Institut für
Mechatronische Systeme
imes Leibniz Universität Hannover

Leibniz
Universität
Hannover

Object's Motion Estimation from Observer's Monocular Vision

Master Thesis M-08/21-1057

Fang Xiao
Matrikelnummer 10017948

Hannover, 16. August 2021

First Examiner	Dr.-Ing. Mark Wielitzka
Second Examiner	Prof. Dr.-Ing. Eduard Reithmeier
Supervisor	Karl-Philipp Kortmann, M.Sc.
	Priv.-Doz. Dr.-Ing.habil. Dirk Joachim Lehmann

Ich, Fang Xiao, versichere hiermit, dass die vorliegende Arbeit selbstständig verfasst wurde, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen hat.

Hannover, 16. August 2021

(Fang Xiao)

Institut für
Mechatronische Systeme
Leibniz Universität Hannover
imes

Master Thesis

Mr. Fang Xiao, Matr. No. 10017984

Object Motion Estimation from Observer's Monocular Vision

(Dt.: Schätzung von Objektbewegungen aus der monokularen Sicht des Beobachters)

Background:
The collision avoidance system (CAS) has been bearing more importance due to the thriving of autonomous driving (AD) in recent years. There are several favored technologies that have been developed to adapt to more real-world conditions. And one of them is monocular vision (MV), which utilizes visual information collected by a single camera for decision making of the CAS. There are stubborn drawbacks in MV, e.g. the scale ambiguity, and difficulties in the so-called egomotion compensation. Nonetheless, it's still welcomed in the industrial world due to the lower cost and convenient practice. Also, the prosperity of Deep Learning (DL) methods in recent years are giving the MV much potential to be improved. It is desired that a satisfying movement estimation of the objects of interest (OI) can be accomplished with solely MV with reasonable speed for practical use.

The goal of this thesis is to investigate algorithms related to the movement/motion estimation (ME), including object detection and tracking, as well as egomotion estimation, and selectively perform modification and integration in a plausible way. Specifically, the prediction (forecasting) of the near-future object position (relative to moving camera itself) need to be considered and addressed. "Near-future" here means to predict within a time window of up to one second in the future, depending on the context's dynamic.

Task:
Within the scope of this thesis, a new method for MV based ME should be developed, optimized and evaluated based on the state of the art. The task is divided into the following subtasks:

1. Literature research on the current state of the art in MV based ME (with focus on DL methods).
2. Research and reasoned selection of one or more suitable reference data sets from the literature. Especially the availability of published analyses of the chosen data by means of MV based ME should be considered.
3. Selection and adoption / extension of at least one MV based ME method. The choice between several methods should be based on the literature research and comprehensible criteria. The extensions should address at least one of the following issues: position/state forecasting, object tracking, egomotion estimation.
4. Implementation of the chosen and extended method in a suitable programming language and preparation of an (graphical) user interface.
5. Quantitative comparison of the own method with a model or published results from literature (e.g *CVPR2017 Velocity Estimation Challenge*) on the chosen data set. The evaluation metrics need to be explained and chosen reasonably and should consider the addressed extension(s) (see 3.). If no competing/benchmarking model is available, this task can be substituted by an extended (hyper-) parameter study (optimization) of the own model.
6. Preliminary conception of an up-following ME forecasting with a time horizon up to one second.

Duration: 900 h
Start Date: 26.02.2021 Deadline: 26.08.2021

Supervisors: PD Dr. Dirk J. Lehmann (IAV),
Karl-Philipp Kortmann, M. Sc. (imes)

....
(Dr.-Ing. Mark Wielitzka) (Fang Xiao)
(First Examiner)

Abstract

In this thesis, objects' distance and velocity relative to the camera are estimated based on monocular video data, where object detection, tracking, depth map estimation are conducted continuously. The velocity, defined as the derivative of position with time, can then be acquired.

A series of neural networks are chosen to perform the sub-tasks mentioned above, respectively. Considering that there are differences between sub-tasks, the choice of end-to-end system is discarded. Therefore, the networks for components are independently selected and trained, which then collaboratively produce the final motion estimates as a complete object motion estimation system.

Quantitative evaluation is conducted firstly on each sub-task separately to locate the advantages and weaknesses for the pipeline, and then on the overall pipeline to test the feasibility of this system. Qualitative results are also analyzed, which supplement the practicality that is not thoroughly covered by the quantitative results.

Although we have not achieved outstanding scores in benchamarking, the highlight of our system lies in the practicality that most of the the network's training is unsupervised, i. e. requires no labeling. Also, the pipeline has demonstrated flexibility that allows convenient substitution for each component, which is vital for (continuous) future updates.

Contents

Glossary

Acc	accuracy
BN	batch normalization
BBOx	bounding box
CS	coordinate system
CNN	convolutional neural network
CV	computer vision
CPU	central processing unit
CSPNet	cross-stage partial network
Det	detection
D-IOU	distance-intersection-over-union
DL	deep learning
disp	disparity
DME	depth map estimation
EST	estimated
FPS	frame per second
FE	feature extractor
FP	false positive
FN	false negative
FC	fully-connected
GPU	graphics processing unit
GT	ground truth
IOU	intersection over union
ID	identity
IA	identity association
KF	Kalman filter
LiDAR	light detection and ranging
MOT	multi-object tracking
MLP	multi-layer perceptron
NN	neural network

NMS	non-maximum suppression
OD	object detection
OMD	object motion derivation
OME	(the overall system of) object motion estimation
PA	path aggregation
pp	principal point
PH	projection head
Re-ID	re-identification/re-identify
RGB	red-green-blue color mode
ResNet	residual network
ResBlock	residual block
RAE	relative absolute error
RSE	relative squared error
RMSE	root-mean-square error
RMSLE	root-mean-square log error
IDSW	(number of) identity switch
SOT	single-object tracking
SPP	spatial pyramid pooling
TBD	tracking by detection
TP	true positive
TN	true negative
W-ResNet	wide residual network

1 Introduction

1.1 Motivation

Estimating of objects' movement, including spatial position and speed, serves a crucial role in many areas, e. g. autonomous driving, scene understanding and robotics. While the relative distance demonstrates the location of surroundings, the perception of dynamic motion can help extend this perception to a near future. There exist two fundamental principles distinguished by the sensor type, each of which has its own methods, i. e. image- or point-cloud-based techniques.

Stereo vision-based depth estimation, mimicking how human eyes perceive the surroundings from two RGB images taken simultaneously by a stereo camera set, is one of the image-methods. A traditional taxonomy concluded by SCHARSTEIN ET AL. (2001) [SSZ01] is: Same regions between two images are recognized by matching the hand-crafted features (e. g. FAST feature detector proposed by ROSTEN AND DRUMMOND (2006) [RD06]), then a pixel-wise depth map is derived through triangulation. Later works, e. g. by ZAGORUYKO and KOMODAKIS (2015) [ZK15], adopted deep-learning (DL) techniques and treated the feature matching as a one-step data-driven learning task. More recently, new DL-approaches, e. g. the SfMLearner by ZHOU ET AL. (2017) [Zho+17], has freed the task from explicit image-pair feature extraction, enabling the NNs to learn implicit features directly and estimate the depth information from individual image(s), i. e. the so-called *monocular vision*.

The depth estimation based on light detection and ranging (LiDAR), inspired by environment perception of animals like bats, also receives a warm welcome in this field. Instead of passively accepting the visual cues from the environment, a LiDAR system actively emits light pulses and receives their reflections from the surroundings. It then converts the received light pulses into the point cloud, out of which the 3D position, reflection intensity ,and even surface property of the object can be derived. LiDAR data can be utilized for depth estimation just as monocular vision, e. g. QIU ET EL. (2019) [Qiu+19] proposed a point-cloud-based (monocular-vision-guided) NN that estimated the dense depth map.

Inspired b KAMPELMÜHLER ET AL. [KMF18] (2018), this work splits object-wise motion estimation into

- object detection (OD),
- multi-object tracking (MOT),
- depth map estimation (DME),
- and finally the object-wise motion derivation (OMD)

Instead of stacking complicated data sources, this work merely takes single-image as input with minimum data processing complexity.

The involved NN models are mainly trained with KITTI dataset [Gei+13]. The evaluations of *tracking* and *depth map estimation* are conducted in KITTI, while *CVPR'2017 vehicle velocity estimation challenge* (VeloChallenge) is utilized for the evaluation of *object relative velocity*.

1.2 Related Work

As stated in Section. 1.1, the complete objects' motion estimation over time covers aspects of object detection (OD), multi-object tracking (MOT), and depth map estimation (DME) and object motion derivation (OMD), this section will discuss related work in these fields.

The OD is one of the most heavily studied domains in deep learning (DL). There are traditional algorithms based on hand-crafted features, e. g. the histogram of oriented gradient (HOG) feature proposed by N. Dalal, N. and B. Triggs (2005). Compared to DL-based methods, they are generally outperformed in terms of accuracy, speed and adaptiveness to unfamiliar scenes. The DL-based methods for OD are divided into one- and two-stage detectors. The R-CNN family: *R-CNN* [Gir+14], *Fast R-CNN* [Gir15] and *Faster R-CNN* [Ren+17], are representatives for two-stage detectors, as they insert an additional module to propose regions of interest (ROIs) between feature extraction and final regression. The one-stage detectors, such as YOLO-family(*YOLOv3* [RF18], *YOLOv4* [BWL20], etc.), *SSD* [Liu+16], do not explicitly distinguish the regions of background from objects, but directly perform regression tasks and use Jaccard index to determine the error/dissimilarity (see Fig. 1.1).

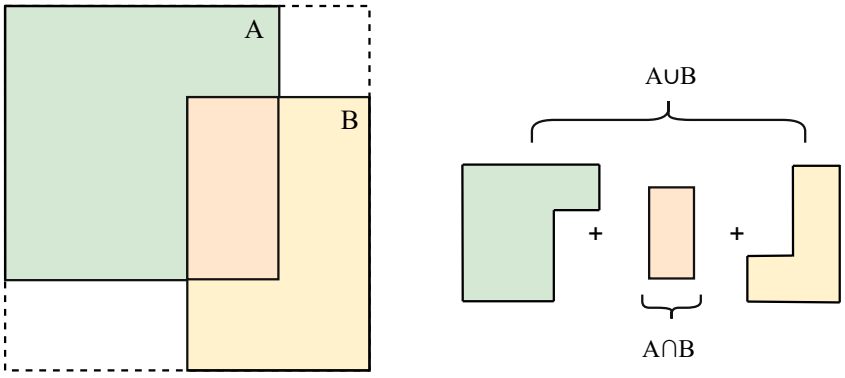


Figure 1.1: Illustration of Jaccard index. A measurement for the (dis-)similarity between two sets. When it comes to two areas, it is identical to the ratio of *intersection over union* (IOU), i. e. the ratio of $\mathrm{A} \cap \mathrm{B}$ to $\mathrm{A} \cup \mathrm{B}$.

The comparison between one- and two-stage detectors is demonstrated in Fig. 1.2. The one-stage detectors have faster inference speed, while they suffer from inaccuracy when it comes to smaller objects. There are also novel models, e. g. *CenterNet* [Dua+19], which treats object locations as *points* instead of *bounding boxes* (BBoxes) and predicts heatmap with peaks to indicate the object position. They perform in even less inference time and can adapt to more crowded scenarios. Object tracking is split into single and multiple object tracking (SOT and MOT), which tracks only one or multiple objects in each image. Traditional algorithms, e. g. *MOSSE* [Bol+10] and *KCF* [Hen+15] tracker, focus on the SOT task, where they use the kernel initialized by the template patch to perform correlation with target image. The tracked area is determined by correlation score and the kernel will be updated progressively. The later work based on DL, such as *SiamRPN* [Li+18] and *SiamDW* [ZP19], achieved more robustness as the network can learn richer and more complex features for matching.

When it comes to MOT, simply stacking the SOT-trackers can form an MOT-tracker, but the difficulty lies in re-identification (Re-ID), i. e. correctly assigning a unique ID to the same object overtime. The *DeepSort* tracker proposed by WOJKE ET AL. (2017) [WBP17], is a pioneer work for MOT, which depends on the expert system to coordinate the tasks of ID initialization, restoration and destruction, etc. It also features fast inference time as it only uses Kalman Filters to model the track and a lightweight feature extractor for appearance information. ZHANG ET AL. (2020) [Zha+20] presents a DL-based algorithm *FairMOT*: It constructs a model end-to-end trainable that combines the *CenterNet* detector and feature extractor to multitask the OD and Re-ID.

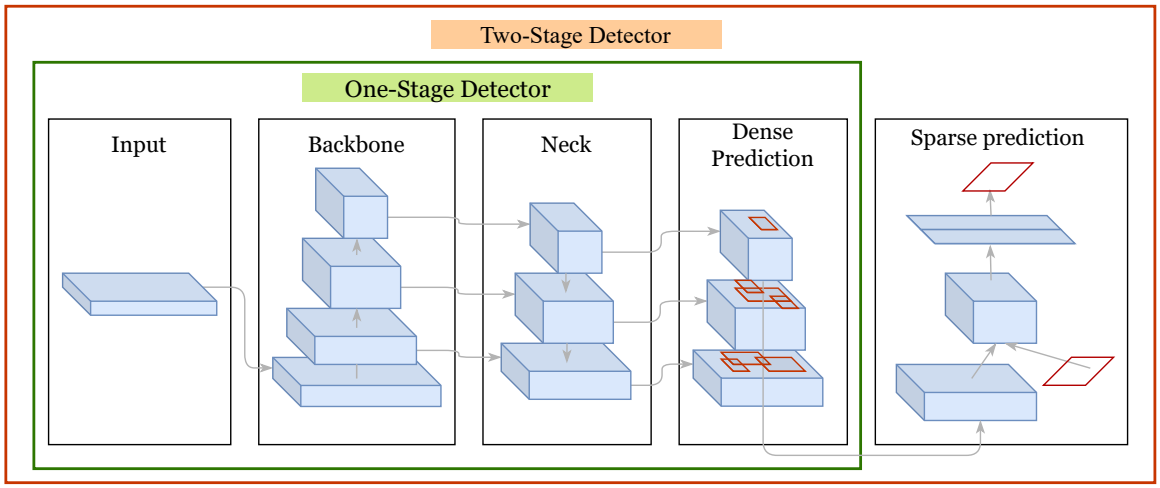


Figure 1.2: Comparison between the general structure of one-stage and two-stage detectors. The former detectors estimate final detection results directly by the extracted feature, while the latter use dense prediction merely as a pre-selection step (to propose region of interest). The arrows indicates how the information flows through the network.

As for the monocular DME, it estimates the depth map,i. e each pixel is the depth value, from a RGB image. Although there are methods using additional training data of stereo image-pairs, e. g. the work by ŽBONTAR AND LECUN (2016) [ŽL16], or LiDAR data [KSL17], these kinds of ground-truth (GT) are not convenient to obtain in practice. Therefore, the self-supervised-learning methods manifest greater practicability, e. g. the *SfMlearner* [Zho+17] utilizes only monocular image, and the work by GORDON ET AL. (2019) [Gor+19] takes a further step to learn the camera intrinsic parameters.

However, it is still an ill-posed problem for object-wise motion estimation, since it must combine OD, MOT, and DME together to produce the final estimates on individual objects, and then additionally derive the velocity. A typical pipeline, like in [KMF18] or [Son+20], treats the problem as a regression task after DME or optical flow estimation to concatenate a small multi-layer perception (MLP) to derive the individual depth and velocity estimates. However, the final MLP makes the pipeline dependent of GT data, constraining the capability for unfamiliar scenes.

1.3 Structure of the Thesis

This thesis is organized as follows:

- Chapter 2 introduces the fundamental knowledge as well as general concepts regarding the research.
- Chapter 3 illustrates the mechanisms of specific algorithms chosen for each module in this work.
- Chapter 4 elaborates the motion estimation pipeline built on the combination of the specified algorithms.
- Chapter 5 introduces the evaluation setup and process, including the dataset choices, evaluation methods, and metrics. Main contributions are highlighted.
- Chapter 6 discusses the evaluation results and provides insight for future research.

2 Fundamentals

This chapter is organized as follows:

- Section 2.1 introduces the geometric knowledge involved in this work.
- Section 2.2 illustrates the Deep Learning basics heavily used in this work, including convolutional neural network and residual network.
- Section 2.3 defines two visual recognition tasks, namely object detection and tracking.
- Section 3.4 gives a general introduction to the motion components, including egomotion, and relative distance and velocity.

2.1 Basic Geometry

In this section, geometric knowledge involved in this work will be introduced. Subsec. 2.1.1 introduces the geometric transformation of a rigid body moving in three-dimensional space. Subsec. 2.1.2 and 2.1.3 describe the pinhole model and epipolar geometry, respectively. At last, *image warping*, an important mechanism used in our depth map estimation model, is presented.

2.1.1 Geometric Transformation

The movement includes the change in position and orientation, and can be decomposed into translation and rotation. The coordinate system (CS) is used to describe the process mathematically, which can be defined through orthogonal basis $\boldsymbol{e} = (\boldsymbol{e}_{\mathrm{x}}, \boldsymbol{e}_{\mathrm{y}}, \boldsymbol{e}_{\mathrm{z}})$, where the subscripts are referred to as the x, y and z axes. As shown in Fig. 2.1There are two fundamental types of CS:

- the world CS, denoted by $(\mathrm{CS})_0$. It is a fixed, absolute reference to any object. Any point position can be described in global CS by a vector $\boldsymbol{v} = (\boldsymbol{e}_{\mathrm{x}}, \boldsymbol{e}_{\mathrm{y}}, \boldsymbol{e}_{\mathrm{z}})\,(v_{\mathrm{x}}, v_{\mathrm{y}}, v_{\mathrm{z}})^{\mathrm{T}}$,

which stands for a vector pointing from the CS origin to the target point. The vector $(v_{\mathrm{x}}, v_{\mathrm{y}}, v_{\mathrm{z}})$ are referred to as the coordinates of this point.

- the local CS, denoted by $(\mathrm{CS})_{\mathrm{B}}$. It is bounded to a target and originated in a specified point on this object. The CS orientation as well absolute position moves along as the object moves.

The linear translation, e. g. from a point A to B with global coordinates, namely vectors $(X_{\mathrm{A}}, Y_{\mathrm{A}}, Z_{\mathrm{A}})$ and $(X_{\mathrm{B}}, Y_{\mathrm{B}}, Z_{\mathrm{B}})$ can be described in $\boldsymbol{t} = (\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z)\,(\mathrm{X}_A - \mathrm{X}_B, \mathrm{Y}_A - \mathrm{Y}_B, \mathrm{Z}_A - \mathrm{Z}_B)^{\mathrm{T}}$. The translation process expressed as $\boldsymbol{B} = \boldsymbol{A} + \boldsymbol{t}$

When it comes to rotation, it is often represented through changes in orientation of local axes. There exists a variety of conventions. Here are two mostly used ones:

- the Euclidean angles decompose an arbitrary orientation change into threefold subrotations around the three axes of a reference CS, denoted by three angles α, β, γ. The reference CS can either be the fixed world CS or dynamic local CS. Since different rotating sequences lead to different ending orientations, the usage of Euclidean angles must always be specified with its sequence.

- the *axis-angle* representation. Since any change in orientation can be achieved by rotating the body around a specific axis by a certain angle (the so-called axis angle), this representation is defined as the vector $(\alpha, \beta, \gamma, \theta)$. The first three members form a unit vector (α, β, γ) that defines the *axes* in $(\mathrm{CS})_0$, while the last θ defines the *rotation angle*. Note that the Axis-angle representation is equivalent to the rotation vector, which merges the θ into the former three elements, namely $(\alpha \cdot \theta, \beta \cdot \theta, \gamma \cdot \theta)$ in the same case.

2.1.2 Perspective Camera Model

The perspective camera model, or better known as the pinhole model, describes how a three-dimensional object is projected onto an image plane. Fig. 2.2 shows a simplified geometry model: If there is a lens with focal length f, a target point with distance $S_1 > f$, its projection will be on the image plane with distance S_2 parallel to the lens. Place an actual screen or image sensor there, and a real image will be formed.

The relationship between the three is mathematically expressed through the *Lensmaker equation* (Eq. 2.1), while the proportion of the ratio of actual dimension H_1 to projected

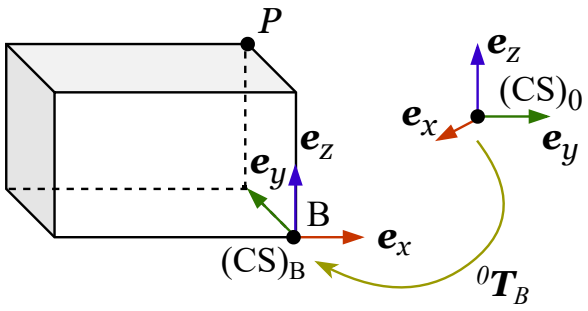


Figure 2.1: The world and the local coordinate system, denoted by $(CS)_0$ and $(CS)_B$. Their bases The P is an arbitrary point on the object. Their base is composed of (e_x,e_y and e_z. The arrow indicates the transformation ${}^0T_B$ from $(CS)_0$ to $(CS)_B$. As needed, an arbitrary point P can be described in $(CS)_0$ or $(CS)_B$.

dimension H_2 is proportional to the ratio of S_1 to S_2 (Eq. 2.2):

$$\frac{1}{S_1} + \frac{1}{S_2} = \frac{1}{f}, \tag{2.1}$$

$$\frac{H_1}{H_2} = \frac{S_1}{S_2}. \tag{2.2}$$

In this process, as Fig. 2.3 illustrates, there are mainly three coordinate systems involved, namely the world $(CS)_0$, the camera (a rigid body) $(CS)_C$, and the image plane $(CS)_I$. $(CS)_0$ is where we locate both the observer/camera and the target, while the latter two is how we connect an object's representation between $(CS)_0$ and $(CS)_I$.

Denoting coordinates of the origins of the world and camera CS by $\boldsymbol{X_0} = (X_0, Y_0, Z_0)$ and $\boldsymbol{X_C} = (X_C, Y_C, Z_C)$, respectively, their relationship can be expressed as follows:

$$\boldsymbol{X}_C = [\boldsymbol{R} \,|\, \boldsymbol{t}] \cdot \boldsymbol{X}_0, \tag{2.3}$$

where $\boldsymbol{R}$ and $\boldsymbol{t}$, as before, represent the rotation and translation, respectively.

The image plane is two-dimensional and bounded to the camera CS. The projection of the target from the camera CS to the image plane, the coordinates will be $\boldsymbol{X}_I = (x, y)$, losing the information in the Z-axis direction. In other words, each pixel coordinate represents a determined 2-D angle in a 2-D image rather than a position measurement. Given the focal

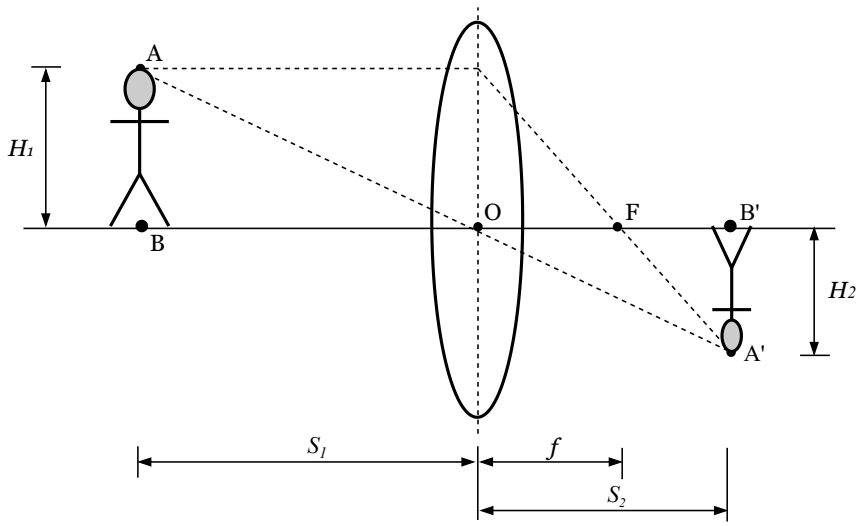


Figure 2.2: (Simplified) Image formation by an optical lens. The $A - B$ and $A' - B'$ denote the actual and projected object with height of H_1 and H_2 respectively. O is the lens center, while F is the focus. The S_1, S_2, f are the horizontal distance from O to F, $A - B$ and $A' - B'$, respectively.

length in x- and y-axis direction (f_x and f_y), the relationship is expressed as

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{Z_\mathrm{C}} \begin{pmatrix} f_\mathrm{x} \\ f_\mathrm{y} \end{pmatrix} \begin{pmatrix} X_\mathrm{C} \\ Y_\mathrm{C} \end{pmatrix} + \begin{pmatrix} C_\mathrm{x} \\ C_\mathrm{y} \end{pmatrix}, \tag{2.4}$$

where (C_x, C_y) are principle point in x and y axes, since the CS's origin is often set on the upper-left in practice.

In addition, the imaging process is done with an image sensor, e. g. a charge-coupled device (CCD). The image plane on the sensor is often originated on the upper-left instead of the principal point, and measures lengths in pixels. Therefore, the 2.4 can be further transformed as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{\mathrm{d}x} \\ \frac{1}{\mathrm{d}y} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{2.5}$$

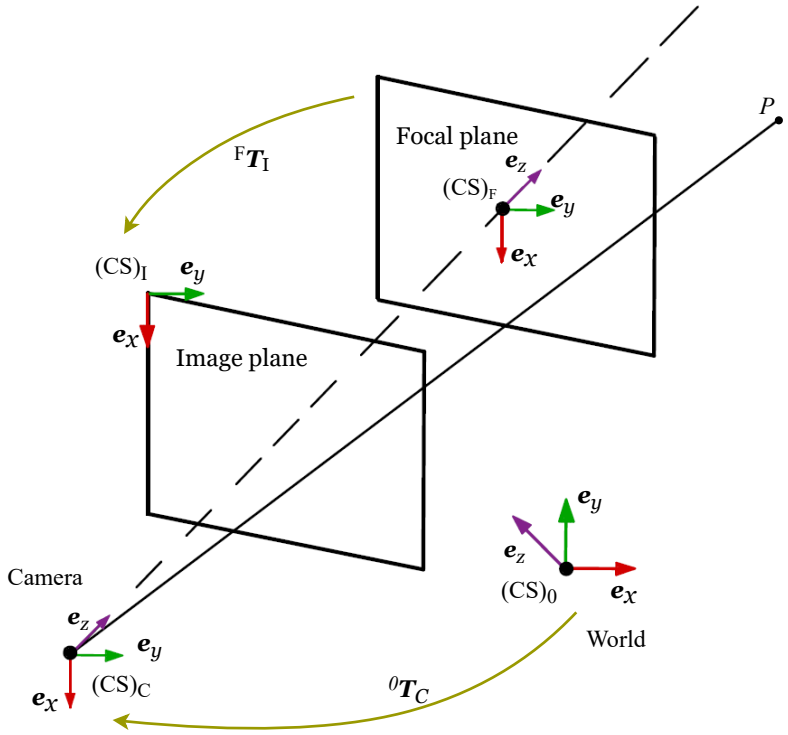


Figure 2.3: The relationship between the world, local (camera), focal-plane and image-plane coordinate systems. Notice that the image plane is denoted with a 2-D coordinate system, because the final image is projected into the 2-D plane.

The whole process can be summarized in homogeneous matrices as

$$Z_{\mathrm{C}}\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\mathrm{d}x} & 0 & u_0 \\ 0 & \frac{1}{\mathrm{d}y} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f_{\mathrm{x}} & 0 & 0 & 0 \\ 0 & f_{\mathrm{y}} & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{pmatrix}, \tag{2.6}$$

where $u_0 = \frac{1}{\mathrm{d}x}C_{\mathrm{x}}$, represents the principle point on the sensor plane, and the same goes to the v_0.

2.1.3 Triangulation with Epipolar Geometry

As introduced in Subsec 2.1.2, the information regarding the depth (Z) dimension is lost in the process of projection. In order to recover the target 3D position in the world CS from the

2D image plane, more than one projections (i. e. image planes) are needed. This is referred to as triangulation.

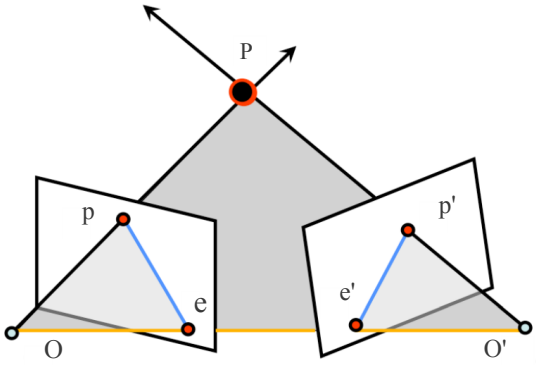


Figure 2.4: The epipolar geometry [HS17]. Two observers' origins are denoted by O and O', and the target is P. The light rays between $O - P$ and $O - P'$ intersect with their respective focal planes at point p and p, while the line $\overline{OO'}$ intersect on their focal planes at e and e' respectively. The $\overline{ep}$ and $\overline{e'p'}$ are *epipolar lines* while the $\overline{OO'}$ is the *base line*.

Fig. 2.4 shows the ideal case of triangulation, where two cameras (a stereo camera set) construct two pinhole models respectively. The ray of light coming from the world point P will go through their focal points O and O', and the projections on the respective image planes land at points p and p'. The two lines defined by $\overline{O\,p}$ and $\overline{O\,p'}$ will intersect at point P, whose 3D position can be derived through these constraints.

However, the practical situation is far more complicated than the ideal case mainly in following aspects:

- The inherent distortion resulting from the make the projected point inaccurate;
- The imaging process in a digital camera measures in a discrete unit, so are the pixel intensity values. Therefore, the reconstruction through interpolation leads to inaccuracy;
- When matching y_l and y_r by extracted features (e. g. *Harris corner* [HS88] and *ORB* feature [Rub+11]), the local region instead a single point is used, which also leads to inaccuracy.

Therefore, more than a pair of points are needed to reduce the error if one intends to solve the problem traditionally by linear algebra, e. g. Five-Point algorithm [Nis04] which utilizes at least five matched features to produce reliable estimates.

In order enable these traditional methods, there has to be enough matched feature points. This becomes a potential shortcoming when it comes to a highly dynamic scene, or the object area contains a large texture-less region with scarce features to be found. However, this property turns into an advantage, when the scene is good enough to provide valid feature matches. Therefore, the current DL-based methods would always actively add geometric constraints to enhance the robustness of the model.

The DL-based methods used in this work is elaborated in Section 3.3.

2.1.4 Image Inverse Warping

The image warping, including forward and inverse warping, is essentially how to obtain an image from the other image through a certain transformation. This section will use the forward one as a transition to better illustrate the inverse one.

The affine transformation $\boldsymbol{T}(\boldsymbol{\theta})$ includes rotation, translation, scaling, mirroring, and sheering an image. Since 2-D projection of the 3-D coordinates loses the depth dimension, the image is denoted as $\boldsymbol{I}(x, y)$, and the transformation $\boldsymbol{T}(\boldsymbol{\theta})$ working on each pixel coordinate from source/input (x_s, y_s) to target/output (x_t, y_t) image can be defined as

$$\begin{aligned}\begin{pmatrix} x_\mathrm{t} \\ y_\mathrm{t} \end{pmatrix} &= \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_\mathrm{s} \\ y_\mathrm{s} \\ 1 \end{pmatrix} \\ &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{pmatrix} x_\mathrm{s} \\ y_\mathrm{s} \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}. \end{aligned} \tag{2.7}$$

Specifically, the second row of the equation above is the decomposition of the $\boldsymbol{T}(\boldsymbol{\theta})$ regarding its effect: The parameters (e, f) controls the translation, while the other parameters can define rotation, sheering, mirroring and scaling.

As the figure shows, the forward warping copies *each* pixel value at position (x, y) to the position (x', y') (with interpolation, if the target coordinates are not integers). In other words, the forward warping finds corresponding positions on the *target* image for *source* coordinates.

The inverse warping, on the contrary, utilizes the inverse transformation $\boldsymbol{T}^{-1}(\boldsymbol{\theta})$ to match a corresponding *source* coordinates for each *target* coordinate. And then the $\boldsymbol{I}_s$ pixels are copied to $\boldsymbol{I}_t$ as in forward warping.

However, the transformation $\boldsymbol{T}^{-1}(\boldsymbol{\theta})$ could result in float-point coordinates while a digital image is defined in integer coordinates (i. e. the grid), the matched points needs re-sampling to be integers.

The mechanism of any sampling/interpolation method is to calculate the *weighted-sum* the values in the local/neighbor region around the target position to estimate the possible value in that position. The common choices for 2-D data are nearest-neighbor, bilinear, Gaussian, etc. The choice is about the trade-off between computation cost (i. e. speed) and accuracy. In this work, the bilinear sampling is chosen to perform the re-sampling (the application is in Section 3.3).

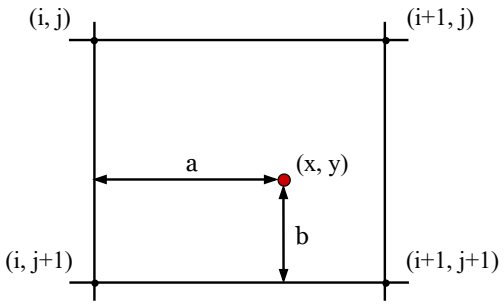


Figure 2.5: Illustration of bilinear interpolation. The distances (a, b) to the nearest pixel coordinates are less than the unit length.

The bilinear sampling in a 2-D scenario, as Fig. 2.5 illustrates, takes all four vertices of the grid unit around the target position. The weight, i. e. contribution of each vertex to the final result, depends on the distance in between. The calculation of the sampled value is as follows:

$$\begin{aligned} f(x,y) = &(1-d_i)\cdot(1-d_j)\cdot f(i,j) \\ &+ d_i\cdot(1-d_j)\cdot f(i+1,j) \\ &+ d_i\cdot d_j\cdot f(i+1,j+1) \\ &+ (1-d_i)\cdot d_j\cdot f(i,j+1), \end{aligned} \tag{2.8}$$

where $(i,\ j)$ and its alike are the pixel coordinates for the source image, their unit is the pixel size; d_i and d_j is the sub-pixel distance between two directions: $d_i = x - i$, and $d_j = y - j$.

To conclude, the inverse warping use the inverse transformation $\boldsymbol{T}^{-1}(\boldsymbol{\theta})$ to: 1) construct mapping each coordinate of the target back to the source coordinate, 2) sample the source image to get the pixel value, 3) and copy the sampled value to the target coordinate.

2.2 Deep Learning Basics

In this section, the construction of the convolutional neural network and residual network will be introduced.

2.2.1 Convolutional Neural Network

In the field of DL-based computer vision (CV), the convolutional neural network (CNN) has proven to be exceptionally effective since the success of the *AlexNet* model proposed in 2012 [KSH12]. This subsection will cover the basic components of a CNN.

A CNN usually consists of four major components: input image, convolutional layer, pooling layer and fully-connected layer.

Input layer

The input layer is, by name, the input for a CNN, which is usually an image. Since an image can be represented by one or more channels, e. g. a colored image can be expressed in red, green and blue (RGB) channels, the dimension of an input image is height(H_{img}), width(W_{img}) and depth (C_{img}, channel number). And a *depth map* has only one channel ($C_{\text{img}} = 1$) and each pixel value represents the actual depth value in physical world.

Convolutional layer

The core of a CNN is the convolutional layer. It is a set of convolution kernel/filters. To explain its function, the more general form of 1D convolution is given here. Denoted by $*$, the 1-D convolution in general can be mathematically expressed as follows:

$$s(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(\tau) g(x - \tau) \mathrm{d}\tau, \tag{2.9}$$

where s, f and g stand for the output, target input and the filter respectively, while x and τ can be interpreted as a global position in an input and a local offset in filter.The ∞ implies this filter has an infinite range. When it comes to real-world CV application where 2-D

discrete form is used, the Eq. 2.9 can be expressed as follows:

$$s(i,j) = (f * g)(i,j) = \sum_{m,n} f(m,n)g(i-m, j-n). \tag{2.10}$$

As shown in Fig. 2.6, f and g, stand for input image with the size of $(H_{\text{img}}, W_{\text{img}})$ on the bottom, and the convolutional filter with the size of $(H_{\text{conv}}, W_{\text{conv}})$ on the top. Specifically, for each coordinate (i, j) in f, the whole filter g takes out a patch from f centered at (i, j) in the same size, and calculates the sum of product, which results in one value $s(i, j)$. The filter will similarly scans through each position in f to produce the final output s. Since the elements in a kernel are called weights, the core operation in convolution is weighted sum of weights and original pixel intensities.

A convolutional layer consists of a certain number of 3-D convolutional kernels. It has the following dimensions to be marked:

- height and width, namely $(H_{\text{conv}}, W_{\text{conv}})$ as mentioned above;
- depth/channel number (for an individual kernel). It should be the same as the input channel number. For example, a RGB-image will be processed with a kernel with size of $(H_{\text{conv}}, W_{\text{conv}}, C_{\text{in}} = 3)$. Since it is implicitly determined by its input dimension, it is not a necessity to be stated in definition, e. g. as in *Tensorflow*.
- output dimension, denoted by C_{out}. This is solely determined by the number of kernels in this layer. Combined with the previous point, it can be derived that the channel number of the output in this layer will determine the depth of convolutional kernels in the next layer.

To summarize, the dimension of a convolutional layer is defined as $(C_{in}, H_{\text{conv}}, W_{\text{conv}}, C_{\text{out}})$. Moreover, the process of weighted sum can be interpreted as *feature extractor*, because the choice of kernel weights will emphasize (and omitted) certain information in the input. Therefore, the output of a convolutional layer is often referred to as the *feature map*.

Pooling Layer

The pooling layer serves to extract or filter useful information from the output of convolutional layer, therefore it often reduces the size of feature maps. There are many choices of pooling, but they can be categorized concerning its mechanism into two types:

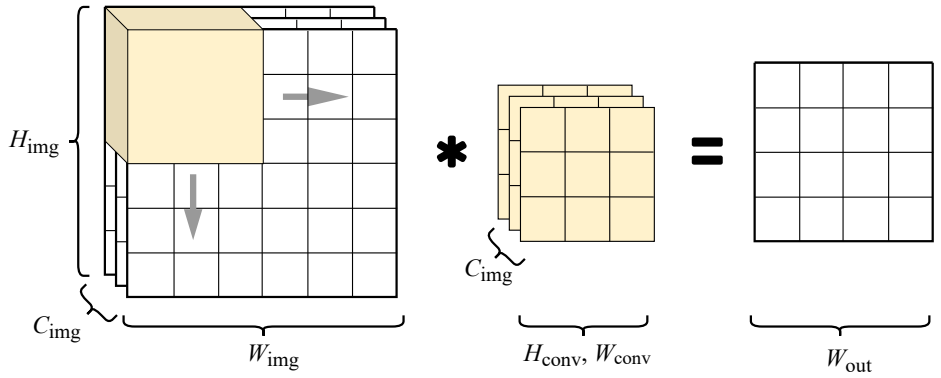


Figure 2.6: Visualization of a simple convolution. From symbol * denotes the convolution operation. In this example, an image with dimension (W_{img},H_{img},C_{img}) is convoluted with a *convolutional kernel* with the dimension (W_{conv}, H_{conv}, C_{img}). Notice that channel number C of each kernel must equal the image's, namely C_{img}=3 in this example). The kernel will slide with a certain step size (in this example, 1) to traverse the whole image. The resulting output is called *feature map* with size of $(W_{\text{conv}}, H_{\text{conv}}, D_{\text{conv}})$. The D_{conv} is the amount of the kernels (not shown here for simplicity), and in this example the there is only one kernel

.

- *selecting*: Within the pooling are (could be local or global), it selects a representative element as result. The mostly used one is the *max pooling*, which simply chooses the largest the number for output.
- *merging*: Opposed the selecting a single element, the merging type utilizes the whole area. The *average pooling* is the most common choice.

The max-pooling is more associated with the extraction of low-dimensional features, while the merging pooling is more often used to extract high-dimensional ones, which contains the so-called semantic information.

In practice, the kernel number used in a convolutional layer will be increased, leading to a larger number (deeper) of channels, while the size of $(H_{\text{conv}}, W_{\text{conv}})$ gets smaller and smaller with pooling layers. To summarize, the original low-dimensional features will be eventually extracted in depth-dimension as semantic information.

Fully-Connected Layer

The fully-connected (FC) layers are often arranged as the head of a CNN, where the high-dimensional semantic features are ready for downstream tasks, e. g. classification, detection, localization, etc.

2.2.2 Residual Network

As introduced in Section 2.2.1, the usage of stacked convolutional layers is intended to extract deeper semantic information out of image data. However, it was observed in practice, that overly cascading the convolutional layers will eventually make it difficult for the model to converge.

To tackle this problem, HE ET AL. (2016) [He+16] proposed the a new unit structure to construct neural networks, the so-called the *Residual Block* (ResBlock), and a neural network built based on ResBlocks is called the *Residual Network* (ResNet).

Residual Block

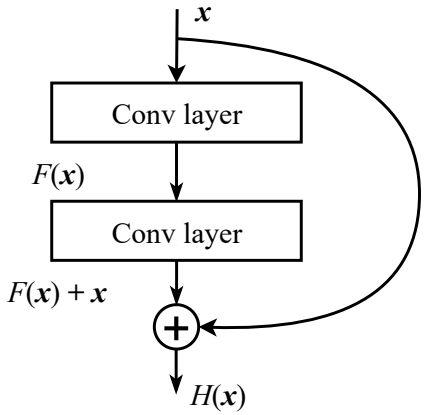
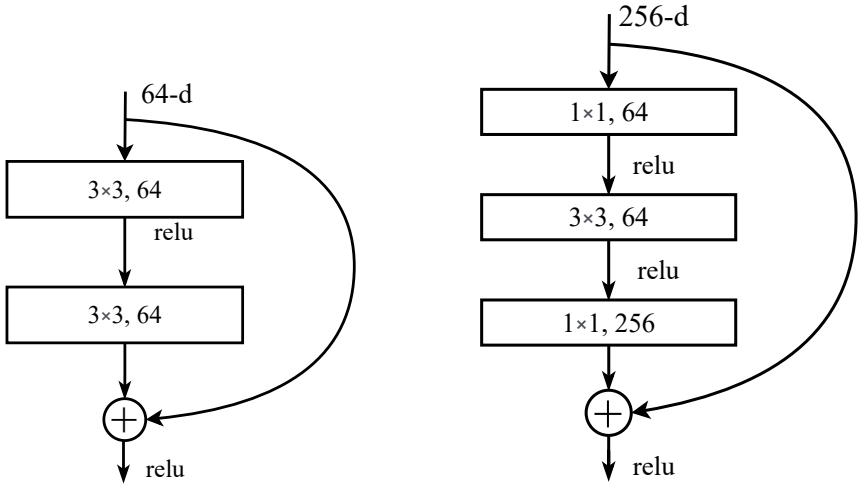


Figure 2.7: The basic structure of a residual unit. The input ($\boldsymbol{x}$) takes two paths simultaneously: one will go through convolutional layers, while the other path let the $\boldsymbol{x}$ be directly added to the output of the previous path $F(\boldsymbol{x})$, which makes the final output $H(\boldsymbol{x}) = F(\boldsymbol{x}) + \boldsymbol{x}$. The other auxiliary operations of the *convolutional layer*, e. g. *activation*, is not shown here for simplicity.

To define how a ResBlock works, it is more understandable when compared with normal flow of a CNN. As the Fig. 2.7 describes, if the input is defined as $\boldsymbol{x}$ and all operations inside CNN as $\boldsymbol{F}$, the output $\boldsymbol{H}(\boldsymbol{x})$ will be $\boldsymbol{H}(\boldsymbol{x}) = \boldsymbol{F}(\boldsymbol{x})$. However, in a ResBlock, the identity of x takes a shortcut and is added to the output of the CNN, making the overall output $\boldsymbol{H}(\boldsymbol{x}) = \boldsymbol{F}(\boldsymbol{x}) + \boldsymbol{x}$.

Although the operations in CNN are the same, the learning target is not $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{H}(\boldsymbol{x})$ anymore, but $\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{H}(\boldsymbol{x}) - \boldsymbol{x}$. This is also the reason that it is called the *Residual* Block.



(a) double-layer residual unit (b) triple-layer residual unit

Figure 2.8: The illustration of double-layer (a) and triple-layer residual unit (b), respectively. The triple-layer structure is to reduce the parameters of the convolutional layer.

There are generally two types of ResBlock categorized by layer number, namely double- and triple-layer structure. As Fig. 2.8 displays, the trick lies in the first and last convolutional layer, where the channel is decreased from 256 to 64, and increased back to 256 to match the channel number of the identity. The intention here is to reduce the parameter number and thereby the computation, specifically in this example:

- For the double-layer ResBlock, the total number of parameters is $2 \times (3 \times 3 \times 256 \times 256) = 1.18\text{e}6$.
- For the triple one, the number is $1 \times 1 \times 256 \times 64 + 3 \times 3 \times 64 \times 64 + 1 \times 1 \times 64 \times 256 = 69,632$.

The double-layer ResBlock is often used in shallower ResNet (e. g. ResNet-18 and ResNet-

34), while the latter is often used in deeper ResNet (e. g. ResNet-101 and ResNet-152). There are many variants for the sequence of ReLU activation [NH10] and Batch Normalization [IS15] relative to the convolutional layers.

Formulation of a Residual Network

A ResNet is, in short, the cascading of the ResBlocks. The construction pattern can be summarized into three parts in order:

1. first convolutional layer and max-pooling for low-dimensional features;
2. Repetition of ResBlocks. The parameters in brackets describe one specific ResBlock, where $\times N$ defines the number of cascading of this kind of ResBlock.
3. average-pooling, FC layer and softmax is for the downstream task, e. g. object detection. They are often referred to as the *top*.

The Fig. 2.9 is an example of the first few layers of ResNet-34. Specifically, "/2" stands for *downsample* operation that shrinks the input feature map to the half of its original size. They can be achieved with a (1×1) convolutional kernel with stride of 2. In addition, the dashed arrow means that there will be an operation to adjust channel dimension to make the input-identity's channel matches the output's.

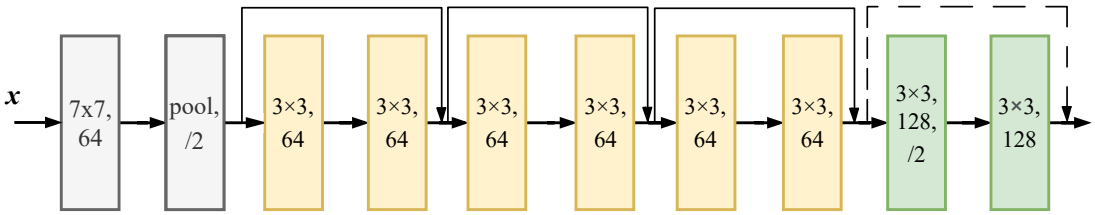


Figure 2.9: The first ten layers of ResNet-34: Except that the 2nd layer is *pooling*, the rest are all convolutional layers. For each cell, the first row (N×N) denotes the kernel size, the second is the kernel number, and the third row is the *down-sampling* by the factor of 0.5. The arrow stands for the residual connection (as in Fig. 2.8). The dashed-arrow indicates an operation to adjust channel dimension to make the input-identity's channel matches the output's.

The naming of the ResNet-X (X referred to as the ResNets's depth) is based on the number

of important convolutional layers. Taking ResNet-34 as an example (see Table 2.1 for its composition), the following components define the ResNet's depth:

- the first convolutional layer with 7×7 kernel;
- the $2 \times (3 + 4 + 6 + 3) = 32$ (each block has two) layers in the main structure;
- the final FC layers.

The 34 *key* layers mentioned above form the ResNet-34. Notice that the *downsample*-convolutional layers are not counted here.

Table 2.1: Illustration of ResNet construction using ResNet-34 as example. Each row corresponds to a block, and the last column parameterizes the convolutional layer. For example, the *conv3.x* block is composed of four repetition of two convolutional layers with identical kernel shape of $(C_{\text{in}}, H = 3, W = 3, C_{\text{out}} = 256)$.

block index	output size	34-layer
conv1	112×112	7×7, 64, stride=2
conv2.x	56×56	3×3, max pool, stride=2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$
conv3.x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$
conv4.x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$
conv5.x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$
	1×1	average pool, 1000-D FC, softmax

2.3 Visual Object Recognition

Visual object recognition includes object detection for single images and tracking across time. The following subsections will focus on these two domains.

2.3.1 Object Detection

Object detection (OD) is the most representative application for CNNs. The detection here is ambiguous, because the modern OD models are often expected to achieve at least one of the following goals:

- Decision on the presence of a specified target, along with the confidence on the decision;
- Classification;
- Localization of the object, often by a rectangle (better known as the *bounding box* or BBox).A more advanced localization task is to generate a segmentation mask, e. g. in Mask R-CNN [He+18], which produces a mask that covers the actual shape of the object.

In general, there are two major phases in the workflow of a detector. For starters, the (CNN-based) backbone network, e. g. ResNet [He+16], VGG [SZ15], serves to extract feature maps from raw image data. Afterwards, these feature maps will be further processed to achieve specific tasks. According to how to design the latter part, the OD can be categorized into *one-stage* and *two-stage* detection.

For a one-stage detector, the extracted features will be directly used as the input of final FC layers, so that the goals like BBox-coordinates are regressed. These FC layers are also called the *head*. The representative algorithms are YOLO series (e. g. YOLOv4 [BWL20]) and Single-Shot Detector (SSD [Liu+16]).

For two-stage detectors, an additional module, e. g. Region Proposal Network (RPN) in Faster-RCNN [Gir15], will be inserted after the backbone. The purpose here is to propose preliminary regions of interest (ROIs). The ROIs will be pooled (or *aligned*, in later Mask R-CNN algorithm [He+18]), in order to extract fixed-length feature vectors for later FC-layers.

The process of proposing ROIs can be understood as a special kind of sliding-window approach. For each point in the feature map, several anchor rectangles (windows) are applied to outline the local patch. Afterwards, a classifier will make a rough decision on whether the patch contains the target class(es), while a BBox-regression layer will determine a rough region that encloses the object. The resulting BBoxes are applied to the image to crop out regions of interest (ROIs). The famous example is the RCNN family, including R-CNN [Gir+14], Fast R-CNN [Gir15], Faster RCNN [Ren+17] and Mask R-CNN [He+18].

The ROI-proposal mechanism in two-stage detectors shows advantages in terms of precision and ability to detect objects with a small size relative to the full image. However, the point-by-point proposing leads to much slower running speed compared to the one-stage detectors, which makes it impossible for real-time application.

2.3.2 Object Tracking

Object tracking determines the same identity across a series of single frames, which enables processing in temporal order and bridges the gap between object-wise detection results and image-wise depth map estimates. The object tracking includes single-object (SOT) and multi-object tracking (MOT). MOT is required in this work, since we need track each object of interest on sight. For the completeness, SOT is introduced at first.

Single Object Tracking

Single-object tracking (SOT) across consecutive frames is the most basic tracking task. Typically, the task is to find the region in a new/target image, given a patch image as the template. Early on, this was treated as a correlation problem.

Thanks to the rapid development in DL, numerous neural-network-based SOT trackers have become leading roles at least in academic fields. Instead of using conventional methods that focus more on low-dimension features (e. g., pixel-level) correlation, NN models (e. g. the *SiamRPN* [Li+18]) utilize CNNs to extract higher-dimension features from a template and target frame for comparison. In this way, useful features are implicitly learnt as each convolutional kernels are updated through the back-propagation method (by RUMELHART ET AL. in 1986 [RHW86]) in the training stage.

DL-trackers generally tend to manifest higher precision when the applied scenarios are similar to the training stage. Since CNNs don't rely on hand-crafted features, they could adapt to more scenes, as long as the supervision is provided.

However, they have generally following shortcomings, such as:

- Their performance tends to deteriorate when the object has more shifts in orientation and size.
- The inference speed relies on the hardware, e. g. GPU, while conventional ones could

almost run at any modern CPU.

Note that the discussion on SOT here is for completeness. Since this work focuses on Multi-Object Tracking (MOT), a very different subject than SOT, there will be no further discussion on SOT.

Multi-Object Tracking

Multi-Object tracking (MOT) intends to keep track of multiple targets across different frames. The difficulty here lies in the mechanism of *re-identification* (Re-ID), i. e., to re-recognize multiple targets in different images and distinguish between their identities. MOT can be categorized into two types:

- Detection-based tracking, or tracking-by-detection (TBD). The trackers rely on the detected results (often in the form of BBox) to perform Re-ID. This style is more practical, because the detectors not only produce more precise BBox but also respond immediately when the object is missing (i. e., a failure report).
- Detection-free tracking. Automatically tracking multiple targets with one single initialization on the first frame. This is how most of the conventional SOT tracker (MOSSE [Bol+10], KCF [Hen+15], etc.) functions. This method fails to perform self-correction in time when, e. g. the object's size changes fast on the image., or even disappears.

One of the most classic TBD-trackers is the *DeepSort* algorithm presented by WOJKE ET AL. (2017). [WBP17]It establishes the modern TBD framework, which decomposes the task into *detection*, *feature encoding*, *data association* and *updating*. Generally for each frame:

1. The detected BBoxes are generated by the specified detector. The images will be cropped out as patches by BBoxes, each containing a target.
2. The patches are fed to a feature extractor (FE) to generate a feature vector.
3. The BBoxes and feature vectors, together denoted as *Detections*, will be used to match with previously recognized objects. If no match found, it would be treated as a newcomer and assigned with a new identity.

Different algorithms handle the above-mentioned procedure differently. For DeepSort, the framework is mainly conventional, since it matches the movement of BBoxes by Mahalanobis

distance, and feature vectors by cosine distance. The only DL-based components here are the off-the-shelf detector and FE. A more recent milestone, the *FairMot* presented by ZHANG ET AL. (2020) [Zha+20], constructs an end-to-end learning task to regress the identity directly.

In this paper, DeepSort is chosen because of its excellent flexibility of self-defining detector and feature extractor, as well as possibilities to extension/enhancement. More details are in Section 5.1.

Matching Metrics Used in DeepSort

One of the core operations in tracking is to match between previously tracked identities and newly detected ones, also known as data association. This work involves three metrics, namely Mahalanobis distance, cosine distance, intersection over union (IOU) or distance-IOU, which will be used in different matching tasks.

When it comes to positional and geometric distance/dissimilarity, the Euclidean distance (Eq. 2.19) is often used. However, for multi-dimensional data, there could be differences regarding unit-scale and variance between dimensions.

For example, for vectors $\boldsymbol{x}_i = (a_i, b_i)^{\mathrm{T}}$, where i is the index, a_i and b_i have the unit $[\mathrm{m}]$ and $[\mathrm{Pa}]$ respectively, simply calculating the sum of squared difference will overlook their unit and variance. Therefore, a transformation beforehand is needed, which brings the Mahalanobis distance (d_{mah}) defined as:

$$\boldsymbol{d}_{\mathrm{mah}}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}} \boldsymbol{S}^{-1} (\boldsymbol{x} - \boldsymbol{y})}, \tag{2.11}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are two vectors standing for new observation and previous states respectively. They are assumed to follow the same distribution with covariance matrix $\boldsymbol{S}$. The Mahalanobis distance generalizes the multi-dimensional data to be unitless and scale-invariant, and S also helps take the possible correlation between dimensions into account.

Cosine distance, on the other hand, is to measure the similarity between vectors. It is known that a NN will map the input (e. g. an image) into feature maps/vectors. The values in a feature vector has no physical unit, and an individual value can hardly represent any meaningful information. Therefore, the relative divergence between two vectors is calculated

through their included angles. The cosine distance ($\boldsymbol{d}_{\cos}$) is defined as follows:

$$\boldsymbol{d}_{\cos}(\boldsymbol{x}, \boldsymbol{y}) = 1 - \cos(\boldsymbol{x}, \boldsymbol{y}), \tag{2.12}$$

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|}. \tag{2.13}$$

Since output of $\cos(\cdot)$ ranges $[-1, 1]$, the cosine distance $\boldsymbol{d}_{\cos}$ value ranges $[0, 2]$.

The intersection over union (IOU) is dedicated to measuring the relative position, or intersection extent, between two rectangles (BBoxes). As the Figure presents, the IOU by name is the ratio of intersection area to united area. Define the two rectangles (i. e. the areas they cover) as A and B, the IOU is expressed as:

$$\text{IOU} = \frac{A \cap B}{A \cup B}. \tag{2.14}$$

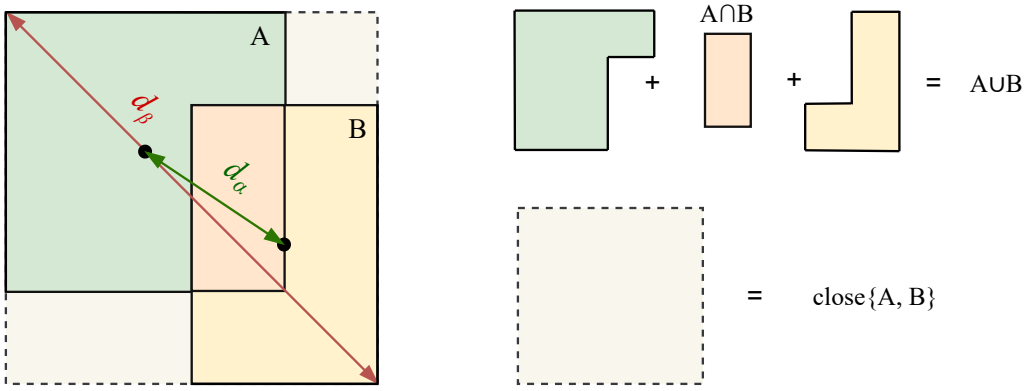


Figure 2.10: The scheme of the components for distance-intersection-over-union (D-IOU) calculation. Besides the traditional IOU, D-IOU additionally takes the following factors into account: 1) the relative distance d_α; 2) relative scale $\frac{d_\alpha}{d_\beta}$, 3) minimum closure area denoted by close{A, B} in the figure.

However, the major flaws of IOU are:

- It will always equal zero (the minimum value) when there is no intersection, making it impossible to distinguish the relative dissimilarity once BBoxes are no longer intersected.

- It will always equal one ((the maximum value) when two BBoxes fully intersects. This does not take the geometric characteristics into account, e. g. the BBox size and orientation.

Therefore, an improved metric based on the IOU, namely the distance-IOU (DIOU) is developed. It is defined as above:

$$\text{DIOU} = \text{IOU} - \frac{d_\alpha^2}{d_\beta^2}, \tag{2.15}$$

where d_α is the Euclidean distance between the center of the two BBoxes A and B, while d_α is the diagonal length of their minimum closure area. The Fig. 2.10 illustrates the mechanism: The extra portion d_α^2/d_β^2 punishes dissimilarity in geometric characteristics and their relative position.

2.4 Motion Estimation

Motion estimation includes two aspects: the egomotion (the camera's movement) and the motion of targets relative to the camera. Also, the classic method Kalman Filter [Kal60] to derive motion is presented in the last subsection.

2.4.1 Egomotion Estimation

To estimate the motion of a target relative to the observer, the observer's movement, often referred to as *egomotion*, should also be considered. For static observation, e. g. the traffic surveillance, there is not egomotion involved. This work, however, focuses on dynamic scenes where the camera is on a moving vehicle.

In the process of driving, the camera moves in a 3-D space with six degrees of freedom, which includes translation along and rotations around x, y and z axes, respectively (as introduced in Section 2.1.1). In addition, as Fig. 2.11 depicts, the rotation angles around lateral, longitudinal and vertical axes are also often referred to as *pitch*, *roll* and *yaw* angles, respectively.

Since the camera is taking images in a temporal sequence, the egomotion could be theoretically derived by comparing neighbor images.

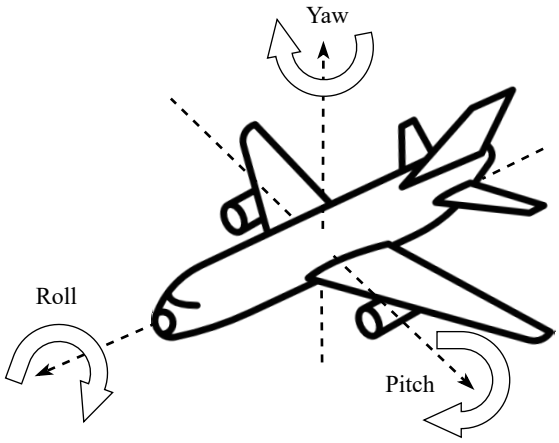


Figure 2.11: Illustration of the 3-D movement with six degrees of freedom. For each axis, i. e. roll, yaw and pitch, there are translation and rotation movement.

A common pipeline for conventional methods is

1. detect the feature points on two neighbor frames. The typical choices of hand-crafted features include ORB[Rub+11], SIFT [Low99], etc.

2. match the hand-crafted feature between two frames and localize their pixel coordinates.

3. given camera's intrinsic parameters, the egomotion can be estimated using geometric constraints using relevant algorithms.

To be specific, egomotion is traditionally solved by utilizing constraints based on epipolar geometry (introduced in Section 2.1.3). A typical choice is the *Five Point algorithm* [Nis04], which takes at least five points (and camera intrinsic parameters) to estimate the essential matrix. The essential matrix can then be decomposed into rotation and translation matrix. Note that the translation here is merely up to scale, since the monocular projection will lose depth information.

Since these algorithms use merely geometric constraints by positions of matched points between two images, feature-point matching is of vital importance. In practice, the more accurately matched point pairs are provided, the better the estimation will be. This demand is challenging to be met, since the quality of visual-based methods is subject to the following defects:

- The blurring effect caused by movement. Since the shutter speed is often not fast

enough compared to the car's movement, the exposure will capture visual information for a relatively long period.

- The pixel noises caused by an environment with awful conditions, e. g. in a dim weather. Because the visual sensor can only differentiate lighting intensity in a limited range.
- Certain circumstances are not suitable for feature matching, e. g. a scene where few texture exits, such as the cloudless sky.

Therefore, this method will be robust only when the image sequences are taken in a less dynamic scene rather than in a moving car. Moreover, there is no valid failure report for feature matching, making it more difficult to determine how many matches can be used. Last but not least, the calculation of hand-crafted features requires certain amount of time, e. g. with an Intel® Core™ i7 Processors (7th Gen) the processing speed is only about 30 frames per second (FPS).

With the above-mentioned reasons considered, the conventional methods of egomotion estimation are not used. Furthermore, since the ultimate goal is to estimate the motion of the targets (instead of the ego vehicle), we argue that an explicit egomotion estimation should not be a part of the inference pipeline. In Section 3.3, an auxiliary NN for egomotion will be introduced, which is only utilized during the training of the NN for depth estimation.

2.4.2 Distance and Velocity Estimation

The physical definition of distance: Place the origin of the (Cartesian) coordinate system at the observer, the distance to an object is often defined as the Euclidean distance. Given the coordinate (x_0, y_0, z_0) of the target A, the distance/depth d_0 to the observer (the origin) is calculated by:

$$d_0 = \sqrt{x_1^2 + y_1^2 + z_1^2}. \tag{2.16}$$

The projection magnitudes, x_0, y_0, z_0 also clarify the distance in respective directions. Note that this work will not involve movements in the vertical direction (z_0), thereby only depth, and 2-D distance in longitudinal and lateral directions denoted as d_x and d_y, will be considered.

In a 2-D planar coordinate system, the Eq. 2.16 is simplified to following connections:

$$d_0 = \sqrt{x_1^2 + y_1^2}, \tag{2.17}$$

$$x_0 = d \cdot \cos(\theta), \tag{2.18}$$

$$y_0 = d \cdot \sin(\theta), \tag{2.19}$$

where θ is defined as the angle between the vector $(x_0, y_0)^{\mathrm{T}}$ and x axes, thereby $\theta = \arctan2(y_0, x_0)$. Therefore, the 2-D distance can be derived with depth, as long as the angle θ can be estimated.

The depth can also be connected to the disparity in stereo vision. For two cameras, like human eyes, the position of a target will appear at a slightly different position (in horizontal direction). The farther the target is, the less noticeable the difference becomes. And such difference is referred to as the *disparity*. For a stereo camera set, the disparity, denoted as disp, is defined as

$$\mathrm{disp} = \frac{f \cdot b}{d \cdot \mathrm{ps}}, \tag{2.20}$$

where f is the focal length; b stands for baseline length, e. g. the length between two cameras; ps is the pixel size. Many networks estimate the disp as the output and derive the depth by scaling the reciprocal, since $\mathrm{disp} \propto \frac{1}{d}$.

To avoid confusion between depth d and its 2-D projections d_{x} and d_{y}, we will clearly state the 2-D distance projections as *2-D distance*. Otherwise, d stands for Euclidean distance or its general concept regardless of dimension.

Distance is the integral of velocity over time, namely $d = \int v \cdot \mathrm{d}t$. Although the velocity can be continuously changing, but in a real-world application, we often take a short period unit (e. g. 0.1 second), within which the velocity is presumed to be constant. This presumption is based on the fact that there would not be abrupt velocity change within that time because of the inertia of a vehicle. Such discretization also simplifies analysis within a time interval to compensate error of one single estimation.

As we analyze the video frame-wise, the concept of frame-rate is more often used than time unit: Define frame-rate as frame-per-second (FPS in $[\mathrm{s}^{-1}]$), the relationship between distance

change Δd in $[\mathrm{m}]$ and velocity in $[\mathrm{m} \cdot \mathrm{s}^{-1}]$ can be expressed as:

$$v = \Delta d \cdot \mathrm{FPS}, \tag{2.21}$$

where velocity here is, as mentioned above, presumed to be constant within at least each frame; and Δd is the distance change taking place between two frames. In other words, the velocity estimation can be derived during frame-wise distance estimation (within known frame interval).

2.4.3 Kalman Filter

Kalman Filter (KF) in essence serves to fuse data from multiple sources to optimize the measurement. An example is: Estimate a car's motion state by utilizing multiple sensors (e. g. depth sensors, visual sensor, inertial sensors, etc.). The KF will take their inaccuracy (noise) into account to model the motion and continuously update the model with ongoing measurement. In this work, the *motion state* includes only the distance and its derivative with time (i. e. velocity), so the acceleration will not be considered.

In short, the KF will firstly predict the next state depending on the previous knowledge, and the new observation will be utilized to optimize the raw prediction and give the final estimate of the motion state. The following is the main procedure of how KF predict-and-correct to optimize the state. Note that all the symbols used in this section will not occur in other sections unless specified otherwise.

To begin with, the motion at time step t ($\boldsymbol{x}_\mathrm{t}$) is predicted by the previous step $\boldsymbol{x}_{\mathrm{t}-1}$, namely the *prior prediction* calculated as Eq. 2.22. In the meanwhile, new measurement $\boldsymbol{z}_\mathrm{t}$ of the current status $\boldsymbol{x}_\mathbf{t}$ arrives, expressed in Eq. 2.23.

$$\boldsymbol{x}_t = \boldsymbol{F}_\mathrm{t}\boldsymbol{x}_{\mathrm{t}-1} + \boldsymbol{B}_\mathrm{t}\boldsymbol{u}_\mathrm{t} + \omega_\mathrm{t}, \tag{2.22}$$

$$\boldsymbol{z}_\mathrm{t} = \boldsymbol{H}_\mathrm{t}\boldsymbol{x}_\mathrm{t} + \boldsymbol{v}_\mathrm{t}, \tag{2.23}$$

where

- $\boldsymbol{B}_\mathrm{t}$ and $\boldsymbol{u}_\mathrm{t}$ is the control matrix and movement measurement of the target. Together, they measure how the target is steered, e. g. how far do the four wheels travel respectively. As there is no measurement for the camera's motion in this work, we will omit

this part afterwards.

- $\boldsymbol{F}_{\mathrm{t}}$ is the state transformation matrix, which describes how the target's state is transformed from time-step $t-1$ to t. The parameters for the state $\boldsymbol{x}$ is self-defined. In this work, we model the distance and velocity, namely $(d_{\mathrm{x}}, d_{\mathrm{y}}, v_{\mathrm{x}}, v_{\mathrm{y}})$. But for simplicity of the introduction in this section, we refer the motion as $\boldsymbol{x}_{\mathrm{t}} = (x_{\mathrm{t}}, \dot{x}_{\mathrm{t}})$
- ω_{t} is the *process noise* that reflects prediction inaccuracy caused by non-linearity of the system, with covariance denoted by $\boldsymbol{Q}_{\mathrm{t}}$.
- $\boldsymbol{z}_{\mathrm{t}}$ is the measurement from the sensors, while $\boldsymbol{H}_{\mathrm{t}}$ is the transformation matrix that maps the $\boldsymbol{z}_{\mathrm{t}}$ to the same unit as the $\boldsymbol{x}_{\mathrm{t}}$, e. g. from millimeter to meter.
- $\boldsymbol{v}_{\mathrm{t}}$ is the noise in measurement with covariance $\boldsymbol{R}_{\mathrm{t}}$, reflecting the inaccuracy of the sensor reading.

The *prediction* result of a variable at t based on $t-1$ is expressed with the subscript: $\mathrm{t}|\mathrm{t}-1$. For example $\boldsymbol{x}_{\mathrm{t}|\mathrm{t}-1}$ stands for the raw prediction of the $\boldsymbol{x}_{\mathrm{t}}$ based on the final estimate of $\boldsymbol{x}$ at time $t-1$. In addition, the inter-connection inside the state parameters are represented by the covariance $\boldsymbol{P}$:

$$\hat{\boldsymbol{x}}_{\mathrm{t}|\mathrm{t}-1} = \boldsymbol{F}_{\mathrm{t}}\, \hat{\boldsymbol{x}}_{\mathrm{t}-1}, \tag{2.24}$$

$$\boldsymbol{P}_{\mathrm{t}|\mathrm{t}-1} = \boldsymbol{F}_{\mathrm{t}} \boldsymbol{P}_{\mathrm{t}} \boldsymbol{F}_{\mathrm{t}}^{\mathrm{T}} + \boldsymbol{Q}_{\mathrm{t}}. \tag{2.25}$$

It is obvious that the process of prediction increase the uncertainty by $\boldsymbol{Q}_{\mathrm{t}}$ because of motion-measurement uncertainty is introduced.

Next, the measurement is taken to *correct* the raw prediction:

$$\hat{\boldsymbol{x}}_{\mathrm{t}} = \hat{\boldsymbol{x}}_{\mathrm{t}|\mathrm{t}-1} + \boldsymbol{K}_{\mathrm{t}} \left(\boldsymbol{z}_{\mathrm{t}} - \boldsymbol{H}_{\mathrm{t}}\, \hat{\boldsymbol{x}}_{t|t-1} \right), \tag{2.26}$$

$$\boldsymbol{K}_{\mathrm{t}} = \boldsymbol{P}_{\mathrm{t}|\mathrm{t}-1}\, \boldsymbol{H}_{\mathrm{t}}^{\mathrm{T}} \left(\boldsymbol{H}_{\mathrm{t}}\, \boldsymbol{P}_{\mathrm{t}|\mathrm{t}-1}\, \boldsymbol{H}_{\mathrm{t}}^{\mathrm{T}} + \boldsymbol{R}_{\mathrm{t}} \right)^{-1}, \tag{2.27}$$

where $\boldsymbol{K}_{\mathrm{t}}$ is called the *Kalman gain* during the correction step at time t.

When ignoring the transformation matrix $\boldsymbol{H}$ that just serves to adjust the unit, the $\boldsymbol{K}$ reflects the relationship between $\boldsymbol{P}$ and $\boldsymbol{R}$, namely the uncertainty from prediction and measurement respectively, and $\boldsymbol{K} \in [0, 1]$. Moreover, according to Eq. 2.26, the smaller measurement uncertainty $\boldsymbol{R}$ is, the more the $\boldsymbol{K}$ will approach 1, thereby the more will the *new measurement* dominate the final estimation result.

At last, the $\boldsymbol{P}_{\mathrm{t}}$ will be updated as the $\boldsymbol{x}$ has been updated from $\boldsymbol{x}_{\mathrm{t|t-1}}$ to $\boldsymbol{x}_t$:

$$\boldsymbol{P}_{\mathrm{t}} = \boldsymbol{P}_{\mathrm{t|t-1}} - \boldsymbol{K}_{\mathrm{t}}\boldsymbol{H}_{\mathrm{t}}\boldsymbol{P}_{\mathrm{t|t-1}}. \tag{2.28}$$

The corrected/updated variables with the subscript: t will be used for the next time step $t+1$,which forms a predict-correct cycle.

3 Applied Algorithms

In this chapter, applied algorithms, including object detection, multi-object tracking, depth map estimation, and object-wise motion estimation, will be introduced in sequence.

3.1 Object Detection with YOLOv4

YOLOv4, the fourth version of You-Only-Look-Once algorithm, is an one-stage object detection algorithm that absorbs state-of-the-art ideas into its network design and training strategy. It features high detection speed and achieves an outstanding trade-off between speed and accuracy.

To present a thorough introduction, the section will be split into three parts: 1) the network architecture; 2) post-processing and losses; 3) essential training tricks. In order to illustrates in a *top-down* manner, the focus will be firstly placed in the main structure and skip single convolutional blocks for clarity.

Network Architecture

As Fig. 3.1 shows, the YOLOv4 network consists of the following three parts: The backbone: CSPDarknet53, the neck: SPP with PAN, and the head for three tasks (the predictions on location, class and confidence).

The CSPDarknet53 combines Cross Stage Partial Network (CSPNet) and Darknet-53 backbone (without FC-layer). To begin with, the cross-stage-partial network (CSP, presented by WANG ET AL. in 2020 [Wan+20]) is used, which lowers the computation cost and improves the performance compared to a similar structure of traditional ResNet (for more details see Appendix .1).

The concatenated CSP-units extract feature maps in a higher dimension, and each follows a convolution with a step of 2 in the end, for down-sampling and expanding the receptive field.

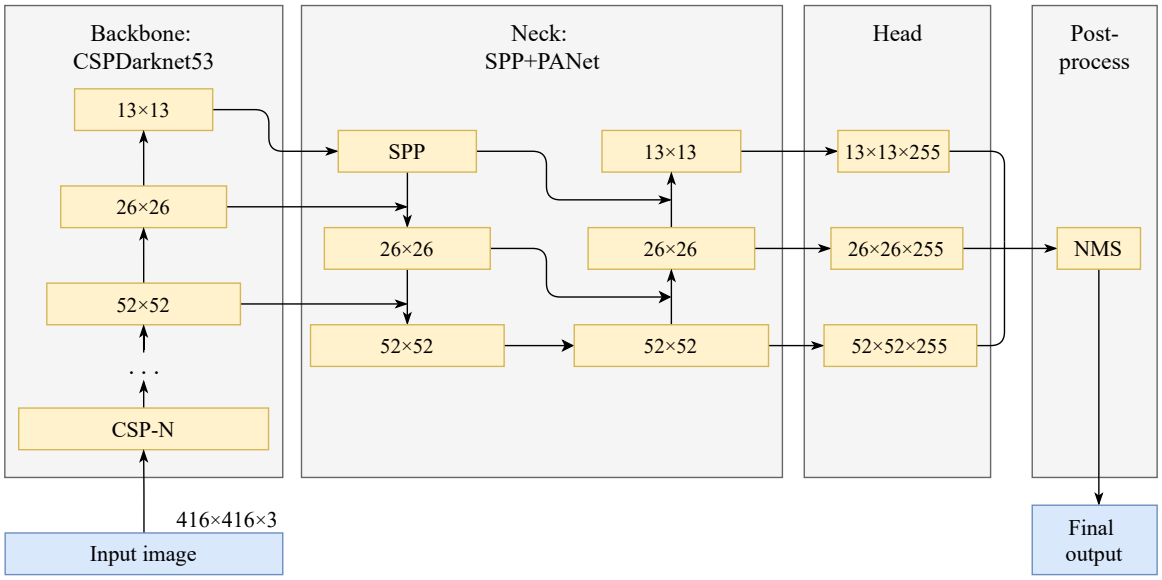


Figure 3.1: The overall architecture of YOLOv4. The network is composed of the *backbone*: CSPDarknet53; the *neck*: spatial pooling pyramid (SPP) and path-aggregation network (PANet); the *head*; and Non-Maximum Suppression (NMS) as *post-processing* method.

In addition, the last two CSP units will copy its output to the following neck structure, which allows information fusion in multiple dimensions.

The neck is formed by the combination of Spatial Pyramid Pooling (SPP), and the combined structure of Feature Pyramid Network (FPN) and Path Aggregation (PA) method:

- As Fig. 3.2 demonstrates, the SPP takes the output of CSPDarknet53 and branches into copies, which pass *max-pool* operation with kernel size of $(1, 1)$, $(5, 5)$, $(9, 9)$, $(13, 13)$, respectively. Afterwards, they are concatenated again as one feature map.
- The PANet, i. e. the combined FPN-PA structure, enhance the multi-scale fusion further.
  - As shown in the left part of the *Neck* block in Fig. 3.1, the FPN, is a series of *top-down* operations. It takes *up-sampled* the high-dimension output to merge with lower-dimension features (intermediate output of the backbone).
  - Afterwards, a series of *bottom-up* PA operations follows the structure shown in the right part of the *Neck* in Fig. 3.1.This is a similar procedure to the FPN but in

reversed order, i. e. the feature will be merged from low to high dimension.

The two components: SPP and PANet, enable the fusion of multi-scale features. Specifically, the features at lower scale tend to carry more location and information because of their limited receptive field, while the those at a higher scale carry more semantic information, because they have the grasp of the global information.

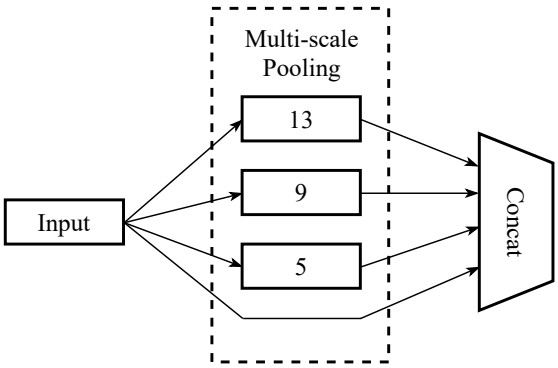


Figure 3.2: Scheme of spatial pyramid pooling (SPP). The input feature map is applied with multiple pooling layers with different kernel sizes. The size of the kernel determines the local area size to be pooled, e. g. a 13 (square) kernel will take 13×13 grid units into account.

The heads of YOLOv4 are simply a series of convolutional blocks. Taking the three outputs at different scales from the YOLOv4-neck, and generate the raw predictions (BBox coordinates, classification and confidence) respectively. Take an image with shape $(416, 416, 3)$ as an example: the resulting size of the three heads will be $(13, 13), (26, 26)$ and $(52, 52)$ respectively. The channels of the outputs correspond to the dimension of the one prediction: $4 + 1 + N_{\text{class}}$, where 4 is the dimension of a BBox coordinate, 1 is the confidence, and $+N_{\text{class}}$ is the total number of class options. The official code is trained on COCO dataset with 80 classes, which makes the output shapes $(13, 13, 85), (26, 26, 85)$ and $(52, 52, 85)$, respectively. As result, the amount of raw predictions are $13^2 + 26^2 + 52^2 = 3549$

Post-processing and Losses

The post-processing, including 1) proposing triple boxes by anchors and 2) *Non-Maximum Suppression* (NMS, presented by J. CANNY in 1986 [Can86]), will produce the final predictions. The losses will be calculated based on the prediction errors, which will then be used to

back-propagate to update the weights.

The outputs at three scales of the YOLOv4 heads are merely raw proposals. Each prediction will be multiplied with three anchors to generate final proposals, which triples BBox-predictions in one location. As Fig. 3.3 demonstrates, the anchors are simply a set of fixed rectangle sizes (width and height) that tend to fit the sizes of most of the objects in the dataset. For example, the car should have a larger size in horizontal direction, while a person should be of larger size in vertical direction.

Specifically, the anchor sizes are chosen by running *K-Means* algorithm [Llo82] on all labels of the COCO dataset. The nine resulting anchor sizes are $(10, 13), (16, 30), (33, 23), (30, 61), (62, 45), (59, 119)$ $(116, 90), (156, 198), (37, 326)$ in (width, height) manner.

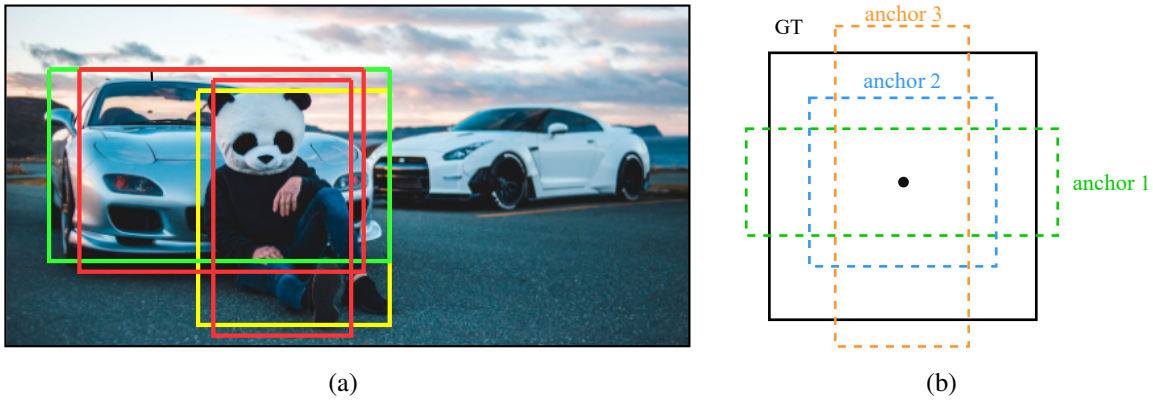


Figure 3.3: The mechanism of the anchor box. (a) is an example of an image with ground-truth boxes in yellow for the person and green for the car, respectively. Since two objects overlap, it is difficult to estimate boxes with similar center points. The anchor boxes with fixed aspect sizes help propose possible bounding boxes that better enclose the object. (b) is an example of the anchor boxes generated at one center point. The "anchor 2" fits the corresponding GT better than other anchor boxes.

At this point, there are $3549 \times 3 = 10647$ predictions, which will be filtered by NMS. The NMS serves to: 1) directly delete the predictions with low scores; 2) only preserve the one with the highest score among a group of overlapped BBoxes. Noticeably, the NMS utilizes the Distance-IOU (DIOU [Zhe+19], introduced in Section 2.15), instead of traditional IOU, to determine the overlapping/redundancy extent.

Afterwards, the losses will punish the errors regarding three aspects:

- The BBox coordinate. CIOU-loss L_{ciou} is chosen as the metric. Compared to DIOU, the CIOU punishes the aspect-ratio error, which is defined as:

$$\text{CIOU} = \text{DIOU} - \alpha \cdot v, \tag{3.1}$$

$$v = \frac{4}{\pi^2} \left(\arctan2\left(w^{\text{gt}}, h^{\text{gt}}\right) - \arctan2(w, h) \right)^2, \tag{3.2}$$

$$L_{\text{ciou}} = 1 - \text{CIOU}. \tag{3.3}$$

- The classification correctness. It is defined by a binary cross-entropy loss L_{cls} for one single prediction is:

$$\begin{aligned} L_{\text{cls}} &= -\sum_{i=1}^{2} p_i \log\left(f\left(\hat{p}_i\right)\right) \\ &= -p_i \log\left(f\left(\hat{p}_i\right)\right) - \left(1 - p_i\right) \log\left(1 - f\left(\hat{p}_i\right)\right), \end{aligned} \tag{3.4}$$

  where $f(\cdot)$ is the sigmoid function (Eq. .1); p_i and $\hat{p}_i$ are predicted and GT class respectively.

- The confidence score p_{conf}. This score indicates the existence probability of any object, which ideally should equal 0 when there is no object inside the BBox (i. e. the background), and 1 when there is a valid object. The object-confidence loss L_{conf} is defined as:

$$L_{\text{conf}} = -p_{\text{pos}}^{\text{gt}} \log\left(p_{\text{conf}}\right) - \left(1 - p_{\text{pos}}^{\text{gt}}\right) \log\left(1 - p_{\text{conf}}\right), \tag{3.5}$$

  where p_{pos} is the GT binary value for the positive (has-object) instance.

Finally, the three losses, i. e. L_{ciou}, L_{cls} and L_{conf} will be averaged prediction-wise to form the final loss value.

Data Augmentation

Data augmentation is generally the operations that adjust and create new pseudo data based on the raw data. It is only applied in the training phase to simulate more various scenarios than the ones that original data could represent, and it is turned off in the testing/inference phase. In the field of Computer Vision, the data augmentation manipulates images. Commonly there

are:

- Changes in global representation: random scaling, cropping, flipping and rotation.
- The lighting condition adjustments, i. e. random adjustment on brightness, contrast, hue, saturation and image noise (e. g. the Gaussian noise added to the original pixel values).
- Randomly erasing regions (replacing pixel values with a constant) using *GridMask* proposed by CHEN ET AL. in 2020 [Che+20a].
- Randomly merging different images into a new one, referred as *Mosaic* [BWL20]. Precisely, it resizes four images and puts them together like a jigsaw puzzle, which significantly enriches the information compared to one raw image.

Data augmentation artificially increases the complexity of the image, helping the trained model adapt to more complicated scenes in real world that are not represented by the training set.

3.2 DeepSort Tracking

This work adopts *DeepSort* [WBP17] algorithm for multi-object tracking (MOT) which has established a classic framework for the so-called *tracking-by-detection* style tracking.

It evolves from the predecessor, the Sort algorithm [Bew+16] which is the first to combine Kalman Filter by R. E. KALMAN [Kal60] and Hungarian algorithm by KUHN AND YAW [KY55]) to achieve fast inference speed (up to 260 Hz). The high speed benefits because it solely utilizes IOU metric (i. e. geometric and location characteristics). However, such simplicity makes it difficult to perform Re-ID in dynamic scenes, whose previous BBox will deviate greatly from the current one. Neither is it capable of recovering a temporarily-lost identity.

Therefore, the DeepSort enhances the Sort by taking the appearance information into account, and applying more metrics for Re-ID task. In addition, the detection and appearance feature extraction is achieved by a separate detector and feature extractor, which is independent of the DeepSort structure. Since the object-detection algorithms have been introduced in Section 2.3, this section will only cover the main structure of DeepSort and the feature extractor.

3.2.1 Tracking Pipeline

This section will firstly introduce the overall pipeline, and then offer a more detailed explanation of its most crucial procedure, i. e. the *cascade matching*. The corresponding flowcharts will be presented for better clarification.

Overall Structure

Fig. 3.4 describes how DeepSort functions in general, where The block area stands for a procedure, and the rest is either a component or a status. Above the dashed line separates the main workflow and working mechanism of individual Tracks.

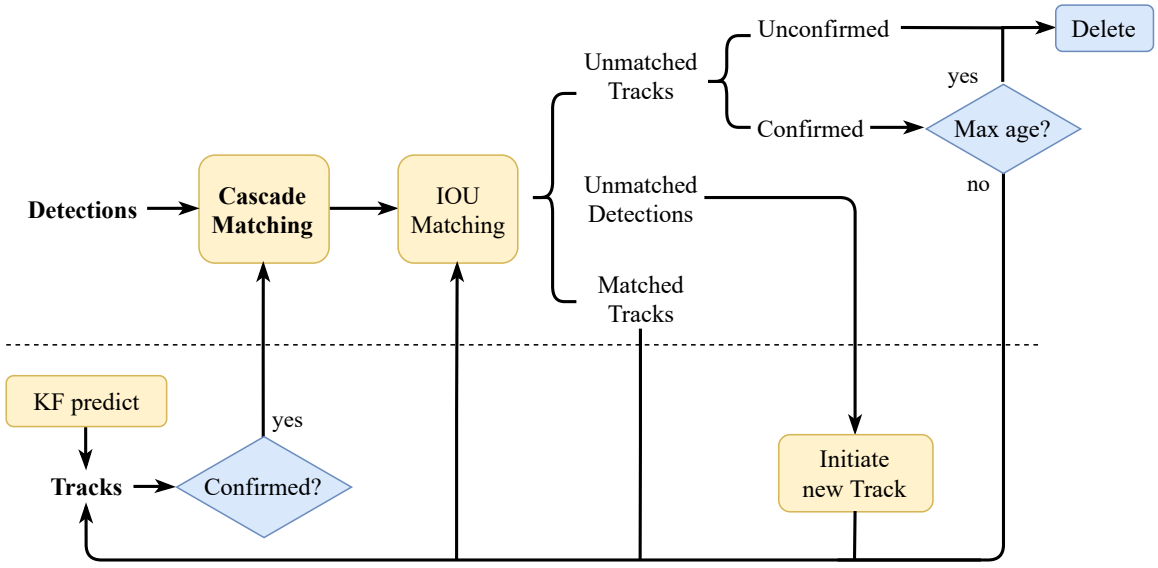


Figure 3.4: The pipeline of DeepSort algorithm. Detections go through two-stage matching, i. e. cascade and IOU matching. The *Detections* of new identities are used to initiate new *Track* (and given with a unique ID-number). In turn, the previously initiated Tracks will be used to match with new *Detections*. The matched Track will be updated with the detected results, while the continuously unmatched Track will expire after a specific time. Each Track contains the geometric and appearance characteristics of the identity, represented by a BBox and a feature vector respectively.

There are following critical components/concepts worth extra explanation, of which capital letters are used for keywords to distinguish them from normal words:

- *Detection*. This is a set of the results from the object detector and the feature extractor. Specifically, the former provides the BBox coordinates, while the latter passes the feature vector in a fixed dimension., e. g. 128.

- *Track*. Each Track carries the necessary information of one tracked identity. The geometric and positional characteristics are modelled by a Kalman Filter (KF), while the feature vectors will be simply stored as a list in temporal order. The KF parameterises each BBox by its center coordinates (c_x, c_y), aspect ratio r_a, height h, and their velocities respectively. So the full parameters are: $(c_\mathrm{x}, c_\mathrm{y}, r_\mathrm{a}, h)$. For each new time step, the KF will firstly predict by itself, giving an a-prior result of where the new BBox could be. The result will be corrected by the actual measurement (the newly detected BBox).

- (Un-) matched status. Matching takes place between a Track and a Detection. There are feature and IOU matching in DeepSort, which take place between feature vectors and BBoxes respectively. An unmatched Track means it cannot be matched with any new Detection in this frame. And an unmatched detection is the other way around.

- (Un-) confirmed status. For a Track, a successful match in the current frame will be marked as *confirmed*. And if a Track cannot find a match for certain successive frames, it is considered to be lost (unconfirmed) and deleted. As for a KF, the *unconfirmed* status is given in the initialization phase. Since KF is initialized with large variation (of a Gaussian distribution, to be exact), it needs certain rounds of new measurements to be stabilized.

The DeepSort algorithm can be summarized in the manner of how data flows inside the algorithm:

1. Detections, including BBox and extracted features, are passed in.

2. The KF for each Track will predict at each step independently, giving an a-priori prediction on the BBox characteristics, i. e. $(c_x, c_y, r_\mathrm{a}, h)$.

3. In cascade matching, feature vectors in new Detections Detections and confirmed Tracks generate a cost matrix. In the meanwhile, the Mahalanobis distance (d_mah) between new Detections and the Tracks. d_mah here serves as a gating mechanism for the feature-similarity cost matrix to exclude objects whose trajectories have gone astray.

4. The unmatched Tracks and Detections are then passed to the next round of matching, i. e. IOU matching (Eq. 2.14). In this procedure, IOU metric measures how well the BBoxes between Detections and Tracks match each other.

5. After two rounds of matching as above mentioned, the leftover Detections are deemed to be new-comers, thus a new Track will be initiated according to its information (BBox and feature vector). Notice that newly-initiated Track will be marked as "Unconfirmed"/"Tentative", until it is continuously matched for a certain frames.

6. At last, the unmatched Tracks will be considered to have disappeared in this frame, therefore it will be hidden or deleted (if lost for too long).

Cascade Matching

The cascade matching is the major contribution of the DeepSort compared to Sort tracking, because it combines the geometric similarity between BBoxes and the appearance similarity between feature vectors.

As Fig. 3.5 shows, the Mahalanobis and cosine dissimilarities are obtained, and forms the final cost matrix by weighted summation. For example, N Tracker candidates with M new Detections will form a $N \times M$ cost matrix.

However, according the author, it would be more practical to simply use the Mahalanobis distance to *gate* the cosine distance/cost, i. e. set the distance to infinite large if d_{mah} is over the threshold. In this way, trivial tuning of the weight λ can be avoided. And the results are practically more robust.

After the final $N \times M$ cost matrix is obtained, Hungarian Algorithm [KY55] match Tracks and Detections. Since each Detection should only be matched to one Track (and there is naturally no internal matching between Detections themselves), it can be understood as a bipartite graph. And our goal is to find one matched Track for each Detection, and minimize the overall cost. Denote cost matrix as $\boldsymbol{C}(n, m)$ and the (binary) mask matrix as $\boldsymbol{X}(n, m)$, where $\boldsymbol{X}(N, M) = 1$ means index-N Detection is assigned to index-M Track, then the problem can be described as:

$$\arg\min \left(\sum_{n} \sum_{m} \boldsymbol{C}(n, m)\, \boldsymbol{X}(n, m) \right), \tag{3.6}$$

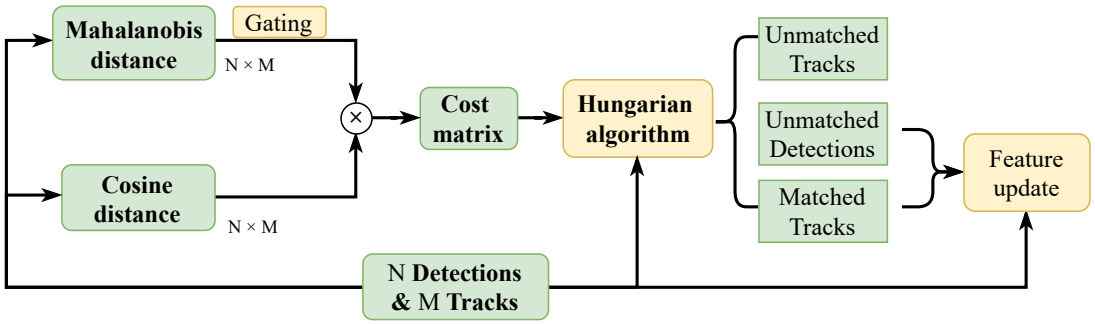


Figure 3.5: The *cascade matching* of DeepSort. the Detection carries the newly detected BBox and appearance-feature vector, while the Track, representing an identity, contains a Kalman Filter (KF) and a list of past feature vectors. The KF-predicted BBox by each track will be used to calculate *Mahalanobis distance* (Eq. 2.11) with BBox in each Detection, while the feature vectors will be used to calculate the *cosine distance* (Eq. 2.12). This cross comparison leads to two cost matrices. A binary mask will be generated by thresholding the Mahalanobis distance, and be applied to the cosine distance. Afterwards, the *Hungarian algorithm* [KY55] is applied to the resulting cost matrix to handle the matching task.

Note that the Detections excluded by the gating mechanism will be directly marked as unmatched, and not participate in matching.

3.2.2 Feature Extractor

The feature extractor in DeepSort serves to extract appearance information of the target. Essentially, a CNN-based model generally consists of a backbone to extract high-dimension feature maps and head(s) to perform downstream tasks, such as classification. Therefore, any backbone model is qualified as a feature extractor.

However, modern models tends to be equipped with increasingly deep CNNs to enhance representational power. Heavy computational cost comes with a price, making the inference slow and incapable of real-time execution. Therefore, a lightweight model is needed.

In the original proposal, DeepSort takes the *wide residual network* (W-ResNet, presented by ZAGORUYKO AND KOMODAKIS in 2017 [ZK17]) as the feature extractor, which processes 32 BBoxes within merely 30 ms on a Nvidia GeForce GTX 1050 mobile GPU. On the contrary to the ResNet, the W-ResNet leverages the width of feature planes instead of

stacking ResBlocks in depth.

As the width w of a normal ResBlock is defined as $w = 1$, and the multiplicative factor as k, a W-ResNet with $w = k$ and depth of l can be denoted as W-ResNet-k-l. Experiments indicate that ResBlock that extended in width has following major advantages:

- Comparable performance with simpler (shallower) architecture. A widened ResBlock (W-ResBlock, $w = k$) can achieve better performance with similar parameter amount but much less depth. To be specific, the author observes that W-ResNet-28-10 outperforms ResNet-1001 on CIFAR-10 classification task with margin of 0.65% regarding classification error($\%$).
- Significant advantage over computation efficiency. The author reports a 2-time-faster training speed with 50-time-less layers in general. This results from the fact that GPU is better in parallel computations when handling large tensors.
- Since the previous point implies that a W-ResNet with a similar amount of parameters to a ResNet can actually run faster, it allows the W-ResNet to be further widened (up to W-ResNet-28-10, according to the author) to achieve an even better result with comparable inference time.

Therefore, a W-ResNet is a proper option for lightweight feature extractor. In this work, a W-ResNet-2-10 with a dense-layer head to form the feature extractor (FE). Contrastive learning (the *SimCLR* algorithm presented by CHEN ET AL. in 2020 [Che+20b]) is employed to train this FE in a self-supervised fashion.

3.3 Monodepth2 for Depth Estimation

The *Monodepth2* model, presented by GODARD ET AL. (2019) [GAB19], is a state-of-the-art self-supervised depth estimation algorithm, dedicated to generating depth map(i. e. per-pixel depth estimates). The whole algorithm consists of two models for training stage: a Depth Net for pixel-wise depth map, and a Pose Net to derive egomotion. Only Depth Net will be used for inference, while the Pose Net assists in constructing geometric constraints.

3.3.1 The Depth Net and Pose Net

Both Depth Net and Pose Net are composed of encoder-decoder models. Table .1 in Appendix .1 shows the architecture of their decoders. Two models share the same encoder architecture: the ResNet-18 (introduced in Section 2.2.2) with input in the shape of $(H = 192, W = 640, 3)$, where H and W are height and width respectively.

For Depth Net, the encoder-decoder adopts *short-cut* connections proposed in the *U-Net* model, proposed by OLAF ET AL. (2015) [RFB15], which bridge the encoder and decoder. This allows low-dimensional features to be taken into account when decoding the high-dimensional feature from the bottleneck.

In decoding phase, the Depth Net will first use a Conv-layer to decode the feature map, apply up-sampling to increase its resolution, and merge with the short-cut feature passed from the encoder. Finally, the a disparity map is generated by the Conv-layer (denoted as *disp-conv*). The final disparity map has the same size as the original input image, but with only one channel for depth, i. e. $(192, 640, 1)$.

For Pose Net, the encoder takes a pair of RGB-images as input, therefore the input shape is $(\text{height}, \text{width}, 6)$. In practice, we duplicate the first Conv-layer in depth (channel dimension) to allow such adjustment, and leave the other rest of the ResNet architecture unchanged. The architecture is displayed in the second part of Table .1 in Appendix .1.

the decoder of Pose Net will decode the bottleneck without short-cut connections. The final estimation are six egomotion components: $(t_\mathrm{x}, t_\mathrm{y}, t_\mathrm{z}, \theta_\mathrm{x}, \theta_\mathrm{y}, \theta_\mathrm{z})$, representing for 3-D translation and axis-angles (introduced in Section 2.1.1), respectively.

3.3.2 The Training Strategy

The Monodepth2 is a self-supervised-learning algorithm, meaning that the loss is not by comparing prediction with GT, but by constructing losses from the predictions (and input data) themselves. This makes the construction of losses the very essence of the training strategy. Therefore, the relevant losses will be studied first, before the overall training strategy is presented.

For clarity, let us state some basic denotations: the *target* (current) frame is denoted as t, its *source* (neighbor) frame as t', which makes the input image pair as $\boldsymbol{I}_\mathrm{t}$ and $\boldsymbol{I}_{\mathrm{t}'}$. Naturally, the

$t' = t \pm 1$ can either be the previous or the next frame to frame t. The egomotion from the t' to t is expressed as $\boldsymbol{T}_{t \to t'} = (t_\mathrm{x}, t_\mathrm{y}, t_\mathrm{z}, \theta_\mathrm{x}, \theta_\mathrm{y}, \theta_\mathrm{z})$.

Photometric Reprojection Loss

The photometric reprojection loss, $\boldsymbol{L}_{photo}$, is calculated pixel-wise difference/error between two images. As introduced in Section 2.1.4, *image (inverse) warping* can pixel-wise map the the source image $I_{t'}$ pixel back to its position on the target image $\boldsymbol{I}_t$, as long as the depth map $\boldsymbol{D}_t$, transformation matrix calculated from egomotion $\boldsymbol{M}_{\mathrm{t} \to \mathrm{t}'}$, and intrinsic parameters $\boldsymbol{K}$ are acquired. The inverse-warped image from source to target $\boldsymbol{I}_{t' \to t}$ is calculated by

$$\boldsymbol{I}_{t' \to t} = \boldsymbol{I}_{t'} \left\langle \operatorname{proj} \left(\boldsymbol{D}_\mathrm{t}, \boldsymbol{M}_{\mathrm{t} \to \mathrm{t}'}, \boldsymbol{K} \right) \right\rangle , \tag{3.7}$$

where *proj*$(\cdot)$ is the operation to project $\boldsymbol{D}_t$ in $\boldsymbol{I}_t$, while $\langle \cdot \rangle$ here is the operation of sampling (e. g. the bilinear sampling introduced in Section 2.1.4). The original author follows the concept of *Spatial Transform Network* presented by JADERBERG ET AL. (2016) [Jad+16], and uses the bilinear sampling to sample the $\boldsymbol{T}_{t'}$.

To include the similarity of the scene into the loss, instead of simple local pixel comparison, the structural similarity (SSIM) introduced by WANG ET AL. (2004) [Wan+04] is used (for more details see Appendix .1). The SSIM models the errors of brightness, contrast and scene structure separately. Specifically, mean represents the brightness, variance is for contrast, and covariance is for the similarity of the structure.

Finally, the $\mathrm{err_p}$ is defined as the weighted sum of SSIM and pixel-wise difference (L1 loss):

$$\mathrm{err_p} \left(\boldsymbol{I}_\mathrm{t}, \boldsymbol{I}_{\mathrm{t}' \to \mathrm{t}} \right) = \frac{\alpha}{2} \left(1 - \operatorname{SSIM} \left(\boldsymbol{I}_\mathrm{t}, \boldsymbol{I}_{\mathrm{t}' \to \mathrm{t}} \right) \right) + (1 - \alpha) \left\| \boldsymbol{I}_\mathrm{t} - \boldsymbol{I}_{\mathrm{t}' \to \mathrm{t}} \right\|_1 , \tag{3.8}$$

where $\alpha = 0.85$ is hyper-parameter empirically recommended by the author.

Disparity Smoothness Loss

To encourage local continuity for the estimated disparity, a smoothness loss ($\boldsymbol{L}_{smooth}$). In addition, the loss will be weakened with edge-awareness when facing actual discontinuity in

object border, through the term of image gradient ∂I.

$$\boldsymbol{L}_{\text{smooth}} = |\partial_{\text{x}}\mathbf{disp}^{*}_{\text{t}}|\, e^{-|\partial_{\text{x}}\boldsymbol{I}_{\text{t}}|} + |\partial_{\text{y}}\mathbf{disp}^{*}_{\text{t}}|\, e^{-|\partial_{\text{y}}\boldsymbol{I}_{\text{t}}|}, \tag{3.9}$$

where the $\mathbf{disp}^{*}_{\text{t}}$ is disparity map $\mathbf{disp}_{\text{t}}$ normalized by its mean. According to the work by WANG ET AL. in 2018 [Wan+18], it punishes the drastic declining of the depth estimates (represented by the inverse of the disparity).

Refine The Losses

As photometric error, $\mathbf{err}_{\text{p}}\left(\boldsymbol{I}_{\text{t}}, \boldsymbol{I}_{\text{t}'\to\text{t}}\right)$, is computed twice because t' can be both $t-1$ and $t+1$, the author proposes 1) *minimum projection loss*; 2) *auto-masking*; 3)*full-resolution multi-scale refinement*, to further refine the $\boldsymbol{L}_{\text{photo}}$ and $\boldsymbol{L}_{\text{smooth}}$.

To begin with,*minimum projection loss* works as follows: Only pick the minimum between $\mathbf{err}_{\text{p}}\left(\boldsymbol{I}_{\text{t}}, \boldsymbol{I}_{\text{t}-1\to\text{t}}\right)$ and $\mathbf{err}_{\text{p}}\left(\boldsymbol{I}_{\text{t}}, \boldsymbol{I}_{\text{t}+1\to\text{t}}\right)$, instead of simple averaging. The photometric projection loss is refined to:

$$\mathbf{err}'_{\text{p}} = \min_{t'\in(t-1,\,t+1)} \mathbf{err}_{\text{p}}\left(\boldsymbol{I}_{\text{t}}, \boldsymbol{I}_{\text{t}'\to\text{t}}\right). \tag{3.10}$$

Auto-masking is to mask out outlier regions: The self-supervised depth estimation algorithms share a common assumption, that the scene is static while the camera is moving. In practice, however, it is inevitable that there will be moving objects (e. g., cars, pedestrians), or the camera stops moving (e. g. holding still facing a traffic light). These outliers will cause degeneration of the training results, manifesting a hollow *hole* in disparity map and indicates an infinity value (see Fig. 3.6). Based on the observation that same pixel value implies a camera holding still, or a moving object with similar relative speed, the author proposes a binary mask to mask out these particular regions

$$\mu = \left[\min_{\text{t}'} \mathbf{err}_{p}\left(\boldsymbol{I}_{\text{t}}, \boldsymbol{I}_{\text{t}'\to\text{t}}\right) < \min_{\text{t}'} \mathbf{err}_{p}\left(\boldsymbol{I}_{\text{t}}, \boldsymbol{I}_{\text{t}'}\right)\right], \tag{3.11}$$

where $[\cdot]$ is the Iverson bracket. The resulting binary mask μ will be applied to the $\boldsymbol{L}_{\text{photo}}$ to avoid the photometric loss to be contaminated by the outlier regions.

As for **full-resolution multi-scale refinement**, it is employed to refine the details of the disparity map. It has been observed (in Monodpeth2 [GAB19]), that following flaws will

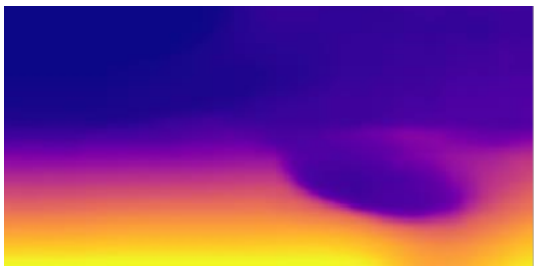


Figure 3.6: Example of hollow hole caused by: 1) an object with significantly different speed relative to the camera; 2) large texture-less region.

occur when only final disparity map is utilized for $\boldsymbol{L}_{\mathrm{p}}$ calculation: 1) the large texture-less region tends to cause a hole on the disparity map (see Fig. 3.6); 2) texture-copy artefacts, which means that texture in RGB image will be mistakenly transferred to the estimated disparity map.

The author therefore proposes to combine photometric loss at multiple scales. Since the Depth Net encoder has four short-cut connections from encoder to the decoder besides the final output, the decoder can generate an intermediate disparity map $\mathbf{disp}_s$ after each connection.

The subscript s stands for *scale*, e. g. $s = 0$ indicates the original resolution $(192, 640)$. And as the s increases by one, the resulting resolution will be decreased by a factor of two, e. g. $\mathbf{disp}_{s=2}$ has the size of $(96, 320)$.

Disparity maps at each scale will be up-sampled to the original resolution, and participants in calculation of $\mathbf{err}'_{\mathrm{p}}$ by Eq. 3.10. And finally $\boldsymbol{L}_{\mathrm{photo}} = \sum_s \mathbf{err}'_{s,\mathrm{p}}$.

In addition, the smoothness loss will also be refined by this multi-scale technique. Specifically:

$$\boldsymbol{L}_{s,\mathrm{smooth}} = \boldsymbol{L}_{\mathrm{smooth}}/s^2. \tag{3.12}$$

Overall Training Strategy

The overall training strategy can be illustrated with the Fig. 3.7. Since $I_{\mathrm{t}-1}$ and $I_{\mathrm{t}+1}$ have the same behavior in training, they are all referred as *source* image ($I_{\mathrm{t}'}$) to simplify the notation. And the I_t is referred as the target image.

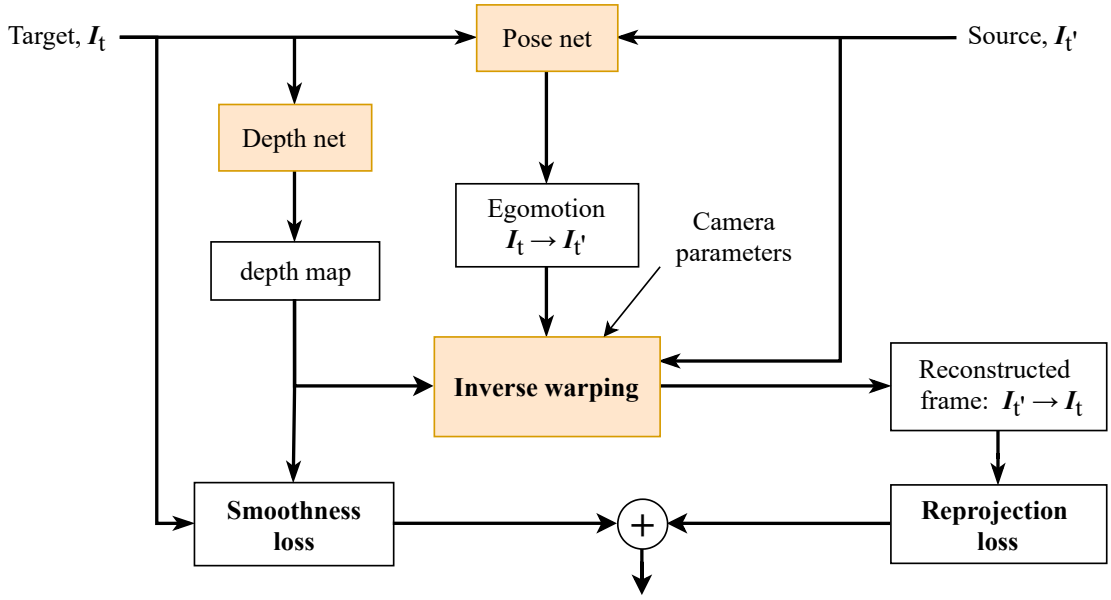


Figure 3.7: The self-supervised training pipeline of the *Monodepth2* algorithm [GAB19]. The Depth net estimates the depth map for target view $\boldsymbol{I}_{\mathrm{t}}$, while the Pose net estimates the egomotion from target to source view. Combining with camera parameters, a bilinear sampling function will be calculated for the *inverse warping*, which reconstructs the $\boldsymbol{I}_{\mathrm{t}}$ by $\boldsymbol{I}_{\mathrm{t}'}$. Comparing the reconstructed target with original target image will lead to *reprojection loss*. In addition, the *smoothness loss* is calculated through the depth map generated by $\boldsymbol{I}_{\mathrm{t}}$. The two losses will form the final loss.

The training pipeline can be summarized as

1. The source-target pair is fed to the Pose Net and produce the egomotion, i. e. the translations and axis-angles $(t_{\mathrm{x}}, t_{\mathrm{y}}, t_{\mathrm{z}}, \theta_{\mathrm{x}}, \theta_{\mathrm{y}}, \theta_{\mathrm{z}})$, which are combined into the transformation matrix $\boldsymbol{M}_{\mathrm{t}\rightarrow\mathrm{t}'}$. To keep the temporal order, $\boldsymbol{I}_{\mathrm{t}'}$ and $\boldsymbol{I}_{\mathrm{t}}$ are swapped when $t' = t - 1$.

2. Meanwhile, the target I_t alone will be fed to Depth Net and generate the raw estimation of disparity map ($\mathbf{disp}_{\mathrm{t}}$), which is then inversed and normalized to depth map ($\boldsymbol{D}_t$) by Eq. 2.20.

3. The $\boldsymbol{D}_{\mathrm{t}}$, $M_{\mathrm{t}\rightarrow\mathrm{t}'}$, known camera intrinsic parameters $\boldsymbol{K}$, are used to calculate the projected depth $\boldsymbol{D}_{\mathrm{t}\rightarrow\mathrm{t}'}$. The inverse warping is then conducted by Eq. 3.7 to re-project the $\boldsymbol{I}_{\mathrm{t}}$ as $\boldsymbol{I}_{\mathrm{t}\rightarrow\mathrm{t}'}$.

4. The photometric reprojection loss $\boldsymbol{L}^{s}_{\text{photo}}$ at each scale is calculated. And with D_t and $\boldsymbol{I}_{\text{t}}$, The smoothness is calculated by Eq. 3.9 and refined by Eq. 3.12 at each scale.

5. The final loss $\boldsymbol{L}_{\text{total}}$ is then calculated by $\boldsymbol{L}_{\text{total}} = \sum_{s=0}^{4} (\boldsymbol{L}_{s,\text{photo}} + \boldsymbol{L}_{s,\text{smooth}})$.

With training methods introduced, it is reasonable to discuss the pros and cons over the choice of Monodepth2:

- The input data for both training and inference are simple. It requires only monocular image (and camera parameters) as inputs, no additional information needed, such as *stereo pairs* in the work by GODARD ET AL. (2017) [GAB17]; or *segmentation masks* in the *Struct2Depth* model presented by CASSER ET AL. (2019) [Cas+19].
- The components and structure of training is less complicated. There is no auxiliary motion modelling (e. g., modelling each moving object) like in the *GeoNet* model presented by YIN AND SHI (2018) [YS18].
- Due to its simplicity, the amount of hyper-parameters is relatively small, thereby more convenient to be tuned.
- It also supports training with stereo pairs (or mixed monocular images), which extends possibility to further study. In addition, according to the author, the *minimal projection* method will be extra beneficial to training result.

3.4 Object Motion Derivation

The *full motion* includes both distance and velocity. Since the velocity is the derivative of the position/distance with time, it can be derived directly from the distance/depth estimates with known time interval or FPS of the video. Generally there are two directions, depending on whether GT of the distance (and velocity, if available) is known, in other words, supervised or direct-derivation methods. The following sections describe two methods, respectively.

3.4.1 Supervised-Learning Method

A supervised network can be applied when there is enough GT data of motion (at least distance). If not directly available, the GT data can still be obtained with known FPS with Eq. 2.21. Considering that our depth estimation is achieved self-supervised (see Sec-

tion 3.3 for details), where the losses are elaborately designed regarding the photometric and geometric constraints, it is impractical to extend the architecture for further purpose.

However, there exist other works, e. g. [KMF18], [Son+20], that employ end-to-end networks to estimate the full-motion directly from the image data with pre-computed detection results. Their methods can be summarized as follows:

1. Combine the pre-computed BBoxes and motion networks, i. e. depth- and optical-flow-estimation network. There are two ways to combine:

   a) Use the BBoxes to crop out image patches into patches (regions of interest), and feed the patches into motion network;

   b) Alternatively, the whole image is fed to motion networks respectively, and the BBoxes are used to crop out motion maps, i. e. the depth and optical-flow map.

2. After patches of motion maps are obtained, a Multi-Layer Perception (MLP) head is used to derive the full motion. To contextualize this kind of MLP, it will be referred as the *Motion-MLP*.

Motion-MLP

Inspired by the works mentioned above, it is reasonable to concatenate a Motion-MLP after the Monodepth2 to bridge the gap between depth map (from Monodepth2) and final motion estimates. Similar to the method in [Son+20], the motion map (depth only, in our case) is cropped into patches by detected-BBoxes, and then the MLP is used to leverage the following information:

- The patches of pixel-wise motion estimates from the current and the previous frame. The same identities will be tracked by DeepSort (see Section. 3.2) to form a pair of patches for each identity in temporal order.
- The six geometric hints: $\boldsymbol{g}_i = \left(\frac{f_\mathrm{x}}{r_i - l_i}, \frac{f_\mathrm{y}}{b_i - t_i}, \frac{l_i - c_\mathrm{x}}{f_\mathrm{x}}, \frac{t_i - c_\mathrm{y}}{f_\mathrm{y}}, \frac{r_i - c_\mathrm{x}}{f_\mathrm{x}}, \frac{b_i - c_\mathrm{y}}{f_\mathrm{y}}\right)$, where (l, t, r, b) stand for left-, top-, right- and bottom coordinates of a BBox, i is the BBox identity, $(f_\mathrm{x}, f_\mathrm{y})$ are focal lengths, and $(c_\mathrm{x}, c_\mathrm{y})$ represent the principal point of the image plane.
- Finally, the FPS, a single digit as the hint of time.

Specifically, each pair of depth patches will be *max-pooled* into a 2-D feature map with shape

of $(13, 13)$, and *flattened* into the final 1-D feature vector $\boldsymbol{f}_i$ with shape of $(1, 169)$.

The input dimension of the Motion-MLP is then obtained: $2 \times (169 + 6) + 1 = 351$. And we construct the MLP with 3 hidden layers, whose amount of units are in descending order; $256, 128, 64$. And the output dimension are four-digits of full-motion denoted by $\boldsymbol{m}_i$, i. e. $\boldsymbol{m}_i = \left(d_{\mathrm{x}}^i, d_{\mathrm{y}}^i, v_{\mathrm{x}}^i, v_{\mathrm{y}}^i\right)$.

The pipeline of Motion-MLP usage can be summarized as Fig. 3.8.

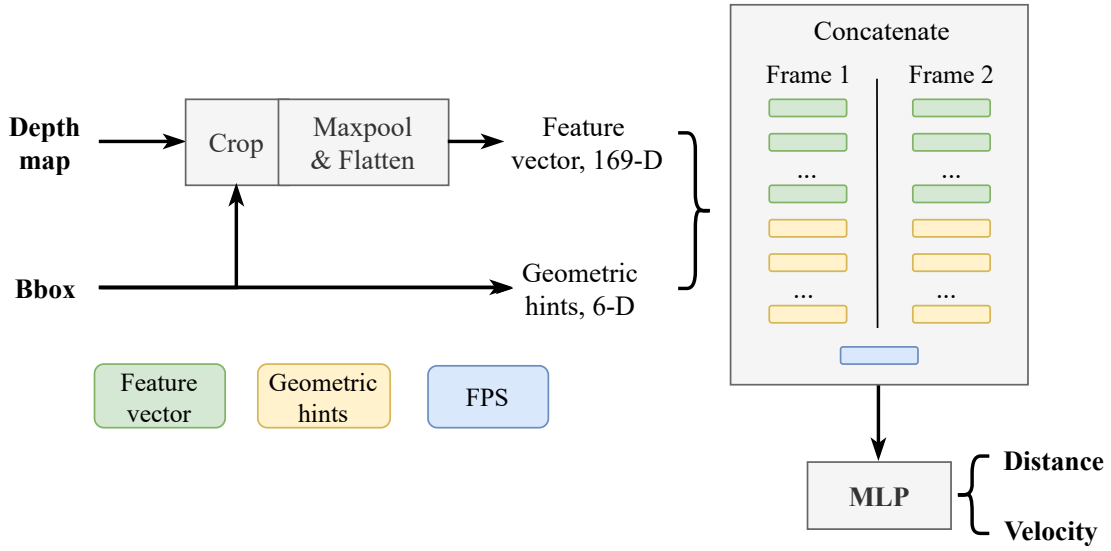


Figure 3.8: The schematic of the usage of Motion-MLP. The BBOx is used to crop out object-containing regions out of depth map. Feature vectors are composed of both depth and geometric features. Pair-wise feature vectors and FPS are passed to MLP to estimated the distance and velocity of the object represented by the BBox.

3.4.2 Generalized Method for Unfamiliar Scenes

For the case where the GT motion is available, supervised methods are always feasible. However, the motion sensors, e. g. the LiDAR, are not always available in real-world applications. Besides, the performance for supervised-learning methods will deteriorate when facing a scene not similar to that of the training data. Therefore, even if the model is already well-trained with a dataset with GT, it cannot be tuned when facing an unfamiliar scene.

With limitations of supervised-learning methods taken into account, it is necessary to utilize a method that can directly derive velocity from the depth estimates and temporal hints. In this way, the full-motion derivation can be completely supervision-free. The overall pipeline will be firstly presented, and then follows the elaboration:

1. Use the BBox to crop out the depth-map patch of each identity. And the patch is processed into a final estimate of depth.

2. To project the depth into 2-D distance, he BBox and camera intrinsic parameters are used to estimate the yaw angle (horizontal rotation angle between the camera and the target).

3. The Kalman Filter is then used to derive the full motion out of the raw depth estimates in the first step.

The cropped depth map is cropped by a square BBox instead of a fine segmentation mask, and needs to be processed in to one final value. Since background or overlapped objects will inevitably be included, it is necessary to alleviate contamination from these alien objects. The observation, demonstrated in Fig. 3.9, is as follows:

- The contamination from the background, which will be estimated to be far larger (farther) depth values than the target object. And it is normally around the borders, not likely to be in the central area.

- The occlusion, however, is the objects and pedestrians coming from the horizontal direction that might cover some of the central area. And their depth estimates will be smaller (nearer) than the target.

- The depth map itself also has flaws: The upper-part of the patch will sometimes fade into the background and get a smaller value; The estimation near the boundaries of the image will not be as accurate as the rest.

To deal with the problems stated above, we first simply *shrinking* the cropped area towards the center, which filters out the most of the background and some of the occlusion. As the Fig. 3.9(b) shows: To handle the fading-into-the background problem, larger portion (20%) on the upper region is taken than the other three direction (10%), leaving 50% the area of the original patch to be further processed.

Afterwards, *the mean of the middle interval* of the shrunken patch will be calculated as the final estimate. Specifically, we sort the depth estimates by value, and average the 30% of the

(a) (b)

Figure 3.9: Examples of contamination when averaging the depth-map patch by BBoxes. (a) Since a BBox can include extra objects, which might be the occlusion or background, the mean value of the depth-map patch cannot accurately reflect the object's depth estimate. (b) The depth-map patch's upper region is slightly smaller compared to the main area, sometimes even fade into the background. Therefore, before calculating the mean depth, the BBox is shrunken to handle above-mentioned flaws.

middle interval to be the raw estimate representing the depth of the target. This can,,to some extent, filter out outlier values from occlusion and fading edge areas.

Next, the KF will model the full motion $\boldsymbol{m}$ from the raw estimate of depth), where $\boldsymbol{m} = [d_x, d_y, v_x, v_y]$, are parameters to be modelled as described in Section 2.4.3. For each frame, the raw depth estimate with its identity will be passed in, where new identities initiate a new KFs. Previously existing KFs will perform prediction-and-correction to produce the final estimate on depth and its derivative with time (i. e., the velocity). Naturally, this method also works for 2-D distance, which will produce the corresponding 2-D velocity.

Since the KF will be initiated with a large variance to indicate the uncertainty in the initial phase, the KF-estimated full motion is not robust nor stable enough for direct use. Therefore, a waiting phase needs to be set before the estimates are actually usable. Since we process the image frame-wise, one way to set the phase by frame number is to introduce the FPS to calculate the time [s], which is invariant against different FPS settings for different cameras.

4 Proposed Object Motion Estimation System

4.1 Overall Pipeline

As shown in Fig. 4.1, this work dissembles the task into the following sub-tasks:

1. Object Detection (OD) and multi-object tracking (MOT), which are carried out with detectors introduced in Section 3.1 and Section 3.2. Since our first choice for detector is YOLOv4, the phrase *detector* is by default for YOLOv4, unless stated otherwise.
2. The depth map estimation achieved by Monodepth2 (Section 3.3).
3. Combine the 1 and 2 to estimate the rough depth for individual objects, and derive their final estimates on depth and distance through methods described in Section 3.4.

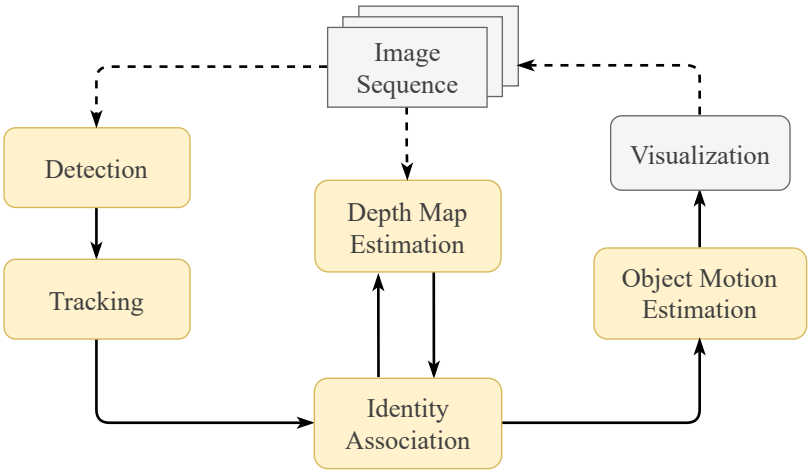


Figure 4.1: The task breakdown of our motion estimation pipeline. For each image/frame, object detection and tracking are applied to generate tracked BBoxes; meanwhile a depth map is estimated. The tracked BBoxes are then used to determine the regions of interest in the depth map, which leads to depth-map patches. Finally, the object-wise motion estimation is achieved based on the patch.

4.1.1 The General Solutions

Since each algorithm's detailed mechanisms have been elaborated in the previous chapter, the rest of the section will focus on how those algorithms connect to or interact with each other at each step.

Object Detection

To begin with, a RGB image $\boldsymbol{I}(t)$ at time step t, with shape $(H_{\text{in}}^{\text{od}}, W_{\text{in}}^{\text{od}}, 3)$, is passed to the detector. The detection results are denoted as Det^{od}, in total of N. Note that the Det_i ($i = 1, 2, 3, \cdots, N$) will be enriched with new members, as each step contributes new results, thereby a subscript is given to distinguish between different steps.

A Det_i^{od} has the following members for each Det_i^{od}:

1. the BBox defined by the coordinates of its two vertices on the diagonal, which could follow the convention of left-top-right-bottom ($ltrb$) but can also be in other conventions as needed;
2. the class label represented by one integer. The COCO-dataset, on which the detector is trained, has 80 classes. However, we only need car-alike categories: car, bus and truck, while categories like person, motorbike and bicycle are not in the scope of this work;
3. the confidence score, which range between $[0, 1]$ to indicate the probability of the detection is valid.

Object Tracking

The DeepSort, the Multi-Object tracker (MOT), is placed after the OD. The DeepSort will first utilizes the *feature extractor* (FE) to generate a 1-D feature vector with $\boldsymbol{f}_i^{\text{trk}}$ shape $(1, 128)$. Specifically, the each BBox in Det_i^{od} is used to crop out a patch P_i out of $\boldsymbol{I}(t)$ to feed the FE. The resulting feature vectors $\boldsymbol{f}_i^{\text{trk}}$ are added to the corresponding Det_i^{od} that forms $\text{Det}_i^{\text{trk}}$ as the input of DeepSort.

If the DeepSort can find a match for a current detection $\text{Det}_{i,\text{in}}^{\text{trk}}(t)$ in previous detections $\text{Det}_{i,\text{in}}^{\text{trk}}(t - T)$, it will be called a tracked detection. Note that T is the search range/lifespan that the DeepSort will take into account. Upon the first successful tracking, a unique ID is

given to the detection, representing the same object tracjecting with time. A set of $\mathrm{Det}(t)$ throughout time steps with the same ID is referred to as a **Track**.

The are IDs are then added to each detection $\mathrm{Det}^{\mathrm{trk}}_{i,\mathrm{in}}(t)$, which is updated to $\mathrm{Det}^{\mathrm{trk}}_{i,\mathrm{out}}(t)$. At this point, the object visual recognition (OD with MOT) of is finished. And $\mathrm{Det}^{\mathrm{od}}_i \in \mathrm{Det}^{\mathrm{trk}}_{i,\mathrm{in}} \in \mathrm{Det}^{\mathrm{trk}}_{i,\mathrm{out}}$.

Depth Map Estimation

Meanwhile, the raw image $\mathbf{I}(t)$ is input to Monodepth2, the depth-map estimator (DME). Since the network requires a fixed aspect ratio for input image, the raw $I(t)$ is center-cropped (denoted by $\boldsymbol{I}^{\mathrm{dme}}(t)$) to reach aspect ratio $r_{\mathrm{a}} = 640 : 192$, and resized to $(192, 640, 3)$ before fed to the network. Specifically:

$$W^{\mathrm{dme}} = W^{\mathrm{od}}, \tag{4.1}$$

$$H^{dme} = W^{\mathrm{dme}}/r_{\mathrm{a}}, \tag{4.2}$$

$$\mathrm{offset} = (H^{\mathrm{od}} - H^{\mathrm{dme}})/2 \tag{4.3}$$

where offset is the half of the cropped-out region in vertical direction. H^{dme} is the actual height (after center cropping) of the Monodepth2 input size. The Monodepth2 then generates the depth map $\boldsymbol{D}(t)$ with shape $(192, 640, 1)$.

To get depth estimation for individual objects, the $\boldsymbol{D}(t)$ is cropped out into patches defined by BBoxes in $Det(t)$. Notice that the BBoxes here is defined according to the raw image size $(H^{\mathrm{od}}, W^{\mathrm{od}})$, which might not fit the size of $\boldsymbol{D}$ with size $(192, 640)$. Therefore, offset and resizing is needed to adjust the BBox coordinates before they are applied on the $\boldsymbol{D}$. The final effect is illustrated in Fig. 4.2.

As a result, the depth patches $\mathbf{D}_i$ that represent the depth map of individual objects will be obtained.

Object-Wise Motion Estimation

At this point, there are two ways to derive the final depth and velocity from $\mathbf{D}_i$ depending on the availability of GT data:

Figure 4.2: Center cropping of the image with a aspect ratio different from Monodepth2's [GAB19] required input size. The yellow lines indicates the new boundary after cropping. The white BBox is the original label for the vehicle, while the red-dashed BBox is the BBox after the cropping.

1. If GT data of an individual's depth is available, the derivation from object-wise depth map ($\mathbf{D}_i$) to depth value ($\mathbf{d}_i$) can be solved by data-driven method, specifically the Motion-MLP (for more details are in Section 3.4.1)

2. In a more general situation where the GT is unavailable, the Kalman Filter (KF) is used (described in Section 3.4.2).

Motion-MLP requires information from both the current and previous frame, denoted by t_c and $t_\mathrm{p} = t - T$ (the choice of T is a hyper-parameter explained in Section 4.1.2). To begin with, the object-wise depth map $\boldsymbol{D}_\mathrm{k}(t_\mathrm{c})$ and $\boldsymbol{D}_\mathrm{k}(t_\mathrm{p})$ for each valid ID $\boldsymbol{k}$ are drawn, where $\boldsymbol{k}$ denotes the unique ID tracked across time.

Afterwards, **D**s are resized to a fixed size, following the *max-pool* that generates the feature vectors $\boldsymbol{f}_\mathrm{k}(t_\mathrm{c})$ and $\boldsymbol{f}_\mathrm{k}(t_\mathrm{p})$. The geometric hints $\boldsymbol{g}_\mathrm{k}(t_\mathrm{c})$ and $\boldsymbol{g}_\mathrm{k}(t_\mathrm{p})$ are calculated with BBoxes $\boldsymbol{B}_\mathrm{k}(t_\mathrm{c})$ and $\boldsymbol{B}_\mathrm{k}(t_\mathrm{p})$ respectively, as described in Section 3.4.1. The actual FPS, defined as:

$$\mathrm{FPS}_\mathrm{act} = \mathrm{FPS}_\mathrm{cam}/T, \tag{4.4}$$

where subscripts cam and act denotes the original FPS of the camera and the actual FPS used in Motion-MLP. T is the time interval defined by number of frames.

Finally, the complete input for the identity k at time t_c is formed:

$\left(\boldsymbol{g}_{\mathrm{k}}(t_{\mathrm{p}})^{\mathrm{T}}\boldsymbol{f}_{\mathrm{k}}(t_{\mathrm{p}})^{\mathrm{T}}, \boldsymbol{g}_{\mathrm{k}}(t_{\mathrm{c}})^{\mathrm{T}}\boldsymbol{f}_{\mathrm{k}}(t_{\mathrm{c}})^{\mathrm{T}}, \mathrm{FPS}_{\mathrm{act}}\right)$. The Motion-MLP outputs the full-motion estimation $\boldsymbol{m}_{\mathrm{k}}(t_{\mathrm{c}})$, where $\boldsymbol{m} = (d_{\mathrm{x}}, d_{\mathrm{y}}, v_{\mathrm{x}}, v_{\mathrm{y}})$ stands for the 2-D projections of the depth and its changing rate.

Kalman Filter (KF) only inputs either depth d or its 2-D projections $\boldsymbol{d}_{\mathrm{2D}} = (d_{\mathrm{x}}, d_{\mathrm{y}})$ for the current time step, where the latter can be derived by the former. For clarification, we assume that 2-D full-motion is our goal. And since KF doesn't require previous information, we will omit the t.

For starters, the square patch $\boldsymbol{D}_{\mathrm{k}}$ is shrunken in all four directions, and the raw depth estimate d^{est} is estimated, as described in Section 3.4.2. The included angle, between the camera-upfront and camera-target, as Fig. 4.3 shows, can be derived through the pinhole model (Section 2.1.2).

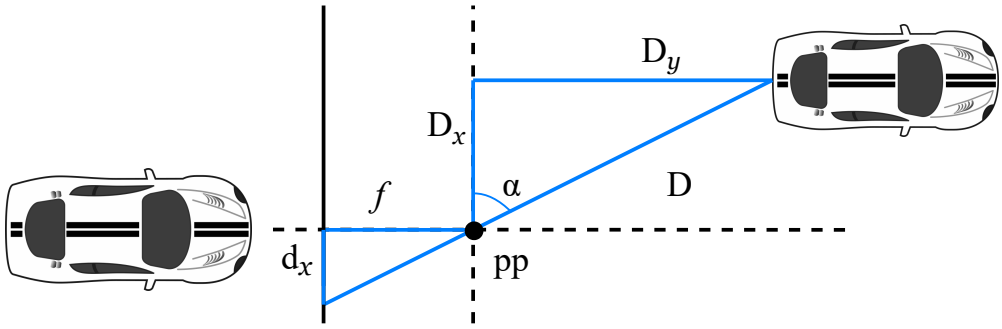


Figure 4.3: Usage of pinhole model to derive the included angle between the camera's principal point (pp) and the object vehicle. The triangulation similarity is established, so that the included angle α can be obtained. In this way, D_{x} and D_{y}, the 2-D projection of depth estimation D, are solved.

Denote the included angle α, camera's principal point coordinates $\mathbf{pp} = (\mathrm{pp_x}, \mathrm{pp_y})$, and define the BBox $\mathbf{B} = (c_{\mathrm{x}}, c_{\mathrm{y}}, \mathrm{w}, \mathrm{h})$ by *center-width-height* convention. The α can be calculated as follows:

$$b_{\mathrm{cx}} = c_{\mathrm{x}} + w/2, \tag{4.5}$$

$$b_{\mathrm{cy}} = c_{\mathrm{y}} + h/2, \tag{4.6}$$

$$\alpha = \tan^{-1}\left(\frac{b_{\mathrm{cx}} - pp_{\mathrm{x}}}{b_{\mathrm{cy}} - pp_{\mathrm{y}}}\right), \tag{4.7}$$

where (b_{cx}, b_{cy}) is the bottom-center position. Then the 2-D projections can be easily

calculated: $\mathbf{d}_{\mathrm{2D}} = [d\sin(\alpha),\ d\cos(\alpha)]$

Afterwards, the $\mathbf{d}_{\mathrm{2D}}$ is fed to KF as the *measurement* to *correct* its prediction and *update* the KF-model. The output will be $(d_{\mathrm{x}}, d_{\mathrm{y}}, v_{\mathrm{x}}, v_{\mathrm{y}})$ for 2-D input or (d, v) for 1-D depth input.

4.1.2 Implementation Details

In this subsection, implementation details regarding the algorithms mentioned in Section 4.1.1 will be elaborated. They are mainly

- important hyper-parameters of the networks/algorithms;
- the data processing, including the data augmentation in pre-processing, and the refinement methods in post-processing;
- the modifications/improvement based on the original proposals.

Notice that this subsection concentrates on the **procedure**, which follows the same sequence as the last subsection, namely *OD*, *MOT*, *DME* and *OMD*. The critical **reasoning** of parameterization and modifications will be continued in Section.4.2.

Object Detection

The *YOLOv4* [BWL20] described in Section3.1 is the default detector in this work, because it has excellent inference speed and good trade-off for accuracy. The original proposal for YOLOv4 use the input size of $(416, 416)$. The network produces numerous of detection proposals (BBoxes with class and confidence score), which needs to be further filtered by Non-Maximum Suppression (NMS). The lower IOU-threshold in NMS is, the less false-positive (FP) is, but the chance of false-negative (FN) cases will also increase. Compared to the default value of 0.3, the IOU threshold in this work is set to 0.38 to allow a little trade-off between TP and sensitivity to small/farther objects.

Although the thresholding by confidence is optional in NMS of YOLOv4, it is recommended in later object tracking. Therefore, the confidence threshold of default 0.5 is also included, omitting the detection with confidence less than 0.5.

Since the official work has released a well-trained model that includes the vehicle-related classes, namely car, bus and truck, there is no additional training in this work.

Object Tracking

As introduced in Section 3.2, the core of DeepSort tracking lies in two-stage matching, namely *cascade matching* and *IOU matching*.

Cascade Matching includes two metrics: *Mahalanobis Distance* (d_{mah}) to match parameterized BBoxes (center coordinates, aspect ratio and height); and *Cosine Distance* ($d_{\cos}$) to match the appearance-feature vectors.

We follow the default settings for $d_{\cos} = 0.3$, and d_{mah} equals 9.49 for 2-D mode and 5.99 for 1-D depth mode. Specifically, the d_{mah} is taken from the 95% confidence interval computed from the inverse χ^2 distribution [Dod08] with 4 and 2 degree of freedom for 2-D and 1-D mode, respectively. In theory, $d_{\cos}$ and d_{mah} should be combined by weighted summation, but experiment results have shown that using d_{mah} as a gating threshold will be more practical for a dynamic scene.

IOU Matching is a more straightforward process. The IOU values are calculated between each newly-detected BBox and the KF-predicted BBox in each **Track**. Notice that the KF-predicted BBoxes are not actually from detection results, but merely rough predictions based on previous updates of a KF model (see Section 2.4.3 for details of Kalman Filter).

The IOU-distance threshold in the original proposal $d_{\mathrm{IOU}} = 0.7$. However, the IOU is actually not an optimal metric, especially in a dynamic scene where location drift for a target is the combination of target motion and the camera's egomotion. Hence, the IOU metric is replaced with *DIOU* (Eq. 2.15), allowing two BBoxes with no intersection to be matched. The threshold d_{IOU} is thereby replaced with 1.0.

There are hyper-parameters worth mentioning regarding **background processing** of a individual **Track**: the *budget* (i. e. the maximum number) of the feature vectors, and its maximum *age*:

- A *budge* set for a **Track** to only preserve a fixed number of the newest appearance. This helps reduce the unnecessary comparison between newly-detected BBoxes with features too old to reflect the latest appearance. It is by default set to 70 for tracking the *people*, which are obviously slower and stays longer in the image. The vehicles, however, tend to have a shorter stay in the image. Hence *budget* is set to 35, which also increases the running speed slightly.
- The *maximum age* is the maximum time allowance for a **Track** to be lost, i. e. once

the **Track** cannot find a match in *age* frames, it will be reckoned as lost and be deleted. The default setting of $age = 3$ is adopted in this work.

As for the **feature extractor** (FE), the *W-ResNet* [ZK17] with the depth of 10 and width of 2 is chosen.

In the original proposal of DeepSort, the FE is trained through the *Deep Cosine Metric Learning* proposed by WOJKE AND BEWLEY (2018) [WB18], which is a supervised-learning method dedicated to person Re-ID problem. In this work, we utilizes a off-the-shelf method of *contrastive learning*, specifically the *SimCLR* [Che+20b] algorithm. It is a self-supervised-learning framework that pre-trains any backbone/encoder network to learn feature representation based on raw data. Specifically:

- It temporarily removes the head of the original network, and concatenate a new projection head (PH). Following the original proposal, the PH is a MLP composed of two hidden layers. The Unit numbers, 128 and 64, are chosen to match the dimension of the W-ResNet output. In short, each RGB image is finally processed to be a 64-D feature vector.
- The *cosine similarity* is calculated between each objects within the batch and form the final loss.

As the Fig. 4.4 demonstrates, the resulting new model can be (pre-)trained without any supervision, since it learns the feature representation by comparing individual objects. The unlabelled data is from the **KITTI** dataset [Gei+13]. After training is finished, the first hidden layer of the PH is preserved as the head that projects a feature map (output of the W-ResNet backbone) to a 128-D vector. This feature vector will be used as appearance information, which is matched with the metric *cosine distance*.

Depth Map Estimation

The *Monodepth2*, as introduced in Section 3.3, is a self-supervised learning algorithm that estimates the depth map. During the training stage, the input are image pairs and Depth-Net and Pose-Net (both with encoder-decoder structure) work together to construct the losses. But for inference, only one single image and Depth-Net are needed. **Although the official code is released, we re-implement a simplified version and obtained comparable results by training from scratch**. The following will elaborate on the implementation details, which

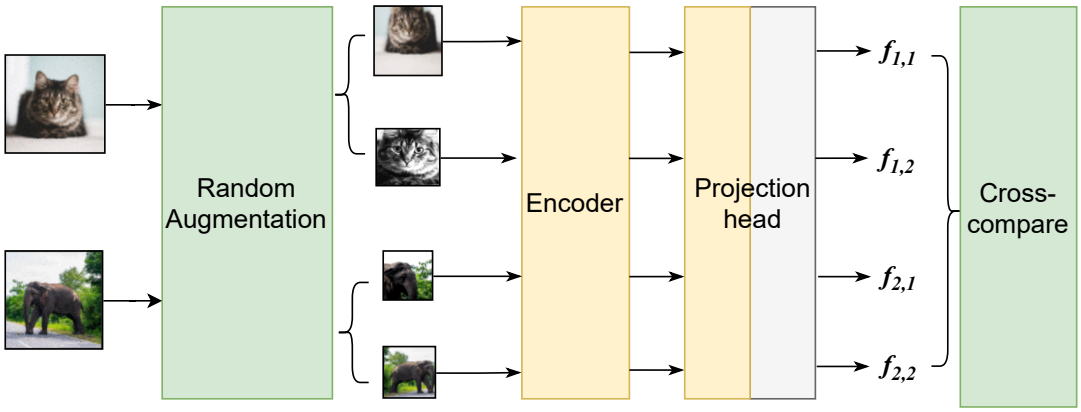


Figure 4.4: The SimCLR model is composed of: 1) a self-defined encoder/backbone, 2) two additional dense layers as *projection head*. The pipeline of the contrastive training by SimCLR framework. Each time, a pair of images is applied with random augmentation, resulting in four variants. Each variant will be passed to the SimCLR model, which generates a fixed-length feature vector. Loss is computed by cross-compare the cosine similarity between each vector. The encoder will gain the capability to learn feature representation. In this work, the first dense layer of the projection is directly taken as the head to project the feature map into a 128-D feature vector.

generally follow the original proposal unless stated otherwise.

KITTI dataset [Gei+13] is chosen to train the model (see Section 5.1.1 for details regarding is dataset). The raw image size of **KITTI** is $(1242, 375)$ which generally fits the aspect-ratio requirement of the network input $(640, 192)$. Hence, no center-cropping is needed. The complete dataset includes 93 thousand depth maps with corresponding raw LiDAR scans and RGB images. Although the former is not needed in training, it could still serve as a qualitative indication of how well the model has been trained.

We follow the **Eigen split** [EPF14] to re-group the dataset into the training, validation and testing set. Pre-selection proposed in *SfMLearner* [Zho+17] is adopted to filter out static scenes, which leads to 39,810 triplets (i. e. images in previous-current-next order, denoted by $\boldsymbol{I}_{t-1}, \boldsymbol{I}_t$ and $\boldsymbol{I}_{t+1}$) are perserved for training and 4,424 for validation. The camera intrinsic parameters, including focal length and principal point are provided and treated as known parameters. No distortion effect is taken into account.

During **data preparation**, the loaded images are firstly grouped into triplets (in temporal

order). Afterwards, triplets are (for 50% chance) applied with **augmentation** of *brightness, contrast, saturation* with a factor randomly chosen from $[-0.2, 0.2]$, and *hue* with random factor between $[-0.1, 0.1]$. In addition, left-to-right flipping is randomly applied to the images with 50% probability. Notice that all the augmentation within one triplet is the same, which allows the $\boldsymbol{I}_{\mathrm{t}-1}, \boldsymbol{I}_{\mathrm{t}}$ and $\boldsymbol{I}_{\mathrm{t}+1}$ to be augmented the same way.

The total **loss** is the weighted sum of reprojection loss and smoothness loss (elaborated in Section 3.3.2):

$$L_{\mathrm{total}} = \alpha_{\mathrm{p}} \cdot L_{\mathrm{photo}} + \alpha_{\mathrm{s}} \cdot L_{\mathrm{smooth}}, \tag{4.8}$$

where $\alpha_{\mathrm{p}} = 1.0$ and $\alpha_{\mathrm{s}} = 1 \times 10^{-3}$. The values are chosen experimentally by the author of *Monodepth2* [GAB19].

The **training** is carried out with learning rate (*lr*) of 10^{-4} for first step of 10 epochs with a *decay* of 0.1 for fine-tuning phase. After training by the code of our own version, it has been observed that the improvement is scarce after 10 epochs and the quality is already at a high level. Nonetheless, it is still slightly outperformed by official results (quantitative comparison in Section 6).

Apart from the straightforward training strategies mentioned above, the author [GAB19] argued that no additional tricks (auto tuning the *lr*, freezing layers, etc.) are needed.

Object-Wise Motion Estimation

As introduced in Section 4.1.1, the object-wise motion derivation can (OMD) be achieved with Kalman Filters (KFs) or a Motion-MLP, depending on the usage of GT data.

The **KF method** models an object's motion and keeps updating with time. To begin with, let us define the dimensions of the input and full state as $\dim_z$ and $\dim_{\mathrm{x}}$, respectively. For 2-D motion, KF is fed with 2-D raw distance estimate, and outputs final 2-D distance and velocity estimates, i. e. $\dim_{\mathrm{x}} = 4$ and $\dim_{\mathrm{z}} = 2$. Similarly for 1-D mode, $\dim_{\mathrm{x}} = 2$ and $\dim_{\mathrm{z}} = 1$. In addition, the update time step $\mathrm{d}t = 1$, indicating a frame-wise update.

As described in Section 2.4.3, the hyper-parameters are the initial state (i. e. $t = 0$) of the following parameters:

- The covariance $\boldsymbol{P}_0$ that reflects uncertainty of the state estimate. Since there is no

pre-knowledge for an object's motion, it is set to a diagonal matrix with large numbers $\boldsymbol{P}_0 = \textbf{diag}(500, 500)$ for 1-D motion.

- The covariance $\boldsymbol{Q}_0$ that represents the *process noise* is initialized with a discrete constant white noise model, explained in BAR-SHALOM's book (2001) [BLK01]. Specifically for 1-D mode:

$$\boldsymbol{Q_0} = \begin{bmatrix} 0.5 \cdot \mathrm{d}t^4 & 0.25 \cdot \mathrm{d}t^3 \\ 0.25 \cdot \mathrm{d}t^3 & \mathrm{d}t^2 \end{bmatrix} \sigma_v^2, \tag{4.9}$$

  where $\sigma_v = 0.05$ is interpreted as the *acceleration increment*. Since we assume a minimal change of acceleration, the value is set to a small number.

- The covariance $\boldsymbol{R}_0$ stands for the *measurement noise*. Considering the physical meaning of the input is depth, which roughly ranges [0, 100], $\boldsymbol{R}_0 = \textbf{diag}(5, 5)$ is chosen to initialize the noise for 1-D mode.

4.2 Highlights and Main Contributions

In this section, important contributions and modifications based on the original proposals of respective algorithms will be highlighted and justified. For a more continuous understanding, the introduction will still follow the flow of the pipeline.

Note that the YOLOv4 and Faster-RCNN **detectors** are off-the-shelf models and will not be further discussed. The discussion on Kalman Filter (KF) will neither be extended, since there is not much to alter. In addition, **the following illustration will assume a 2-D motion mode** unless stated otherwise.

For DeepSort **MOT-tracking** algorithm, there are following points worth mentioning:

- The direct-gating mechanism is replaced with *soft-gating* in the *cascade matching* step. Originally, the d_{mah} between BBoxes from detection and KF-prediction, but the KF model is initialized with high uncertainties ($\boldsymbol{Q}_0$ and $\boldsymbol{P}_0$). Therefore, a certain frames of *buffer time* is needed before the KF-prediction is usable. We set this buffer-time of 0.5s, calculated by multiplying the camera's FPS with the number of frames.

- The choice of *contrastive training* over supervised training for FE. Although learning from GT data often yields better performance regarding the same dataset, the work

of labelling is even burdensome for tracking task than detection. Hence, a quick and unsupervised training is of better practical sense.

- The IOU metric is replaced with DIOU. As introduced in 3.2, it handles more dynamic scenarios by taking the non-intersection case and BBox geometric characteristics into account.

- (Optional) *assistant tracker* serving for each **Track**. To handle the occasional failures of object detection, e. g. due to smaller size or darker lighting conditions, a KCF tracker [Hen+15] is initiated by the BBox of each tracked detection. Since the KCF only requires GT-BBox in the initiation, it could automatically update for each frame. The implementation of KCF is done by OpenCV library [Bra00]. However, it proved it to be a failure through following qualitative observation:

  – The failure report has too many false-positive cases, i. e. it sometimes continues to update a random BBox even though the vehicle has left the image.

  – The frequent, object-wise initiation is costly, which slows the overall running speed significantly.

  Therefore, it is deemed impractical.

As for the Monodepth2 **depth-map estimation**, there is no structural change made. Although tentative ideas borrowed from other similar algorithms have been experimented, yet unfortunately, **none of them is proved to be effective compared to the original performance**. For the purpose of exploration, they are still listed below and will be shortly analyzed (see Chapter 5 for quantitative analysis):

- The additional *rotation-consistency* loss is added based on the Pose-Net results. In the original proposal, the axis-angles and translations are estimated by the Pose-Net. The resulting transformation matrix $\boldsymbol{T}_{\mathrm{t}\to\mathrm{t}'}$ with rotation matrix $\boldsymbol{R}_{\mathrm{t}\to\mathrm{t}'}$, along with estimated depth map $\boldsymbol{D}_{\mathrm{t}}$ are used for the *image warping* process (see Section 3.3). Naturally, the transformation in reverse temporal order should be symmetric. Therefore, we swap the image-pair sequence and estimate the reversed pose, where the axis-angles are combined to a reversed rotation matrix $\boldsymbol{R}_{\mathrm{t}'\to\mathrm{t}'}$. Ideally, the two rotation matrices should have the following relationship (Eq. 4.10), thereby the rotation-consistency loss

L_{rot} as Eq. 4.11:

$$\boldsymbol{I} = \boldsymbol{R}_{\text{t}\to\text{t}'}\boldsymbol{R}_{\text{t}'\to\text{t}}, \tag{4.10}$$

$$L_{\text{rot}} = \alpha_{\text{rot}} \frac{\|\boldsymbol{R}_{\text{t}\to\text{t}'}\boldsymbol{R}_{\text{t}'\to\text{t}} - \boldsymbol{1}\|^2}{\|\boldsymbol{R}_{\text{t}\to\text{t}'} - \boldsymbol{1}\|^2 + \|\boldsymbol{R}_{\text{t}'\to\text{t}} - \boldsymbol{1}\|^2}, \tag{4.11}$$

where $\boldsymbol{I}$ is the identity matrix; α_{rot} is the weight in the final loss. Setting a larger α_{rot} (greater than 1×10^{-3}) makes the training difficult to converge, while a smaller value (e. g. 1×10^{-4}) seems to have no observable influence at all.

- An additional *pseudo-depth* channel added to the input of Pose-Net. Inspired by the work of WANG ET AL. (2019) [Wan+19], we insert the depth map estimated by Depth-Net to image, resulting to a pseuo-RGBD image with shape $(192, 640, 4)$. Theoretically, it provides Pose-Net with richer information which should result in an improved pose estimate, thereby the precision of inverse-warping, the reprojection loss, and finally yielding an improved training result for depth estimation. But unfortunately, this modification damages the performance.

- A new loss based on the *real-world-size constraint* was also experimented. Inspired by *Struct2Depth* model [Cas+19], we use pre-computed BBoxes to crop out depth-map patches during training, and roughly estimate the object-wise depth value (D_{i}). Since BBoxes height h could reflect the real-world size when the focal length and vehicle height D_{approax} are roughly known as in Eq 4.12, we impose a constraint loss L_{sz} on the object-containing region.

$$D_{\text{approx}} \approx f_{\text{y}} \frac{H}{h}, \tag{4.12}$$

$$L_{\text{sz}} = \frac{1}{\bar{D}} \sum_{i=1}^{N} \left| D - D_{\text{approx}} \right|, \tag{4.13}$$

where the H_{obj} is the real height in world units; N is the number of detected objects; $\bar{D}$ is the mean of the depth patch as a normalizer. With no luck, the qualitative observation demonstrated severe degeneration of the depth map quality. Therefore, no further evaluation was carried out with this idea.

The Motion-MLP for OMD is an idea inspired by [KMF18] and [Son+20], where the velocity is estimated from the motion-related features (e. g. the patches of optical flow and depth map). We propose a certain simple **data augmentation** methods during its training stage,

leveraging the pre-knowledge of basic physics.

Assume that we have GT distance (1-D mode) of an object for one frame t, denoted by $\boldsymbol{m}_{\mathrm{t}} = (d_{\mathrm{t}}, v_{\mathrm{t}})$ and $\boldsymbol{I}_{\mathrm{t}}$, respectively. By estimating the depth of the same object for this frame and its previous frame ($\boldsymbol{I}_{\mathrm{t}-1}$), we have estimated depth $\hat{d}_{\mathrm{t}-1}$ and $\hat{d}_{\mathrm{t}}$, and thereby the velocity $\hat{v}_{\mathrm{t}}$.

The following augmentation could artificially create more scenes based on the above-mentioned scene:

- *A static scene*. By changing the GT: $\boldsymbol{m}_{\mathrm{t}} \rightarrow (d_{\mathrm{t}}, 0)$ and replace $\boldsymbol{I}_{\mathrm{t}-1}$ with $\boldsymbol{I}_{\mathrm{t}}$, we have a scene indicating the vehicle stands still.

- *A scene with reverse movement*. As d_{t} and v_{t} are given, it is natural that the previous distance $d_{\mathrm{t}-1} = d_{\mathrm{t}} - v_{\mathrm{t}} \cdot \mathrm{FPS}$. By changing the GT: $\mathbf{m}_{\mathrm{t}} \rightarrow (d_{\mathrm{t}-1}, -v_{\mathrm{t}})$ and swap the $\boldsymbol{I}_{\mathrm{t}-1}$ and $\boldsymbol{I}_{t}$, we have scene where the vehicle travels backward compared to original scene.

- *A scene with changed speed*. As introduced, the time interval is expressed by FPS, which helps calculate the velocity (Eq. 2.21). In other words, given a fixed distance difference ($\Delta d = d_{\mathrm{t}} - d_{\mathrm{t}-1}$), a longer unit of time interval will result in smaller velocity. Therefore, by altering the original FPS to FPS' and $\mathbf{m}_{\mathrm{t}} \rightarrow (d_{\mathrm{t}}, v_{\mathrm{t}} \cdot \frac{\mathrm{FPS}'}{\mathrm{FPS}})$, we create a scene with a different GT-speed

Supervised-learning algorithms whose training results depend greatly on the quality and amount of the GT data. The data augmentation as mentioned above can artificially create three alternative scenes for each original scene, which "enlarge" the dataset by three times, and thereby should be theoretically beneficial to the model robustness.

Unfortunately, the Motion-MLP in this work merely serves as an alternative to utilize the GT. Since it is not integrated into the training of depth estimation (Monodepth2, in our work), the Motion-MLP alone has very limited learning potential for further improvement. Therefore, Our primary choice is still the KF-method, because it allows the object motion estimation system to be constructed in a self-supervised way (detector not included).

4.2.1 Additional Features

This subsection introduces some of practical features: the motion visualization, the graphical user interface (GUI), and the preliminary considerations on the (linear) motion forecasting function.

Visualization

As introduced in previous sections, the proposed OME system estimates relative distance and speed for each object on sight. As Fig. 4.5 shows, **the results can be visualized for better understanding in practical use**.

Figure 4.5: Visualized results of the proposed object motion estimation system. The left-top label is the relative motion of velocity with unit $[\mathrm{m/s}]$ and distance with unit $[\mathrm{m}]$, respectively in longitudinal and lateral directions. The red vector at the BBox center is the visualized 2-D relative velocity, whose magnitude and included angle corresponds to the velocity values. The red dot points out the BBox's bottom center, which highlights the coordinates of this identity on the image. The label "ID: 4" at the right-bottom corner is the tracking identity.

Notice that we visualize the velocity through a vector, including its magnitude and included angle, to conveniently express the general information regarding the speed. The label at the left-top corner are the concrete object motion estimates. Moreover, the *color* can express additional information, e. g. whether the the threshold of (safe) distance or velocity has been exceeded.

Graphical User Interface

For better experimentation and using experience, a graphical user interface (GUI) has also been developed, which mainly consists of

- The *video player* panel shown in Fig. 4.6.
- The *function tab* panel that enables control over options, e. g. whether to display BBox or ID number, etc. The snapshots are placed in Appendix (Fig. .2).
- Auxiliary setting interface for, e. g. the loading of the video.

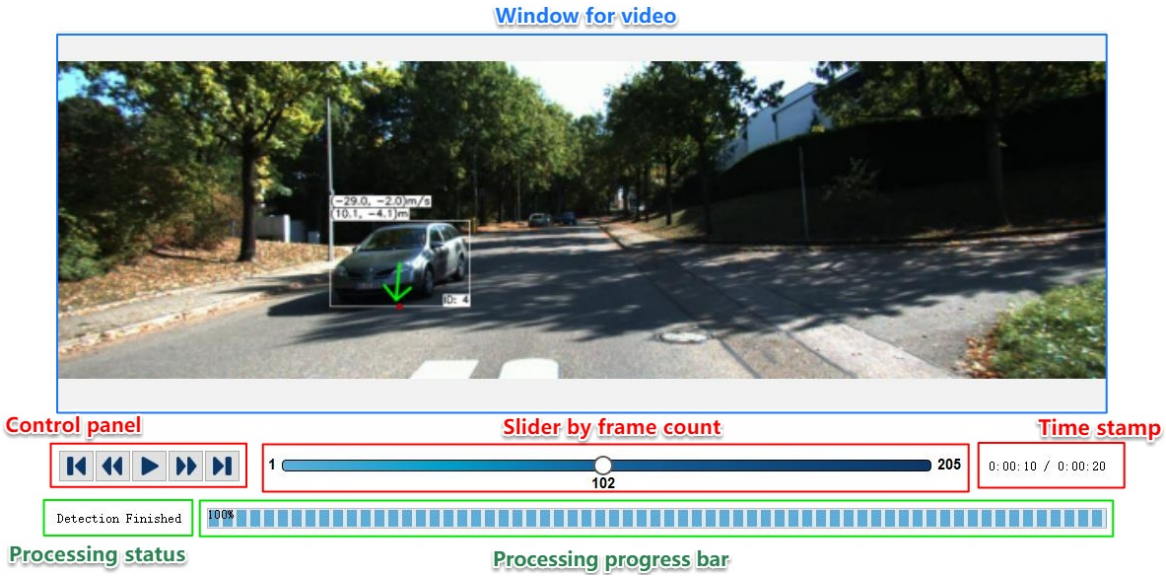


Figure 4.6: The video player panel of the developed GUI. The slider can be dragged back and forth to inspect the intermediate results, while the progress bar indicate the detection progress. The labels for the vehicle is introduced in Fig. 4.5.

Motion Forecasting

Up till this point, the proposed OME system is able to estimate relative distance and speed. Assuming that a vehicle's motion, in the normal driving process, does not abrupt within unit time, there exists the potential of forecasting a near-future motion (position and speed) for it.

As Fig. 4.7 conceptualizes, the forecasting includes two aspect; the motion data, and its change of appearing size on the image (e. g. larger as it gets closer).

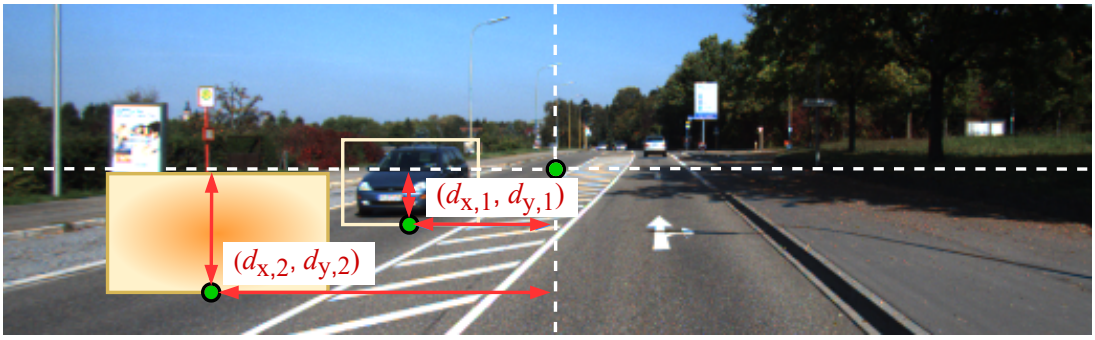


Figure 4.7: Visualized concept of the forecasting motion of a vehicle. The filled BBox in orange color is the forecast BBox. The small green circles on the image center is the principal point, while the circles on the bottom center of BBoxes represents the coordinates/locations of the vehicles in a pinhole model.

Defining the unit time $\mathrm{d}t$, for a time interval T, there are $N_T = T/\mathrm{d}t$ frames. We define a set $\mathbf{V}(t_0)$ that contains the latest-one-second velocity estimates up till t_0, i. e. for $N_{1\mathrm{s}}$ frames in total. If the motion estimates $\boldsymbol{m}(t_0) := (\boldsymbol{d}(t_0), \boldsymbol{v}(t_0))$ at time t_0 is obtained, then the forecast distance $\boldsymbol{d}_T$ at $t_T = t_0 + T$ can be estimates by

$$N_{1\mathrm{s}} := 1/\mathrm{FPS}, \tag{4.14}$$

$$\mathbf{V}(t_0) := \{\boldsymbol{v}(t_0), \boldsymbol{v}(t_{-1}), \cdots, \boldsymbol{v}(t_{-N_{1\mathrm{s}}})\}, \tag{4.15}$$

$$\boldsymbol{d}(t_T) = \boldsymbol{d}(t_0) + N_T \cdot \frac{\sum_i^{N_{1\mathrm{s}}} \mathbf{V}(t_0)}{N_{1\mathrm{s}}}, \tag{4.16}$$

where FPS is the video's frame rate, e. g. FPS=10 for KITTI dataset [Gei+13]). Notice that we use the *average* value of past the latest set of velocity estimates to determine the current velocity, which could theoretically alleviate unstable fluctuation.

Define 2-D projections of vehicle motion as $\boldsymbol{d} := (d_x, d_y)$. As Fig. 4.7 depicts, the BBox's bottom center represents its coordinates in the pinhole model. Then the coordinates for the current and forecast 2-D distance are acquired, namely $(d_{x,1}, d_{y,1})$ and $(d_{x,2}, d_{y,2})$), where the latter is the forecast position by Eq. 4.16.

According to the relationship between the coordinates/position on the image (in px) and in reality (in meter), as introduced in Fig. 4.3, the position of the forecast BBox can be obtained. Similarly, the size of the forecast BBox are the original size multiplied with the factor $S_{\mathrm{BBox}} = |\boldsymbol{d}(t_1)| \,/\, |\boldsymbol{d}(t_0)|$.

Admittedly, the forecasting mechanism mentioned above is still preliminary and can only be employed for vague indication. Nonetheless, this is still the first step to forecast *relative* distance and velocity, which is still an ill-posed subject in the domain of autonomous driving.

5 Evaluation

5.1 Datasets

This section will present KITTI [Gei+13] and CVPR'17 Velocity Challenge (VeloChallenge) dataset. The KITTI dataset is used for training and evaluation of both *tracking* and *depth map estimation*. The VeloChallenge is used for object motion estimation.

5.1.1 KITTI Dataset

The KITTI dataset is the product of the *KITTI Vision Benchmark Suite* project, which provides labelled data for stereo, optical flow, visual odometry, 2-D/3-D object detection and 2-D/3-D tracking. The detailed introduction to equipment and set-up of data acquisition of KITTI project is documented in Appendix .3.

For image data, the raw size is 1242×374 and recorded at 10 Hz. However, the raw depth values collected by rotating laser scanner are unevenly spaced. Therefore they do not reflect actual depth values on the image.

For each task, the project provides recommended metrics and standard evaluation procedure (with code), so that researchers can benchmark their performances. This work evaluates the developed object motion estimation (OME) system in terms of *2-D tracking* and *depth map estimation* on KITTI dataset.

5.1.2 CVPR'17 Velocity Challenge Dataset

In order to evaluate the performance of object-wise motion estimation (OMD), the CVPR'17 Velocity Challenge (VeloChallenge) dataset is chosen. The training data of the dataset is composed of 1074 2-second video clips recorded at 20 Hz (i. e. 40 images per clip), while the testing set has 270.

The annotation is limited in following aspects:

- The target number is limited to four, i. e. the annotation is not complete but selective.
- Only the last frame of each clip is annotated, while the first 39 frames are only provided as contextual information. In other words, there are only 1074 image pairs with valid motion label, which is far less than KITTI dataset (with over 93,000 raw images).
- The annotation includes Bbox, 2-D distance and 2-D velocity.
- Additionally, 5000 single images are provided as supplementary material. However, they are not in temporal order and are only annotated with BBox (i. e. no motion data).

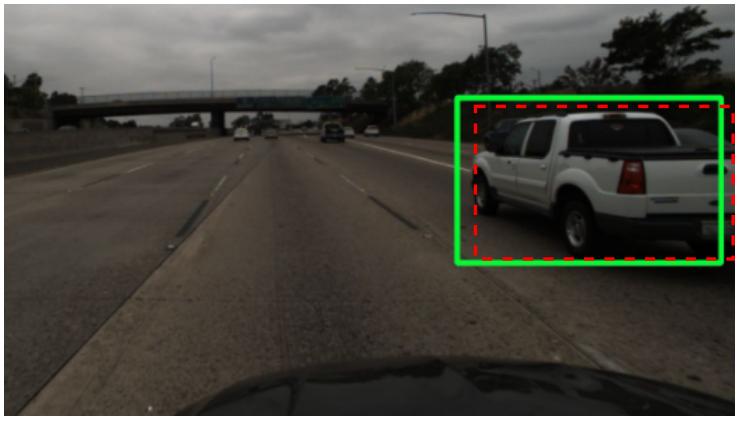
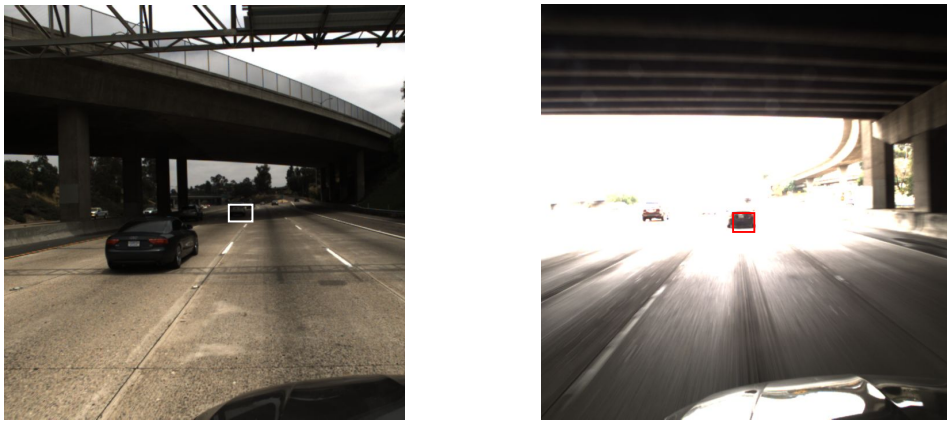


Figure 5.1: Example of an inaccurate BBox annotation in CVPR'17 Velocity Challenge dataset. The visualized GT-label shows that BBox does not precisely enclose the object. The correct BBox is drawn with red dashed lines for comparison.

After inspection of the data, **we observed certain *flaws* of this dataset**, which has been documented in Appendix .3. Unfortunately, to the best of our knowledge, this is the only dataset that fits the task of motion estimation. Therefore, the dataset is used despite of its flaws and we will conduct more qualitative reasoning for a more thorough analysis.

In addition, the *lighting condition* of this dataset is challenging. Although videos are recorded in daytime, just as in KITTI, but the image quality is much worse, e. g. the camera is confronting the sun or the whether is dim. As shown in Fig. 5.2 as examples, the object in a) is hardly recognizable due to the dimness, while the over-exposure in b) makes the surroundings of the object textureless (i. e. with less contextual information), which makes it difficult for the NN to estimate the depth map.

Last but not least, the *image quality* is also not optimal. As Fig. 5.1 shows, the image seems "blurred". Even though it has a resolution of 1280×720, the visibility of details is far worse than images in KITTI with a similar resolution of 1242×374.



(a) (b)

Figure 5.2: Examples of challenging lighting conditions. a) dim condition with a far-away object makes the object nearly fade into shadow. b) too-bright lighting condition caused by over-exposure or confronting the sun while recording.

5.2 Multi-Object Tracking Evaluation

The evaluation on multi-object tracking (MOT) performance includes many aspects. It is generally centered around correctness of identity assignment/association, and accuracy of detection (represented by BBox). In this work, the evaluation adopts HOTA metric [Lui+20] recommended by KITTI project.

5.2.1 HOTA Metric

The HOTA utilizes the concept of the *Jaccard index* to evaluate

- the *detection accuracy* ($\mathrm{ACC}_{\mathrm{det}}$) defined in Eq. 5.1. TP, FP, FN represent true-positive, false-positive and false-negative, respectively. The true-positive case for localization and (next point of detection) is determined by matching between *the set of all predicted*

and the set of all ground-truth BBoxes by IOU. Note that true-negative case is not involved and thereby is omitted.

$$\mathrm{ACC}_{\mathrm{det}} = \frac{\mathrm{TP}_{\mathrm{det}}}{\mathrm{TP}_{\mathrm{det}} + \mathrm{FP}_{\mathrm{det}} + \mathrm{FN}_{\mathrm{det}}} \tag{5.1}$$

- the *localization accuracy* ($\mathrm{ACC}_{\mathrm{loc}}$). As introduced in 3.2, it is identical to the IOU (denoted by BBox-IOU) between predicted and GT areas. It is defined as

$$\mathrm{ACC}_{\mathrm{loc}} = \frac{1}{\mathrm{TP}_{\mathrm{det}}} \sum_{c \in \mathrm{TP}} \mathrm{IOU}_{\mathrm{bbox}}(c). \tag{5.2}$$

  Note that the $\mathrm{ACC}_{\mathrm{loc}}$ shares the same definition of TP, TF and FN cases.

- The *accuracy of identity association* ($\mathrm{ACC}_{\mathrm{ia}}$), expressed in Eq. 5.4, where a single association metric is denoted by $\mathrm{IOU}_{\mathrm{ia}}$.

$$\mathrm{ACC}_{\mathrm{ia}} = \frac{1}{\mathrm{TP}_{\mathrm{ia}}} \sum_{c \in \mathrm{TP}_{\mathrm{det}}} \mathrm{IOU}_{\mathrm{ia}}(c), \tag{5.3}$$

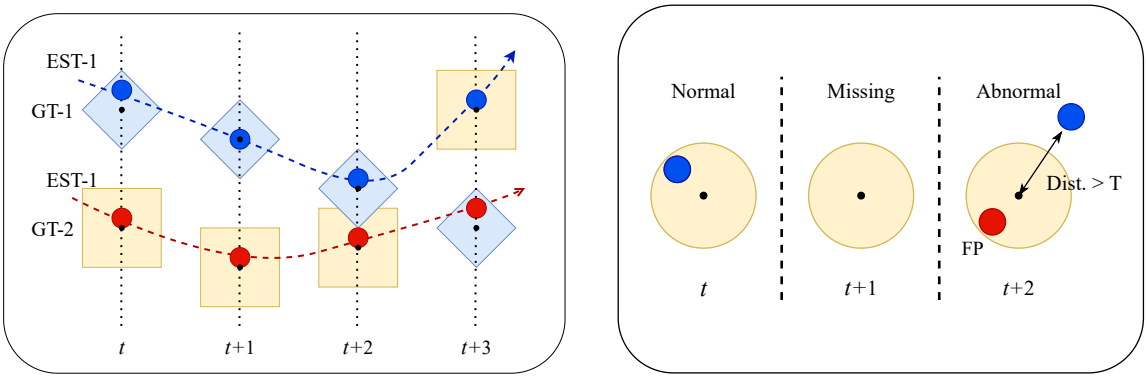
$$= \frac{1}{\mathrm{TP}_{\mathrm{ia}}} \sum_{c \in \mathrm{TP}_{\mathrm{det}}} \frac{\mathrm{TP}_{\mathrm{ia}}}{\mathrm{TP}_{\mathrm{ia}} + \mathrm{FP}_{\mathrm{ia}} + \mathrm{FN}_{\mathrm{ia}}}. \tag{5.4}$$

  The subscript ia is to distinguish the TP case from the similar concept in $\mathrm{ACC}_{\mathrm{det}}$. As shown in Fig. 5.3, GT identity is used for determining the identity.

Combining $\mathrm{ACC}_{\mathrm{det}}$, $\mathrm{ACC}_{\mathrm{loc}}$), $\mathrm{ACC}_{\mathrm{det}}$ together, the HOTA score at IOU threshold $\mathrm{IOU}_{\mathrm{bbox}} = \alpha$ is defined

$$\begin{aligned} \mathrm{HOTA}_{\alpha} &= \sqrt{\mathrm{ACC}_{\alpha,\mathrm{det}} \cdot \mathrm{ACC}_{\alpha,\mathrm{ia}}} \\ &= \sqrt{\frac{\sum_{c \in \mathrm{TP}_{\alpha,\mathrm{det}}} \mathrm{IOU}_{\alpha,\mathrm{ia}}(c)}{|\mathrm{TP}_{\alpha,\mathrm{det}}| + |\mathrm{FN}_{\alpha,\mathrm{det}}| + |\mathrm{FP}_{\alpha,\mathrm{det}}|}}, \end{aligned} \tag{5.5}$$

which leads to the final HOTA score by calculating the integral at different threshold levels



(a) (b)

Figure 5.3: Illustration of data association task for multi-object tracking (MOT) task. (a) is a example case in MOT, where small circles are estimated position and large squares are GT position in temporal order. The colored dashed lines are the trajectories formed by estimated positions. At $t+3$ there is a mismatch, where the identities of EST-1 and EST-2 are switched. (b) illustrates identity assignment. The radius of the big circle indicates the confidence threshold around the GT position. At time $t+1$, there is no valid estimates to be matched, while at $t+2$ there is a false-positive case (red circle) and the estimate of the real identity is out-oft-bound (blue circle).

α, specifically

$$\text{HOTA} = \int_{0<\alpha\leq 1} \text{HOTA}_\alpha \tag{5.6}$$

$$\approx \frac{1}{19} \sum_{\substack{\alpha=0.05 \\ \alpha+=0.05}}^{0.95} \text{HOTA}_\alpha, \tag{5.7}$$

where the second row is the discretized definition used in actual calculation.

In summary, the HOTA metric is the integration of three-fold accuracy in terms of BBox-localization precision, detection correctness, and ID-association correctness. The final score is convenient for benchmarking, while *the three separate sub-metrics can also be analyzed respectively for more details*.

5.2.2 Evaluation on DeepSort

The KITTI MOT tracking benchmark consists of 21 sequences for training and 29 for testing. In order to guarantee a comprehensive evaluation, only *valid* targets participate. The validity is defined in three three aspects: class type and occlusion/truncation condition, and the BBox size of the object.

Only classes *Car* and *Pedestrian*, which contain enough samples, are supported for evaluation. In this work, we only focus on the *Car*-alike classes, therefore classes *Car*, *Bus* and *Truck* in COCO classes are merged together into *Car* class.

The completeness of the visible part of the object is indicated by occlusion and truncation levels. Specifically, there are 0 to 3 ascending levels of occlusion, e. g. 0 for no occlusion and 3 for complete occlusion, while truncation is the percentage of truncated area, e. g. 1.0 truncation means the object is wholly left out of the image. The evaluation applies no limit on occlusion but only take the zero-truncated object into account.

In addition, the object with BBox size less than 25 px does not participate in evaluation.

With above-mentioned filtering conditions applied to GT data, the HOTA score, as well as their sub-metric scores are calculated. Based on mechanisms of DeepSort introduced in Section 3.2, we conducted a series experiments on the hyper-parameters as well small modifications mentioned in Section 4.2. Quantitative results will be analysed in the next chapter.

5.3 Depth Map Evaluation

The evaluation of depth map estimation (DME) is basically comparing pixel-wise difference between GT depth values provided by Velodyne LiDAR and the estimated results by Monodepth2 [GAB19].

The GT depth map, however, only covers a smaller region compared to the full image range. As shown in Fig. 5.4, the cropping will also be applied to the estimated depth map to enable direct comparison and evaluation. Following the solution provided in EIGEN's work (2014) [EPF14], we mask out the region that has not been covered by GT during evaluation.

The metrics involved are relative absolute error (RAE), relative squared error (RSE), root-

mean-square error (RMSE), log of RMSE (RMSLE), accuracy metric δ with a specified threshold T values (δ_T). Define depth value at coordinate (i, j) on depth map as $\boldsymbol{D}(i, j)$, and the metrics mentioned above can be mathematically defined as

$$\text{RAE} = \frac{1}{N \times M} \sum_{i=N}^{N} \sum_{j=M}^{M} \frac{|\hat{\boldsymbol{D}}(i,j) - \boldsymbol{D}(i,j)|}{\boldsymbol{D}(i,j)}, \tag{5.8}$$

$$\text{RSE} = \frac{1}{N \times M} \sum_{i=N}^{N} \sum_{j=M}^{M} \frac{\left(\hat{\boldsymbol{D}}(i,j) - \boldsymbol{D}(i,j)\right)^2}{\boldsymbol{D}(i,j)}, \tag{5.9}$$

$$\text{RMSE} = \sqrt{\frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(\hat{\boldsymbol{D}}(i,j) - \boldsymbol{D}(i,j)\right)^2}, \tag{5.10}$$

$$\text{RMSLE} = \sqrt{\frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(\log(\hat{\boldsymbol{D}}(i,j) + 1) - \log\left(\boldsymbol{D}(i,j) + 1\right)\right)^2}, \tag{5.11}$$

$$\delta_T = \frac{1}{N \times M} \sum_{i}^{N} \sum_{j}^{M} \left[\max \left[\frac{\hat{\boldsymbol{D}}(i,j)}{\boldsymbol{D}(i,j)}, \frac{\boldsymbol{D}(i,j)}{\hat{\boldsymbol{D}}(i,j)} \right] < T \right]_{\text{ivs}}, \tag{5.12}$$

where $\hat{\boldsymbol{D}}$ denotes depth map estimates and $\boldsymbol{D}$ is the GT. Additionally for Eq. 5.12, T is the threshold value, and $[P]_{\text{ivs}}$ is the Iverson bracket that returns 1 when the condition P is True, otherwise 0.

5.4 Object Velocity Evaluation

The evaluation of velocity estimation of individual objects is conducted on the CVPR'17 Velocity Challenge dataset. As introduced in Section5.1, the GT only records the velocity of the *last frame* of each video.

Unlike previous evaluation, where we only focus on one sub-task (e. g. object detection), the individual velocity is evaluated on the whole pipeline, including object detection (OD). tracking, depth map estimation (DME) and object motion derivation (OMD).

As described in 4.1, we can choose either Kalman Filter (KF) or a multi-layer perception (MLP) to derive the motion data from the depth-map patch. The two methods are tested and

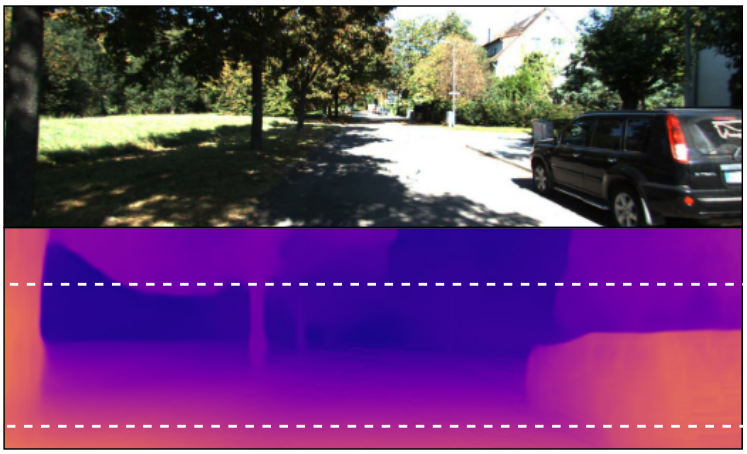


Figure 5.4: The illustration of raw image (upper), the estimated depth map (lower), and the actual region covered by GT depth (region highlighted by white dashed lines).

compared with other algorithms that have participated in this challenge.

In addition, we do not evaluate the individual distance, because there is no available benchmark for this task until this point . In our main method (KF), individual velocity is derived from the raw estimates of the depth-map patch. Therefore, we argue that the evaluation of DME should suffice as a substitute.

Evaluation results will be discussed in Section 6.3.

6 Results

6.1 Multi-Object Tracking Evaluation Results

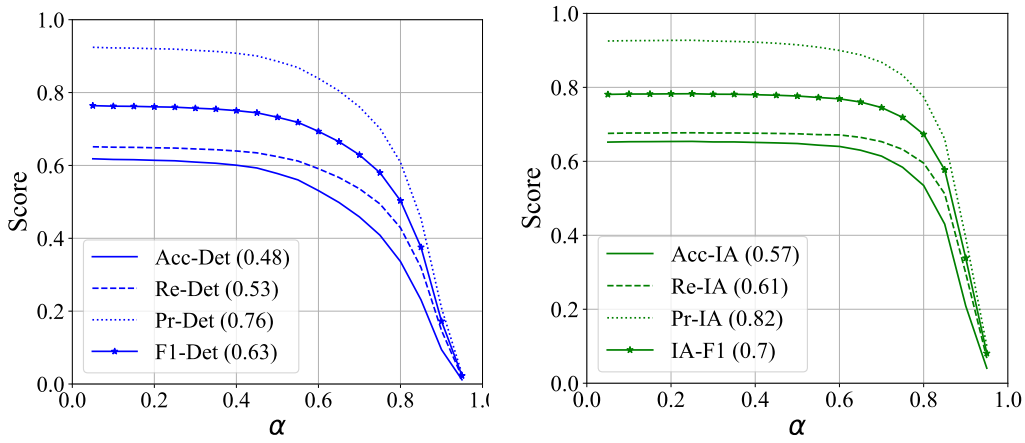
As introduced in Section 5.2.1, the evaluation on multi-object tracking (MOT) task is conducted on KITTI dataset [Gei+13] with HOTA [Lui+20] metric. The HOTA score alone is merely used for benchmarking and ranking, i.e. the analysis will be based on its sub-metrics regarding *detection*, *BBox-location*, and *identity-association* (IA).

Table 6.1: Comparison between DeepSort and other methods taken from KITTI ranking [Gei+13], in terms of HOTA and accuracy metrics for detection, identity association and BBox-localization. Underlined scores are the ones that are closest to our results (lower and higher respectively). The last column is the total number of ID switches.

Rank	Name	Year	HOTA	Acc-Det	Acc-IA	Acc-Loc	IDSW
78	ODAMOT [GV15]	15	0.37	0.47	0.3	0.79	1110
72	SCEA [Yoo+16]	16	0.43	0.45	0.42	0.82	371
65	mbodSSP [LGU15]	15	0.51	0.59	0.45	0.81	770
50	extraCK [GA18]	18	0.60	0.65	0.55	0.84	520
39	SRK [MBP20]	20	0.64	0.75	0.56	0.86	491
31	IMMDP [XAS15]	15	0.69	0.68	0.70	0.85	211
14	CenterTrack [ZKK20]	20	0.73	0.76	0.71	0.87	254
2	PermaTrack [Tok+21]	21	0.78	0.78	0.78	0.87	258
61	ours	17	0.52	0.48	0.57	0.84	384

There are in total 83 participants in KITTI [Gei+13] MOT tracking task, 24 of which is online and based on pure monocular vision, while others leverage stereo or laser points as additional cues. Therefore, as Table 6.1 shows, we display part of the rankings evenly taken in an ascending manner. For each metric, we underline two algorithms with closes results to us (with lower and higher scores respectively).

We scored worst on accuracy of detection (Acc-Det). Based on further inspection on the



(a) Detection (b) Identity Association

Figure 6.1: The F1-score, precision, recall and accuracy for (a) detection and (b) ID-association tasks. The x-axis label α is the IOU-threshold to determine whether a predicted BBox counts as a true-positive case. The mean score of a metric is marked at legend entry, e. g. 0.57 for IA-accuracy. It is noticeable that *precision* is better than *recall* in both tasks.

recall and *precision* in Fig. 6.1(a), it is due to bad recall, which indicates our system is subject to false negative (FN) cases, i.e. some objects have not been successfully detected. This results from the detector (YOLOv4 [BWL20] in this work), because detecting small/far objects is more difficult for one-stage detectors. However, the good score BBox-location (Acc-Loc) indicates that YOLOv4 gives precise BBox prediction.

Regarding the Acc-IA, we outperform SRK [MBP20] algorithm which ranks higher. The credits should go to the design of the two-step matching process of DeepSort (described in Section 3.2). As Acc-IA is decomposed into recall and precision as Fig. 6.1(b), recall metric is also worse, which is similar to the Acc-Det. This is not coincidence, because the GT-objects missed by detector will also lead to direct failure of matching procedure.

In addition, the total number of *identity switch* (IDSW) measures how many times the tracker confuse one identity with another. This mistake often takes place when two objects come across each other, e. g. at a crossroad. Our method outperforms almost the half of the participants. This is surprising, because the IA mechanism in DeepSort, unlike most of the others, is not data-driven (feature extractor excluded) but a straightforward two-step matching process.

More discussion over the potential of improvement, cross-comparison and reasoning is continued in Section 6.4.

6.2 Depth Map Evaluation Results

As introduced in 5.3, the evaluation for depth map estimation (DME) is pixel-wise comparison. As Table 6.2 shows, we group results according to the *Type* of training data, i.e. monocular image (M), or stereo pair (S)/additional depth supervision (D). Since the code is our own re-implementation of official code [GAB19], we add the official results (Monodepth2*) into the table.

Table 6.2: Performance of depth map estimation algorithms. The *Type* column denotes the training data: "M" for monocular vision, "S" for stereo vision, and "D" for depth supervision. The underlined values are the closest ones to our results (lower and higher respectively), while the bold ones are the best in the corresponding group. Except that higher $\delta_{1.25}$ (see Eq. 5.12) stands for better performance, others are the opposite.

Name	Type	RAE	RSE	RMSE	RMSLE	$\delta_{1.25}$ (↑)
SfMLearner [Zho+17]	M	0.183	1.595	6.709	0.270	0.734
GeoNet [YS18]	M	0.149	1.060	5.567	0.226	0.796
DDVO [Wan+17]	M	0.151	1.257	5.583	0.228	0.81
Struct2Depth[Cas+19]	M	0.141	1.026	5.291	0.215	0.816
Monodepth2* [GAB19]	M	**0.115**	**0.903**	**4.863**	**0.193**	**0.877**
Eigen [EPF14]	D	0.203	1.548	6.307	0.282	0.702
Garg [Gar+16]	S	0.152	1.226	5.849	0.246	0.784
DVSO [Yan+18]	D, S	0.097	0.734	4.442	0.187	0.888
DORN [Fu+18]	D	**0.072**	**0.307**	**2.727**	**0.12**	**0.932**
ours	M	0.150	1.112	5.5361	0.224	0.797

In the monocular-vision (M) group, the official result outperforms all others. Unfortunately, our re-implemented version has not achieved such precision. The Struct2Depth [Cas+19] is in general the second best, but it comes to our notice that it utilizes additional segmentation masks and also models individual moving objects. The DDVO[Wan+17] and GeoNet [YS18] have similar results to use (e. g. only 0.01 difference regarding RAE). SfMLearner [Zho+17] serves as the baseline algorithm.

Moreover, other DME algorithms that has been trained on depth supervision or stereo pairs

(but inference on monocular images) are also presented for cross-comparison. The DORN [Fu+18] is outstanding in both groups that have outperformed Monodepth2 by a clear margin (e. g. 66.0% in terms of RSE), and exceeds the DVSO [Yan+18], which uses both stereo pairs and depth supervision.

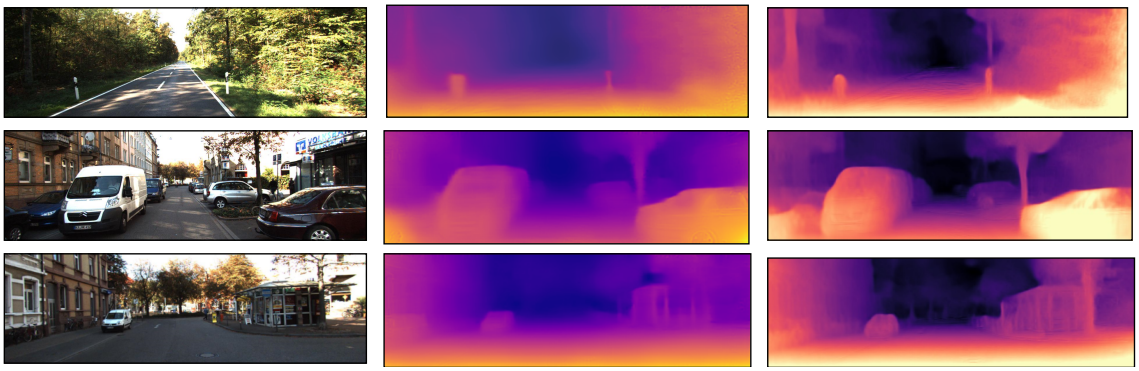


Figure 6.2: Qualitative comparison between our results and DDVO. The first column is the original images from KITTI [Gei+13], the second column is the results from DDVO. It is clear our disparity map demonstrates more details and the edge between background and objects are more sharp.

Although we didn't reach the performance of officially reported Monodepth2, our training results are still satisfying. Qualitative comparison with DDVO [Wan+17] demonstrates that our disparity highlights more details, and has sharper edges between background and objects. Furthermore, Fig. 6.3 has shown more sharpness of the disparity map produced by the official Monodpeth2 [GAB19].

6.3 Object Velocity Evaluation Results

As described in Section 5.4, evaluation has been conducted with both KF-based and MLP-based methods. The results are shown in Table 6.3. We use our KF-method as the baseline, since it is a direct derivation without any training, i.e. no utilization of any GT data. The MLP-based method is an experimental product dedicated to utilizing the GT data to increase competitiveness.

Due to limited information, we only find the data source for the Rank-1 (by the time of submission) and the later work by SONG ET AL. [Son+20]. Our MLP-method has similar

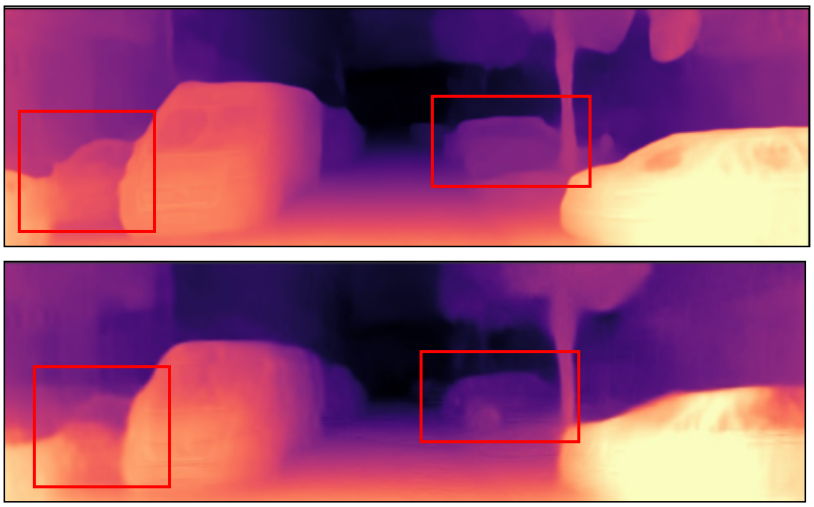


Figure 6.3: Qualitative comparison between ours (lower) and official Monodepth2 [GAB19] (upper). The latter demonstrates more sharp details regarding the edges.

performance to Rank-2 team but with a slight advantage in terms of far-scenario (distance greater than 80 m).

Table 6.3: Evaluation results on object-wise velocity estimation. The metric is mean-square error (MSE), but is divided into four groups (columns): near, middle, far and average, according to the actual distance. The first two rows are our Kalman-Filter (KF) method as the baseline. The following rows are the participants' performances ranked in ascending sequence, and the last row is our MLP-based method. Bold numbers are the best in the corresponding entry, while underlined numbers are the ones closest to our MLP-based method.

Methods	MSE (mean)	MSE (< 20 m)	MSE (< 45 m)	MSE (< 90 m)
ours, KF	3.91	1.59	4.73	5.42
Rand-3	2.90	0.55	2.21	5.94
Rank-2	1.50	0.25	0.75	3.50
Rank-1 [KMF18]	1.30	0.18	0.66	3.07
Song et al. [Son+20]	**0.86**	**0.15**	**0.34**	**2.09**
ours, MLP	1.56	0.68	1.66	2.36

Nonetheless, it has come to our attention that the best two algorithms [Son+20], [KMF18] **both use pre-computed detection and tracking results and only evaluate on motion estimation**, which corresponds to DME and OMD in our pipeline. For example, before performing motion estimation, [KMF18] firstly used GT-BBoxes and off-the-shelf SOT

trackers to manually generate well-tracked BBox. **We, on the contrary, run the whole pipeline which directly derives the velocity (and distance) from raw images**.

Despite the fact that KF-method is outperformed quantitatively, we argue it has more practicality and therefore it is still reckoned as our main method. More reasoning is carried on in Section 6.4.

6.4 Discussion

In this section, a thorough discussion over quantitative evaluation results along with practical sense will be conducted. Notice that we do not

The multi-object tracking evaluation has revealed that the weakest point of DeepSort lies in *detection recall*. This is due to the "missing" case, i. e. our YOLOv4 detector have skipped many valid objects. This mainly results from the inherent drawback for one-stage detectors that they tend to overlook the small objects, namely the far-away vehicles on the image. Although KITTI [Gei+13] tracking evaluation has limited the minimum size to 25 px, it is still too small for YOLOv4 to guarantee a robust detection at that scale.

Furthermore, such "missing" also influences the performance the identity association (IA) in terms of *IA recall*. Specifically, when the vehicle size is within a *range of instability*, approximately 25 to 35 px, the occasional missing makes DeepSort unable to decide the correct ID, or even repeatedly assigning new ID to it.

Besides detection input, the feature extractor (FE) is also vital to IA. As introduced in Section 3.2 the cascade matching heavily depends on the performance of FE. In the original proposal, the author utilizes GT-data to extract identity-wise trajectory across time and conduct supervised learning [WB18], which helps FE learn visual representation from different perspectives of actual observation.

However, such well-labelled data is not always available in real-world applications. Therefore, we have used the unsupervised-learning method (SimCLR [Che+20b]) to directly train an encoder that learns feature embeddings. Since we only use data augmentation, e. g. random cropping, changes in lighting conditions, etc., it still might not suffice to simulate a realistic new perspective. Also, the second matching process based on D-IOU is limited: Once there is occlusion that abruptly changes BBox shapes, the matching will be more prone to failing.

Apart from the data augmentation, the feature type could also be enriched, e. g.

- the optical-flow embeddings, which helps IMMDP [XAS15] achieve outstanding results, especially as an algorithm proposed back in 2015.
- the RGB color histograms used by extraCK [GA18] is also an alternative.

There exists other possibilities of course, e. g. PermaTrack [Tok+21] employs DL-based method with a spatial-temporal recurrent unit to utilize the full history of estimates.

Apart from the shortcomings, DeepSort is still a practical framework for MOT. To begin with, its two-step matching process is based on simple calculation (the lightweight FE can be ignored), which makes computation cost considerably low, reaching over 300 FPS. Also, the integration of detector and FE is flexible, i.e. we could always replace them with better algorithms in the future, which promises the potential of continuous improvement.

The evaluation on **depth map estimation** has confirmed that our re-implemented Monodpeth2 achieves satisfying results. Even though the quantitative results are not superior to other competitors based on monocular vision, but the margin is not significant: only approximately 7% to 9% lower than Struct2Depth [Cas+19] and official Monodepth2 [WB18]. And qualitative comparison in Fig. 6.2 has demonstrated its advantage over the clarity on objects' edges.

It should be mentioned that Struct2Depth [Cas+19], whose results are slightly better than ours, utilizes additional GT segmentation masks to: 1) mask out moving objects; 2) add additional object-distance loss based on pre-knowledge of real-world size. These auxiliary constraints might help the model learn more precise depth map. Although similar ideas have been experimented with as introduced in Section 4.2, we did not employ GT segmentation mask but YOLOv4-predicted BBoxes as rough masks to keep the work in *unsupervised* style. Unfortunately, these experiments did not return good results, and therefore we have to forfeit them for now.

The object motion estimation evaluation reveals our weakness regarding precision. However, it is argued that the quantitative results do not suffice to undermine the effectiveness of our object motion estimation (OME) system. In other words, there are flaws centered around the CVPR'17 Velocity Challenge dataset as well as its evaluation settings.

First of all, realistic motion estimation should be a continuous process. This evaluation of dataset, however, is only conducted on one single frame per clip. This setting overlooks the

significance of **stability** in practical use.

As we apply our MLP-based OME system to a video clip, the velocity estimates are highly incoherent. Although we have not experimented on other competitors' algorithms ([Son+20] and [KMF18]), this problem is very probable, because either of them takes estimation history into account.

This is exactly the reason we still reckon KF-based method as the primary choice, despite that it scores lowest in this evaluation. Because KF, as introduced in Section 2.4.3, models the motion trajectory of an object and is updated with time. Therefore, the final motion data is estimated based on both new and historical estimates. This also leads to a steady velocity estimation across time, which fits the fact that the vehicle's speed will not abruptly change between each frame (i. e. within an interval shorter than 0.1 s).

Nonetheless, there is still much potential for improvements, for example: Frame-wise update for KF might not be enough; long-time occlusion cannot be well handled, especially if it is too near to the camera. Since OME reflects the combined performance of OD, MOT, DME and OMD, any improvement on the above-mentioned components should lead to rise of overall performance.

7 Conclusion

In this work, a pipeline is established to estimate relative velocity and distance estimation for each object based on single images recorded from a moving camera. Although algorithms dedicated for *object detection* (OD), *tracking*, *depth map estimation* (DME) have been well-developed in their own domains, there has not been a well-developed framework flexible enough to integrate them all together to estimate individual object motion.

To begin with, we investigated representative algorithms in the domains mentioned above in Chapter 3. In addition, two methods, dependent of the availability of GT, are proposed for the final procedure of *object motion derivation* (OMD).

In Chapter 4, the implementation details of the overall object motion estimation (OME) pipeline is elaborated, which integrates separate algorithms together. And in Section 4.2 we presented our main contributions, including important experiments and modifications.

In Chapter 5, evaluations are carried out firstly on respective algorithms' performance regarding their own task to find our their own pros and cons. And finally the object velocity estimation is evaluated on CVPR'17 Velocity Challenge dataset for the performance of the OME pipeline.

The evaluation results are presented in Chapter 6. It reveals that the main weak spots lie in

- detector's sensitiveness to the smaller objects;
- robustness of feature-vector matching mechanism for in tracker;
- more integration between each components for globally finest estimates.

As expected, it is found that our system can produce stable motion estimates for vehicles up to the middle range (with a size of approximately greater than 40 px on the image). And the inference speed reaches 10 Hz on a 7th-Gen Intel® Core™ i7 processor and Nvidia GTX 1060 Max-Q GPU. Last but not least, **our OME pipeline (except for the detector) can be trained/tuned in an unsupervised manner**.

7.1 Future Work

In this section, practical considerations for possible improvements are discussed.

First of all, there exists the alternative that four components (OD, tracking, DME and OMD) can be integrated into two or three. For example, detection and tracking might be able to merged into one, e. g. the FairMOT [Zha+20], where a globally optimal estimates for tracked BBOx can be directly obtained. Similarly, the DME and OME could also be merged, like in SONG ET AL. [Son+20].

And if we stick to the current pipeline, the detector could be replaced with one that produces instance segmentation, e. g. [Bol+20] instead of mere BBox. Since the segmentation mask will enclose the actual profile of the object, it could not only help feature extractor of tracker produce a better visual representation, but also help the MLP-based OMD methods to focus on the object area.

Furthermore, the KF could be utilized in more flexible way. In this work, we follow the most typical implementation where KF is updated per frame. However, adjustments on the update interval might provide more temporal hints from the previous trajectory, especially when vehicle-related objects' motion usually does not fluctuate dramatically.

Appendix

.1 Additional Details on Algorithms

The sigmoid function is a mathematical function with numerous variants that produces "S" curve. The variant used in this work is the *logistic function*:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{.1}$$

The cross-stage partial network (CSPNet) reconstructs the flow of the data as the Fig. .1: Instead of letting the data flow directly through the dense layers (Conv-layer, BN, etc.), the base input layer is now split into two parts, where one part is for normal dense-layer processing while the other is directly copied and concatenated with dense-layer output to form the output of one block. This iterates for a designated number of dense blocks. The output of the all dense blocks will be concatenated again with the first half of base input layer, and go through the final transition layer to produce the final output.

The CSPNet can help normal backbones (e. g., ResNet50) reduce a great amount of computation by about 20% and thereby increase the inference speed. In addition, the it helps the original network achieve better performance, because the cross-connection manage to fuse information at different levels, which enhance the learning ability of the CNN.

Table .1 displays the network structure of decoders of depth net and pose net of Monodepth2 [GAB19] (introduced in Section 3.3).

The structural similarity index measure (SSIM) between two images A and B is defined as

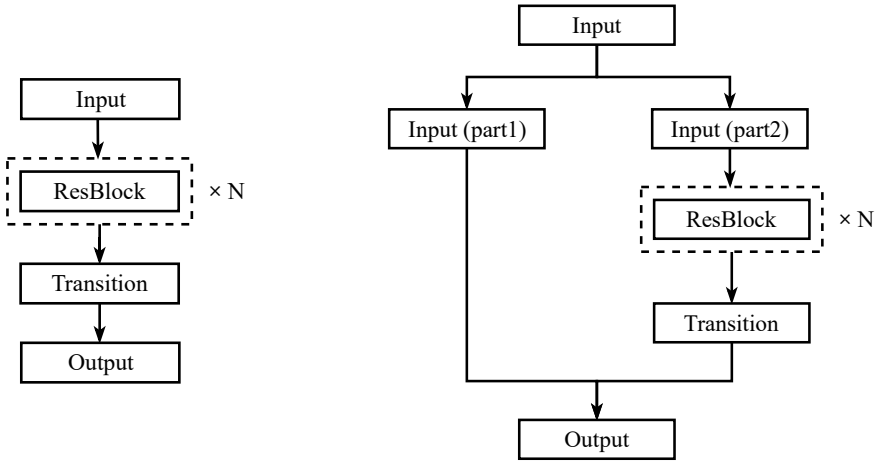


Figure .1: Comparison between the architecture of a normal ResNet (a) and the one modified according to CSP (b). In a normal ResBlock the input is fully fed to the one or more ResBlock(s), and passed through a *transition* layer which down-sample the feature maps. For a CSP-modified ResNet, the Input is split into two parts, and only one part will go through the operations in (a), while the other part is directly concatenated to the output.

follows:

$$L(A,B) = \frac{2u_{\mathrm{A}}u_{\mathrm{B}} + C_1}{u_{\mathrm{A}}^2 + u_{\mathrm{B}}^2 + C_1}, \tag{.2}$$

$$C(A,B) = \frac{2\sigma_{\mathrm{A}}\sigma_{\mathrm{B}} + C_2}{\sigma_{\mathrm{A}}^2 + \sigma_{\mathrm{B}}^2 + C_2}, \tag{.3}$$

$$S(A,B) = \frac{\sigma_{\mathrm{AB}} + C_3}{\sigma_{\mathrm{A}}\sigma_{\mathrm{B}} + C_3}, \tag{.4}$$

$$\mathbf{SSIM}(A,B) = L(A,B) \cdot C(A,B) \cdot S(A,B), \tag{.5}$$

where u and σ stands mean and variance; σ_{AB} is the covariance of two images; and the Cs are constants to avoid zero value of denominator. According to the author's experiments, there is simplification $C_3 = C_2/2$, which leads the Eq. .5 to the simplified form:

$$\mathbf{SSIM}(A,B) = \frac{(2\mu_A u_{\mathrm{B}} + c_1)\,(2\sigma_{\mathrm{AB}} + C_2)}{(\mu_{\mathrm{A}}^2 + \mu_{\mathrm{B}}^2 + C_1)\,(\sigma_{\mathrm{A}}^2 + \sigma_{\mathrm{B}}^2 + C_2)}, \tag{.6}$$

where empirically $C_1 = 0.01, C_2 = 0.03$.

Table .1: The overview of decoders of Depth Net and Pose Net, respectively. The parameters *k* (kernel), *s* (stride) and *chs* (channel numebr) defines the kernel shape of convolution layer. e-convN is the skip connection from the encoder at scale N (scale 0 is the encoder output). The upconv is the intermediate output and ↑ is the up-sample operation, while ↓ scale is the down-scaled resolution relative to the original resolution. The disp-convN is the output layer to estimate the disparity map at scale N.

Depth Decoder						
layer	k	s	chs	↓ res	input	activation
upconv5	3	1	256	32	e-conv5	ELU [CUH16]
d-conv5	3	1	256	16	↑ upconv5, e-conv4	ELU
upconv4	3	1	128	16	d-conv5	ELU
d-conv4	3	1	128	8	↑ upconv4, e-conv3	ELU
disp-conv4	3	1	1	1	d-conv4	Sigmoid
upconv3	3	1	64	8	d-conv4	ELU
d-conv3	3	1	64	4	↑ upconv3, e-conv2	ELU
disp-conv3	3	1	1	1	d-conv3	Sigmoid
upconv2	3	1	32	4	d-conv3	ELU
d-conv2	3	1	32	2	↑ upconv2, e-conv1	ELU
disp-conv2	3	1	1	1	d-conv2	Sigmoid
upconv1	3	1	16	2	d-conv2	ELU
d-conv1	3	1	16	1	↑ upconv1	ELU
disp-conv1	3	1	1	1	d-conv1	Sigmoid
Pose Decoder						
layer	k	s	chs	↓ res	input	activation
p-conv0	1	1	256	32	e-conv5	ReLU [NH10]
p-conv1	3	1	256	32	p-conv0	ReLU
p-conv2	3	1	256	32	p-conv 1	ReLU
p-conv3	1	1	6	32	p-conv3	−

.2 Additional Implementation Details

Specifically for a detected BBox, $\boldsymbol{B}^{\mathrm{od}} = [b_{\mathrm{l}}^{\mathrm{od}}, b_{\mathrm{t}}^{\mathrm{od}}, b_{\mathrm{r}}^{\mathrm{od}}, b_{\mathrm{b}}^{\mathrm{od}}]$, defined in $ltrb$ convention, the the corresponding $\boldsymbol{B}^{\mathrm{dme}} = \boldsymbol{B}^{\mathrm{od}}$ is firstly initialized, then the adjustment occurs in vertical

direction (i. e. b_t and b_t) in following order:

$$s := W^{dme}/640, \tag{.7}$$
$$b_\mathrm{t}^\mathrm{dme} := \max\left[0, b_\mathrm{t}^\mathrm{od} - \mathrm{offset}\right]/s, \tag{.8}$$
$$b_\mathrm{b}^\mathrm{dme} := \min\left[H^\mathrm{dme} + \mathrm{offset}, l - \mathrm{offset}\right]/s, \tag{.9}$$
$$\boldsymbol{B}^\mathrm{dme} := [b_\mathrm{l}^\mathrm{od}, b_\mathrm{t}^\mathrm{dme}, b_\mathrm{r}^\mathrm{od}, b_\mathrm{b}^\mathrm{dme}] \tag{.10}$$

where s is the scaling factor to resize the center-cropped image $\boldsymbol{I}^\mathrm{dme}$. Since the center cropping only takes place before the scaling, the same scaling factor can be applied to height and width.

.3 Additional Evaluation Details

As the Fig. .3 illustrates, the data is collected through the wagon equipped with sensors. Specifically, the cameras provide image data, while ground truth of depth and global position is provided by a Velodyne laser scanner and a GPS, respetively.

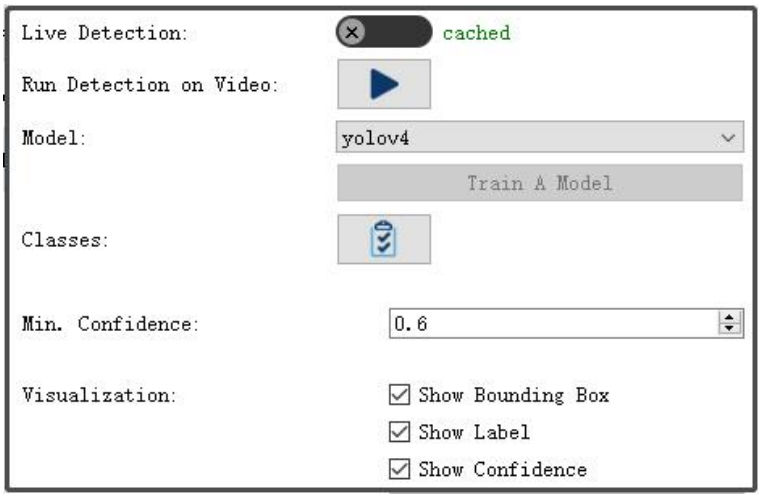
KITTI's camera intrinsic parameters are given in form of matrix

$$\boldsymbol{K}_\mathrm{kitti} = \begin{bmatrix} f_\mathrm{x} = 720.0 & 0 & \mathrm{pp}_\mathrm{x} = 641.0 & 0 \\ 0 & f_\mathrm{y} = 720.0 & \mathrm{pp}_\mathrm{y} = 192.0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathrm{px} \tag{.11}$$

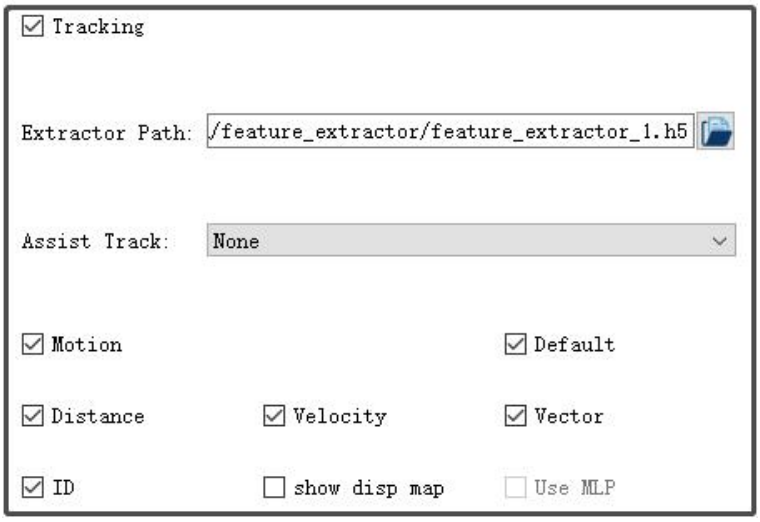
The camera intrinsic matrix of VeloChallenge dataset is

$$\boldsymbol{K}_\mathrm{velo} = \begin{bmatrix} f_\mathrm{x} = 714.2 & 0 & \mathrm{pp}_\mathrm{x} = 713.9 & 0 \\ 0 & f_\mathrm{y} = 710.4 & \mathrm{pp}_\mathrm{y} = 376.3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathrm{px} \tag{.12}$$

- The provided principal point is inaccurate, which has been confirmed by KAMPELMÜHLER's work [KMF18]. Therefore, we adopt modification on the principal point position suggested in his work. Specifically, pp_x is corrected to from 675.6 px to 713.9 px (px is the unit of pixel grid).



(a)



(b)

Figure .2: The function panel of our GUI for (a) *detection*, and (b) *tracking, depth estimation* and *object motion derivation.* The function panel of detection controls the main estimation system, while the other panel contains more detailed visualization options.

- The BBox, as shown in Fig. 5.1 annotation is not accurate enough.
- There exist many repeated or highly-similar video clips, which further undermines the richness of the data.

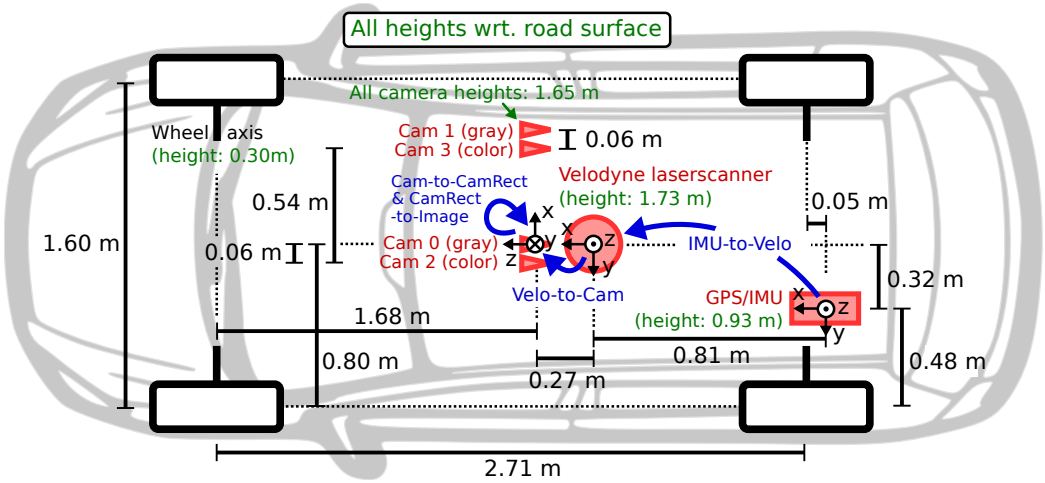


Figure .3: Equipment setup for data acquisition of KITTI dataset. There are two stereo camera sets for grayscale and RGB images, respectively. A GPS is used to collect position data in world coordinate system, while a IMU is to measure the egomotion of the vehicle. This figure is from the original work [Gei+13].

As the Fig. .4 shows, the HOTA metric scores 0.52, while the accuracy scores of detection. location and IA are 0.45, 0.57 and 0.84 respectively.

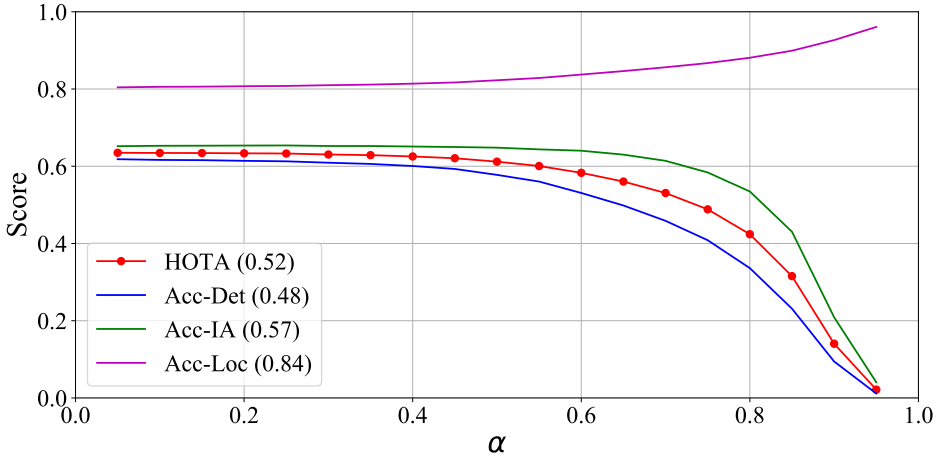


Figure .4: The scores of HOTA of sub-metrics (i. e. accuracy of detection, location and IA). The x-axis label α (Eq. 5.5) is the IOU-threshold to determine whether a predicted BBox is counted as a true-positive (TP) case. As α increases, the location accuracy (Acc-Loc) grows, because location is also measured by BBox-IOU. However, for other metrics, a high threshold for BBox means low tolerance for potentially matched BBox. Therefore, more TP cases are reckoned as FN cases as α increases.

Bibliography

[Bew+16] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. “Simple online and realtime tracking”. In: *2016 IEEE International Conference on Image Processing (ICIP)* (Sept. 2016). DOI: 10.1109/icip.2016.7533003.

[BLK01] Y. Bar-Shalom, Xiaorong Li, and T. Kirubarajan. “Estimation with Applications to Tracking and Navigation: Theory, Algorithms and Software”. In: 2001.

[Bol+10] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, and Yui Man Lui. “Visual object tracking using adaptive correlation filters”. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2544–2550. DOI: 10.1109/CVPR.2010.5539960.

[Bol+20] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. “YOLACT++: Better Real-time Instance Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1. ISSN: 1939-3539. DOI: 10.1109/tpami.2020.3014297.

[Bra00] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[BWL20] Alexey Bochkovskiy, Chien-Yao Wang, and H. Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *ArXiv* abs/2004.10934 (2020).

[Can86] J Canny. “A Computational Approach to Edge Detection”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.6 (June 1986). ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851.

[Cas+19] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. “Depth Prediction without the Sensors: Leveraging Structure for Unsupervised Learning from Monocular Videos”. In: *Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*. 2019.

[Che+20a] Pengguang Chen, Shu Liu, Hengshuang Zhao, and Jiaya Jia. “GridMask Data Augmentation”. In: *ArXiv* abs/2001.04086 (2020).

[Che+20b] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. arXiv: 2002.05709 [cs.LG].

[CUH16] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2016. arXiv: 1511.07289 [cs.LG].

[Dod08] Yadolah Dodge. "Chi-square Distribution". In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 70–72. ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_54.

[Dua+19] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. "CenterNet: Keypoint Triplets for Object Detection". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 6568–6577. DOI: 10.1109/ICCV.2019.00667.

[EPF14] David Eigen, Christian Puhrsch, and Rob Fergus. *Depth Map Prediction from a Single Image using a Multi-Scale Deep Network*. 2014. arXiv: 1406.2283 [cs.CV].

[Fu+18] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. *Deep Ordinal Regression Network for Monocular Depth Estimation*. 2018. arXiv: 1806.02446 [cs.CV].

[GA18] Gültekin Gündüz and Tankut Acarman. "A lightweight online multiple object vehicle tracking method". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2018, pp. 427–432.

[GAB17] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. *Unsupervised Monocular Depth Estimation with Left-Right Consistency*. 2017. arXiv: 1609.03677 [cs.CV].

[GAB19] C. Godard, Oisin Mac Aodha, and G. Brostow. "Digging Into Self-Supervised Monocular Depth Estimation". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 3827–3837.

[Gar+16] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. *Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue*. 2016. arXiv: 1603.04992 [cs.CV].

[Gei+13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* (2013).

[Gir+14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[Gir15] Ross Girshick. "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[Gor+19] Ariel Gordon, Hanhan Li, Rico Jonschkowski, and Anelia Angelova. "Depth From Videos in the Wild: Unsupervised Monocular Depth Learning From Unknown Cameras". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 8976–8985. DOI: 10.1109/ICCV.2019.00907.

[GV15] Adrien Gaidon and Eleonora Vig. "Online domain adaptation for multi-object tracking". In: *arXiv preprint arXiv:1508.00776* (2015).

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[He+18] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. *Mask R-CNN*. 2018. arXiv: 1703.06870.

[Hen+15] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. "High-Speed Tracking with Kernelized Correlation Filters". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.3 (2015), pp. 583–596. DOI: 10.1109/TPAMI.2014.2345390.

[HS17] K. Hata and S. Savarese. "CS231A Course Notes 1: Camera Models". In: 2017. URL: https://web.stanford.edu/class/cs231a.

[HS88] C. G. Harris and M. Stephens. "A Combined Corner and Edge Detector". In: *Alvey Vision Conference*. 1988.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pp. 448–456.

[Jad+16] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. *Spatial Transformer Networks*. 2016. arXiv: 1506.02025 [cs.CV].

[Kal60] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552.

[KMF18] Moritz Kampelmühler, Michael G. Müller, and Christoph Feichtenhofer. *Camera-based vehicle velocity estimation from monocular video*. 2018. arXiv: 1802.07094.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[KSL17] Yevhen Kuznietsov, Jörg Stückler, and Bastian Leibe. “Semi-Supervised Deep Learning for Monocular Depth Map Prediction”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2215–2223. DOI: 10.1109/CVPR.2017.238.

[KY55] H. W. Kuhn and Bryn Yaw. “The Hungarian method for the assignment problem”. In: *Naval Res. Logist. Quart* (1955), pp. 83–97.

[LGU15] Philip Lenz, Andreas Geiger, and Raquel Urtasun. “Followme: Efficient online min-cost flow tracking with bounded memory and computation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4364–4372.

[Li+18] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. “High Performance Visual Tracking with Siamese Region Proposal Network”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8971–8980. DOI: 10.1109/CVPR.2018.00935.

[Liu+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. “SSD: Single Shot MultiBox Detector”. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Cham: Springer International Publishing, 2016, pp. 21–37. ISBN: 978-3-319-46448-0.

[Llo82] S. Lloyd. “Least squares quantization in PCM”. In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[Low99] D.G. Lowe. “Object recognition from local scale-invariant features”. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[Lui+20] Jonathon Luiten, Aljosa Osep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. “HOTA: A Higher Order Metric for Evaluating Multi-object Tracking”. In: *International Journal of Computer Vision* 129.2 (Oct. 2020), pp. 548–578. ISSN: 1573-1405. DOI: 10.1007/s11263-020-01375-2.

[MBP20] Dmytro Mykheievskyi, Dmytro Borysenko, and Viktor Porokhonskyy. “Learning local feature descriptors for multiple object tracking”. In: *Proceedings of the Asian Conference on Computer Vision*. 2020.

[NH10] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML’10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077.

[Nis04] D. Nister. “An efficient solution to the five-point relative pose problem”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.6 (2004), pp. 756–770. DOI: 10.1109/TPAMI.2004.17.

[Qiu+19] Jiaxiong Qiu, Zhaopeng Cui, Yinda Zhang, Xingdi Zhang, Shuaicheng Liu, Bing Zeng, and Marc Pollefeys. “DeepLiDAR: Deep Surface Normal Guided Depth Prediction for Outdoor Scene From Sparse LiDAR Data and Single Color Image”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[RD06] Edward Rosten and Tom Drummond. “Machine Learning for High-Speed Corner Detection”. In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443. ISBN: 978-3-540-33833-8.

[Ren+17] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031.

[RF18] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *arXiv* (2018).

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0.

[Rub+11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[Son+20] Zhenbo Song, Jianfeng Lu, Tong Zhang, and Hongdong Li. *End-to-end Learning for Inter-Vehicle Distance and Relative Velocity Estimation in ADAS with a Monocular Camera*. 2020. arXiv: 2006.04082.

[SSZ01] D. Scharstein, R. Szeliski, and R. Zabih. "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms". In: *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*. 2001, pp. 131–140. DOI: 10.1109/SMBV.2001.988771.

[SZ15] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556.

[Tok+21] Pavel Tokmakov, Jie Li, Wolfram Burgard, and Adrien Gaidon. "Learning to Track with Object Permanence". In: *arXiv preprint arXiv:2103.14258* (2021).

[Wan+04] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[Wan+17] Chaoyang Wang, Jose Miguel Buenaposada, Rui Zhu, and Simon Lucey. *Learning Depth from Monocular Videos using Direct Methods*. 2017. arXiv: 1712.00175 [cs.CV].

[Wan+18] Chaoyang Wang, José Miguel Buenaposada, Rui Zhu, and Simon Lucey. "Learning Depth from Monocular Videos Using Direct Methods". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2022–2030. DOI: 10.1109/CVPR.2018.00216.

[Wan+19] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Weinberger. "Pseudo-LiDAR from Visual Depth Estimation: Bridging the Gap in 3D Object Detection for Autonomous Driving". In: *CVPR*. 2019.

[Wan+20] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. "CSPNet: A New Backbone that can Enhance Learning Capability of CNN". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2020, pp. 1571–1580. DOI: 10.1109/CVPRW50498.2020.00203.

[WB18] Nicolai Wojke and Alex Bewley. "Deep Cosine Metric Learning for Person Re-identification". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)* (Mar. 2018). DOI: 10.1109/wacv.2018.00087.

[WBP17] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric". In: *2017 IEEE International Conference on Image Processing (ICIP)*. 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962.

[XAS15] Yu Xiang, Alexandre Alahi, and Silvio Savarese. "Learning to track: Online multi-object tracking by decision making". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4705–4713.

[Yan+18] Nan Yang, Rui Wang, Jörg Stückler, and Daniel Cremers. *Deep Virtual Stereo Odometry: Leveraging Deep Depth Prediction for Monocular Direct Sparse Odometry*. 2018. arXiv: 1807.02570 [cs.CV].

[Yoo+16] Ju Hong Yoon, Chang-Ryeol Lee, Ming-Hsuan Yang, and Kuk-Jin Yoon. "Online multi-object tracking via structural constraint event aggregation". In: *Proceedings of the IEEE Conference on computer vision and pattern recognition*. 2016, pp. 1392–1400.

[YS18] Zhichao Yin and Jianping Shi. *GeoNet: Unsupervised Learning of Dense Depth, Optical Flow and Camera Pose*. 2018. arXiv: 1803.02276 [cs.CV].

[Zha+20] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. *FairMOT: On the Fairness of Detection and Re-Identification in Multiple Object Tracking*. 2020. arXiv: 2004.01888.

[Zhe+19] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression*. 2019. arXiv: 1911.08287. URL: https://arxiv.org/abs/1911.08287.

[Zho+17] Tinghui Zhou, Matthew Brown, Noah Snavely, and David Lowe. "Unsupervised Learning of Depth and Ego-Motion from Video". In: *Computer Vision and Pattern Recognition*. 2017. URL: https://arxiv.org/abs/1704.07813.

[ZK15] Sergey Zagoruyko and Nikos Komodakis. *Learning to Compare Image Patches via Convolutional Neural Networks*. 2015. arXiv: 1504.03641.

[ZK17] Sergey Zagoruyko and Nikos Komodakis. *Wide Residual Networks*. 2017. arXiv: 1605.07146.

[ZKK20] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. "Tracking objects as points". In: *European Conference on Computer Vision*. Springer. 2020, pp. 474–490.

[ŽL16] Jure Žbontar and Yann LeCun. *Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches*. 2016. arXiv: 1510.05970.

[ZP19] Zhipeng Zhang and Houwen Peng. "Deeper and Wider Siamese Networks for Real-Time Visual Tracking". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4586–4595. DOI: 10.1109/CVPR.2019.00472.