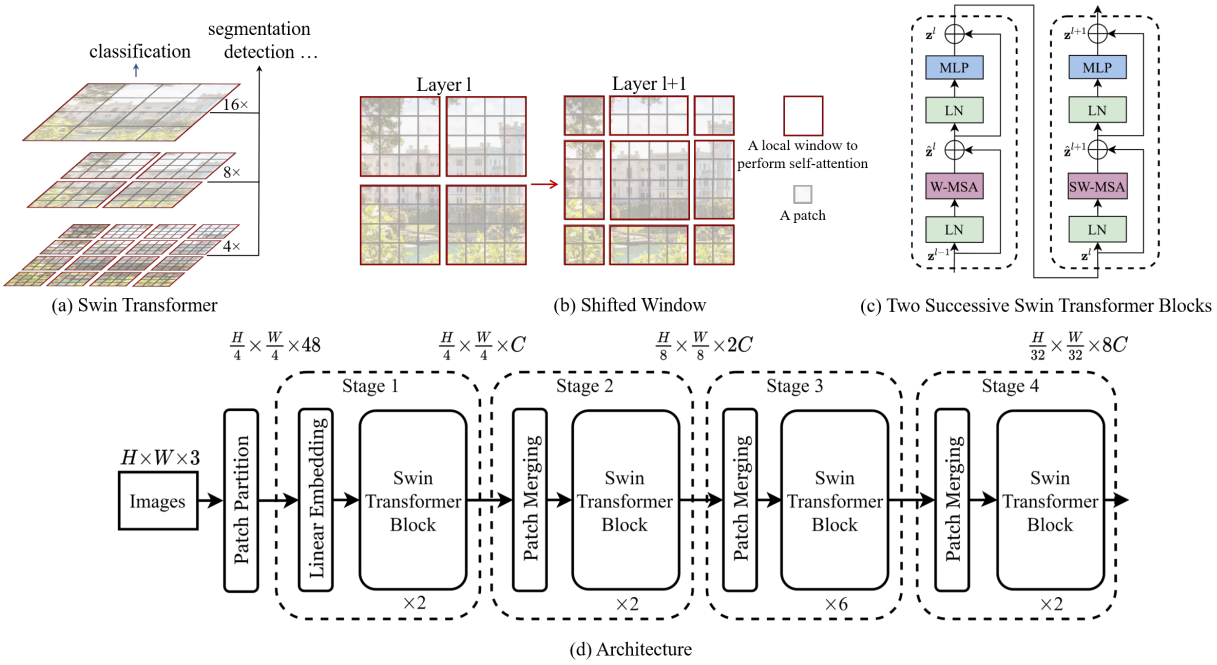


Figure 3.6: Swin Transformer architecture. Source [7].

3.3 Object Tracking

Currently, tracking by detection is one of the most common computer vision tasks since object detection models are increasingly improving in performance and when utilized with a tracker, they allow object identification across an image sequence. There are simple, yet sophisticated ways to track objects. For instance, Bochinski *et al.* created an object tracker based on intersection over union [8]. The algorithm receives detections, frame by frame, and starts by assigning each detection of the initial frame to different object tracks. Furthermore, for newer frames, each detection is compared against most recent bounding box location for each track and assigned to the one with maximum IoU value, as seen in Figure 3.7. Using this metric, assignment thresholds are made possible so that wrongly associations are minimized. This method achieved 16% MOTA and 38.3% MOTP values when combined with R-CNN object detector for UA-DETRAC dataset.

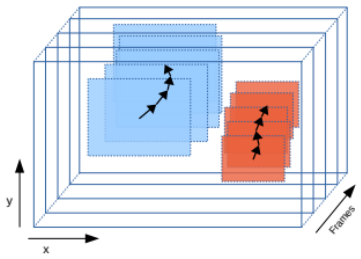


Figure 3.7: Tracking principle from IoU tracker. Source [8].

The same authors extended IoU tracking by adding visual information [9]. This improved version of the algorithm uses the same original mechanism, but with an additional visual tracking system in the event of missing objects detections for a pre-defined maximum number of frames. This principle is depicted in Figure 3.8. Moreover, the objective is to avoid trajectory fragmentation and to reduce the number of ID switches. As a result, this method obtained an MOTA of 30.7%% and MOTP of 37.0% and is the best open-source tracking system on UA-DETRAC leaderboard. In general, this model retrieved more relevant matches than the previous method, but it is only suited for offline tracking since the visual component is initialized on the last known position and then goes backwards in the sequence.

There are works that differ from the concept of designing a tracking algorithm that works on detections from a deep learning object detector. Instead, they intend to develop a single stage method, reducing design complexity and the number of hyperparameters that need to be tuned. For instance, Chu and Ling created FAMNet, a neural network that performs feature extraction, object affinity estimation, and multi-dimensional assignment [10]. The architecture, as seen in Figure 3.9, consists in various specialized modules. The first module is a convolutional sub-network that fuses discriminative appearance and motion information to generate affinity tensors for hypothesized object trajectories. The second one is a R1TA power iteration layer that uses the aforementioned affinity tensors to estimate a set of optimal assignments. Finally, the output of this layer are passed trough an ℓ_1 normalization layer to satisfy data constraints. The reported performance on KITTI-car benchmark, was 77.1% MOTA and 78.8% MOTP, while on UA-DETRAC those values were 19.8% and 36.7% for MOTA and MOTP, respectively.

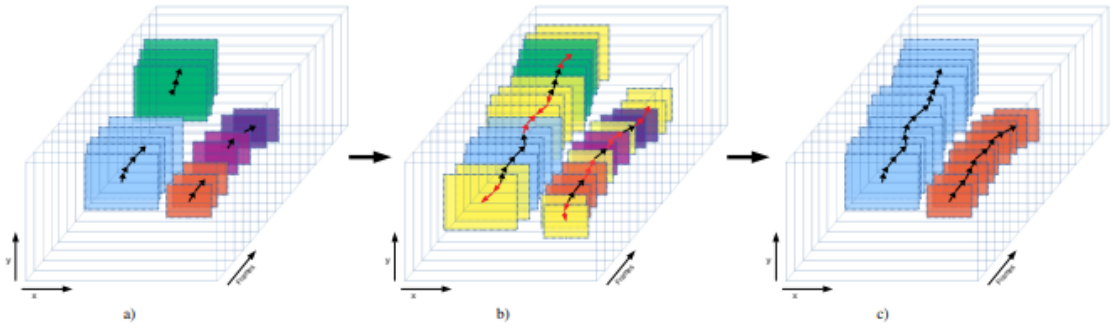


Figure 3.8: Tracking principle from IoU tracker (a) with auxiliar visual system (b) for track defragmentation (c). Source [9].

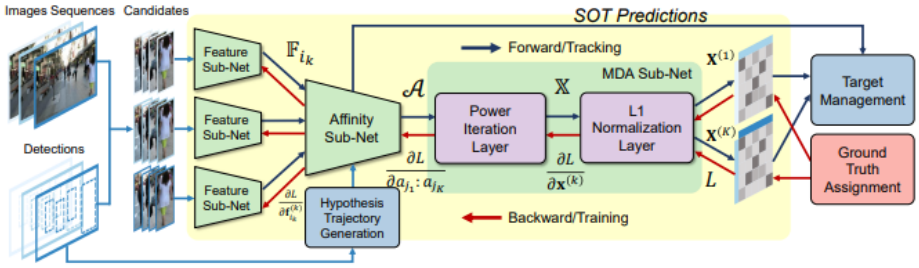


Figure 3.9: FAMNet architecture. Source [10].

Sun *et al.* also developed an end-to-end style system called Deep Affinity Network (DAN) that is capable of associating objects that may not be detected on consecutive frames [11]. This model learns compact, yet discriminant features of pre-detected objects at several layers of abstraction and performs exhaustive pairing permutations of those features in any two frames to predict object affinities. The architecture is depicted in Figure 3.10 and follows a modified version of VGG16 CNN [57], replacing its fully connected layers by convolution layers to better encode spatial features. Furthermore, this network is replicated in a siamese configuration, sharing its weights to receive two different frames at the same time and computing a single concatenated feature tensor. This tensor is then transformed by a compression network composed of five convolutional layers with 1×1 kernels. This approached achieved 53.42% MOTA and 76.90% MOTP for MOT17 tracking dataset. Moreover, for UA-DETRAC benchmark, the model reaches 20.2% MOTA and 26.3% MOTP.

3.4 Homography Matrix Computation

Homography lets us define transformations between two planes given a minimum of four sets of points. The simplest method is to define the system and perform Direct Linear Transform to get the matrix parameters. However, given the choice of points, noise can be introduced into the solution, resulting in inaccurate homography matrices.

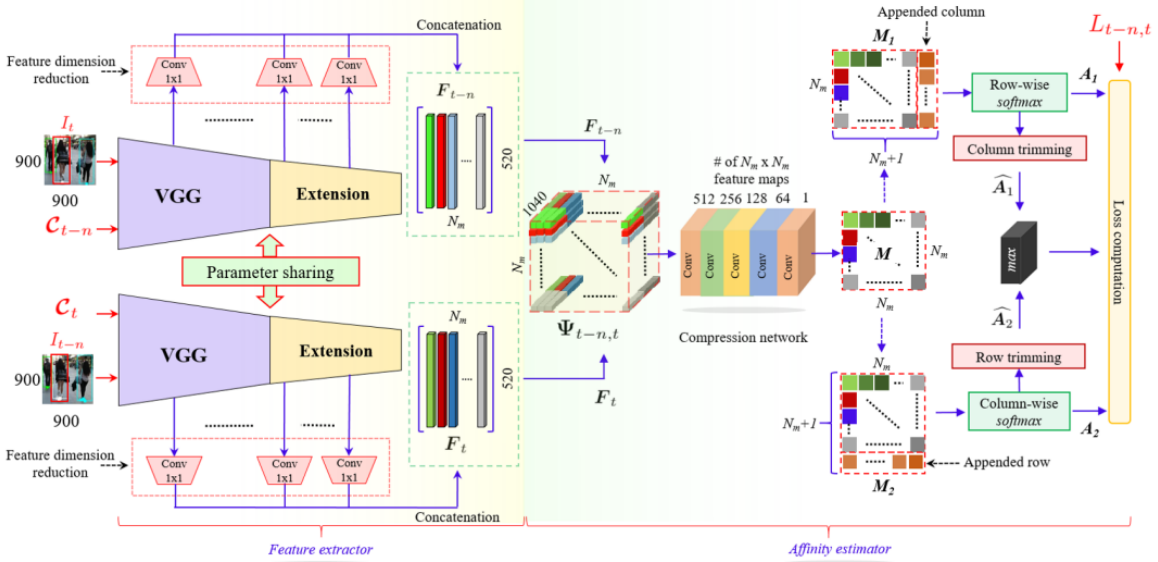


Figure 3.10: Deep Afinity Network (DAN) architecture. Source [11].

There are robust methods that account for this issue, designed as optimization problems where the goal is to minimize the re-projection error from transforming a set of points on one plane into its representation on the other plane.

A survey on planar homography estimation techniques [58] presents some of those methods that, although fall into traditional machine learning worflows, they are still widely used by popular computer vision frameworks like OpenCV [59]. For instance, robust estimation can be achieved using RANSAC [60]. This iterative method finds those points that do not fit into the model, and are labeled as outliers to a gaussian distribution that explains the error on the measurements. Then, if there are points in between planes that are mismatched, the model will discard them. Another robust method is least-squares optimization [61], widely used of its practicality and computational simplicity. Ideally, given correspondences of points in two different planes $(x_i, x'_i), i = 1, 2, ..., n$, the objective is to minimize $Ah = 0$, obtained by converting Equation 2.19 into a homogeneous system. Furthermore vector h contains all entries of the homography matrix H.

Now that deep learning is advancing almost all aspects of computer vision, it is natural for researchers to develop new architectures to solve the problem of finding a homography matrix for a scene with two different perspective views. The work of DeTone *et al.* defines a supervised deep neural network that estimates the relative homography between a pair of images [12]. This model, as depicted in Figure 3.11, is composed of 8 convolutional layers with max pooling between each pair and two fully connected layers, with the last acting as regressor for the homography matrix. This approach achieved a mean average corner error of 9.2 pixels on warped MS-COCO 14 test set.

These novel methods for estimating homography that depart from traditional methods, also poses new problems. Since deep learning is characterized for requiring large volumes of data to be able to generalize its knowledge, the process of compiling such datasets become very difficult. However, in the work of Nguyen *et al.* an unsupervised deep homography estimation is proposed [13]. Inspired by the aforementioned work, it starts with the same architecture, but includes additional layers enabling its unsupervised nature. Specifically,

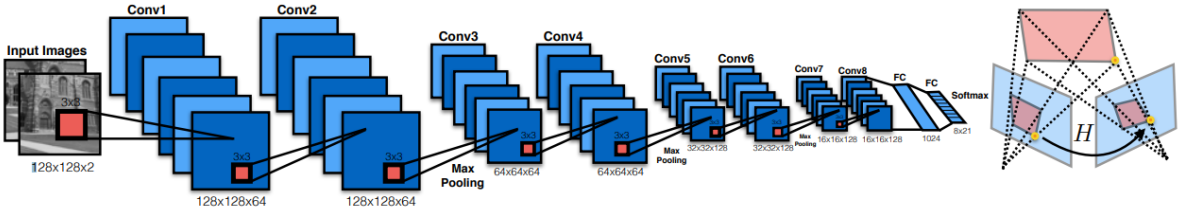


Figure 3.11: Architecture of the supervised deep learning homography estimation model. Source [12].

it includes a tensor Direct Linear Transform (DLT) layer, that maps a fully holography parameterization of 4×4, into a 3×3 version. Then, a spatial transformation layer applies the output from the tensor DLT layer to the pixel coordinates to get warped coordinates. This architecture can be seen in Figure 3.12. Moreover, the results are 15.6 Root Mean Squared Error on aerial UAV images.

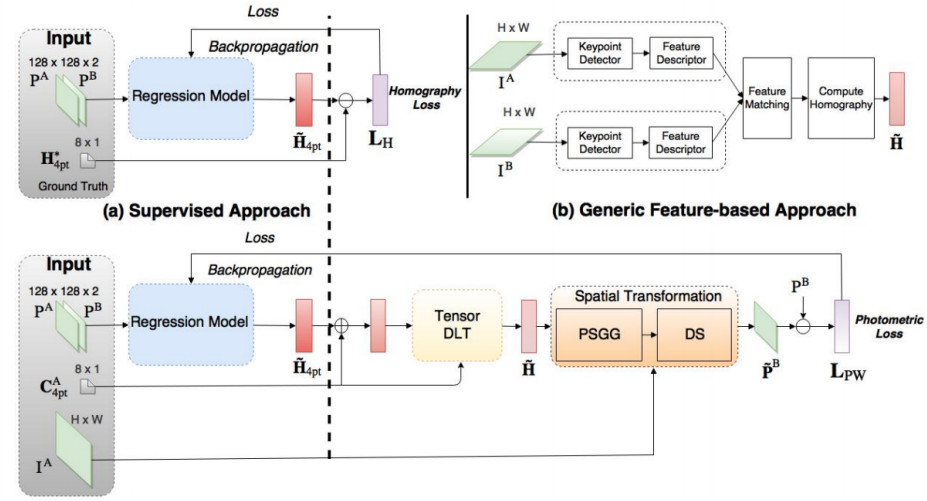


Figure 3.12: Architecture of the unsupervised deep learning homography estimation model. Source [13].

Chapter 4

Methodology

4.1 Overview of the Proposed Solution

With the increasingly fast development of the Internet of Things (IoT), artificial intelligence, and big data, large volumes of information can now be captured in connected cities and processed to solve many problems in ways that were not possible a few years ago. It is common for big cities to have a network of cameras installed in strategic places for surveillance. Some of those places include street intersections, above traffic lights, on freeways, or simply placed at the side of a street pointing to vehicles. Moreover, camera feeds can prove to be valuable sources of information that are still not being properly exploited. Using A.I software for object detection, moving vehicles on the road can be tracked and analyzed. Furthermore, a mathematical model can be used to transform pixel coordinates of the scene into real longitude and latitude measurements. As a result, vehicle speeds can be measured by means of a single camera and a computing machine. Therefore, the departments of transportation on cities that implement this framework could make speed limit enforcement widespread, where surveillance cameras are present, instead of limiting the application to certain areas using lidars, reducing the probability of traffic accidents. In addition, from this video processing scheme, entities in charge of traffic management can make informed decisions based on traffic data to alleviate transportation problems related to traffic congestion. For instance, based on the count of vehicles on streets that meet on an intersection, intelligent traffic lights can be implemented to favor the street that have significantly higher number of cars, while reducing the time where green light is on for the street with fewer traffic. The same system can also use the average speed of the vehicles on a certain street, to indicate whether taking that street is optimal for travel time, or to better take alternate routes, using public displays.

4.2 Algorithm Design

This method has three critical components: homography matrix estimation, object detection, and object tracking. Figure 4.1 depicts the system workflow with the aforementioned components, along with all its inputs and outputs, which are going to be explained in detail in the following subsections.

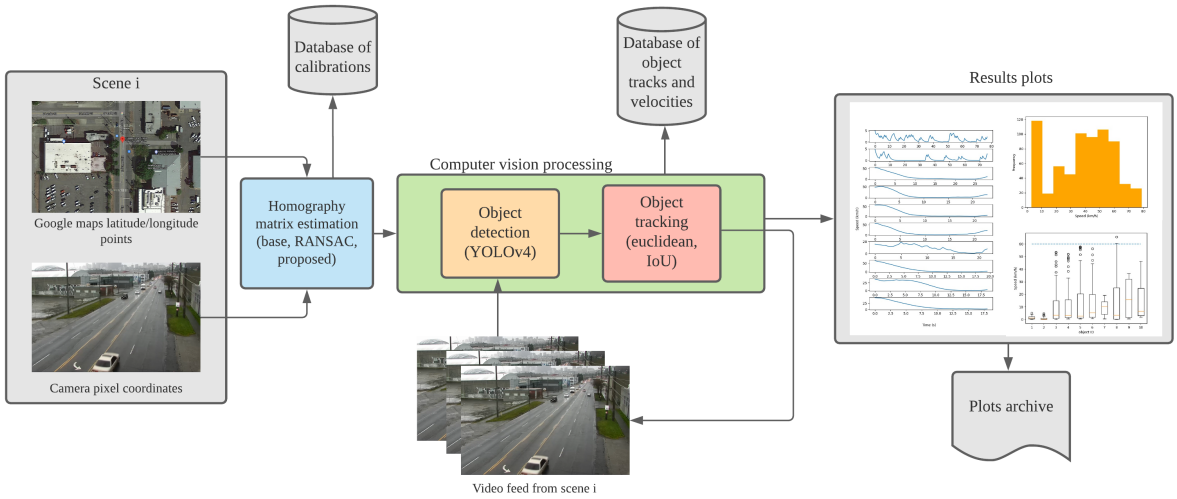


Figure 4.1: Proposed speed estimation workflow.

4.2.1 Homography Estimation for Camera Calibration

The homography estimation is performed to obtain a camera calibration matrix assuming the region of interest lies on a flat road, i.e $Z = 0$. Given a scene i, a set of n points are manually identified both from the pixel coordinates from a reference image of the scene, and from latitude/longitude coordinates taken from Google Maps. Moreover, each set of points between planes should represent the exact same spot on the scene. Furthermore, a system

$$\mathbf{A}\boldsymbol{h} = \boldsymbol{b},$$

is built according to Equation 2.19. This system can be solved using Direct Linear Transformation to find the parameters of $\boldsymbol{h}$, and ultimately derive in homography matrix $\mathbf{H}$. However, in practice, the point correspondences between image and latitude-longitude plane are prone to error since a human operator selects those points manually. Furthermore, a robust method must be utilized to minimize inaccuracies on $\mathbf{H}$. In this work, three methods are evaluated for such purpose. A base homography extraction with no robust method for error minimization, a RANSAC optimized method, and a new technique to find the homography matrix, proposed in this work and explained as follows. The reason for selecting DLT and its RANSAC version for comparison with the proposed calibration, is that these two algorithms are widely known traditional methods implemented in OpenCV [59] and will serve as baselines.

Given the initial manual annotations, the proposed calibration method consists of an iterative localized search for the pixel coordinates that best relates to its longitude-latitude coordinate, guided by the projection error:

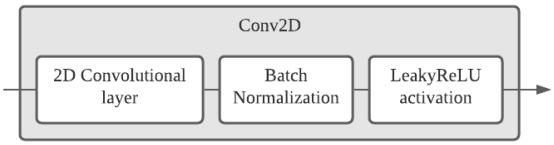
$$\epsilon = \frac{1}{n}\sum_{i=1}^{n} ||p_i - \hat{p}_i||_2,$$

where p_i is a point selected by a human operator on the image plane, and $\hat{p}_i$ is the estimated point given a longitude-latitude coordinate p_w. The search is made by an evolutionary algorithm called Estimation of Density. It works by establishing an initial population N of

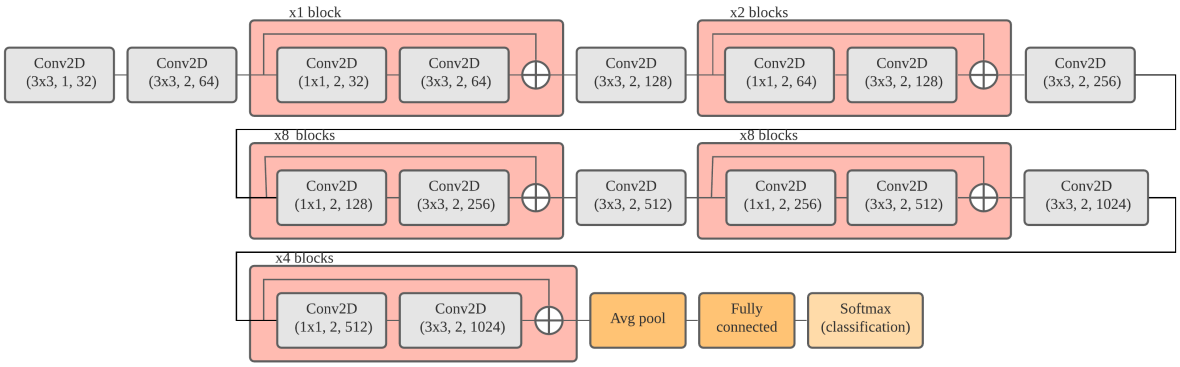
point correspondences with a pre-defined range where a random variation of the original points can be generated. Then, generation by generation, K individuals are selected with the least projection error. Each generation, the mean and variance of the survivors are calculated to generate offspring and the process repeats until error convergence. This process is depicted in Algorithm 1. Note that, although all points from both planes are manually selected, it is enough to correct the annotations on the longitude-latitude coordinates as the points on the image plane serve as ground truth.

4.2.2 Object Detection with YOLOv4

To be able to detect vehicles on the scene, YOLOv4 detection model was utilized. It is a deep convolutional neural network, widely known for its performance in real-time scenarios when compared to other detection models [55], that consists of three main components: backbone, neck, and head. First, a network called CSPDarknet53 is embedded into YOLO as its feature extractor backbone. Based on its original version Darknet-53, it contains 53 convolutional layers with additional residual connections in between, an average pooling layer to convert feature maps into a single vector, a fully connected layer, and a softmax layer for classification. However, these final layers that use vectors are removed before being connected to the other components of the detector, since it is going to be used as a feature extractor for the images and not as a classifier. The architecture of Darknet can be visualized in Figure 4.2.



(a) Conv2D specification.



(b) Darknet-53 layers with Conv2D following (filter size, stride, filters).

Figure 4.2: Darknet-53 architecture, as seen in YOLOv3 [14].

In addition to the previous architecture, CSPDarknet53 implements the Cross Stage Partial Network (CSPNet) [56] concept depicted in Figure 4.3.

Algorithm 1: Proposed homography estimation procedure.

Input : Initial set of image points X' and world points X; Individuals created for each generation N; Individuals to survive generation K; lower boundary $l = (l_x, l_y)$, and upper boundary $h = (h_x, h_y)$ of initial search range; convergence threshold τ

Output: Homography matrix with minimum average back-projection error H

Start
$\bar{\epsilon}_{previous} = \infty$
WorldPointSets = Emptylist()
for *i=0 to N, step=+1* **do**
    $\hat{X}$ = EmptyList()
    for x *in* X **do**
        $\hat{x} = UniformRandomSample(x, l, u)$
        $\hat{X} = \hat{X} \cup \hat{x}$
    $WorldPointSets = WorldPointSets \cup \hat{X}$
while *not converge* **do**
    for $\hat{X}$ *in WorldPointSets* **do**
        $\hat{H}$ = EstimateHomography($\hat{X}$, X')
        $\hat{X}'$ = EmptyList()
        Errors = EmptyList()
        for $\hat{x}$ *in* $\hat{X}$ **do**
            $\hat{x'} = \hat{H}\hat{x}$
            $\hat{X}' = \hat{X}' \cup \hat{x'}$
        ϵ = calcProjectionError($\hat{X}'$, X')
        $Errors = Errors \cup \epsilon$
        Associate ϵ with the corresponding $\hat{X}$ that produced $\hat{X}'$
    $\bar{\epsilon}_{current}$ = Mean(Errors)
    std_ϵ = STD(Errors)
    if $\bar{\epsilon}_{previous} \times \tau > |\bar{\epsilon}_{current} - \bar{\epsilon}_{previous}|$ **then**
        converge
    sort WorldPointSets according to their associated ϵ
    keep first K world points $\hat{x}$ in WorldPointSets

The partial transition layer from Figure 4.3 is designed to maximize the difference of gradient combination by using a hierarchical feature fusion mechanism that truncates the gradient flow to prevent distinct layers from learning duplicate information. It is mainly

```
27
28      σ_x, σ_y = PerCoordVar(WorldPointSets)
29      μ_x, μ_y = PerCoordMean(WorldPointSets)
30      l_x = μ_x − σ_x
31      h_x = μ_x + σ_x
32      l_y = μ_y − σ_y
33      h_y = μ_y + σ_y
34      l = (l_x, l_y); h = (h_x, h_y)
35      for i=0 to N, step=+1 do
36          X̂ = EmptyList()
37          for x in X do
38              x̂ = UniformRandomSamplePerCoord(x, l, h)
39              X̂ = X̂ ∪ x̂
40          WorldPointSets = WorldPointSets ∪ X̂)
41  X̂ = WorldPointSets.firstSet
42  Ĥ = EstimateHomography(X̂, X′)
43  End
```

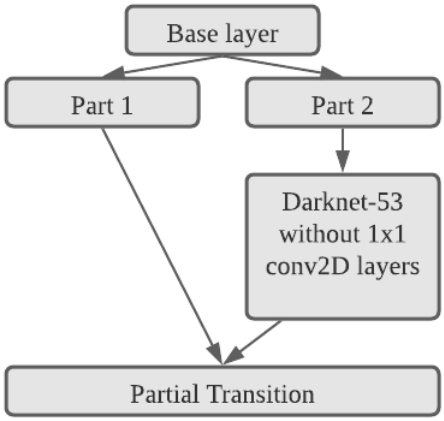


Figure 4.3: CSPDarknet-53 architecture.

composed of a 1×1 convolutional layer, followed by a pooling layer.

For the neck of the detector, a Spatial Pyramid Pooling (SPP) block was included over the backbone. Traditionally, to convert 2D feature maps into vectors a single global average or max-pooling operation is performed. However, this approach results in variable size vectors depending on input images sizes. Thus, fully connected layers will not work in this scenario. On the other hand, SPP blocks generate a fixed-length representation

regardless of image size by pooling in local spatial bins. These spatial bins have sizes proportional to the image size, so the number of bins remain constant [15]. Figure 4.4 shows the architecture of SPP block.

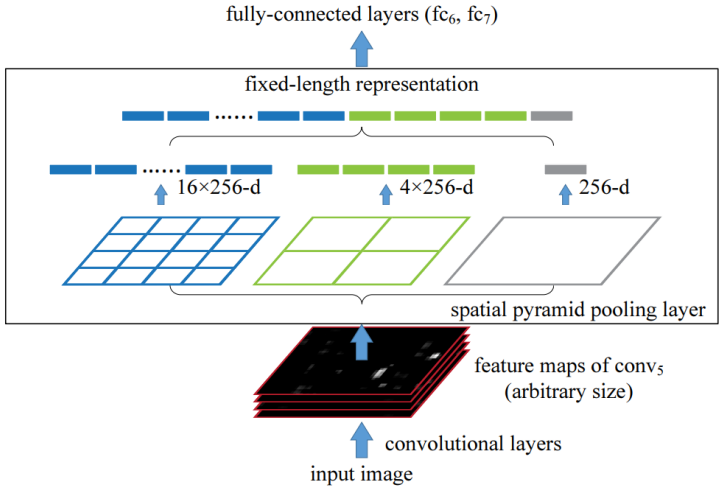


Figure 4.4: Spatial Pyramid Pooling application. Source [15].

YOLOv4 detector also performs feature extraction at three spatial dimensions, to improve detection at varying sizes. The mechanism follows that of a Feature Pyramid Network (FPN) [62] in which the feature maps gradually decrease in the spatial dimension, but are later up-scaled with deconvolution layers. These upscaled feature maps are concatenated with previous feature maps that match in dimensions. In addition, a bottom-up augmented path is also included, starting from the lowest feature level dimension of the FPN and gradually approaching the highest level. In each stage, a 3×3 convolutional layer is employed with stride 2, to decrease spatial dimension. The motivation for this additional path is that neurons in high layers respond to entire objects, while other neurons are more likely to be activated by local patterns. Thus, augmenting the top-down path allows propagation of semantically strong features and enhances localization [63].

Up until this point, the goal was to extract features from the input images, but the head of YOLOv4 is in charge of the actual detection. Its design is taken from YOLOv3, the previous version, in which each output feature map is subjected to a 1×1 convolutional layer with shape $1 \times 1 \times (B(5+C))$, where B is the number of objects that can be detected on each image cell and C is the number of classes. Recall that YOLO divides the images on an $S \times S$ grid, where each cell is in charge of detecting objects. The final output is a tensor of shape $S \times S \times (B(5+C))$ that encodes bounding box coordinates, confidence scores, and class scores for each detected object in each image cell. Furthermore, these coordinates are not estimated directly by the network, as it would have an unstable training process. Instead, pre-defined bounding box anchors in each cell are used, and the network just outputs offsets with respect to those anchors. To predict if an object is in fact present, the confidence needs to be higher than a predefined threshold. The final model is depicted in Figure 4.5.

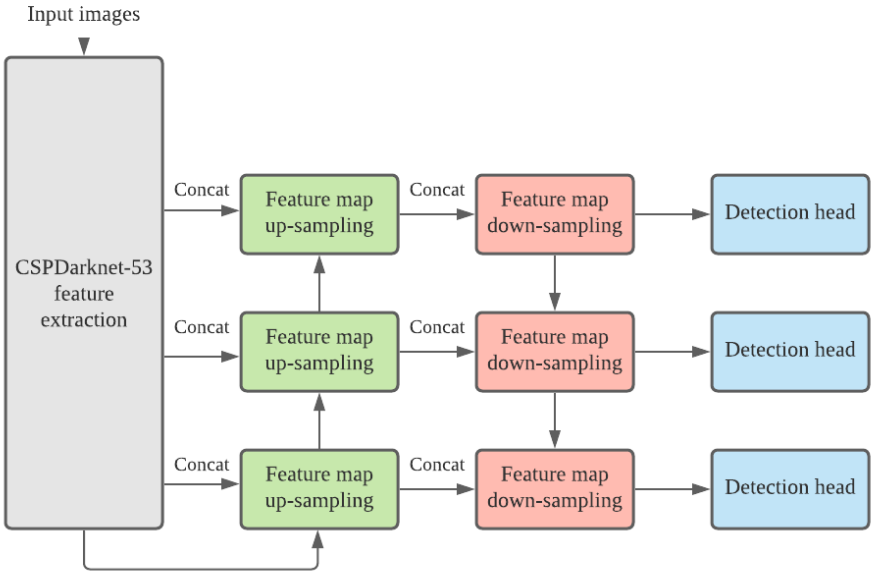


Figure 4.5: YOLOv4 architecture.

4.2.3 Object Tracking Implementations

The object tracker, the last module in the speed estimation system, follows a modified tracking by detection scheme, inspired from the work of Bochinski *et al.* [8] which has the best tracking performance on UA-DETRAC dataset as open-source implementation. It records the location history of all objects frame by frame, but it also keeps track of the speed in km/h along the path of the objects and when crossing a predefined virtual line. At the first image frame, the tracker assigns all detections to new active tracks. Then, the algorithm updates itself each frame by receiving new detections and assigning to each track the new location of the object that is the closest to its last recorded location according to a specific metric. A track may be terminated if no new detection is assigned to it and has a minimum number of locations recorded. Moreover, the metrics implemented in this work are Euclidean Distance

$$d_e(p,q) = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2},$$

using the bounding box centroids and Intersection over Union

$$IOU(\boldsymbol{b_p}, \boldsymbol{b_g}) = \frac{area(\boldsymbol{b_p} \cap \boldsymbol{b_g})}{area(\boldsymbol{b_p} \cup \boldsymbol{b_g})},$$

using bounding boxes directly. However, using bounding boxes in the tracker is beneficial since the assignment of thresholds can be easily implemented to avoid identity hijacking or location jumps and thus, possible unreal peaks in speed estimation. Moreover, the speed estimation process is made every K number of frames for each track independently using:

$$S = \frac{d(p,q)}{t} = \frac{d(p,q)}{f_s^{-1}} \times \frac{3600}{1000},$$

where $d(p, q)$ is distance in meters, f_s is the number of frames per second taken by the camera, and f_s^{-1} represent the time spent each speed estimation. Each time the speed estimation is performed for a vehicle, p takes the value of q, and q becomes the newest location available for that vehicle. To calculate distances, recorded locations in image points have to be transformed to the real world plane as follows:

$$p = \mathbf{H}^{-1}p',$$

where p is the location in world points, $\mathbf{H}$ is the estimated homography matrix, and p' is the location in image points. Then, the distance in meters between the newest real-world location q and the previous real-world point p is computed using the Haversine distance. The reason for using this distance function is that world coordinate system is in longitude-latitude units in radians and the inaccuracies of using planes to represent the world are minimized using this metric. The Haversine distance is expressed as follows:

$$d_h(p, q) = 2arcsin\sqrt{sin^2\left(\frac{p_x - q_x}{2}\right) + cos(p_x)cos(q_x)sin^2\left(\frac{p_y - q_y}{2}\right)},$$

where p_x, q_x are latitudes and p_y, q_y are longitudes. The full tracking algorithm using the IoU metric is depicted in Algorithm 2. Furthermore, the euclidean implementation is similar, but does not apply a threshold to assign detections to objects.

4.3 Implementation

To be able to compute homography matrices, a video dataset with both image and world points representing the same spots is needed or at least contain location metadata of each scene to find these points manually. At the time of writing this work, no such dataset could be found. Instead, a compilation of 14 live feeds from different cameras with unknown hardware specifications were recorded using the Department of Transportation of Seattle portal. These videos have a framerate of 25 frames per second, an average length of 10 minutes (15000 frames), and street addresses from each scene were extracted as well. Then, those addresses were visited on google maps to find the scene and manually extract world points that matched the image plane. Moreover, as the dataset was built from scratch, no beforehand speed annotations of the vehicles are available. Table 4.1 summarizes the dataset metadata.

It can be seen that only 10 videos were usable for the experimentation, as videos 3,4,8, and 10 presented problems. Moreover, all the visual scenes from the remaining videos can be observed in figures 4.6 and 4.7. Each scene has a green line in it, representing the virtual line that will trigger a speed recording. Furthermore, once the data is ready, homographies are estimated using the three methods mentioned in the methodology: base, RANSAC, and proposed, which are implemented in C++ with OpenCV library. Once the camera calibrations are obtained, the best homography method in terms of mean error is picked to be used in the whole workflow. For the object detection module, YOLOv4 was implemented in Python3.8 using Tensorflow 2.4. In addition, the tracking algorithm was initially intended to be in the same workflow as the detection, but due to hardware limitations, detections were precomputed and saved on pickle files. Then, at runtime, the

Algorithm 2: Tracking implementation with IoU metric.

Data: LinePoints (p_1, p_2), Min detections τ, IOU threshold β, Homography matrix H, EMA alpha α, Update rate K, Bounding boxes B, Tracks active T_a

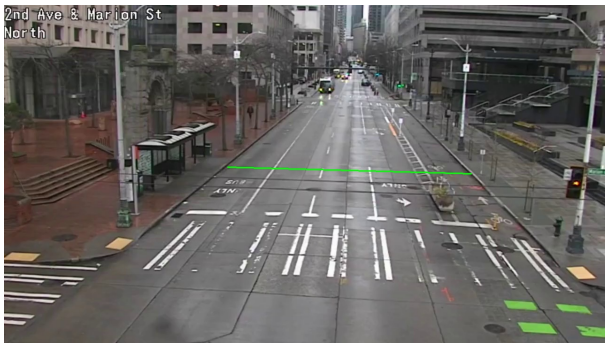
Result: Finished tracks T_f

Start;
T_u = EmptyList()
for t *in* T_a **do**
  if B *is not empty* **then**
    $best\_match = max(\{IOU(t.boxes.last, b)/b \in B\})$
    if $IOU(t.boxes.last, best\_match) \geq \beta$ **then**
      $t.boxes = t.boxes \cup best\_match$
      $p' = centroid(best\_match)$
      $t.newRealLoc = H^{-1}p'$
      $t.count = t.count + 1$
      if $t.count == K$ **then**
        $S = \frac{d(t.prevRealLoc, t.newRealLoc)}{K f_s^{-1}}$
        $t.speed = t.speed \cup S$
        $t.prevRealLoc = t.newRealLoc$
        $t.count = 0$
      if p' *intersects virtualLine* **then**
        $speedsLine = speedsLine \cup t.speed.last$
      $T_u = T_u \cup t$
  if T_u *is empty or* t *is not* $T_u.last$ **then**
    if $t.boxes.size \geq \tau$ **then**
      $T_f = T_f \cup t$
  for *box in remaining boxes* **do**
    $t_{new}.boxes = \{box\}$
    $t_{new}.prevRealLoc = H^{-1} centroid(box)$
    $t_{new}.newRealLoc = \infty$
    $t_{new}.speed = EmptyList()$
    $t_{new}.count = 0$
    $T_n = T_n \cup t_{new}$
  $T_a = T_u \cup T_n$

Table 4.1: Overview of the videos captured from Seattle Dept. of Transportation `https://web6.seattle.gov/travelers/`.

Video Id	Date of recording	Address	Number of Points Extracted	Observation
1	February, 22, 2021	2nd Ave & Marion St	8	None
2	February, 22, 2021	E Marginal Way S & S Idaho St	8	None
3	February, 22, 2021	Alaskan Way & Pike St	0	Points not clearly visible from Google Maps
4	February, 22, 2021	5th Ave N & Broad St	0	Points not clearly visible from image frame
5	February, 22, 2021	23rd Ave E & E Madison St EW	7	None
6	February, 22, 2021	1st Ave S & S Royal Brougham Way	7	None
7	February, 22, 2021	Airport Way S & S Industrial Way	7	None
8	February, 22, 2021	Aurora Ave N & N 103rd St	0	Video feed corrupted
9	March, 10, 2021	23rd Ave S & S Jackson St	4	Placed only 4 points to form a polygon surrounding the road
10	March, 10, 2021	Elliott Ave W & W Mercer Pl	0	Points not clearly visible from image frame
11	March, 10, 2021	Airport Way S & S Lander St	4	Placed only 4 points to form a polygon surrounding the road
12	March, 10, 2021	E Marginal Way S @ Hudson St	4	Placed only 4 points to form a polygon surrounding the road
13	March, 10, 2021	1st Ave & Seneca St	4	Placed only 4 points to form a polygon surrounding the road
14	March, 10, 2021	Fairview Ave & Denny Way	4	Placed only 4 points to form a polygon surrounding the road

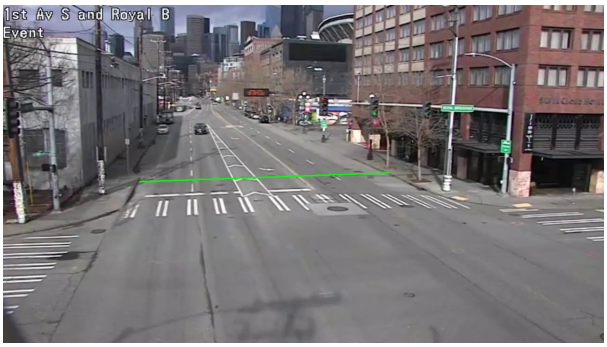
detections were loaded using a traditional loop and fed to the tracker, which is based on the work of [8], modified to perform updates, frame by frame and implementing the speed estimation functionality. Finally, all speed data was serialized and stored on pickle files as well as aggregated for visualizations using Matplotlib library for Python 3.



(a) Video 1

(b) Video 2

(c) Video 5



(d) Video 6

(e) Video 7

Figure 4.6: Capture of the scene of videos 1, 2, 5, 6, and 7, along with the virtual line location.

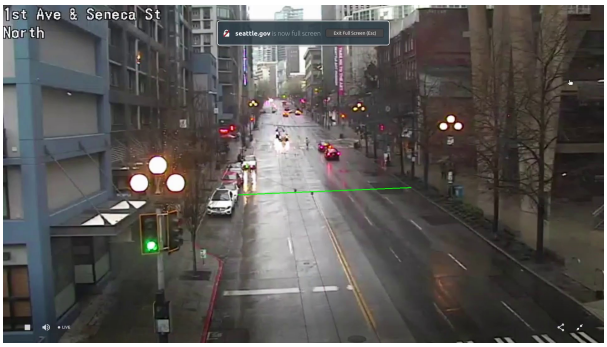
4.4 Experimental Setup

All experimentation was performed on an Acer Aspire-E5-576G laptop running Ubuntu 20.10 with Intel® Core™ i5-8250U Central Processing Unit (CPU), Nvidia GeForce MX150

(a) Video 9

(b) Video 11

(c) Video 12



(d) Video 13

(e) Video 14

Figure 4.7: Capture of the scene of videos 9, 11, 12, 13, and 14, along with the virtual line location.

GPU with CUDA enabled, 12 Gb of DDR4 Random Access Memory (RAM), and 500 GB Solid State Drive M.2 Non Uniform Memory Access (NVME). Although CUDA is enabled, it could not be used for deep learning acceleration due to the limited graphics memory available.

Chapter 5

Results

5.1 Calibration Errors

Upon camera calibration, projection errors from longitude-latitude to pixel coordinates were computed for each video to assess how much the estimated points diverge when the homography matrix is applied. Figure 5.1 presents quatitative results for the base, RANSAC and proposed versions of the calibration.

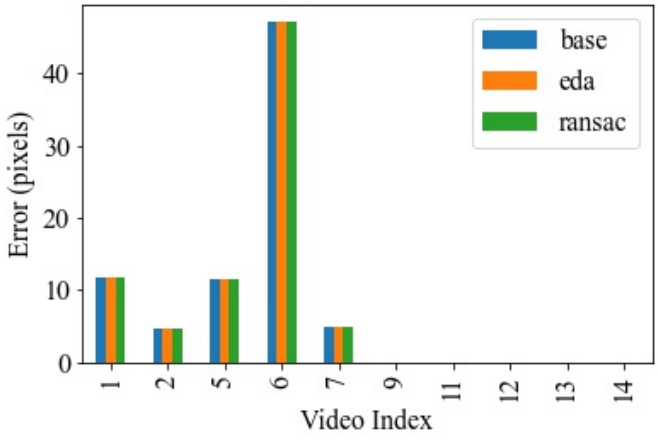
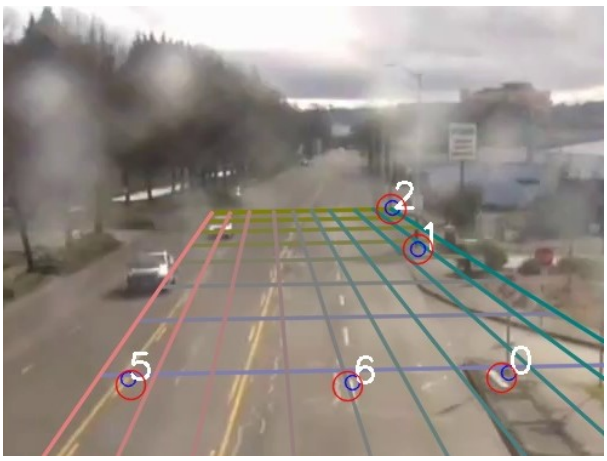


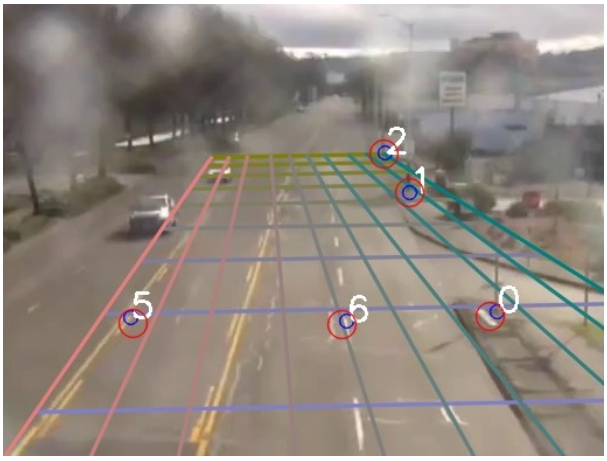
Figure 5.1: Projection error for each video.

In addition, Figure 5.2 depicts some examples of how well the estimated image points, as red circumferences, match the ground truth image points in blue, as well as a grid of points adapted to the perspective of the whole region of interest in a particular scene, using the computed homographies. Some of the most representative scenes in the dataset and their models of the street can also be seen on figures 1, 2, and 3 from Appendix .1 for base, RANSAC, and proposed calibration algorithms, respectively.

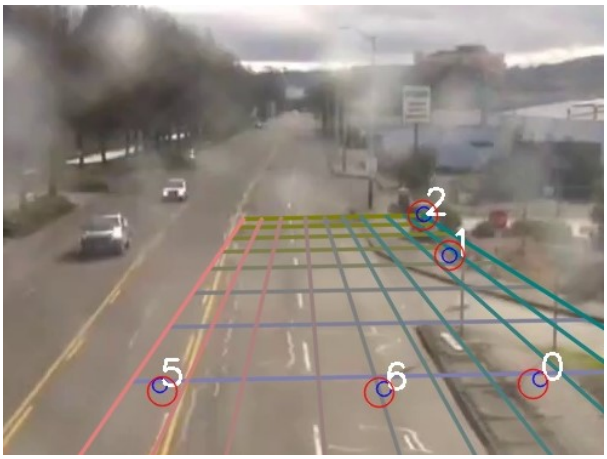
Furthermore, as distance computation is fundamental to computing speed, distance differences were calculated from actual longitude-latitude points and estimated coordinates in the same world units using the haversine distance. These distance errors can be seen in Figure 5.3.



(a) Base calibration.



(b) Ransac calibration.



(c) Proposed calibration.

Figure 5.2: Calibration visualizations for Video 7. Red circumferences indicate the image point estimations, while blue circumferences indicate ground truth. The grid is a virtual model of the street.

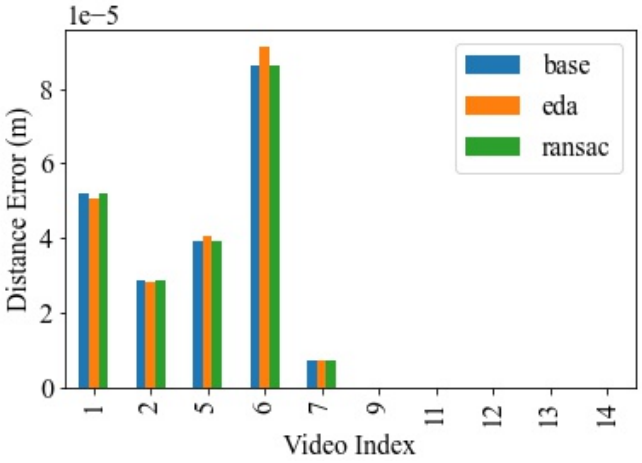


Figure 5.3: Distance error for each video video.

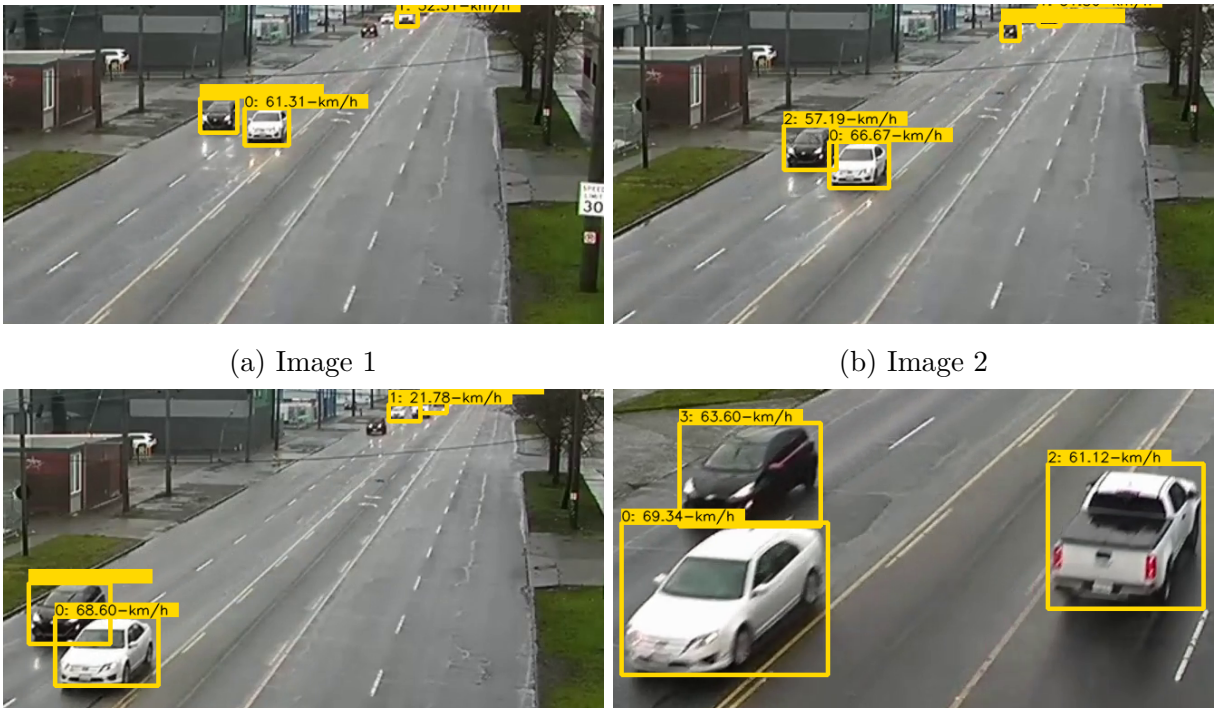
5.2 Graphical Layout of the Method

The proposed method is kept running until the current image sequence is exhausted and the same is performed for each video. In addition, it also has a graphical layout where each detected vehicle is surrounded by a bounding box and the speed value is presented right above it, if available. This layout is available so that an operator can ensure the proper functioning of the system qualitatively. Figure 5.4 depict a sample of images taken while running the proposed methodology on video 11.

5.3 System Run for IoU Tracker Implementation

Separate tests were conducted to analyze the behavior when the tracker implements Intersection over Union (IoU) or euclidean distance to associate tracks. Figures 5.5 and 5.6 depict the histograms of all speeds recorded when vehicles crossed each virtual line per video. All speed values registered throughout the whole region of interest were captured as well for all tracks. Figures 5.7 and 5.8 contain all the distributions of speed recordings per video. Note that for each vehicle track, a speed value is retrieved every half a second, or equivalently, every 25 frames. In addition, the tracking algorithm's ability to maintain stable tracks is inspected through histograms for each video, on figures 5.9 and 5.10. Specifically, the distributions show the frequency of track lengths for each unique object designated by the tracker, and each sample within a track represents a recorded object location. Each location is stored within each track at the time of speed estimation, i.e every timestamp, half a second apart.

On the other hand, measures for analysis of individual objects are also included where each vehicle speed history was recorded independently. However due to the large number of object ids, a set of ten timeseries of the longest tracks were plotted for each video in figures 5.11, 5.12, and 5.13. Those timeseries correspond to the speed recorded on each timestamp. Furthermore, boxplots were computed for the same vehicles to include more statistical insights like outliers, quartiles, mean and media to assess congruence in the



(a) Image 1 (b) Image 2

(c) Image 3 (d) Image 4

Figure 5.4: Sample of frames while runing the methodology.

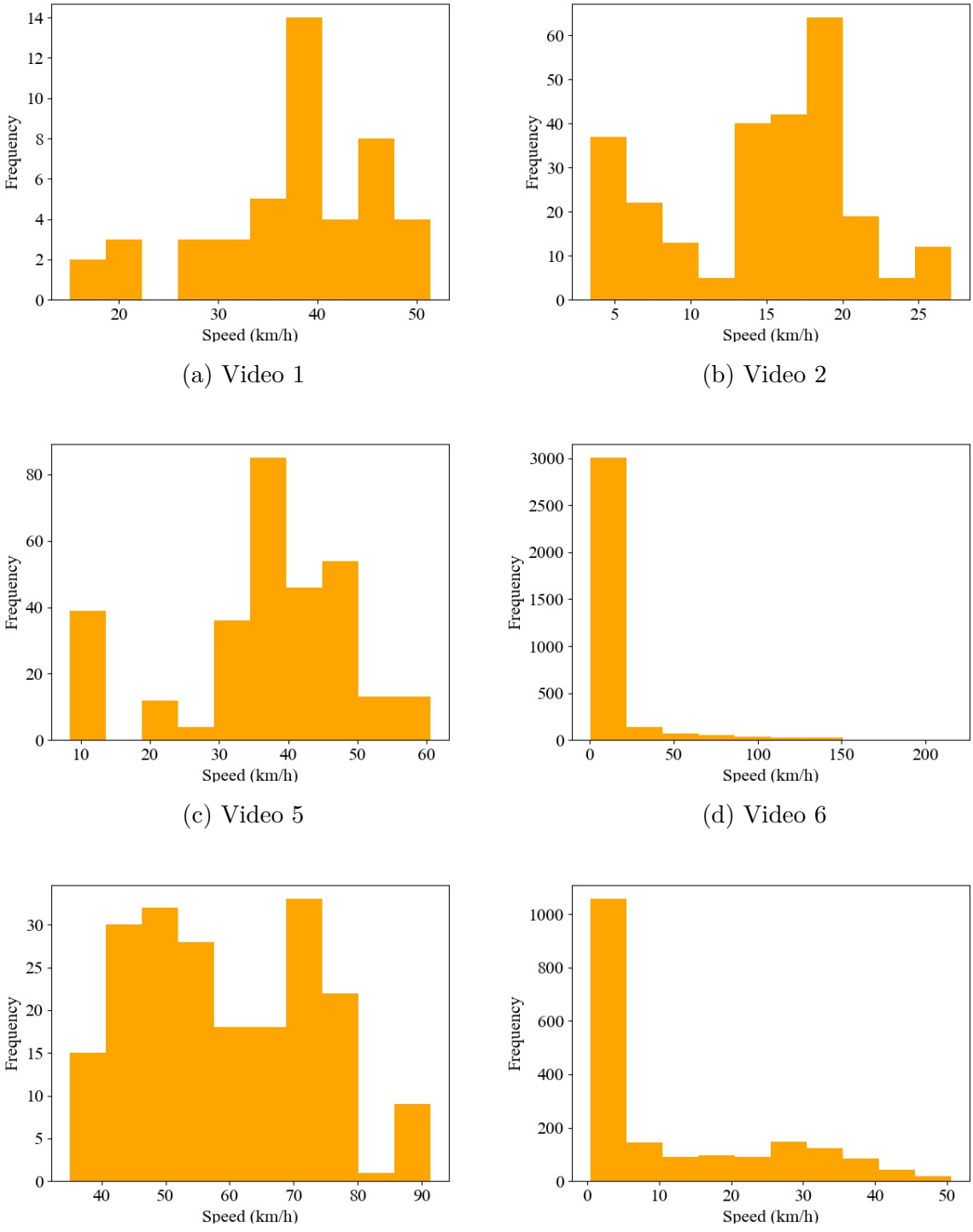
measures. Such boxplots are depicted in figures 5.14 and 5.15.

Finally, three tables were created, to summarize the extensive set of metrics gathered across all videos. Table 5.1 contains mean, median, and standard deviation statistics for the number of samples in each track, Table 5.2 shows also mean, median, and STD for all the speed values recorded and includes the percentage of speed measures that go beyond 70 km/h and 100 km/h. Lastly, Table 5.3 describes the three basic statistics for all measurements captured on the virtual lines.

5.4 System Run for Euclidean Tracker Implementation

The same procedure of extracting metrics for analysis in Intersection over Union tracker was performed in this version, the euclidean distance based tracker. Figures 5.16 and 5.17 show the distribution of speeds recorded when cars were crossing the virtual lines. Figures 5.18 and 5.19 depict the distributions of all speed samples taken across the whole region of interest, i.e the street. For the distributions on the number of samples of vehicle tracks, histograms were computed and presented in figures 5.22 and 5.23. Also, the analysis of vehicles corresponding to the ten longest tracks was computed. Figures 5.24, 5.25, and 5.26 show the timeseries of speed against time for such vehicles and figures 5.20, 5.21 present the boxplots of those same vehicle speed measures to gain additional insights by visualizing median, quartiles, and outliers. Moreover, statistics on the number of samples within the

vehicle tracks, speed on RoI and speed on virtual line are presented in tables 5.4, 5.5, and 5.6, respectively. These tables also have the same structure as the IoU tables.



(a) Video 1

(b) Video 2

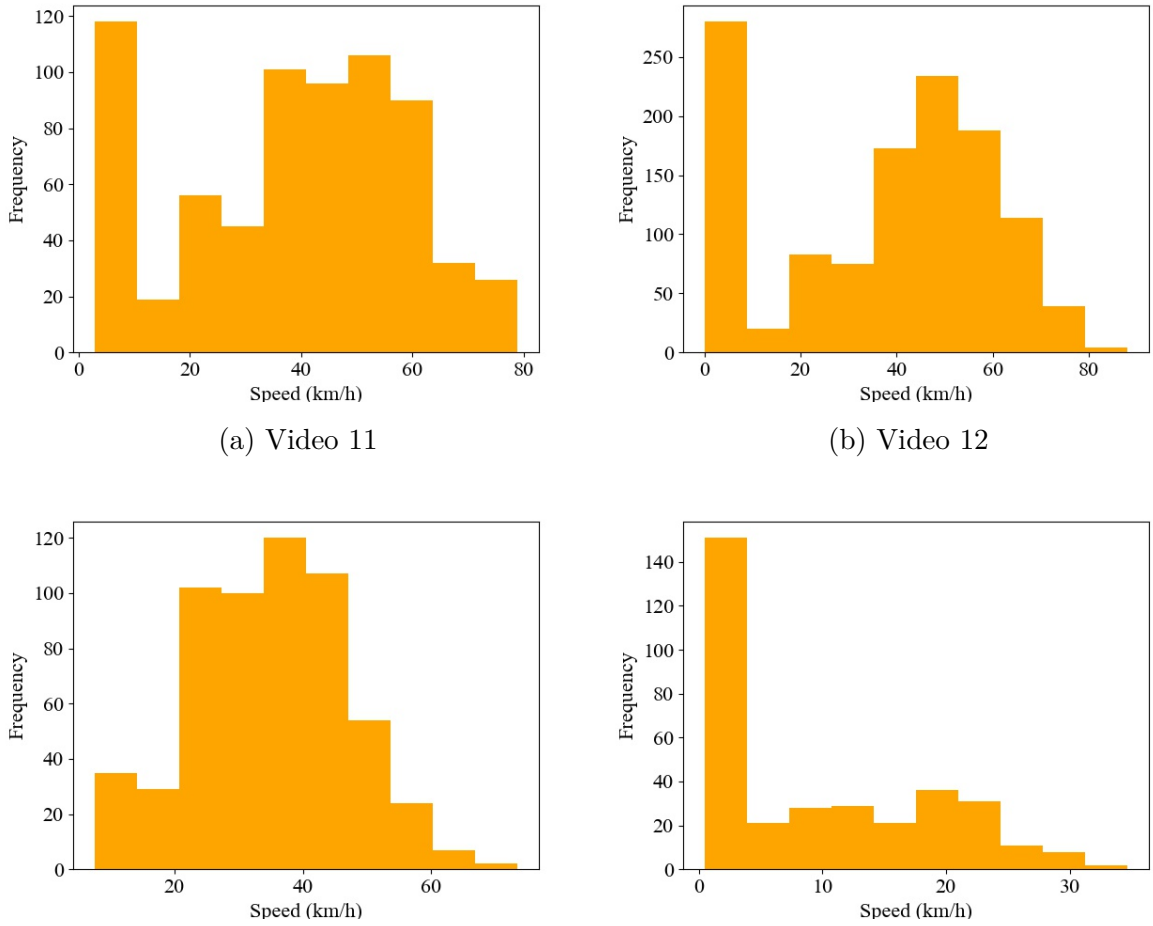
(c) Video 5

(d) Video 6

(e) Video 7

(f) Video 9

Figure 5.5: Histogram of registered speed values using IoU tracker, on virtual lines for videos 1, 2, 5, 6, 7, and 9.



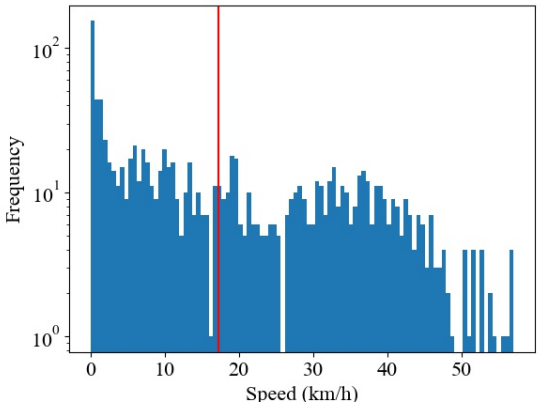
(a) Video 11

(b) Video 12

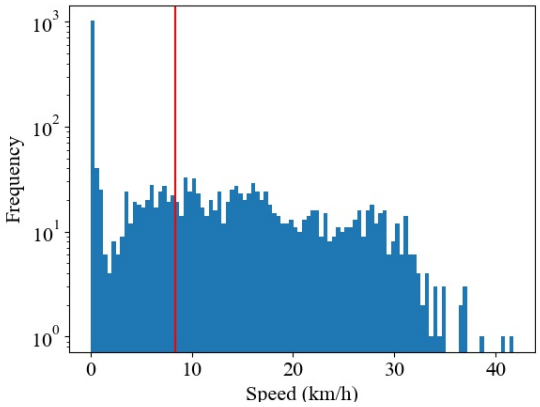
(c) Video 13

(d) Video 14

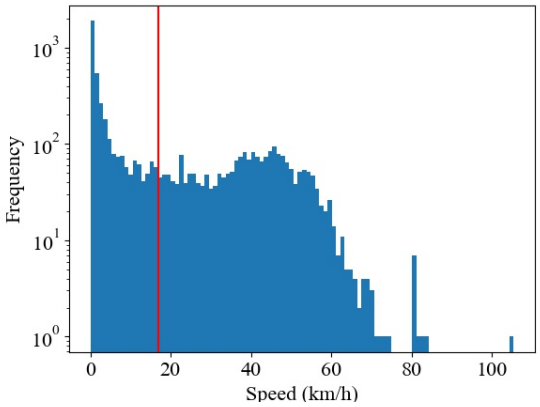
Figure 5.6: Histogram of registered speed values using IoU tracker, on virtual lines for videos 11, 12, 13, and 14.



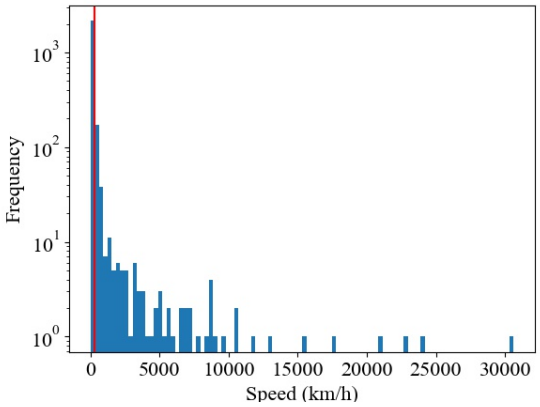
(a) Video 1



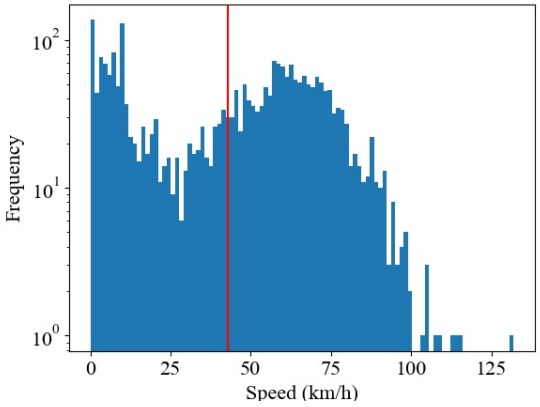
(b) Video 2



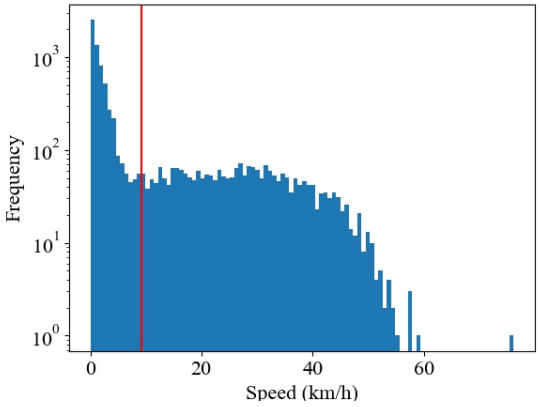
(c) Video 5



(d) Video 6

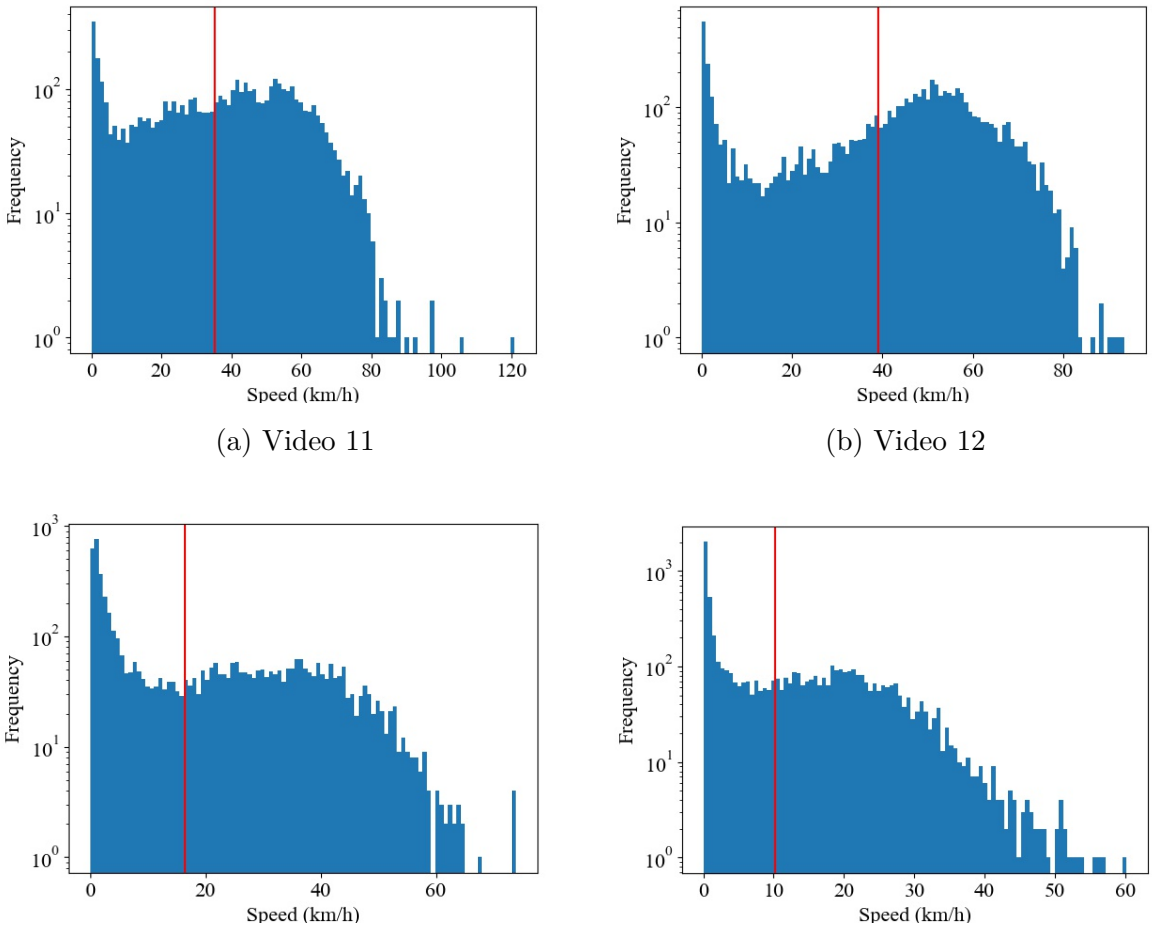


(e) Video 7



(f) Video 9

Figure 5.7: Speed distribution for videos 1, 2, 5, 6, 7, and 9 using IoU tracker.



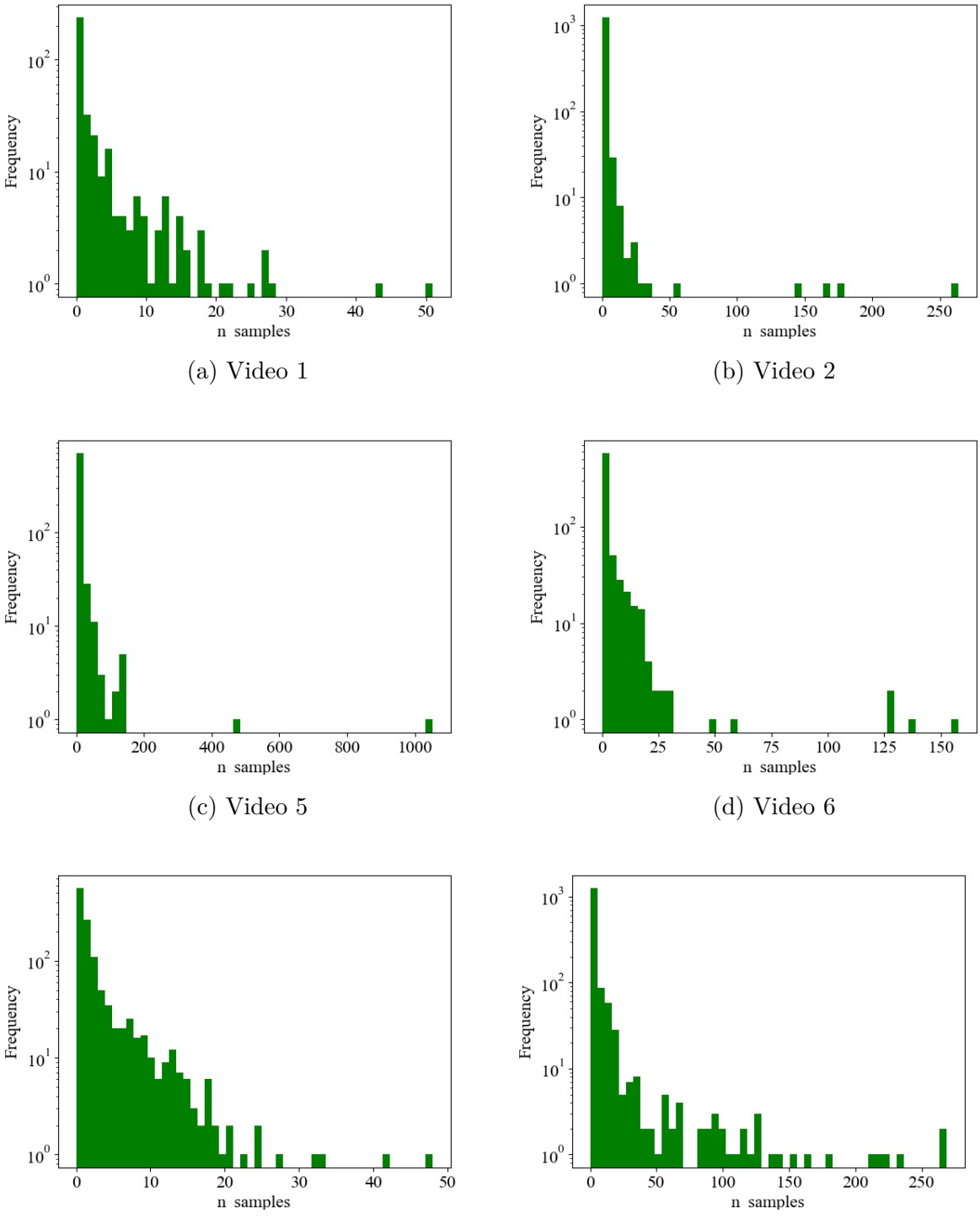
(a) Video 11

(b) Video 12

(c) Video 13

(d) Video 14

Figure 5.8: Speed distribution for videos 11, 12, 13, and 14 using IoU tracker.



(a) Video 1

(b) Video 2

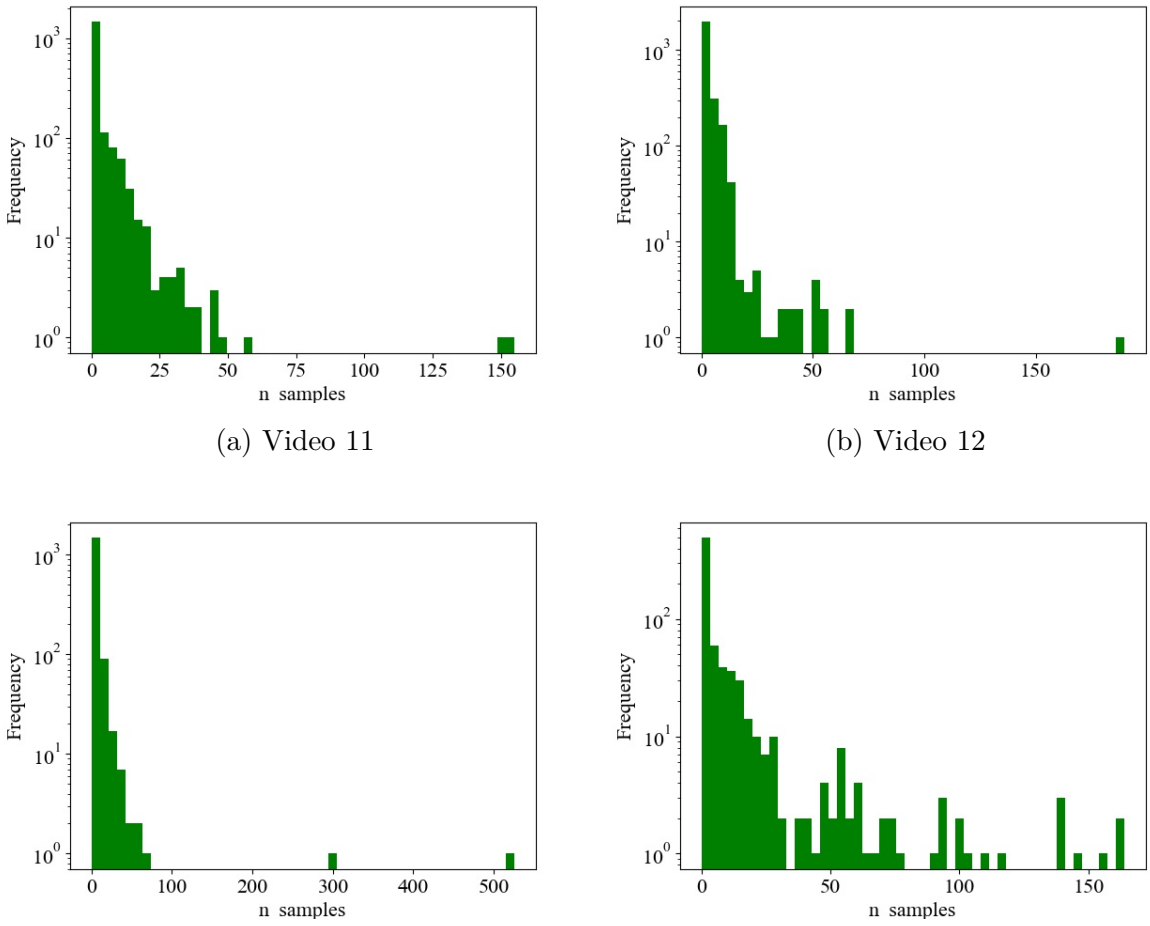
(c) Video 5

(d) Video 6

(e) Video 7

(f) Video 9

Figure 5.9: Histogram of samples per track using IoU tracker for videos 1, 2, 5, 6, 7, and 9.



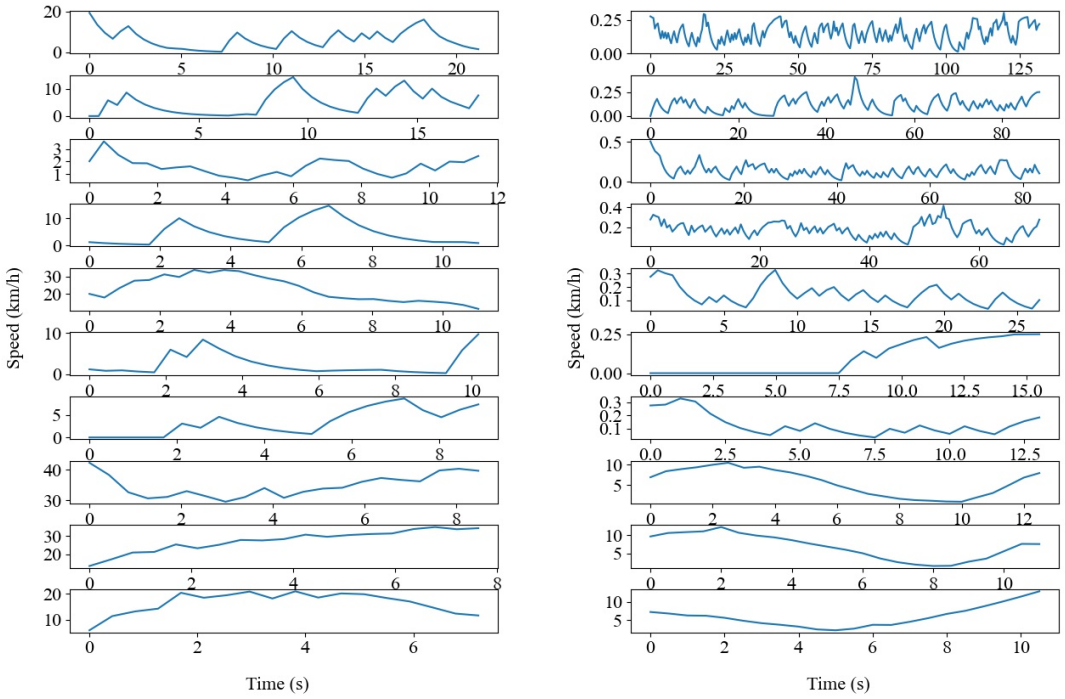
(a) Video 11

(b) Video 12

(c) Video 13

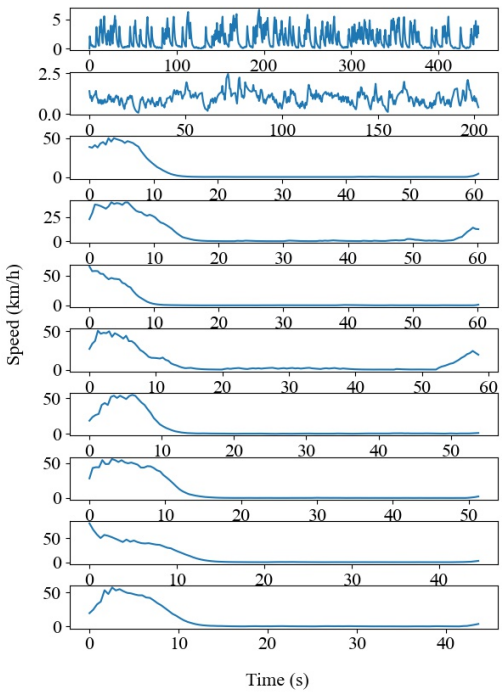
(d) Video 14

Figure 5.10: Histogram of samples per track using IoU tracker for videos 11, 12, 13, and 14.

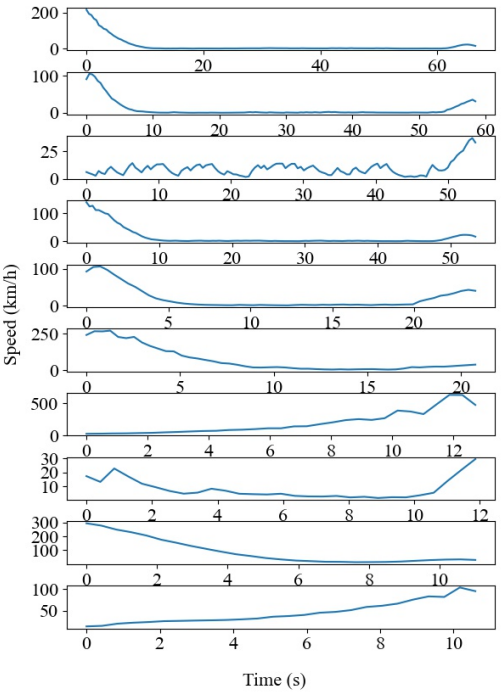


(a) Video 1

(b) Video 2

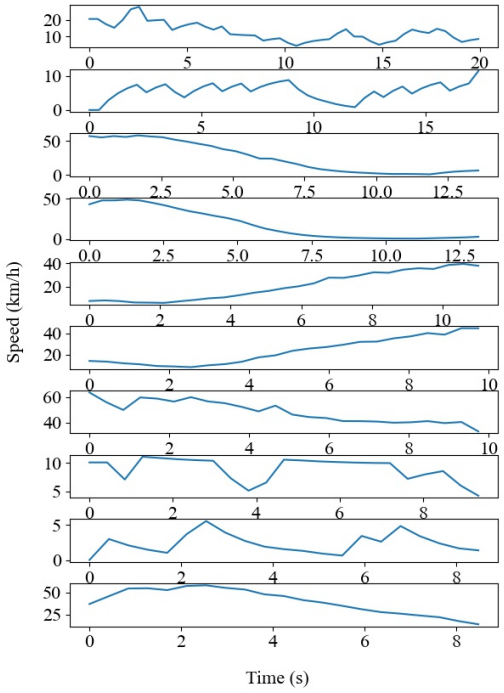


(c) Video 5

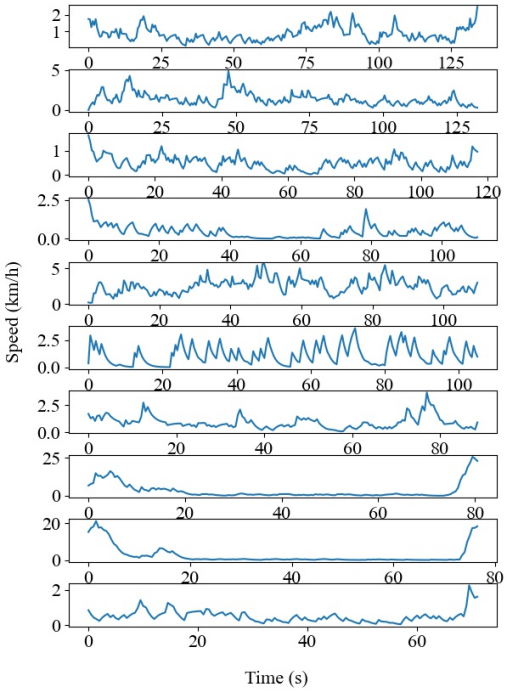


(d) Video 6

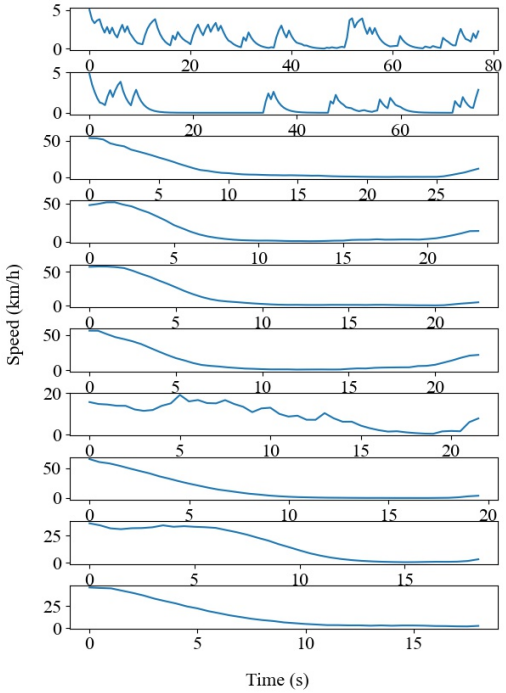
Figure 5.11: IoU tracker speed time series for the 10 longest tracks on videos 1, 2, 5, 6.



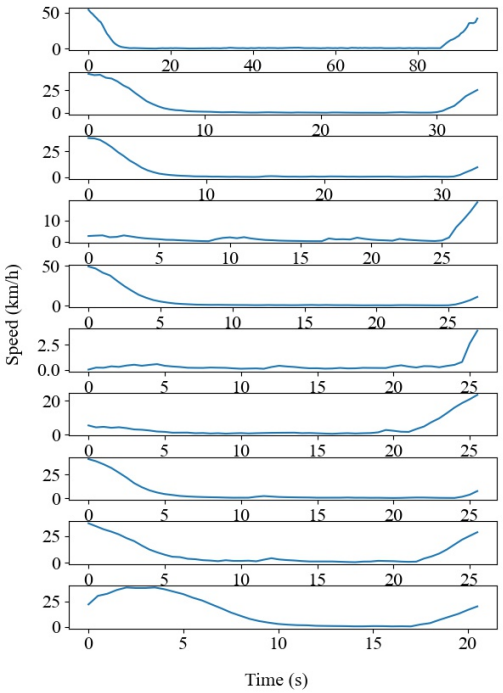
(a) Video 7



(b) Video 9

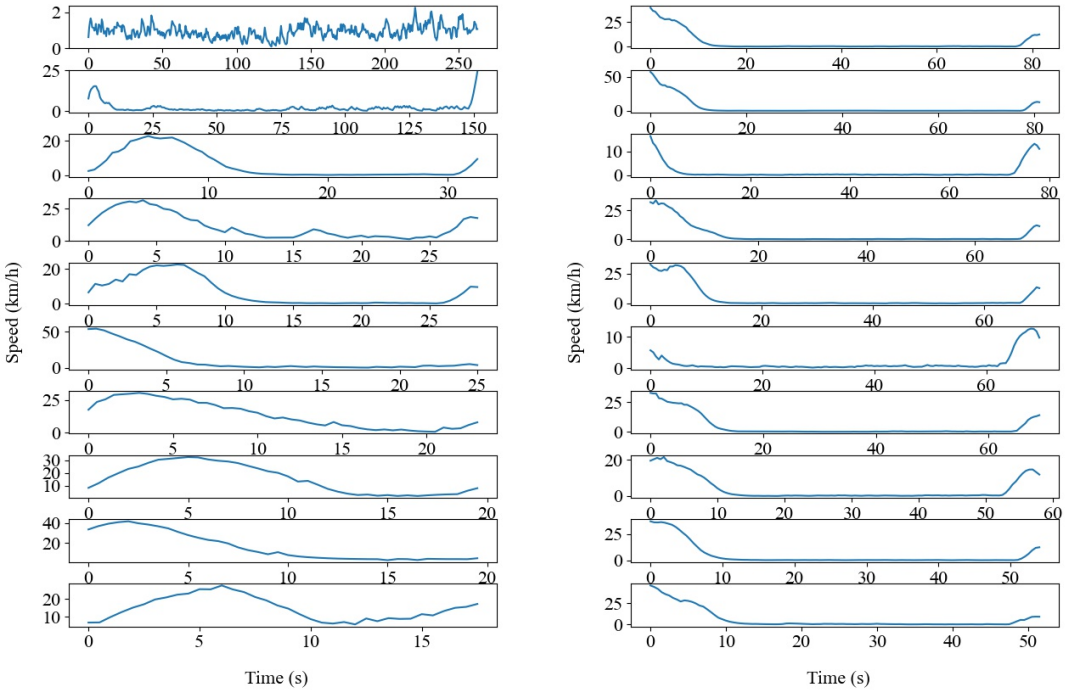


(c) Video 11



(d) Video 12

Figure 5.12: IoU tracker speed time series for the 10 longest tracks on videos 7, 9, 11, 12.

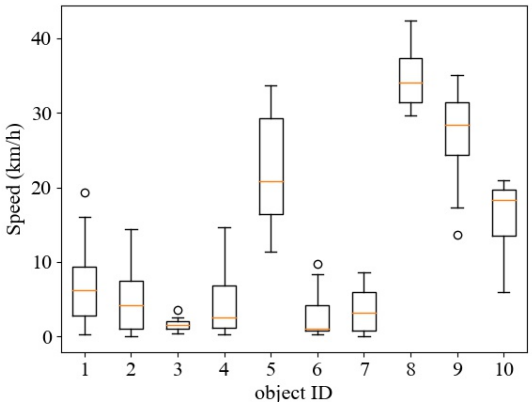


(a) Video 13 (b) Video 14

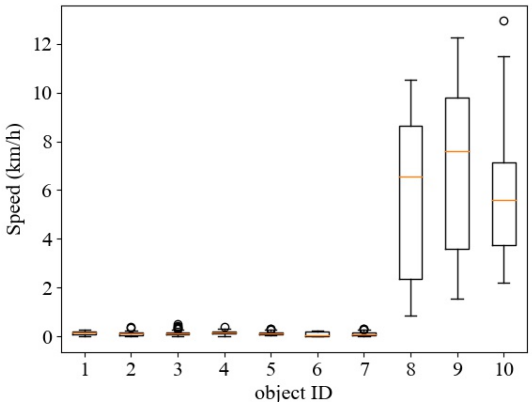
Figure 5.13: IoU tracker speed time series for the 10 longest tracks on videos 13, and 14.

Table 5.1: Statistics computed over vehicles tracks using IoU tracker.

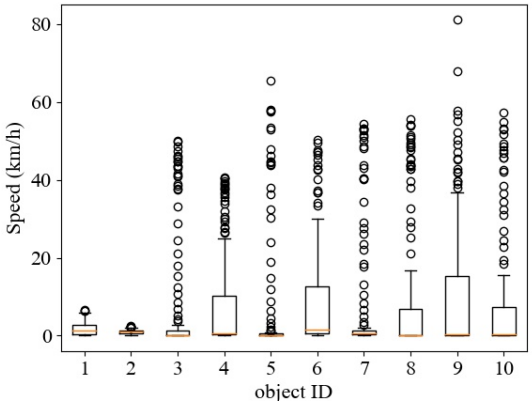
Registered Vehicles	Mean Samples per Vehicle	Median of Samples per Vehicle	Standard Deviation Samples per Vehicle
365	2.873	1.0	5.758
1264	1.810	0.0	11.177
750	7.930	1.0	44.870
716	3.413	1.0	11.431
1190	2.193	1.0	4.347
1483	5.786	1.0	21.700
1795	2.682	1.0	7.327
2501	2.424	1.0	6.222
1599	3.232	1.0	16.175
747	8.661	1.0	21.143



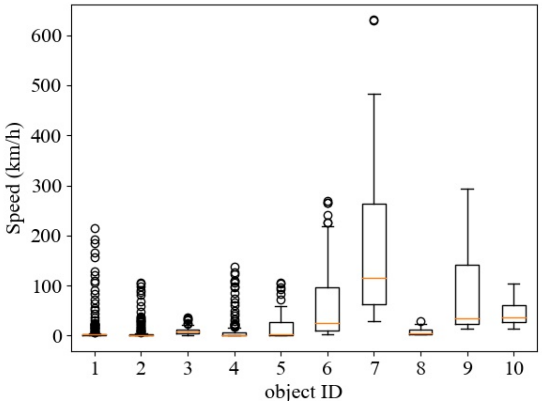
(a) Video 1



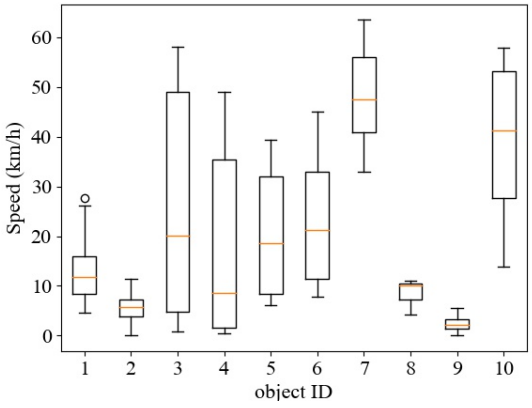
(b) Video 2



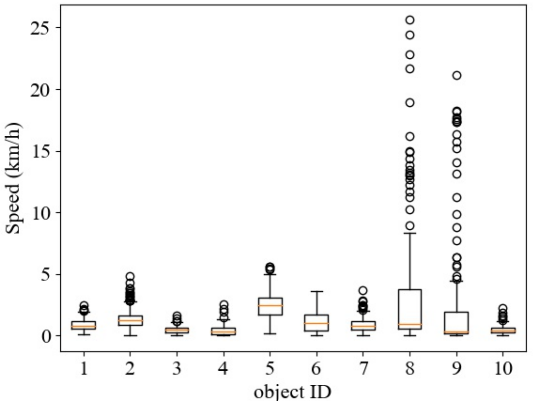
(c) Video 5



(d) Video 6

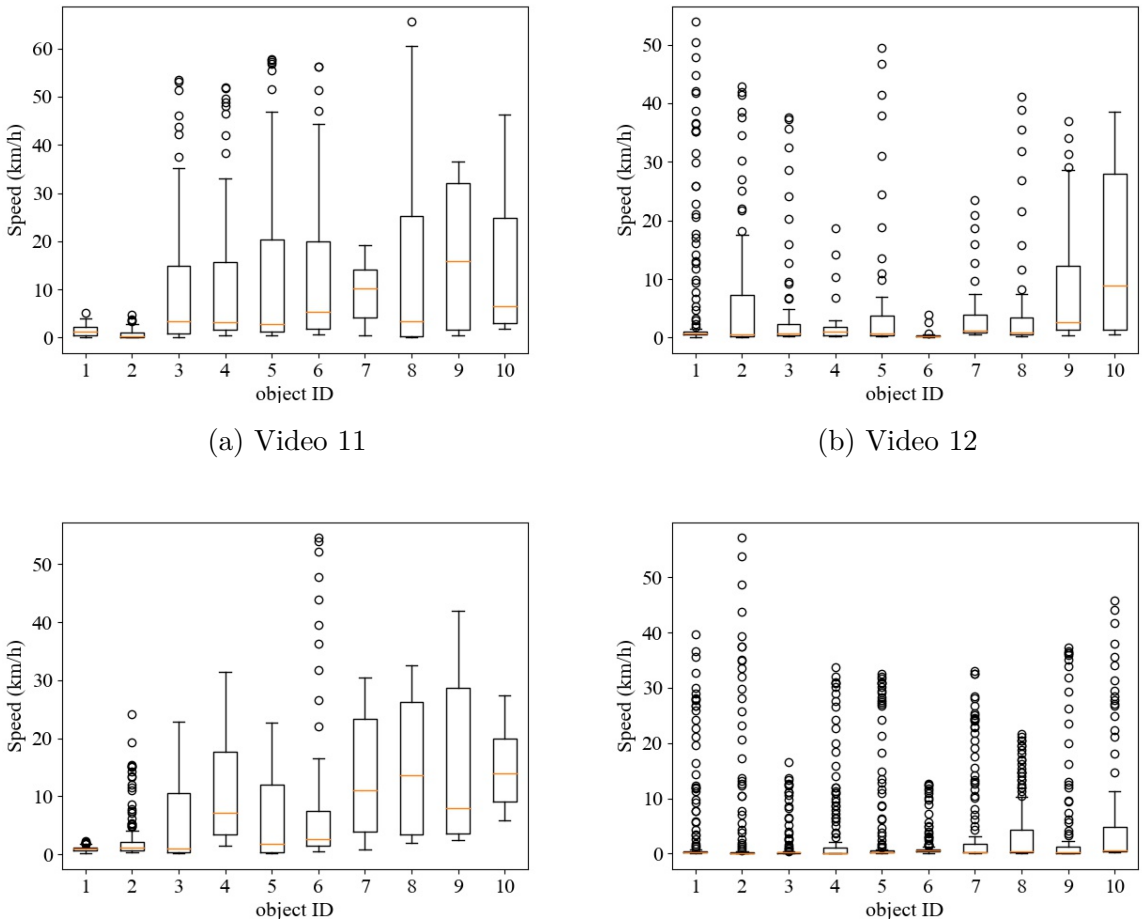


(e) Video 7



(f) Video 9

Figure 5.14: Boxplots of speed for the 10 longest tracks using IoU tracker on videos 1, 2, 5, 6, 7, and 9.



(a) Video 11 (b) Video 12

(c) Video 13 (d) Video 14

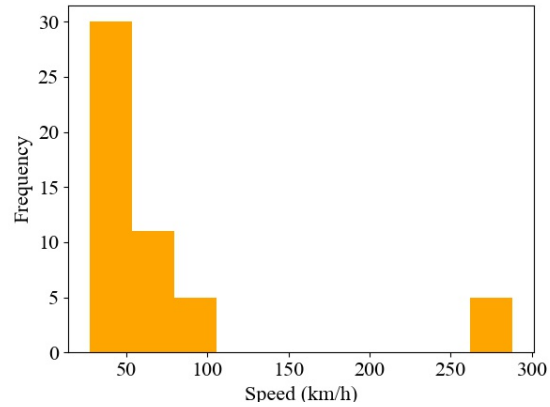
Figure 5.15: Boxplots of speed for the 10 longest tracks using IoU tracker on videos 11, 12, 13, and 14.

Table 5.2: Speed statistics for all videos using IoU tracker.

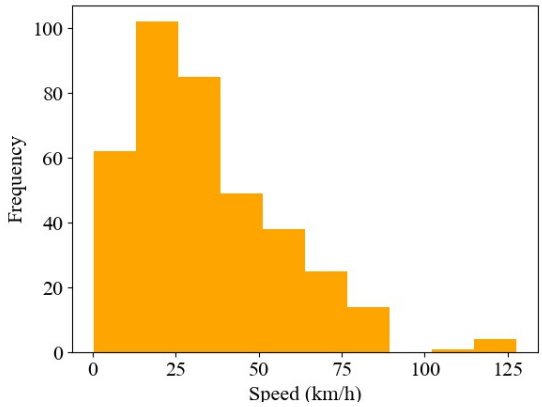
Speed Samples	Mean Speed (Km/h)	Median Speed (Km/h)	Speed STD (Km/h)	Percentage of Speed Samples > 70 km/h	Percentage of Speed Samples > 100 km/h
1049	17.324	13.083	15.590	0.0	0.0
2289	8.385	4.075	9.877	0.0	0.0
5948	16.963	5.067	19.647	0.003	0.000
2444	292.231	71.859	1413.047	0.510	0.375
2610	42.997	49.040	28.072	0.185	0.004
8581	9.140	1.861	13.191	0.0	0.0
4814	35.179	37.634	21.845	0.035	0.0
6063	39.180	45.761	22.991	0.052	0.0
5168	16.445	8.383	17.0	0.0	0.0
6470	10.204	5.168	11.261	0.0	0.0

Table 5.3: Speed statistics for all videos crossing the virtual lines using IoU tracker.

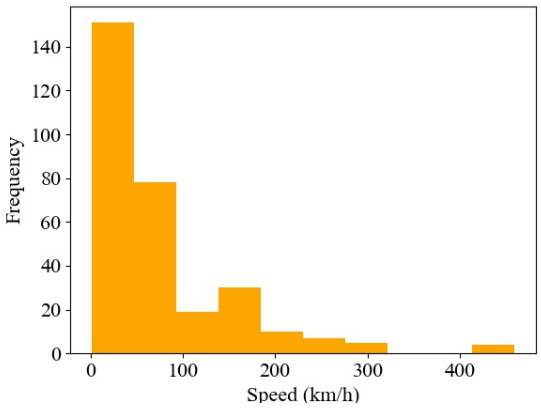
Speed Samples	Mean Speed (Km/h)	Median Speed (Km/h)	Speed STD (Km/h)
46	37.384	38.911	8.797
259	14.641	15.837	6.348
302	36.366	38.218	12.339
3373	8.902	1.267	24.083
206	58.964	57.168	14.148
1897	11.680	4.254	13.628
689	38.776	41.385	19.835
1210	36.773	43.346	23.282
580	34.495	34.954	12.083
338	9.585	5.899	9.304



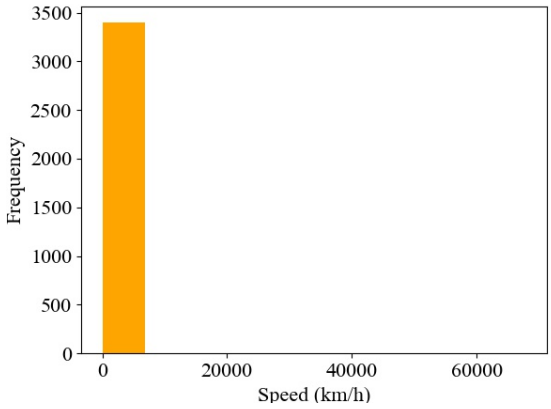
(a) Video 1



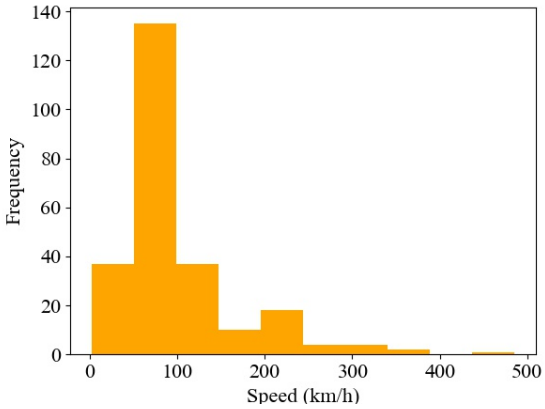
(b) Video 2



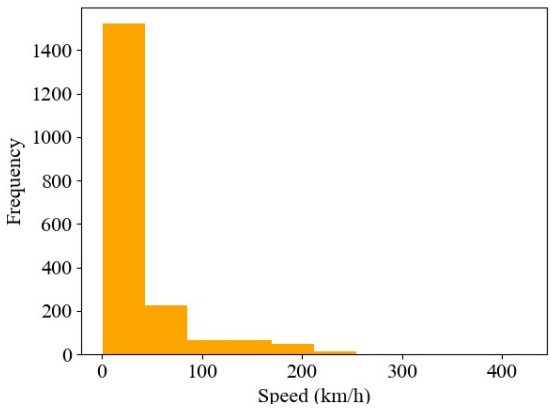
(c) Video 5



(d) Video 6

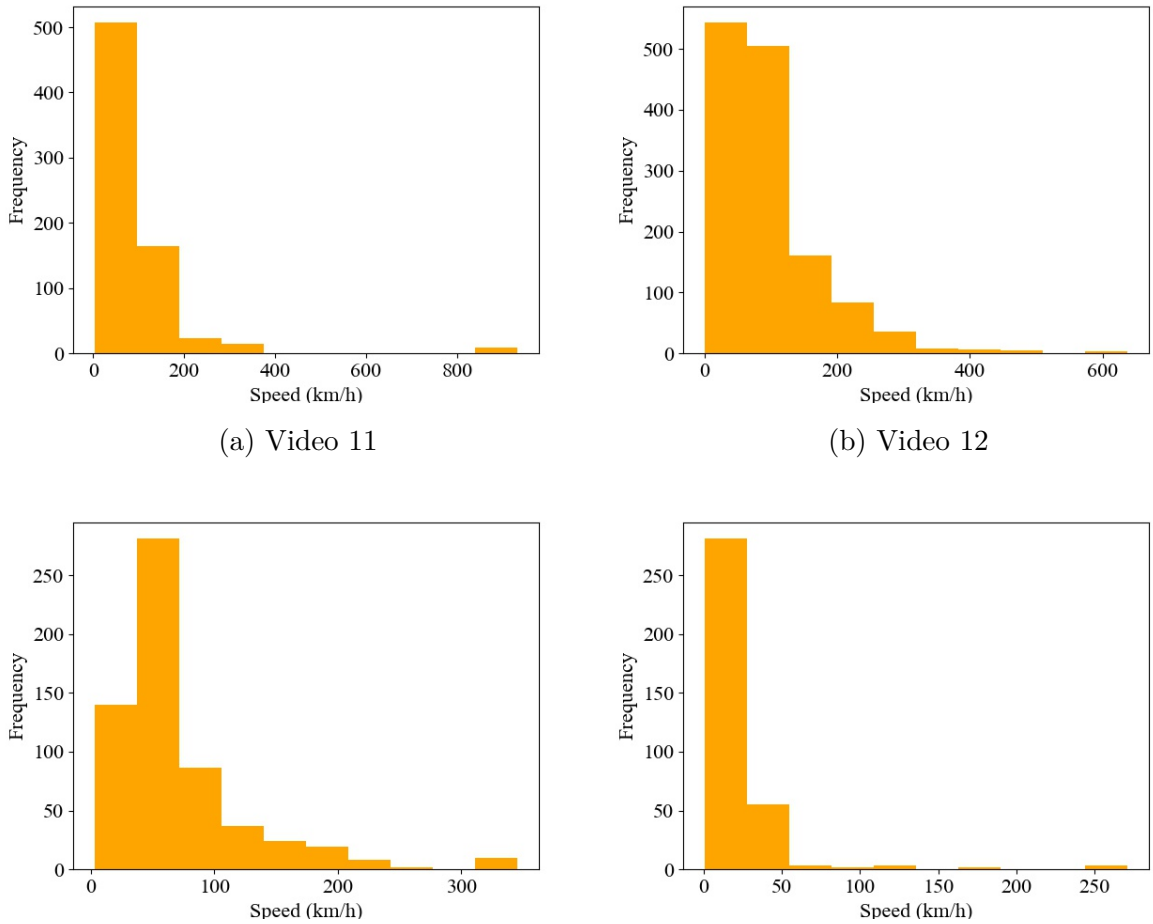


(e) video 7



(f) Video 9

Figure 5.16: Histogram of registered speed values on virtual lines using euclidean tracker for videos 1, 2, 5, 6, 7, and 9.



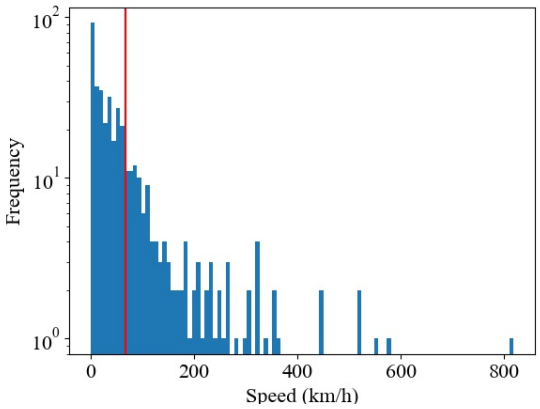
(a) Video 11

(b) Video 12

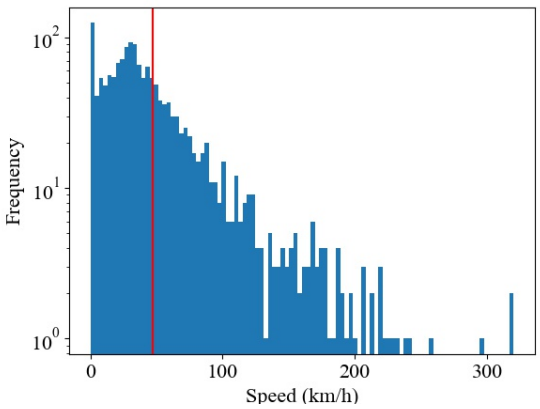
(c) Video 13

(d) Video 14

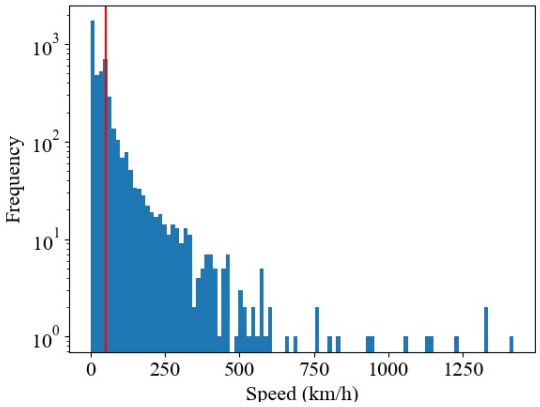
Figure 5.17: Histogram of registered speed values on virtual lines using euclidean tracker for videos 11, 12, 13, and 14.



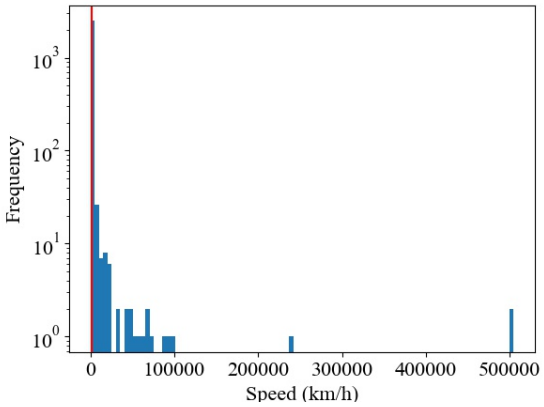
(a) Video 1



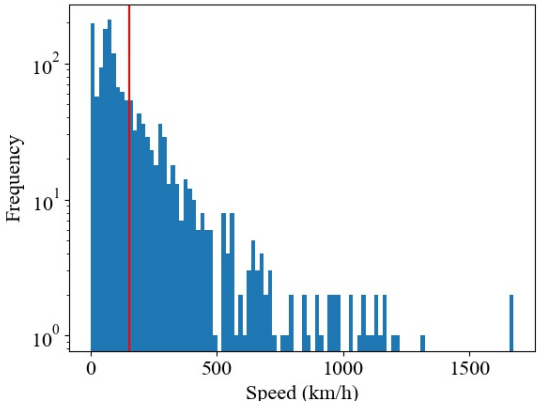
(b) Video 2



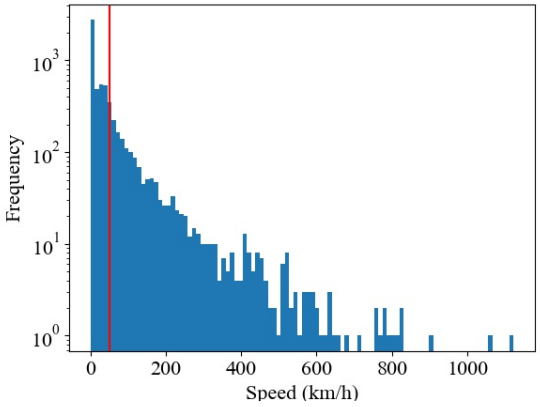
(c) Video 5



(d) Video 6

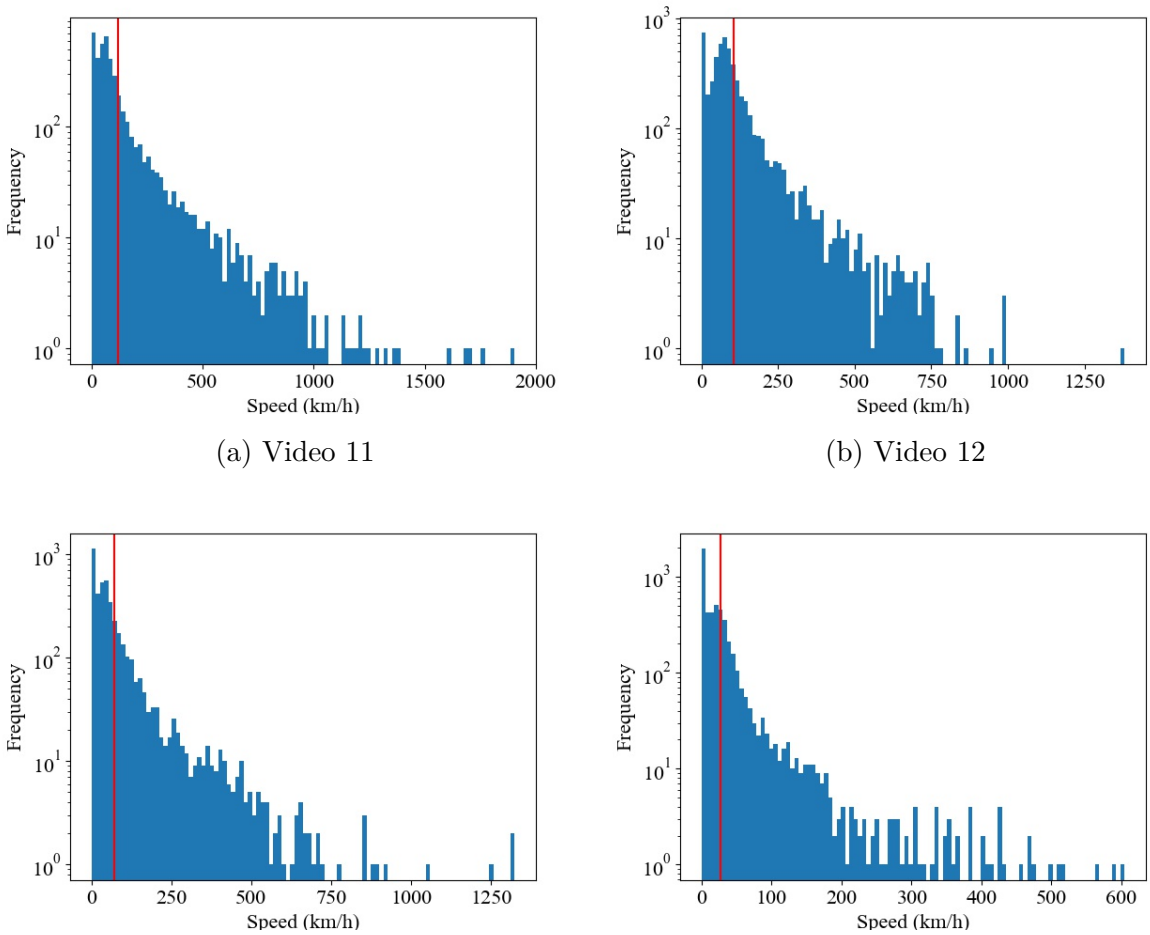


(e) Video 7



(f) Video 9

Figure 5.18: Speed distribution for videos 1, 2, 5, 6, 7, and 9 using euclidean tracker.



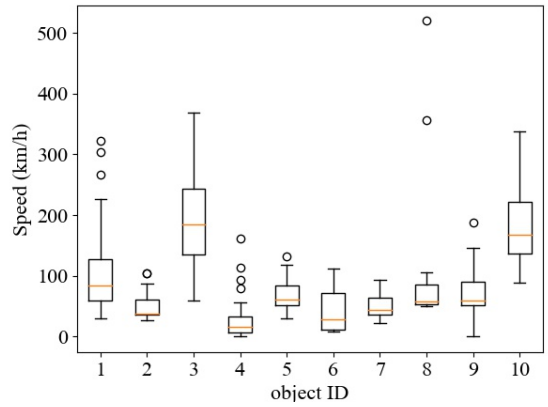
(a) Video 11

(b) Video 12

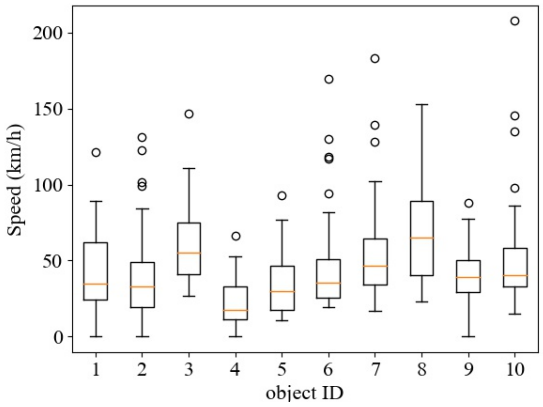
(c) Video 13

(d) Video 14

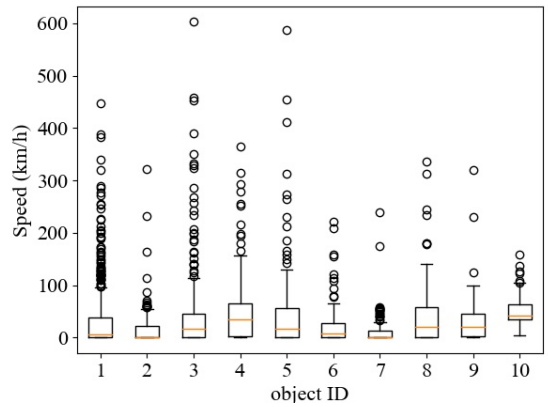
Figure 5.19: Speed distribution for videos 11, 12, 13, and 14 using euclidean tracker.



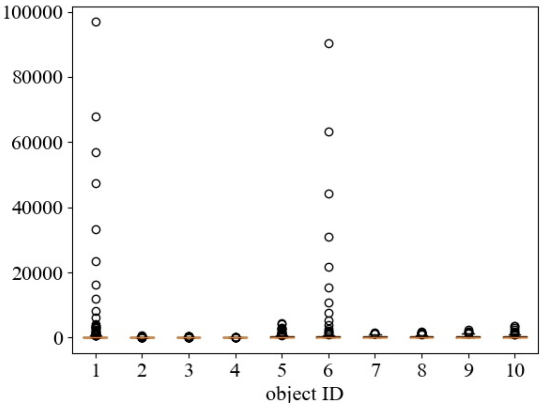
(a) Video 1



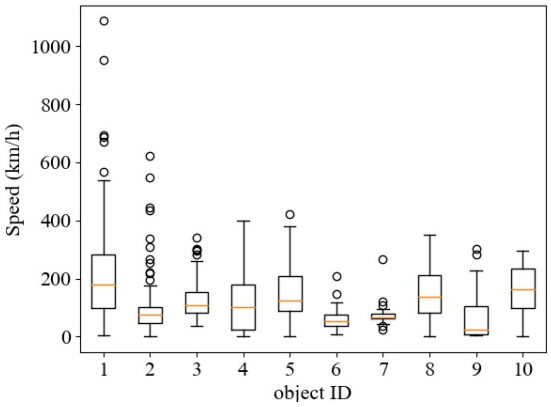
(b) Video 2



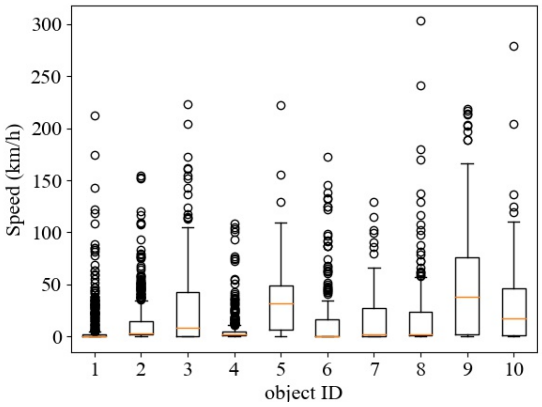
(c) Video 5



(d) Video 6

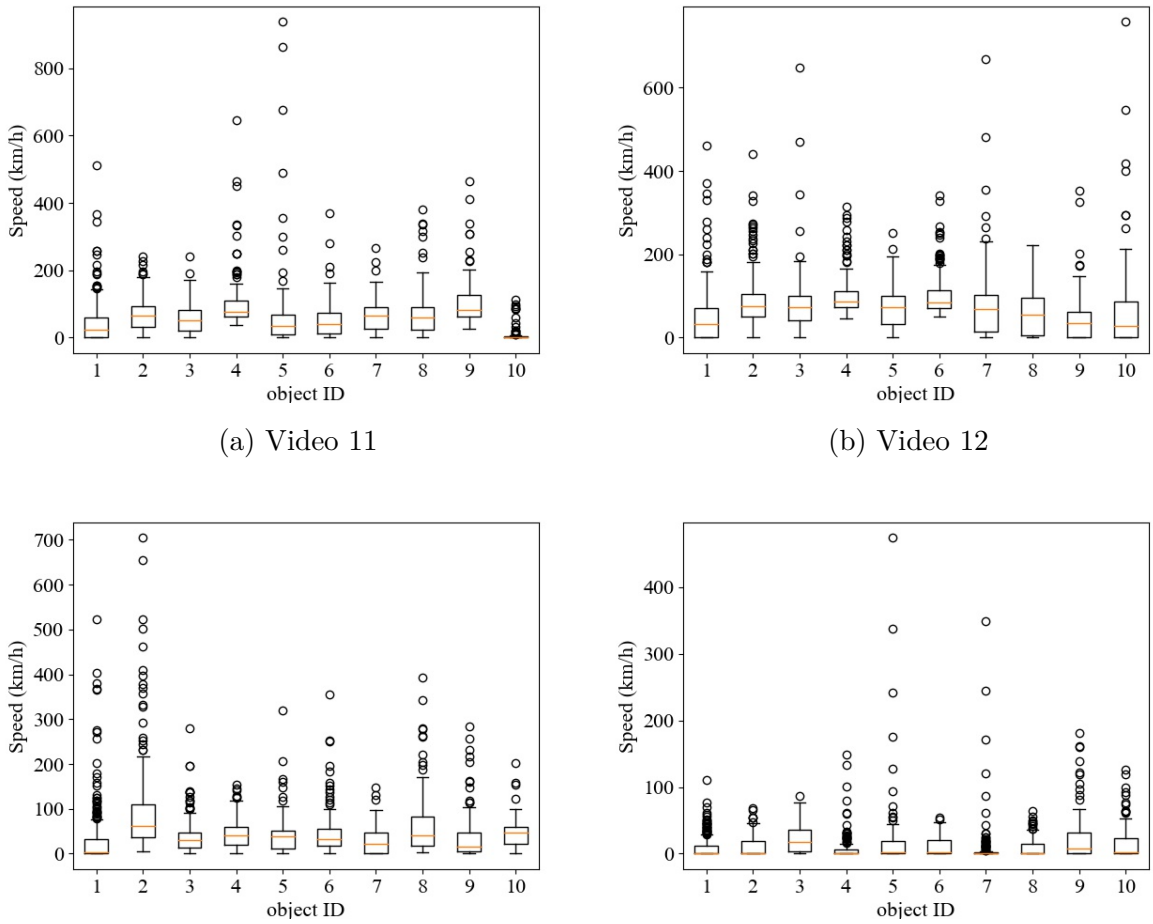


(e) Video 7



(f) Video 9

Figure 5.20: Boxplots of speed for the 10 longest tracks using euclidean tracker on videos 1, 2, 5, 6, 7, and 9.



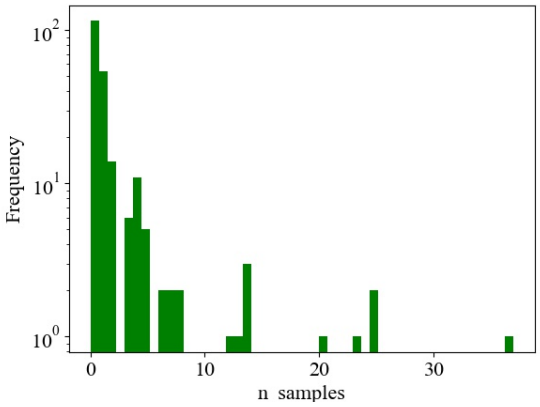
(a) Video 11

(b) Video 12

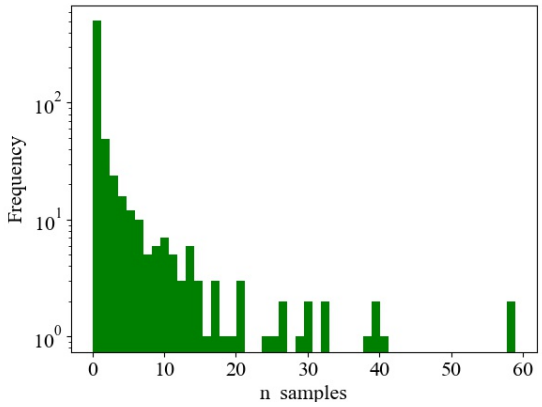
(c) Video 13

(d) Video 14

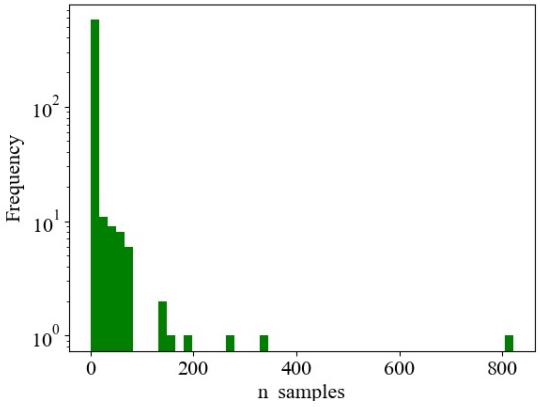
Figure 5.21: Boxplots of speed for the 10 longest tracks using euclidean tracker on videos 11, 12, 13, and 14.



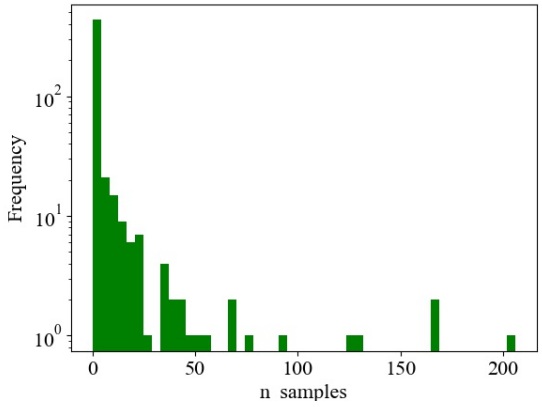
(a) Video 1



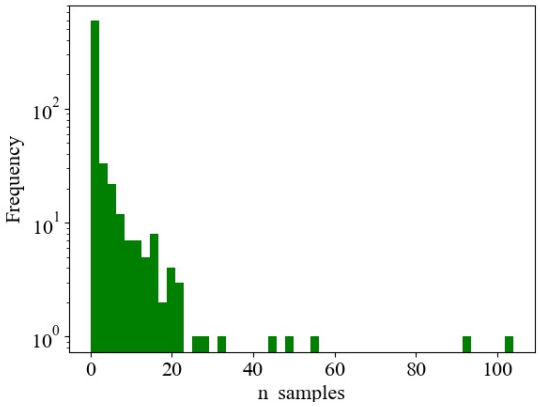
(b) Video 2



(c) Video 5

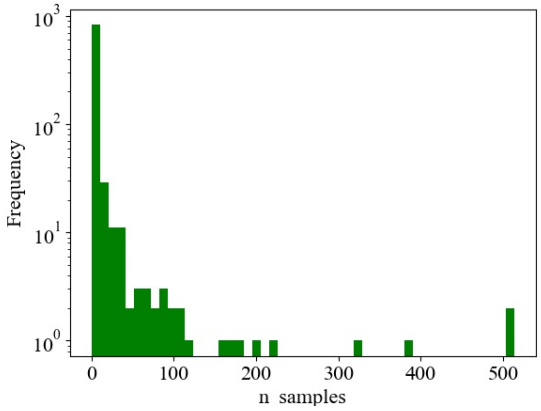


(d) Video 6

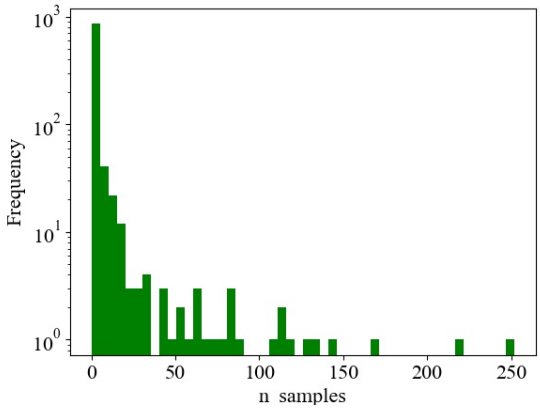


(e) Video 7

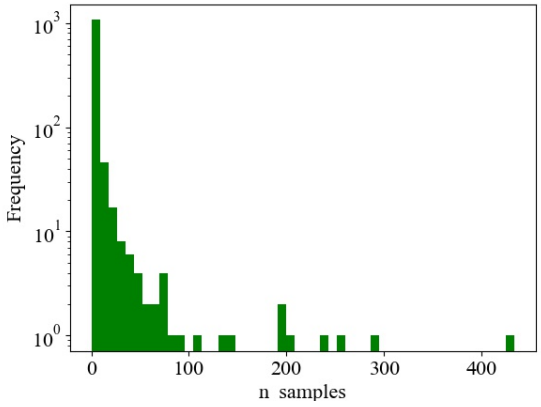
Figure 5.22: Histogram of samples per track using euclidean tracker for videos 1, 2, 5, 6, 7, and 9.



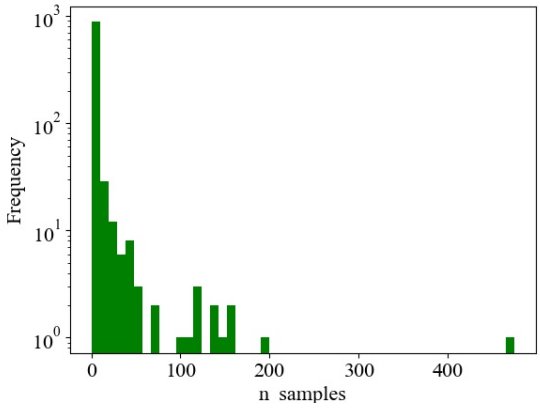
(a) Video 9



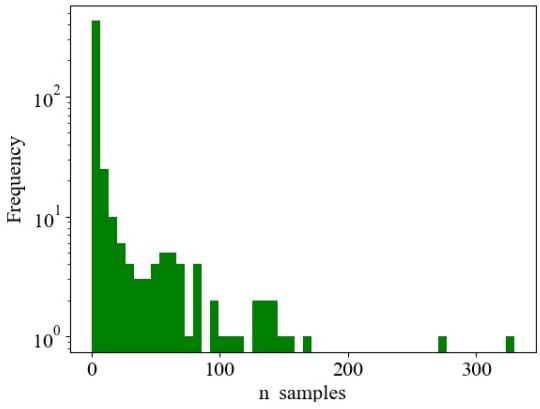
(b) Video 11



(c) Video 12

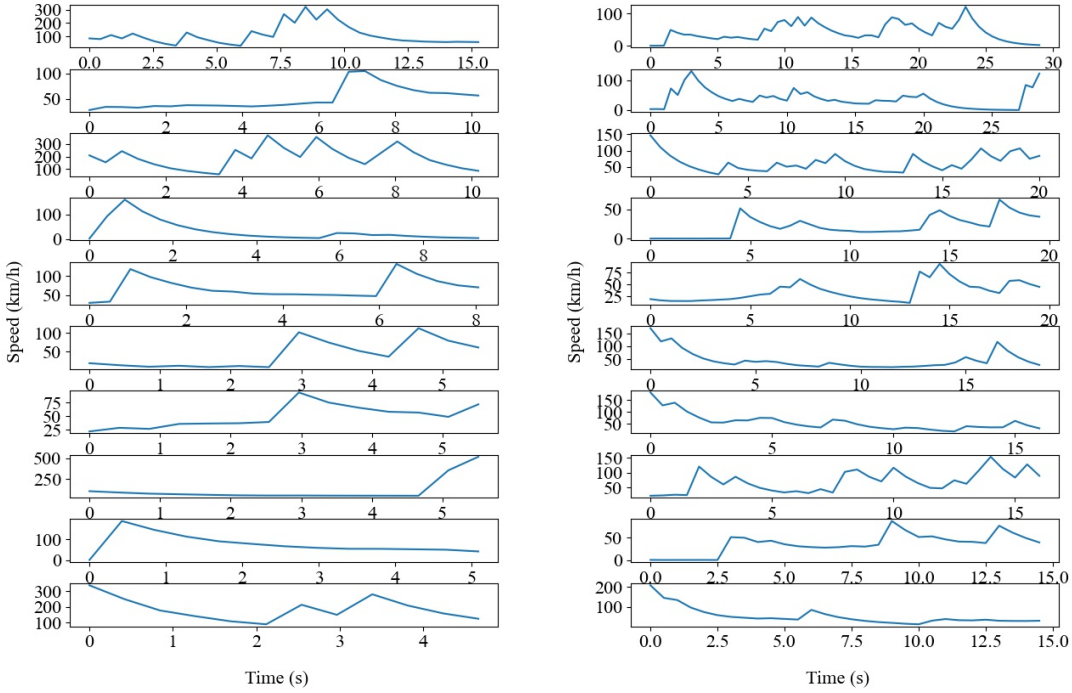


(d) Video 13



(e) Video 14

Figure 5.23: Histogram of samples per track using euclidean tracker for videos 11, 12, 13, and 14



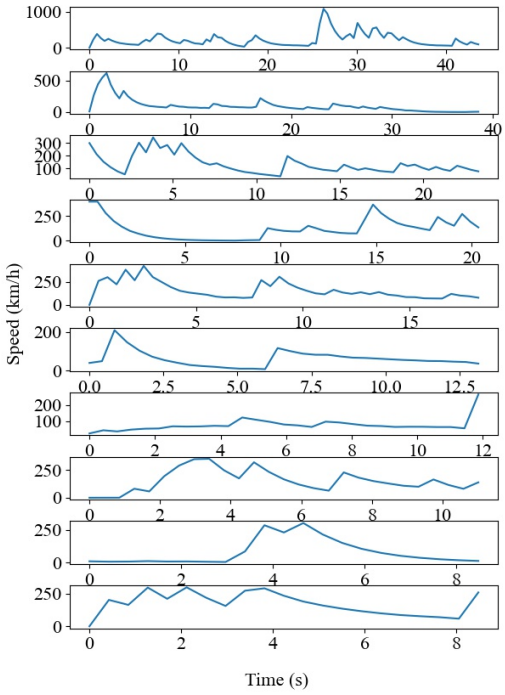
(a) Video 1

(b) Video 2

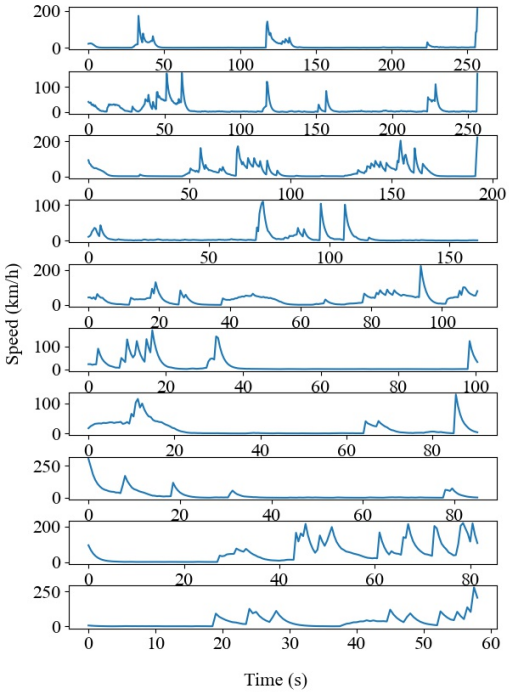
(c) Video 5

(d) Video 6

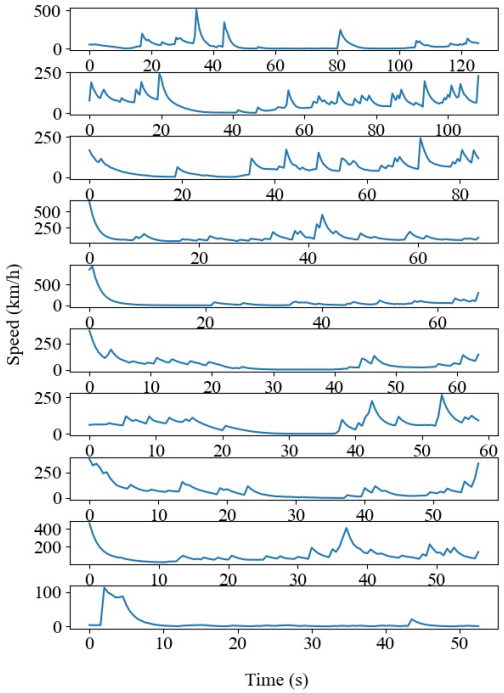
Figure 5.24: Euclidean tracker speed time series for the 10 longest tracks on videos 1, 2, 5, 6.



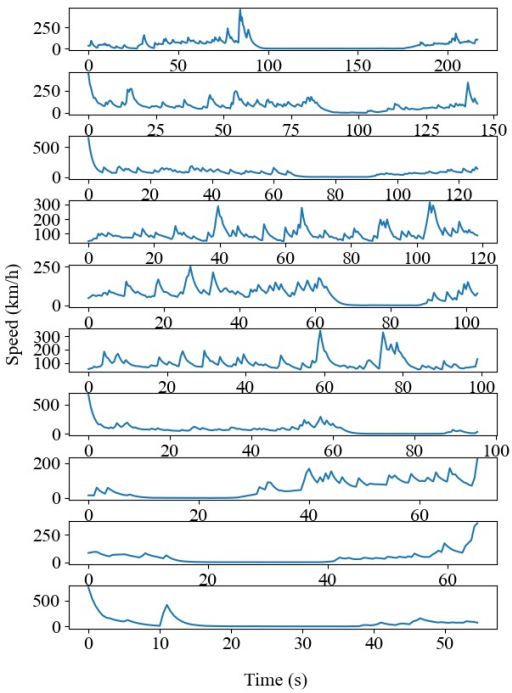
(a) Video 7



(b) Video 9

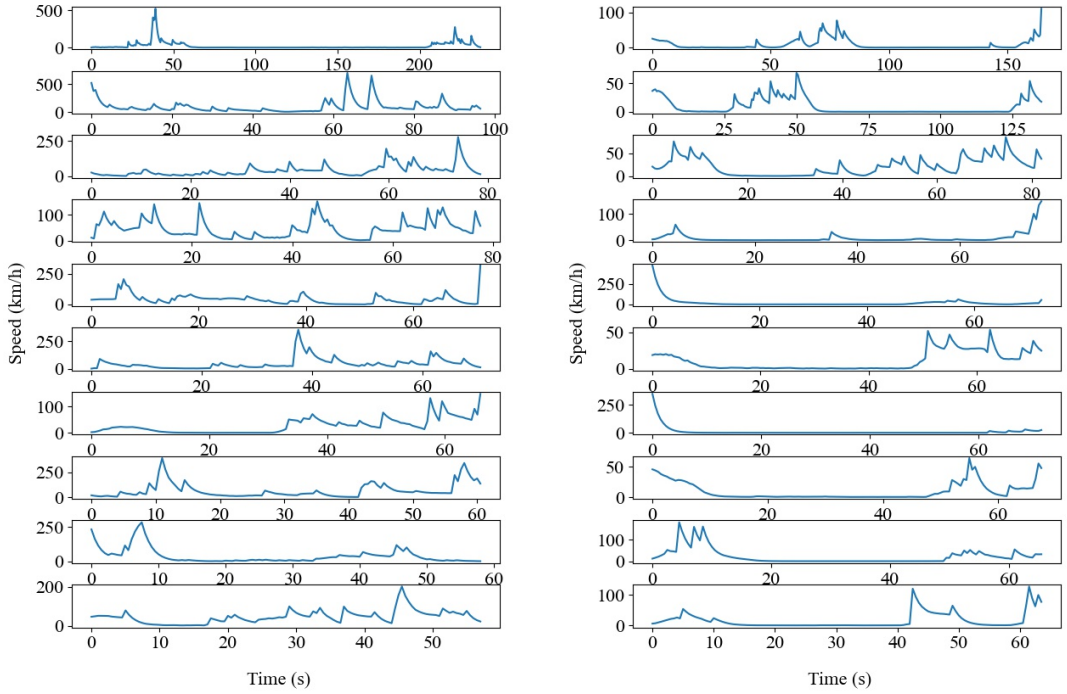


(c) Video 11



(d) Video 12

Figure 5.25: Euclidean tracker speed time series for the 10 longest tracks on videos 7, 9, 11, 12.



(a) Video 13 (b) Video 14

Figure 5.26: Euclidean tracker speed time series for the 10 longest tracks on videos 13 and, 14.

Table 5.4: Statistics computed over vehicles tracks using euclidean tracker.

Registered Vehicles	Mean Samples per Vehicle	Median of Samples per Vehicle	standard deviation samples per vehicle
221	1.846	0.0	4.492
671	2.350	0.0	6.205
614	7.327	1.0	40.721
512	4.980	0.0	18.320
702	2.175	0.0	7.163
905	6.819	0.0	34.355
969	4.413	0.0	18.151
1169	4.654	0.0	23.266
952	4.499	0.0	22.183
516	9.940	1.0	30.615

Table 5.5: Speed statistics for all videos using euclidean tracker.

Speed Samples	Mean Speed (Km/h)	Median Speed (Km/h)	Speed STD (Km/h)	Percentage of Speed Samples > 70 km/h	Percentage of Speed Samples > 100 km/h
408	68.788	36.373	98.829	0.289	0.199
1577	46.994	35.231	43.022	0.197	0.101
4499	50.814	29.481	92.479	0.174	0.117
2550	1406.907	113.450	15741.233	0.607	0.534
1527	155.061	88.454	190.172	0.630	0.446
6171	49.774	18.701	90.911	0.199	0.139
4276	122.166	68.935	171.347	0.492	0.333
5441	106.585	78.694	114.118	0.582	0.352
4283	71.058	40.801	107.023	0.286	0.189
5129	27.123	14.912	49.448	0.075	0.050

Table 5.6: Speed statistics for all videos crossing the virtual lines using euclidean tracker.

Speed Samples	Mean Speed (Km/h)	Median Speed (Km/h)	Speed STD (Km/h)
51	72.303	44.641	72.944
380	33.858	27.882	23.019
304	76.718	46.699	76.344
3400	84.563	1.368	2019.397
248	100.463	78.603	68.332
1946	33.059	15.571	47.390
719	91.475	70.067	109.862
1351	91.876	74.883	81.323
607	69.810	51.377	57.189
349	18.060	10.443	32.400

Chapter 6

Discussion

This chapter describes in detail the results obtained using the proposed workflow that was explained in the previous chapter. Specifically, the main cornerstones of analysis are the homography estimation for camera calibration, YOLOv4 detection model, and both Intersection over Union (IoU) and euclidean tracking implementations.

6.1 Camera Calibration

It can be seen from Figure 5.1 that all evaluated homography estimation methods present no noticeable differences in error. Moreover, videos 1, 2, 5, and 7 have projection errors of approximately 10 pixels using 7 to 8 points correspondences from longitude-latitude to image coordinates. On the other hand, videos 9, 11, 12, 13, and 14 have negligible error values, meaning that estimated image points almost perfectly match the actual chosen coordinates. These videos used only 4 points correspondences for the homography estimation taken from the 4 vertices of the region of interest. However, the video with index 6 had a considerably higher error magnitude, meaning that point correspondences carried more noise and the estimations diverged from the ground truth, affecting the speed estimation capabilities of the system in that particular instance. Observing Figure 5.2, which depicts virtual representations of the street in video 7, for each type of calibration, it can be seen that the proposed method produced a representation that has a direction more in-line with the street than the other methods. The other calibrations have a slight tilt to the left.

In terms of distance errors, Figure 5.3 present similar error patterns as in the projection error plot. Video 6 has the highest error and videos 9, 11, 12, 13, 14 have negligible error magnitudes. It can also be seen that in videos 1 and 2, the proposed method obtained slightly less error magnitudes than both RANSAC and baseline, while in videos 5 and 6 the errors observed were higher. Meanwhile, RANSAC and baseline had no difference in performance whatsoever. Furthermore, all errors in this plot are in the order of centimeters, but it should not ultimately determine the quality of the homography matrix, since distance errors were extracted from a handful of selected points and not from a sufficient number of samples taken at random from the region of interest. The reason for this is that ground truth distances were only available from manually selected points in the calibration.

6.2 YOLOv4 Qualitative Analysis

The detection model YOLOv4 mostly detected all vehicle instances on every video, taken into consideration that all videos were taken in unconstrained environment conditions and image quality correspond to public traffic surveillance cameras, not high-quality hardware. However, when obstacles are present in the image and overlap vehicles, the model easily misses those detections. This limits the performance of the tracking algorithm as it depends on object detections frame by frame to update the tracks. Some instances of the observed obstacles can be seen in figures 4.6 and 4.7. For instance, it can be seen that traffic lights, cables, and even vehicles can occlude other vehicles. It is worth noting that only a qualitative analysis on the detector could be carried out, as no ground truth detections are available for the videos to compute performance metrics and doing so would represent a large amount of time and effort, leaving the opportunity for future work on this component.

6.3 Speed Estimation Using IoU Tracker

The distribution of speed values when cars crossed the virtual lines are mostly condensed within normal driving ranges, according to figures 5.5 and 5.6, of speeds up to 90 km/h. This is not the case for video 6, which has frequent speed measurements in the range of 0 to 50 km/h, but presents outliers that go beyond 100 km/h. It should be noticed that this video in particular presented the highest projection error on the calibration phase.

On videos 6, 9, 11, 12, and 14 it is observed that speeds of almost 0 km/h are high in frequency. This is due to the fact that the cameras are placed next to traffic lights that turn red on certain time-frames in those videos. Hence, vehicles reduce their speed and eventually stop.

Similarly, for the distribution of speeds in figures 5.7 and 5.8 taken from the whole street area, the value ranges are mostly expected on urban streets, which by convention is 70 km/h. However, some videos have densities with inliers that go up to 80 km/h with anomalies that go beyond that magnitude, mainly in videos 5, 7, and 11. In addition, video 6 had major problems in this distribution as well, with values ranging up to 3×10^4 km/h. Moreover, Table 5.2 shows that only videos 6 and 7 had a considerable percentage of speed samples that are over 70 km/h, with 51% and 18.5% respectively, and only video 6 had a noticeable proportion of speeds over 100 km/h, being 37.5% of the whole set of measurements. The rest of videos have negligible or null percentages in speeds beyond 70 km/h.

With a closer inspection to the statistics of each video, also in Table 5.2, it can be seen that the system has registered mean and median values less than 20 km/h for videos 1, 2, 5, 9, 13, and 14. This does not indicate that the system is not performing properly, but rather that the overall statistics are heavily influenced by speed measurements when cars are at full stop temporary or parked on the scene since all cameras were placed on intersections, next to traffic lights. Thus, the previous distributions of speeds give a broader picture when evaluating the behavior of the proposed method.

When looking at the location tracks for each vehicles on figures 5.9, 5.10 and Table 5.1, it is clear that the tracking algorithm present major problems in this area. With a mean number of samples captured per track ranging from 1.846 to 9.94, and different

vehicle IDs ranging from 221 to 1169, it may be that the tracks are being fragmented due to misdetections or by lack of robustness in the method. It should be noted that this tracker finishes a track belonging to a particular vehicle, once it is not detected on a frame. Moreover, the standard deviations indicate that there is also high dispersion in the distribution with ranges from 4, up to 40 samples, each sample taken at half a second. Ideally, as the scenes show only straight streets, the tracks should have similar lengths for all vehicles.

On the other hand, from individual vehicle measurements, figures 5.11, 5.12, and 5.13 show that the tracker is able to record speed changes. For instance when the vehicles are reducing their speed to make a full stop, or when they begin to accelerate. This pattern can be seen clearly on vehicles in videos 5, 6, 11, 12, 13, and 14 that expose soft, downward curves of velocities, then stagnates near 0 and then go up. However, this does not hold for vehicles that present timeseries with upper bounds of speed less than 5 km/h, describing only noise. For instance, most vehicles on videos 2 and 9 show noisy signals. Moreover, the boxplots on figures 5.14 and 5.15 complement the aforementioned timeseries. They show, on videos 5, 6, 9, 12, and 14, that these particular vehicles spent most of their tracking lifecycle on stop, and higher measurements presented as outliers for their very short moving duration. This can be advantageous because vehicle instances can be tracked regardless of their speed, but vehicles that are not of interest and are at full stop on the entire recording will be tracked as well, even if they are not of interest.

6.4 Speed Estimation Using Euclidean Tracker

For the tracking and speed estimation using euclidean distance to assign detections to tracks, every single video presents values that are not expected from urban drivers. Both from the distribution of speeds taken from the whole region of interest, observed in figures 5.18, 5.19, and from distributions of speed recorded in the virtual lines, depicted in figures 5.16, 5.17. For instance, video 6 has outlier values on the virtual lines of up to 60000 km/h. Such high values appear because this implementation has shown to have frequent problems of track identity stealing between instances that do not belong to the same vehicle and that are distant, resulting in incorrect speed measurement, even when the homography projection error is minimal.

Moreover, from Table 5.5, and taking into account the aforementioned distributions, it is clear that a high proportion of samples correspond to speeds greater than 100 km/h, with ranges from 5% for video 14, up to 53.4% for video 6. When comparing the two tracking implementations, it is easy to see that the IoU method is much more robust for speed estimation since it can easily implement thresholds that minimize the chances of assigning tracks to incorrect vehicle detections.

Going further, the euclidean tracker presents the same problem, as the IoU implementation, in terms of not being able to capture complete vehicle tracks and fragmenting to different instances when the object detector fails to locate the vehicles when they are still on the scene. Specifically, Table 5.4 shows that the average location samples on each track for every video ranges from 1.846 to 9.94, but Figures 5.22 and 5.23 show that there is wide variability in the length of the tracks. The tracks should have similar lengths, across vehicle instances, and should not be close to 0 samples within each track.

Looking at individual speed values on figures 5.24, 5.25, and 5.26 of the euclidean tracker, it can be seen various spikes in speed, unlike the IoU tracker which produce softer speed timeseries as seen in figures 5.11, 5.12, and 5.13 that only present noise in very low ranges. Furthermore these spikes from the euclidean version are in speed ranges that are unreal for urban traffic scenes. Finally, the boxplots of the same objects, visualized in Figures 5.20 and 5.21, evidence the problems of this implementation for speed estimation.

Chapter 7

Conclusions

7.1 Performance of the Method

First, for the homography estimation, all three robust methods have proven to be effective, with relatively low projection errors, and no noticeable difference in performance. Moreover, this component of the workflow requires human operation and could be replaced by an automatic deep learning based solution in the future.

Then, the object detection model has shown to be reliable on all videos, but still missed some detections when occlusions are present. The current tracking algorithms implemented in this work are very sensitive to those misdetections, fragmenting vehicle tracks and lowering overall workflow performance.

Furthermore, the speed estimation on the euclidean tracker was very unreliable. It recorded frequent spikes in speed for individual vehicles with speed ranges that are unlikely to happen in urban streets and even impossible for some of the samples captured. On the other hand, the IoU implementation recorded distributions in speed that are well expected on urban areas, near traffic lights. It was also able to successfully track speed changes when vehicles are reducing their speed, at full stop, and when accelerating.

7.2 Evaluation of the Objectives

The workflow indeed only uses a single camera and points from the scene as the only sources of input, without the need for radars or lidars to measure speed. In addition, the video feeds recorded were from live public cameras, so there was no control over setup in this sense, achieving the general objective.

As for specifics, two trackers were implemented and also successfully extended to be able to calculate vehicle speeds. The IoU version managed to produce speed estimations with ranges that are expected on city streets as results show.

Finally, in these speed estimations, homography matrices were also a fundamental part. All three methodologies were tested and have shown to be reliable in practice for transforming image points into world coordinates.

7.3 Challenges

The realization of this work came with many obstacles to overcome. Such a complex system requires adequate hardware, careful software design and implementation, time, and data to evaluate the performance. The resources available were not suited for optimal development, but it was still possible for development as a proof of concept. Some of the challenges faced are described in the following subsection.

7.3.1 Hardware

For a modern object detector, based on deep learning, to perform in near real-time for image sequences, a mid to high end Graphics Processing Unit (GPU) is needed. Otherwise, memory constraints will not allow the model to be loaded, or inference would take long periods of time. Since all the implementation and testing was carried out on a laptop with a low tier GPU, the graphics memory that needed to be allocated for YOLOv4 exceeded its capacity. Thus, some workarounds had to be made.

For instance, the model had to be loaded on conventional RAM instead of the graphics card memory and executed using CPU. This means that massive parallelization provided by CUDA was out of reach, causing a major bottleneck in testing time with processing speeds in less than 1 frame-per-second. Therefore, all detections had to be pre-computed for every video, serialized, and stored on pickle files. Those files would be loaded on every test and detections would be extracted frame by frame and fed to the object tracker. Using this approach, computing all detections took approximately 120 hours, but made all tests much faster.

7.3.2 Software

This project, with the homography estimation, detection, and tracking components, needed a large codebase. Approximately 6261 lines of codes were written. As such, careful review and testing had to be performed on every single component to make sure all behavior worked as expected. In addition, software was also driven by hardware constraints, with initial versions of all components being adapted to the available resources. For instance, all the workflow included object detection at testing time. However, this approach would be proven unfeasible due to low-tier graphics memory and had to be replaced by pre-computed detections. Also, multiple versions of the tracking algorithm were implemented before the final proposed workflow. Initially, all object distances in each frame would be computed on a single matrix and then the tracks would be assigned based on the matrix's indices. Surprisingly, this approach would lead to unexpected behavior for the IoU tracker and had to be replaced by distance comparisons between detections inside loops as the work of [8] proposed.

7.3.3 Data

Although homography estimation, object detection, and object tracking are extensively studied research areas, vehicle speed estimation, using computer vision only, seems not to be a widely studied field, with most publications being clustered within competitions like

Nvidia AI City Challenge [64]. As such, at the time of writing this work, no open access datasets with videos and speed annotations for vehicles were found. Hence, a dataset had to be completely built from scratch to implement and test all the workflow. Specifically, all videos were recorded from live camera feeds available to the public, image and world points were manually annotated, and location metadata for each video were retrieved for camera calibration. As a downside, no speed ground truth was available to assess accuracy, and the performance had to be tested by means of exploratory analysis.

7.4 Lessons Learned

The realization of this work came along with considerable personal knowledge gains. For instance, in the deep learning field with very deep architectures and novel mechanisms to improve the ability of the network to capture patterns and learn feature representations. In addition, object tracking was a completely new field to learn, since detections cannot be associated across frames by the object detectors. However, the most important experience gains came from the engineering aspects of assessing the feasibility of any project given limited resources and modify the workflow configuration when available computing power is less than required. In addition, better planning for future, more efficient implementations in terms of resources, code reuse, and available literature can now be made, now that the initial iteration is complete and project management errors, as well as good practices employed were identified.

7.5 Future Work

There are multiple areas of improvement in this project before it could be implemented in real-world environments. In fact, every single component that the workflow exposes can be improved to achieve better results. Some of these areas are explained in the following subsections.

7.5.1 Data

For future advancement of this work, collaboration with public entities in charge of local security or public surveillance is needed to produce a dataset of videos with speed annotations, and complete object tracks. This would provide the means to assess performance objectively with accuracy metrics, instead of exploratory analysis.

7.5.2 Object Tracking

The next step towards a more robust tracking algorithm is the inclusion of visual features along with the already implemented IoU metric. This would minimize errors derived from wrongly assigning ids to vehicles. In addition, a mechanism to reduce incomplete tracks should be implemented when the object detector fails to find the object. For instance, a method to estimate object position and use it in the event of misdetection on a short span

of frames. Kalman Filters provide a means for that purpose. However, prior information for state transitions should be provided to the model for proper initialization.

7.5.3 Object Detection

Object detection is continuously pushing the state of the art with better neural architectures and training methods. The next implementation for the detector should be robust to low light conditions and occlusions, which would help the tracker minimize performance losses due to missing detections. In addition, the model should be efficient and fast enough to run in real-time with modest hardware specifications. YOLO is said to be able to perform in real-time [6] given a mid-tier graphics card. However, in the tests carried out in this work, minor occlusion problems were enough to hurt its performance.

7.5.4 Homography Estimation

A deep learning based method for homography matrix estimation should replace the current version used in this work to minimize errors derived from human interaction, as it needs manually extracted point correspondences between image and longitude-latitude plane. For instance, the work of [13] used an unsupervised neural network for such purpose, achieving good results.

Bibliography

[1] W. Boulila, M. Driss, M. Al-Sarem, F. Saeed, and M. Krichen, "Weight initialization techniques for deep learning algorithms in remote sensing: Recent trends and future perspectives," *arXiv preprint arXiv:2102.07004*, 2021.

[2] Z. Soleimanitaleb, M. A. Keyvanrad, and A. Jafari, "Object tracking methods: A review," in *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 2019, pp. 282–288.

[3] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[5] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[7] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," *arXiv preprint arXiv:2103.14030*, 2021.

[8] E. Bochinski, V. Eiselein, and T. Sikora, "High-speed tracking-by-detection without using image information," in *International Workshop on Traffic and Street Surveillance for Safety and Security at IEEE AVSS 2017*, Lecce, Italy, Aug. 2017. [Online]. Available: http://elvera.nue.tu-berlin.de/files/1517Bochinski2017.pdf

[9] E. Bochinski, T. Senst, and T. Sikora, "Extending iou based multi-object tracking by visual information," in *IEEE International Conference on Advanced Video and Signals-based Surveillance*, Auckland, New Zealand, Nov. 2018, pp. 441–446. [Online]. Available: http://elvera.nue.tu-berlin.de/files/1547Bochinski2018.pdf

[10] P. Chu and H. Ling, "Famnet: Joint learning of feature, affinity and multi-dimensional assignment for online multiple object tracking," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 6172–6181.

[11] S. Sun, N. Akhtar, H. Song, A. Mian, and M. Shah, "Deep affinity network for multiple object tracking," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 1, pp. 104–119, 2019.

[12] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Deep image homography estimation," 2016.

[13] T. Nguyen, S. W. Chen, S. S. Shivakumar, C. J. Taylor, and V. Kumar, "Unsupervised deep homography: A fast and robust homography estimation model," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2346–2353, 2018.

[14] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[16] "Speed and road crashes," 2018.

[17] J. C. Falcocchio and H. S. Levinson, *The Costs and Other Consequences of Traffic Congestion*. Cham: Springer International Publishing, 2015, pp. 159–182. [Online]. Available: https://doi.org/10.1007/978-3-319-15165-6_13

[18] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[19] R. Farber, *CUDA application design and development*. Elsevier, 2011.

[20] A. Brock, S. De, S. L. Smith, and K. Simonyan, "High-performance large-scale image recognition without normalization," *arXiv preprint arXiv:2102.06171*, 2021.

[21] X. Zhou, V. Koltun, and P. Krähenbühl, "Probabilistic two-stage detection," *arXiv preprint arXiv:2103.07461*, 2021.

[22] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," *arXiv preprint arXiv:2011.08036*, 2020.

[23] H. Pham, Z. Dai, Q. Xie, M.-T. Luong, and Q. V. Le, "Meta pseudo labels," *arXiv preprint arXiv:2003.10580*, 2020.

[24] H. Touvron, A. Vedaldi, M. Douze, and H. Jégou, "Fixing the train-test resolution discrepancy: Fixefficientnet," *arXiv preprint arXiv:2003.08237*, 2020.

[25] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*. PMLR, 2016, pp. 173–182.

[26] W. Han, Z. Zhang, Y. Zhang, J. Yu, C.-C. Chiu, J. Qin, A. Gulati, R. Pang, and Y. Wu, "Contextnet: Improving convolutional neural networks for automatic speech recognition with global context," *arXiv preprint arXiv:2005.03191*, 2020.

[27] T. Wolf, J. Chaumond, L. Debut, V. Sanh, C. Delangue, A. Moi, P. Cistac, M. Funtowicz, J. Davison, S. Shleifer *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 38–45.

[28] S. Sharma and S. Sharma, "Activation functions in neural networks," *Towards Data Science*, vol. 6, no. 12, pp. 310–316, 2017.

[29] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for Activation Functions," oct 2017. [Online]. Available: http://arxiv.org/abs/1710.05941

[30] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.

[31] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[32] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[33] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 886–893 vol. 1.

[34] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[39] H. S. M. Coxeter, *Projective geometry*. Springer Science & Business Media, 2003.

[40] R. V. Petrescu, "Presents some aspects and applications of projective geometry," *Available at SSRN 3445158*, 2019.

[41] G. F. Page, "Multiple view geometry in computer vision, by richard hartley and andrew zisserman, cup, cambridge, uk, 2003, vi 560 pp., isbn 0-521-54051-8. (paperback £44.95)," *Robotica*, vol. 23, no. 2, p. 7–7, 2005.

[42] E. Dubrofsky, "Homography estimation," 2009.

[43] P. Viola, M. Jones *et al.*, "Robust real-time object detection," *International journal of computer vision*, vol. 4, no. 34-47, p. 4, 2001.

[44] P. Piccinini, A. Prati, and R. Cucchiara, "Real-time object detection and localization with sift-based clustering," *Image and Vision Computing*, vol. 30, no. 8, pp. 573–587, 2012.

[45] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[46] R. Padilla, S. L. Netto, and E. A. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. IEEE, 2020, pp. 237–242.

[47] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," *arXiv preprint arXiv:2010.16061*, 2020.

[48] M. Z. Islam, M. S. Islam, and M. S. Rana, "Problem analysis of multiple object tracking system: A critical review," *IJARCCE*, vol. 4, no. 11, pp. 374–377, 2015.

[49] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–10, 2008.

[50] L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang, and S. Lyu, "Ua-detrac: A new benchmark and protocol for multi-object detection and tracking," *Computer Vision and Image Understanding*, vol. 193, p. 102907, 2020.

[51] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[52] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[53] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[54] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[55] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[56] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2020, pp. 390–391.

[57] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[58] A. Agarwal, C. Jawahar, and P. Narayanan, "A survey of planar homography estimation techniques," *Centre for Visual Information Technology, Tech. Rep. IIIT/TR/2005/12*, 2005.

[59] N. Mahamkali and V. Ayyasamy, "Opencv for computer vision applications," 03 2015.

[60] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[61] D. D. Morrison, "Optimization by least squares," *SIAM Journal on Numerical Analysis*, vol. 5, no. 1, pp. 83–88, 1968.

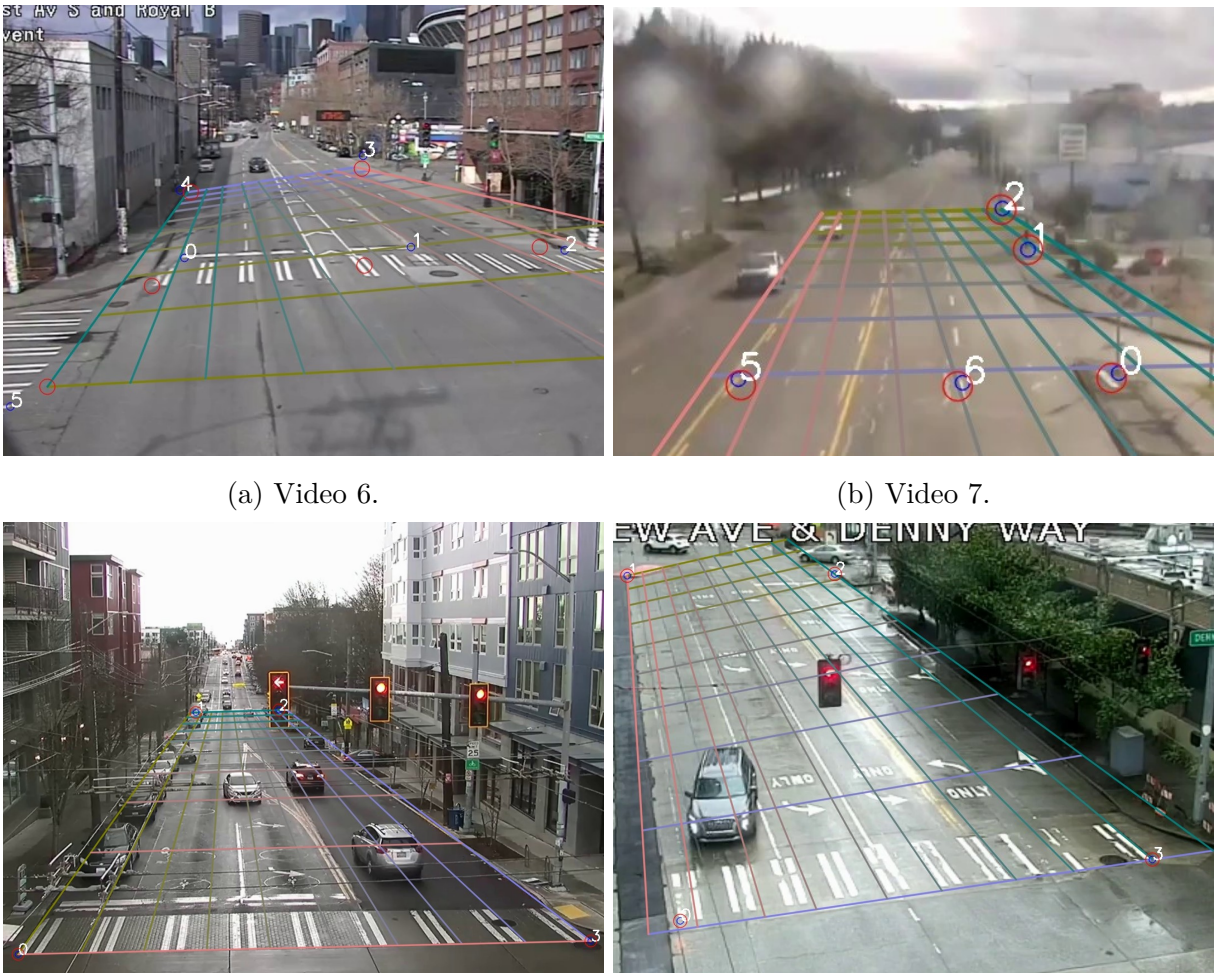
[62] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2017.

[63] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8759–8768.

[64] M. Naphade, S. Wang, D. C. Anastasiu, Z. Tang, M.-C. Chang, X. Yang, Y. Yao, L. Zheng, P. Chakraborty, C. E. Lopez, A. Sharma, Q. Feng, V. Ablavsky, and S. Sclaroff, "The 5th ai city challenge," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2021.

Appendices

.1 Appendix 1.



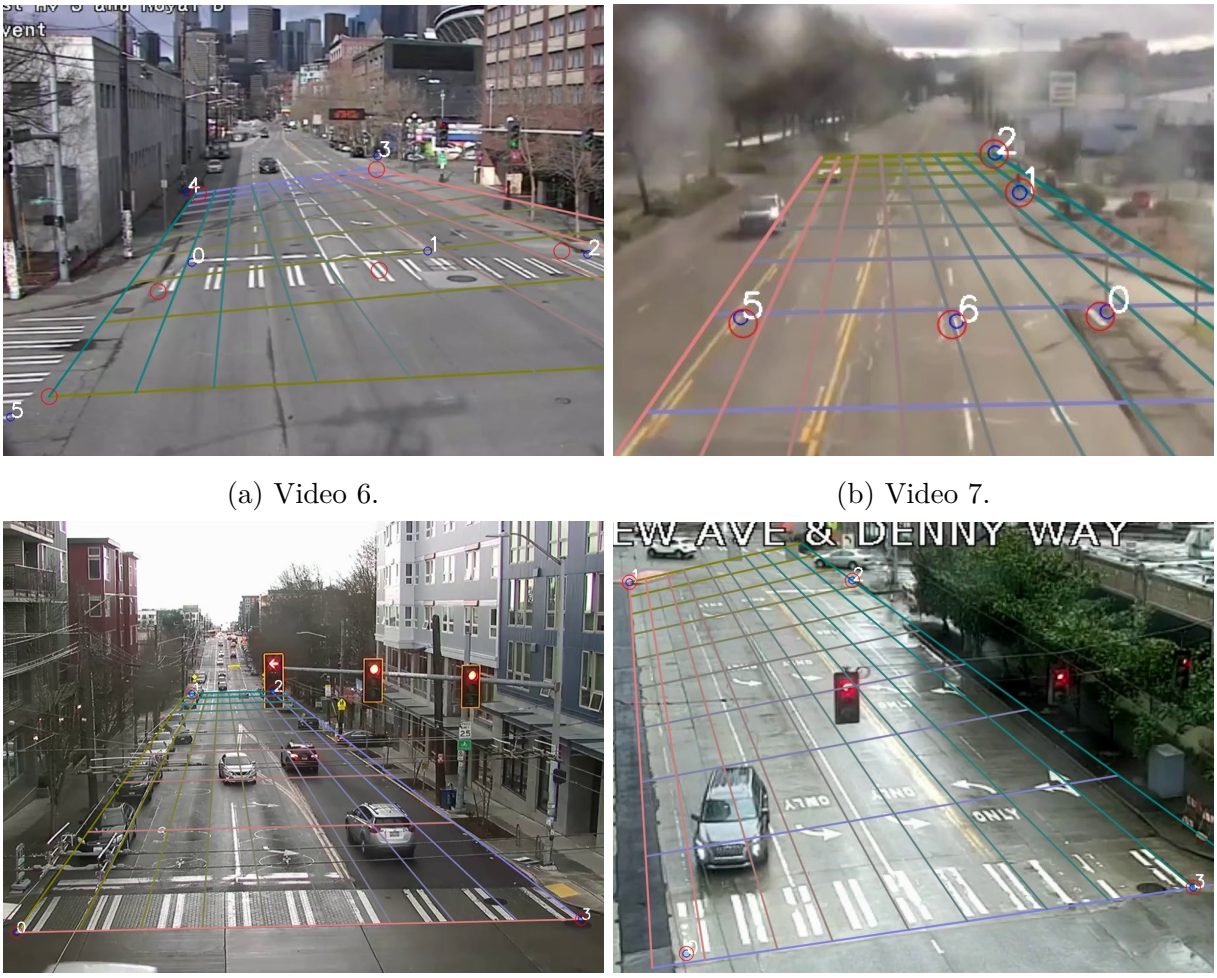
(a) Video 6.

(b) Video 7.

(c) Video 9.

(d) Video 14.

Figure 1: Base calibrations for the most representative videos. Blue circumferences indicate ground truth, while red circumferences indicate image point estimations. The grid is a virtual model of the street.



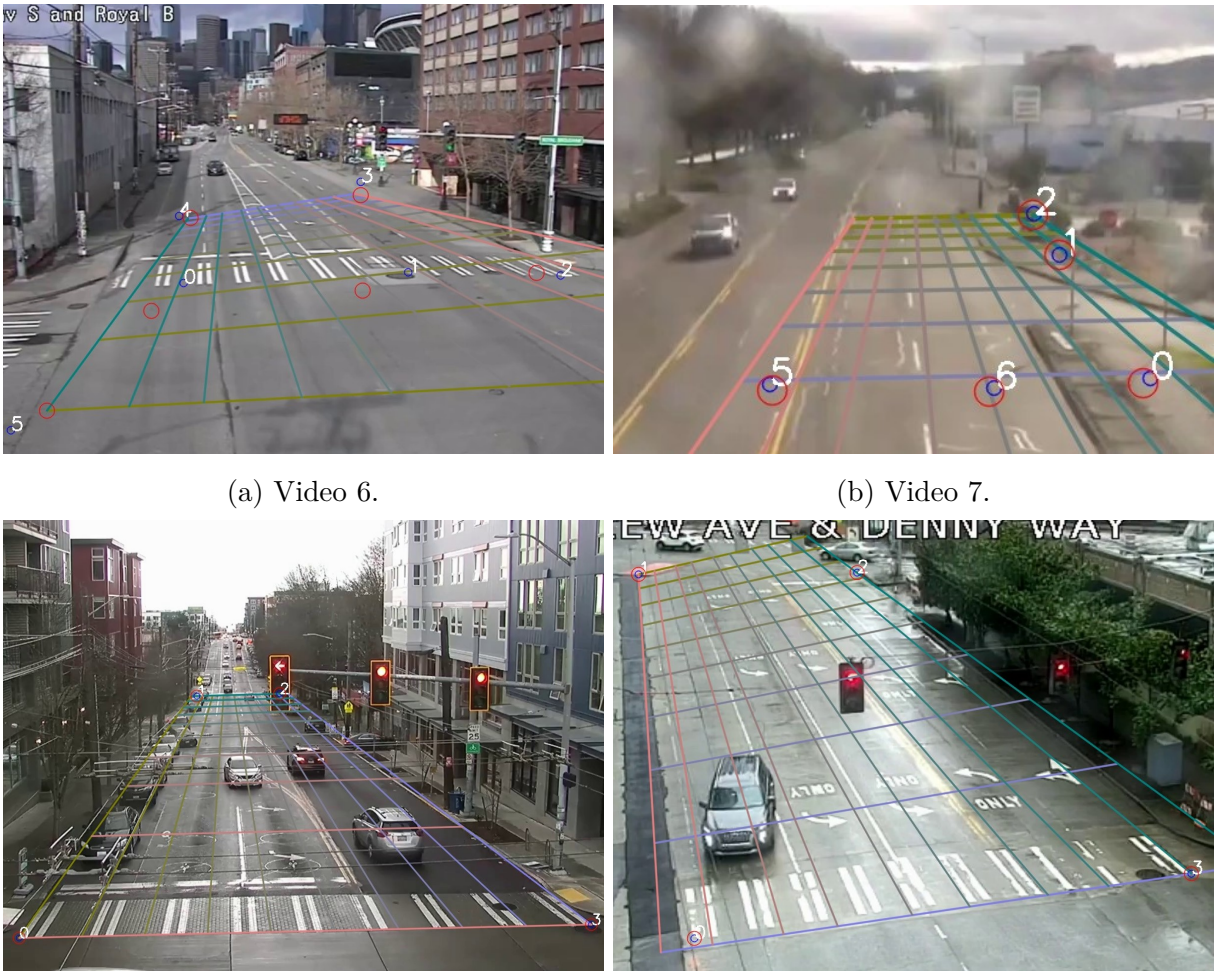
(a) Video 6.

(b) Video 7.

(c) Video 9.

(d) Video 14.

Figure 2: RANSAC calibrations for the most representative videos. Blue circumferences indicate ground truth, while red circumferences indicate image point estimations. The grid is a virtual model of the street.



(a) Video 6.

(b) Video 7.

(c) Video 9.

(d) Video 14.

Figure 3: Proposed method calibrations for the most representative videos. Blue circumferences indicate ground truth, while red circumferences indicate image point estimations. The grid is a virtual model of the street.