

**Combined Learned and Classical Methods for Real-Time Visual Perception in
Autonomous Driving**

**by
Mohamed D. Aladem**

**A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in the University of Michigan-Dearborn
2020**

Doctoral Committee:

**Assistant Professor Samir A. Rawashdeh, Chair
Professor Selim Awad
Assistant Professor Abdallah Chegade
Assistant Professor Alireza Mohammadi**

© Mohamed D. Aladem 2020
All Rights Reserved

Dedicated to my beloved parents

Acknowledgements

First and foremost, I humbly extend my deepest gratitude to my beloved parents. They have always shown me and my siblings unconditional love and support. My -late- father has always implanted the love of science in me and taught me the value of work and dedication. My dear mother, words can never describe how kind, lovely, courageous and supportive she is. She sacrificed a lot in order to see me and my siblings succeed and reach our goals. All I can say is thank you for being the best parents one could ask for. I hope I will always make you proud.

Pursuing a doctorate degree is no easy endeavor. Graduate studies require a significant amount of dedication, patience, hard work, and discipline. I was lucky enough to work under the supervision of a great research adviser, Dr. Samir A. Rawashdeh. Dr. Rawashdeh has helped me tremendously throughout my Ph.D. journey. He helped shape me into an independent, responsible, and caring researcher ready to be a leader in the field. For all the valuable advice and help he selflessly provided, I would like to say thank you.

I would like to thank my great Ph.D. committee, Dr. Alireza Mohammadi, Dr. Abdallah Chehade, and Dr. Selim Awad. Their advice and support proved invaluable throughout my Ph.D. journey. Additionally, I would like to thank my colleagues and friends in the research group, Zaid El-Shair, Sumanth Chennupati, Aaron Cofield, Hisham Alawneh, and Ella Reimann. They became like a family to me. Finally, I would like to thank all the amazing people in the Electrical and Computer Engineering department at the University of Michigan-Dearborn.

This dissertation is but a first step. *Per aspera ad astra.*

Table of Contents

Dedication	ii
Acknowledgements	iii
List of Figures	vii
List of Tables	x
List of Abbreviations	xii
Abstract	xiii
Chapter 1 Introduction	1
Dissertation Goals	5
Dissertation Organization	6
Chapter 2 Background	7
2.1 Image Formation	7
2.2 Multiple View Geometry	10
2.2.1 Epipolar Geometry	11
2.2.2 Triangulation	13
2.2.3 Structure From Motion	13
2.3 Linear Filtering	14
2.4 Image Features	15
2.5 Deep Neural Networks	17
Chapter 3 A Real-Time Visual Odometry Algorithm	20
3.1 Introduction	20
3.2 Related Work	21

3.3	Method	23
3.3.1	Image Sequence	24
3.3.2	Feature Extraction.....	25
3.3.3	Pose Prediction	28
3.3.4	Local Map Tracking (Data Association)	29
3.3.5	Pose Estimation	30
3.3.6	Local Map Maintenance	30
3.3.7	General Implementation Remarks	31
3.4	Evaluation	32
3.4.1	KITTI Dataset.....	32
	Accuracy	32
	Runtime Performance	35
3.4.2	EuRoC Dataset	36
	Accuracy	37
	Effect of Rotational-Invariant Features	37
3.4.3	TUM RGB-D Dataset.....	38
3.5	A Study on Visual Odometry Performance Under Low Illumination	39
3.5.1	Studied Image Preprocessing Techniques	40
	Gamma Correction.....	40
	$L^*a^*b^*$ + CLAHE.....	41
	RG + CLAHE	41
	Bioinspired Retina Model	41
3.5.2	Evaluation.....	42
	Runtime Performance	47
3.6	Summary	47

Chapter 4 A Combined Visual Odometry and Multi-Object Detection and Tracking System	49
4.1 Introduction.....	49
4.2 Literature Survey	51
4.3 Multi-Object Tracking Method.....	52
Keypoints Cost.....	53
IoU Cost	54
4.4 Combined System	55
4.5 Evaluation	57
4.6 Summary	63
Chapter 5 A Single-Stream CNN For Semantic Segmentation and Monocular Depth Prediction	65
5.1 Introduction.....	65
5.2 Literature Survey	67
5.3 OSD-Net	68
5.4 Evaluation and Discussion.....	70
5.5 Summary	75
Chapter 6 Conclusion and Future Work	77
6.1 Dissertation Summary.....	77
The Development Of A Real-Time Visual Odometry Algorithm.	77
A Study On Visual Odometry Performance Under Low Illumination	78
A Combined Visual Odometry And Multi-Object Detection And Tracking System.....	78
A Single-Stream CNN For Semantic Segmentation And Monocular Depth Prediction ..	78
6.2 Future Work	79
Bibliography	81

List of Figures

Figure 1-1 Different types of sensors are used to enable automated driving.....	4
Figure 2-1 Basic pinhole camera model. Because rays of light reflected from the real object pass through the camera aperture, the image is therefore captured flipped.....	7
Figure 2-2 A simplified illustration of the geometry of the pinhole camera model.	8
Figure 2-3 Illustration of the two common radial distortions: (a) Barrel distortion, and (b) Pincushion distortion.	9
Figure 2-4 : [left] Monocular vision where observing an object from one point is not enough to pinpoint its exact position in space. [right] Binocular vision where tracing rays from a stereo camera intersect at the object in space.	10
Figure 2-5 The general setup of epipolar geometry. Gray plane is the epipolar plane. It intersects the two image planes at the epipolar lines (blue color) and contains the 3D point and its projections. The line between the two camera centers is the baseline (orange color). The baseline intersects the image planes at the epipoles.	11
Figure 2-6 The general setup of the stereo rectification problem. Two homographies are computed to virtually transform the images such that epipolar lines become horizontal and image planes become parallel.	12
Figure 2-7 The general setup of structure from motion.	14
Figure 2-8 Example of image convolution where the kernel w was engineered to detect the outline of the input image.	15
Figure 2-9 Common feature types: [TOP] Points or corners, [MIDDLE] Lines, [BOTTOM] Blob or region features [14].....	16
Figure 2-10 A simple illustration of a feedforward artificial neural network.....	17
Figure 2-11 Typical convolutional neural network architecture. This example network consists of two convolutional layers and an output fully connected layer to classify input images as either cats or dogs.	19
Figure 3-1 A high-level demonstration of the main loop of the visual odometry pipeline. Green boxes are inputs and outputs, whereas the blue ones are main processing stages.	24

Figure 3-2 An illustration showing: (a) local 3D map, and (b) detected 2D image features. The goal is to find the correspondence between each 3D map point and detected 2D feature. Sample image from KITTI dataset. (Credit: [23]).	29
Figure 3-3 Estimated trajectory (dashed blue) from LVT against ground truth (solid red) in KITTI dataset. (Credit [23]).	34
Figure 3-4 Average feature age in each frame of KITTI sequence 00.	35
Figure 3-5 Relative runtime speed of the main stages of the visual odometry system. (Credit [23]).	36
Figure 3-6 A sample image from Machine Hall 01 of the EuRoC datasets, where detected features are plotted (red dots). (Credit [23]).	37
Figure 3-7 Results of the four image preprocessing techniques (a) Raw test image without any preprocessing. (b) Gamma correction (c) $L^*a^*b^* + \text{CLAHE}$ (d) $\text{RG} + \text{CLAHE}$ (e) Bioinspired retina model. (Credit [24]).	43
Figure 3-8 The loop traversed while collected the dataset for evaluation. (Attribution: Google Maps).	44
Figure 3-9 Status information from our visual odometry system on different datasets: (a) Number of points in the local map; (b) Number of matched 3D map points with 2D image features to be used in pose estimation; (c) Number of found inliers after pose estimation. (Credit [24]).	46
Figure 4-1 Demonstration of the system output. [top] is estimated poses from visual odometry and 3D tracked objects in world coordinates. [bottom] is detected and tracked objects in the 2D image domain ([68] © 2019 IEEE).	51
Figure 4-2 Keypoints extraction from detected objects. (a) shows that if the mask is set to extract keypoints exactly in the detection bounding box, some keypoints might fall outside the physical object. (b) here I modify the mask to extract keypoints only in the ellipse inscribed in the bounding rectangle. This reduces the number of erroneous keypoints ([68] © 2019 IEEE).	54
Figure 4-3 An overview of the complete combined system and its components. Arrows show the direction of data flow ([68] © 2019 IEEE).	56
Figure 4-4 Qualitative evaluation of our algorithm on selected frames from KITTI dataset ([68] © 2019 IEEE).	61
Figure 4-5 Sample frames showing traffic participants being tracked in 3D world frame ([68] © 2019 IEEE).	61
Figure 4-6 Demonstration of the effect of removing keypoints lying on tracked objects on visual odometry. (a) in most KITTI odometry sequences, cars are stationary. (b) sequence 01 is captured on a highway with moving cars in the scene, so removing those keypoints helps VO estimation ([68] © 2019 IEEE).	63

Figure 5-1 An overview of our network which consists of a single encoder-decoder. It can predict both semantic segmentation and depth in a single output. Assuming the input image has spatial dimensions of $H \times W$, and C classes for semantic segmentation, output has dimensions of $H \times W \times (C+1)$. Credit [98].	69
Figure 5-2 Vanilla Multi-Task Network with one shared encoder and two task-specific decoders. This network was implemented for comparison and evaluation. The two tasks' losses are combined using the arithmetic mean.	71
Figure 5-3 Average validation loss of each model with every training epoch. Credit [98].	72
Figure 5-4 Achieved mean intersection over union (IoU) for semantic segmentation by each model at every training epoch. Credit [98].	73
Figure 5-5 Disparity prediction accuracy by each model at each training epoch. Credit [98].	73

List of Tables

Table 3-1 Visual odometry results on KITTI dataset compared to two other methods.....	33
Table 3-2 Runtime performance measurements.	35
Table 3-3 Computational performance on embedded computers.	36
Table 3-4 RMSE values of absolute trajectory error (ATE) and relative pose error (RPE) on EuRoC datasets.	37
Table 3-5 RMSE values of ATE and RPE on EuRoC datasets using rotational-invariant oriented fast and rotated brief (ORB) features.....	38
Table 3-6 RMSE values of ATE and RPE on TUM RGB-D dataset.	39
Table 3-7 Parameter values used for the parvo channel in the retina model.	42
Table 3-8 Result of the three systems on sequence 3 (Daytime - unprocessed) and the night sequence 28 raw unprocessed and processed with presented techniques. Estimated paths are in blue while driven paths are in red. (Attribution: Google Maps; Credit [24])	44
Table 3-9 Mean \pm standard deviation values of the extracted metric values from our visual odometry system for the different datasets, along with the number of successfully tracked frames.....	47
Table 3-10 Computational performance of the four described preprocessing techniques.....	47
Table 4-1 Metrics used to evaluate multi-object detection and tracking. Based on Ref. [93] [94]. ([68] © 2019 IEEE).....	58
Table 4-2 “CAR” tracking results on KITTI training sequences of my algorithm with two detectors (YOLO and RRC) compared to two different methods ([68] © 2019 IEEE).	59
Table 4-3 Results of each cost alone on KITTI ‘Car’ benchmark ([68] © 2019 IEEE).	60
Table 4-4 “Pedestrian” tracking results on KITTI training sequences of my algorithm with two detectors ([68] © 2019 IEEE).	60
Table 4-5 Visual odometry translation error on KITTI sequences with and without removing features on tracked objects ([68] © 2019 IEEE).....	62

Table 5-1 Quantitative evaluation of our method compared to standard baseline ones.	
Credit [98].	74
Table 5-2 Qualitative results on random frames from the CityScapes dataset. Credit [98].	75

List of Abbreviations

VO Visual Odometry

V-SLAM Visual Simultaneous Localization And Mapping

SfM Structure from Motion

CNN Convolutional Neural Network

MOT Multi-Object Tracking

ADAS Advanced Driver-Assistance Systems

DoF Degrees of Freedom

GPU Graphics Processing Unit

IMU Inertial Measurement Unit

SIMD Single Instruction Multiple Data

XOR Exclusive Or

Abstract

Autonomy, robotics, and Artificial Intelligence (AI) are among the main defining themes of next-generation societies. Of the most important applications of said technologies is driving automation which spans from different Advanced Driver Assistance Systems (ADAS) to full self-driving vehicles. Driving automation is promising to reduce accidents, increase safety, and increase access to mobility for more people such as the elderly and the handicapped. However, one of the main challenges facing autonomous vehicles is robust perception which can enable safe interaction and decision making. With so many sensors to perceive the environment, each with its own capabilities and limitations, vision is by far one of the main sensing modalities. Cameras are cheap and can provide rich information of the observed scene. Therefore, this dissertation develops a set of visual perception algorithms with a focus on autonomous driving as the target application area.

This dissertation starts by addressing the problem of real-time motion estimation of an agent using only the visual input from a camera attached to it, a problem known as visual odometry. The visual odometry algorithm can achieve low drift rates over long-traveled distances. This is made possible through the innovative local mapping approach used. This visual odometry algorithm was then combined with my multi-object detection and tracking system. The tracking system operates in a tracking-by-detection paradigm where an object detector based on convolution neural networks (CNNs) is used. Therefore, the combined system can detect and track other traffic participants both in image domain and in 3D world frame while simultaneously estimating vehicle motion. This is a necessary requirement for obstacle avoidance and safe navigation. Finally, the operational range of traditional monocular cameras was expanded with the capability to infer depth and thus replace stereo and RGB-D cameras. This is accomplished through a single-stream convolution neural network which can output both depth prediction and semantic segmentation. Semantic segmentation is the process of classifying each pixel in an image and is an important step toward scene understanding. Literature survey, algorithms descriptions, and comprehensive evaluations on real-world datasets are presented.

Chapter 1

Introduction

Autonomous robots are one of the main defining themes of next-generation cities and societies. Humans have indeed been relying on robots for a while now in many specialized tasks, such as material delivery robots in warehouses. Mars exploration rovers are another interesting application of mobile robotics in a specialized task. That said, personal robots, such as robotic vacuum cleaners, have been integrating more and more into our daily lives in recent years. Many more personal robotic applications are under active development. This includes, for example, delivery robots which can deliver groceries and purchases, and social robots. Social robots can tend to the elderly or people with special needs for example. They can also be used as educational tools for children. This trend leads to the realization that autonomy, robotics, and *artificial intelligence* (AI) are indeed the defining theme of next-generation societies.

Vehicle automation is perhaps the first large-scale integration of a robotic application into our lives. Vehicle automation refers to automating driving tasks where the vehicle takes control instead of the driver with the ultimate goal of enhancing safety. This has manifested in different *Advanced Driver-Assistance Systems* (ADAS) such as adaptive cruise control, lane departure warning, emergency braking, among others. Moreover, this vehicle automation includes all the way up to fully self-driving cars that are under active development. As a matter of fact, the Society of Automotive Engineers (SAE) has issued a guideline classifying automation into the following levels [1]:

- **Level 0: No automation.** In this level, the human driver has full control over all the driving tasks.
- **Level 1: Driver assistance.** In this level, the human driver is assisted by the system only in specific functions.
- **Level 2: Partial automation.** In this level, the system can perform partial automation. The driver should always be ready to take over.

- **Level 3: Conditional automation.** In this level, the system drives and monitors environments. The driver should be ready to take over when requested by the system.
- **Level 4: High automation.** In this level, the system handles the driving and monitoring environments. The driver is not required after the system takes over.
- **Level 5: Full automation.** In this level, the system handles all aspects of driving.

Vehicle automation is a very promising technology to have a profound impact on our lives. According to the National Highway Traffic Safety Administration (NHTSA) [2], automated vehicles have a great safety benefit. They have the potential to save lives and reduce injuries as 94% of serious accidents are due to human error. The unfortunate number of deaths related to vehicle crashes in 2017 exceeded 37,133. This emphasizes the critical lives-saving role that automated driving could play. Furthermore, automated driving can have additional economic benefits. Motor vehicle crashes in 2010 cost \$242 billion in economic activity. Automated driving has the potential to erase these costs. Moreover, with coordination between different autonomous vehicles, this can reduce traffic congestions and lead to increased speeds. Which means passenger spend less time in traffic, and they can utilize their transit time doing something else. Finally, autonomous driving will increase access to mobility for more people, such as the elderly and people with disabilities. All of these beneficial impacts of automated driving motivated my research to focus on this area.

A self-driving vehicle is a complex machine with many cooperating subsystems. However, on a basic level, an autonomous vehicle must perform the following tasks:

1. **Localization and Mapping**, in which the autonomous vehicle needs to figure out its own location and movement.
2. **Scene Understanding**, in which the autonomous vehicle needs to understand the observed scene. This includes detecting objects and estimating their motion trajectories, detecting drivable areas, ...etc.
3. **Motion Planning**, in which the autonomous vehicle must figure out how to move from a starting point to a destination point. This includes planning paths around obstacles using the hybrid A* algorithm for example, planning paths for switching lanes smoothly, to higher-level route planning.
4. **Human-Robot Interaction**, in which the autonomous vehicle monitors the physical and cognitive state of the passengers and related tasks.

The first two tasks can be collectively classified as *robot perception*. Although computers can do tasks impossible for humans such as millions of complex arithmetic operations per second, even the perception capability of a toddler is still challenging for them. Robust and real-time perception capability is needed for a robot so that it can perform decision making and subsequently motion controls and actuation. Different types of sensors are combined and used in any robotic perception pipeline as shown in Figure 1-1. This include:

- **Global Positioning System (GPS)**, which can provide absolute geolocation information. However, GPS can only provide said location information when it has unobstructed line-of-sight to at least four GPS satellites. This means GPS sensors lose satellite lock in tunnels and urban canyons.
- **Light Detection and Ranging (LIDAR)**, which can provide a dense point cloud of the surrounding. They operate on the time-of-flight principle and can be used for obstacle avoidance or scene interpretation. However, they are active sensors and can interfere with each other and they are still relatively expensive sensors.
- **Wheels encoders**, which can provide motion estimates by dead reckoning. However, they suffer from drift and wheel-slippage depending on the terrain.
- **Inertial Measurement Units (IMU)**, which can provide orientation and acceleration.
- **Cameras**, which can provide a rich representation of the observed scene including texture information. This can be used for semantic and geometrical scene understanding. However, cameras suffer under non-ideal illumination conditions.

Cameras stand out as one of the primary sensors in the robotics perception sensor suite. They are affordable, widely available, can provide rich information of the observed scene, and passive which means they do not interfere with each other when multiple are deployed. In fact, *visual perception* is our most powerful sense to perceive our surroundings as human beings. Our retina is about 10cm^2 and contains millions of photoreceptors that provide a large amount of information at a data rate of approximately $3\text{ Gigabytes/second}$. This means large proportion of our brain processing power is dedicated to visual perception. Computer vision techniques have indeed solved many important perception problems for autonomous robots including object detection, recognition, motion estimation, and mapping. Hence, the focus of my research is on visual perception approaches for robot perception.

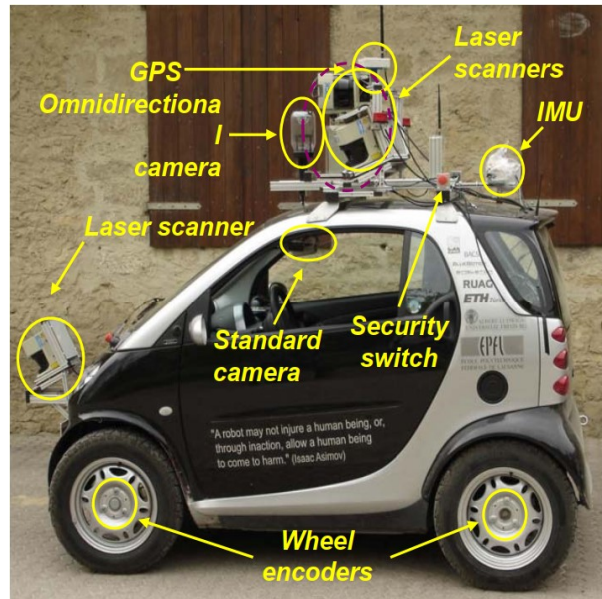


Figure 1-1 Different types of sensors are used to enable automated driving¹.

Computer vision techniques can be classified as either *classical* or *deep learning* [3] methods. Classical computer vision methods are *model-based*. This means a well-defined model based on mathematical and physical properties is developed and used for the target problem. On the other hand, deep-learning-based approaches are *data-driven*. Here, a function to perform the desired task is learned through exposure to a large amount of data.

An example illustrating this is the problem of object recognition. Under classical computer vision approaches, this problem is tackled by first extracting distinctive features using an engineered feature detector such as SIFT [4]. Then, a simple classifier such as *support vector machine* (SVM) is used to classify these extracted features into the target object. However, in the modern deep learning-based methods, a neural network will learn how to extract and classify features all by itself just by exposing it to a large amount of data. That is, it is a data-driven end-to-end method.

Deep neural networks have established their role as the workhorse of modern computer vision. Their star has risen since 2012 when a *convolutional neural network* (CNN) called AlexNet [5] won the ImageNet large-scale image classification challenge [6] beating the best

¹ Image credit: lecture 1 slides, Introduction and Overview, by R. Siegwart in the Autonomous Mobile Robots course at ETH Zurich. URL: https://asl.ethz.ch/education/lectures/autonomous_mobile_robots/spring-2018.html (Accessed Aug. 27, 2019)

classical method with a significant margin. Afterward, winner methods of this competition are all deep learning methods.

Deep learning approaches have indeed achieved unprecedented success on many challenging computer vision tasks that otherwise were very difficult to achieve reasonable accuracies such as object recognition and semantic segmentation. However, classical approaches are not obsolete yet. For example, geometric computer vision such as *structure-from-motion* (SfM) is still model-based and data-driven attempts are still lacking. Moreover, depending on the problem at hand, a deep neural network might be an overkill. For example, if the task is to detect green balls in the scene, using a classical image processing approach would be easier and more efficient than training a neural network to perform this task.

Dissertation Goals

The aforementioned narration leads to the motivation of this dissertation, classical and deep-learning methods should be combined to enable robust real-time visual perception necessary for robotics and autonomous driving. I will start by addressing the fundamental problem of real-time motion estimation of an agent in an uncontrolled and unmapped environment by developing a visual odometry algorithm. Accurate and timely motion estimates is an essential capability enabling decision making and control. My algorithm is model-based relying on principles from geometric computer vision. The algorithm is lightweight and supports both stereo and RGB-D cameras.

The visual odometry algorithm addressed the *where am I?* question. Then I needed to address the *what is around me?* question. For this, I have built on top of my developed visual odometry algorithm, a multi-object detection and tracking system. This combined system has both model-based and deep learning-based components. The object detector is based on a convolutional neural network. This object detector is executed at every frame to detect other traffic participants (vehicles and pedestrians). The multi-object tracker's role then becomes to associate detections across frames. Once we have pose estimates from visual odometry, tracking information in image domain from multi-object tracker, and disparity map from the stereo camera, we can then estimate 3D tracking information in world coordinates. This is an essential requirement for obstacle avoidance and safe navigation.

Afterwards, I tackled a fundamental limitation in the visual sensors used. In my previous work, I have relied on specialized sensors (stereo and RGB-D camera) in order to obtain depth

information. However, with the omnipresence of monocular cameras, it would be beneficial to extend their operational range with the capability of inferring depth. In addition to that, I wanted to address the problem of semantic segmentation which is an important step in scene understanding. Therefore, I have developed a single-stream convolutional neural network model that can simultaneously output semantic segmentation and predict depth from a single input image.

Finally, research in this domain tends to focus on a single specific problem in isolation with the goal of achieving best accuracy metrics for that problem on standard benchmarks. However, the perception system deployed onboard a robot or self-driving car is expected to solve these visual perception problems simultaneously. Thus, I focus on these tasks' integrations and run-time performance as well.

Dissertation Organization

This dissertation proceeds as follows:

- Chapter 2 addresses some fundamental relevant background information in robotic vision.
- Chapter 3 discusses the visual odometry algorithm.
- Chapter 4 discusses the combined visual odometry and multi-object detection and tracking system.
- Chapter 5 discusses the convolutional neural network used to simultaneously perform semantic segmentation and depth prediction from a single image.
- Chapter 6 concludes the dissertation and discusses future research directions.

Chapter 2

Background

In this chapter, relevant background topics on both classical and deep learning computer vision will be reviewed.

2.1 Image Formation

The camera is the primary device in computer vision that is used to capture the observed scene. Computer vision algorithms then work on its output images to extract useful information for target applications. Hence, understanding the geometry of the camera and the image formation process is an essential first step. The most basic imaging device and the dominating model of image formation is the *pinhole camera model* as shown in Figure 2-1.

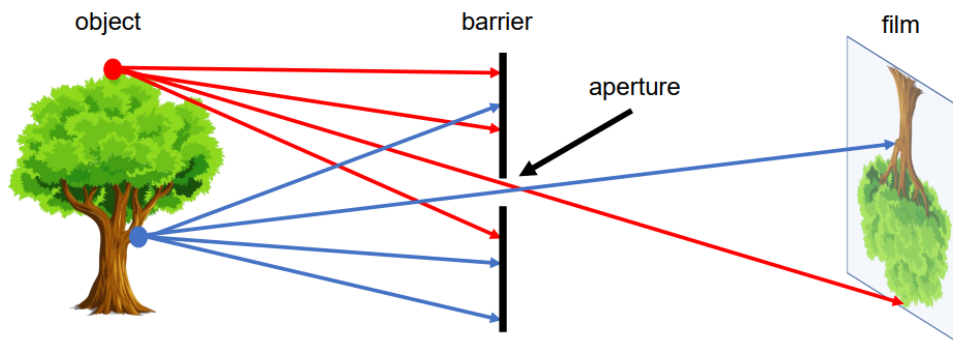


Figure 2-1 Basic pinhole camera model. Because rays of light reflected from the real object pass through the camera aperture, the image is therefore captured flipped.²

In the pinhole camera model, rays of light pass through a small hole, or *aperture*, in one side of a dark chamber and strike an imaging plane where the image is recorded. The aperture is assumed infinitesimally small for the pinhole camera model so that only a few of the light rays reflected from the object pass and hit the sensor or imaging plane. Thus, a one-to-one mapping between the scene object and its image on the film is formed. As a result of the aperture and the

² Image credit: Stanford university CS231A course notes 1 by Kenji Hata and Silvio Savarese. URL: http://web.stanford.edu/class/cs231a/course_notes.html (Accessed Aug. 27,2019)

construction of the pinhole camera, the image formed on the film is flipped as can be seen from Figure 2-1.

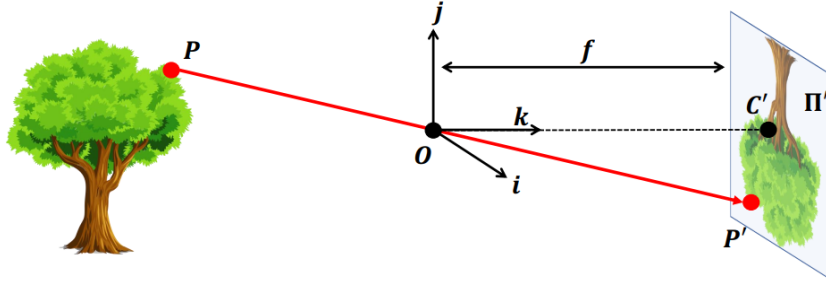


Figure 2-2 A simplified illustration of the geometry of the pinhole camera model.³

A more formal illustration of the construction and geometry of the pinhole camera model is shown in Figure 2-2. The pinhole O is also called the *center of projection* or the *camera center*. The film or sensor is referred to as the *image plane* π' . The distance between the camera center and the image plane is called the *focal length* f . The *camera coordinate frame* $[i \ j \ k]$ is defined with its origin at the pinhole and the k axis is perpendicular to the image plane. The ray \overrightarrow{Ok} intersects the image plane at the *principal point* $C' = (c_x, c_y)$. Mathematically, it is common to represent the image plane on the other side of the camera center which results in the object image to appear upright and not being flipped anymore. This works because of similar triangles and simplifies the geometrical analysis of the camera model.

The projection of objects onto the imaging sensor is a linear mapping between 3D points and image points in homogeneous coordinates. The process can be illustrated in terms of 3D point $P = [x, y, z]^T$, camera parameters which include the focal length (f_x, f_y) and the principal point (c_x, c_y) . Projecting through the pinhole results in the image point $P' = (p_x, p_y)$ in pixels as in the following equations:

$$p_x = f_x \frac{x}{z} + c_x \quad (2.1)$$

$$p_y = f_y \frac{y}{z} + c_y \quad (2.2)$$

Actual real-world cameras differ from the ideal pinhole camera model in that they employ a *lens* at the pinhole. The primary motivation for employing lenses is to overcome the limitation of

³ Image credit: Stanford university CS231A course notes 1 by Kenji Hata and Silvio Savarese. URL: http://web.stanford.edu/class/cs231a/course_notes.html (Accessed Aug. 27,2019)

non-infinitesimally small apertures. Ideal pinhole cameras assume an infinitesimally small aperture which can limit the number of rays passing from each point in the scene. Besides that, lenses are placed for different goals, they are used to allow a sufficient amount of light in, zooming in and out by changing the focal length, changing the sharpness of the image by controlling the aperture, or introducing specific visual effects [7]. The use of lenses, however, can result in distortion in the captured image. In this situation, the image deviates from and cannot be described by the pinhole camera model alone. This image distortion can take a variety of shapes depending on the lens material and manufacturing quality. The most commonly encountered form of distortion is *radial* distortion which can be classified as either *barrel* or *pincushion* distortion as shown in Figure 2-3.

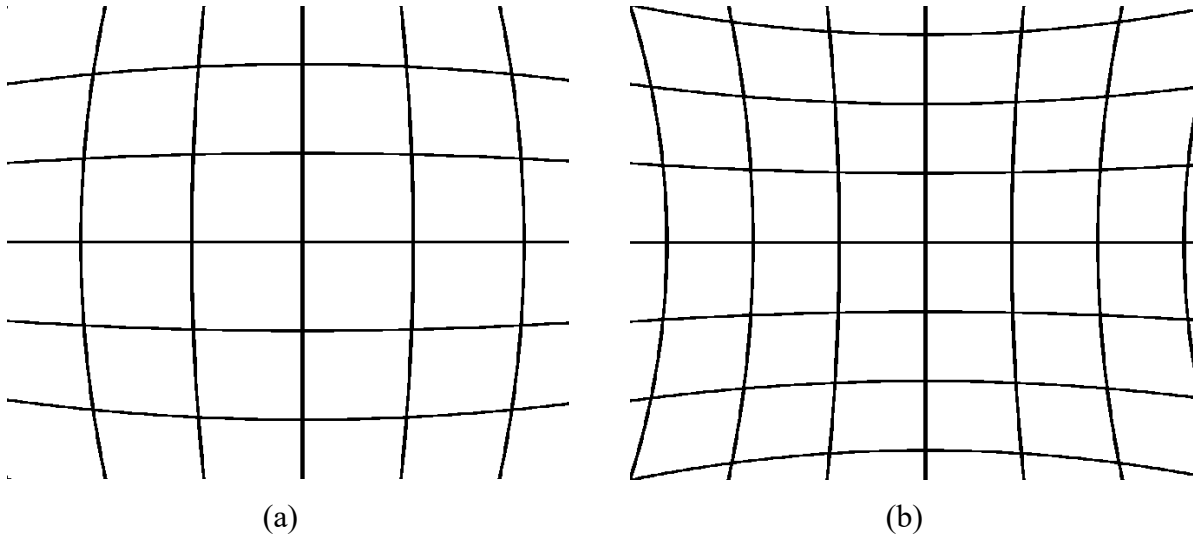


Figure 2-3 Illustration of the two common radial distortions: (a) Barrel distortion, and (b) Pincushion distortion.

It is important to account for the effect of these distortions in order to get accurate results for the computer vision tasks. Radial distortion can be described as a function of the distance to the center of distortion [7] as follows:

$$\begin{pmatrix} u_d \\ v_d \end{pmatrix} = L\left(\sqrt{u_l^2 + v_l^2}\right) \begin{pmatrix} u_l \\ v_l \end{pmatrix} \quad (2.3)$$

where (u_d, v_d) are the distorted image pixels, (u_l, v_l) are the linearized image pixels. $L(r)$ is the distortion function and depends on the distance r to the center of distortion. It is usually approximated by a Taylor series of order n as follows:

$$L(r) \approx 1 + k_1 r + k_2 r^2 + k_3 r^3 + \dots + k_n r^n \quad (2.4)$$

The intrinsic camera parameters, which are the focal length and principal point assuming square pixels, and the coefficients for the lens distortion polynomial are obtained through a process called *camera calibration*. This camera calibration procedure needs to be done for each camera and the obtained parameters cannot be reused even for the same camera model by the same manufacturer.

2.2 Multiple View Geometry

The typical pinhole camera is *projective* in-nature. The 3D scene is projected to a 2D image which results in the loss of depth information. In ideal pinhole camera, lengths and angles information is lost and only straight-line information is preserved. In order to obtain depth information and observed object's dimensions, we need to observe the same object from two different cameras with a know fixed distance between them, usually called the *baseline*. This setup of having two pinhole cameras with a fixed baseline distance between them is known as a *stereo camera*.

The mentioned concept is illustrated in Figure 2-4. When a single pinhole camera captures an image of a tree, it is challenging to pinpoint exactly which tree in the world it was. The captured tree could lie anywhere on the red arrow, and all of them would have been projected to the same 2D tree image. On the other hand, if the tree was captured from a stereo camera, it becomes easier to pinpoint the exact tree in the world. A single pinhole camera is also called *monocular* while a stereo camera is also called *binocular*. The geometry that underlies the stereo imaging process is called *Epipolar geometry*. A great reference that describes single-, two-, and multiple-view geometry is by Hartley and Zisserman [8].

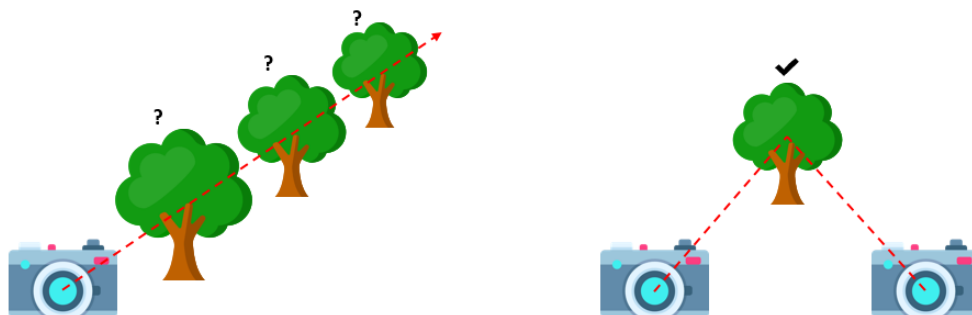


Figure 2-4 : [left] Monocular vision where observing an object from one point is not enough to pinpoint its exact position in space. [right] Binocular vision where tracing rays from a stereo camera intersect at the object in space⁴.

⁴ Tree and camera icons made by Freepik from www.flaticon.com

2.2.1 Epipolar Geometry

The relationships between a 3D point in the scene, the two cameras of a stereo setup, and the projection of this 3D point on the image planes of the two cameras are described by *epipolar geometry*. That is, epipolar geometry is the geometry of stereovision and is demonstrated in Figure 2-5.

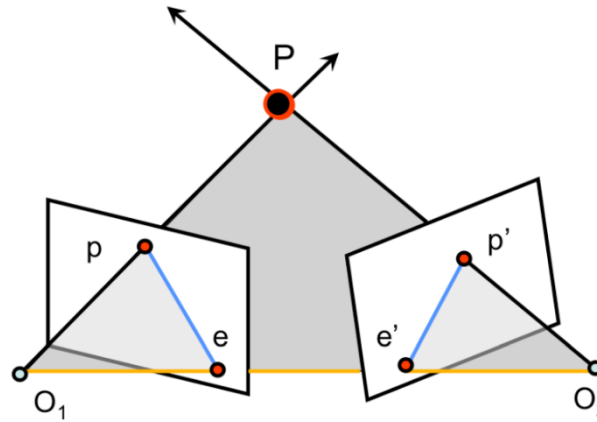


Figure 2-5 The general setup of epipolar geometry. Gray plane is the epipolar plane. It intersects the two image planes at the epipolar lines (blue color) and contains the 3D point and its projections. The line between the two camera centers is the baseline (orange color). The baseline intersects the image planes at the epipoles.⁵

In Figure 2-5, O_1 and O_2 represent the two centers of projection of the left and right cameras respectively of the stereo camera. The two cameras are looking at a 3D point of interest P in the scene. Points p and p' represent the projections of point P onto the image planes of the left and right cameras. Each camera's optical center (O_1 and O_2) projects into a different point in the other camera's image plane. These two image points, e and e' , are called *epipoles* or *epipolar points*. The plane formed by P , O_1 and O_2 is called the *epipolar plane*. It intersects the two image planes at the *epipolar lines*. That is, the line (e' , p') in the right camera's image plane is an epipolar line. This line is also the projection of the 3D line (P , O_1) onto the right camera's image plane. A similar symmetric argument can be made for the left camera. The line between the two camera centers is called the *baseline*.

The most important result of this epipolar geometry is that the observed 3D point P and its two projections p and p' must lie on the epipolar plane. This also means that the projected point on the left camera image plane p must lie on the left camera epipolar line and cannot be anywhere else in the image. This is true for the right camera as well. This requirement is called

⁵ Image credit: Stanford university CS231A course notes 3 by Kenji Hata and Silvio Savarese. URL: http://web.stanford.edu/class/cs231a/course_notes.html (Accessed Aug. 30,2019).

the *epipolar constraint*. As a result of this epipolar constraint, if the two projections p and p' are known, and the relative transformation between the two cameras is known, then the projection rays $\overrightarrow{O_1 p}$ and $\overrightarrow{O_2 p'}$ must intersect at the 3D point P . This process of calculating the 3D point from the two projected image points is called *triangulation*, which constitutes an important concept that forms the basis of 3D reconstruction.

The *stereo correspondence* problem is defined as the task of finding which parts of an image corresponds to in the other one from the stereo camera. Stereo correspondence is solved first to find the point p' in the right camera that corresponds to the point p in the left one. Once the correspondence is found, they can be triangulated into the 3D point P . The epipolar constraint simplifies this stereo correspondence problem as any point in one view must lie on the epipolar line in the other one. Notice how if the two image planes of the left and right cameras were coplanar, then the epipolar lines would be horizontal. In this case, finding stereo correspondences will be further simplified since the search for any point from one image would be restricted to the same horizontal pixel row in the other one.

Because it is difficult in practice to build a perfect stereo camera with coplanar image planes, a transformation process called *stereo rectification* is usually performed on the recorded stereo images to achieve the same desired effect. In this process, the captured images are virtually transformed such that the epipolar lines are horizontal and parallel to the stereo baseline. This stereo rectification task is illustrated in Figure 2-6.

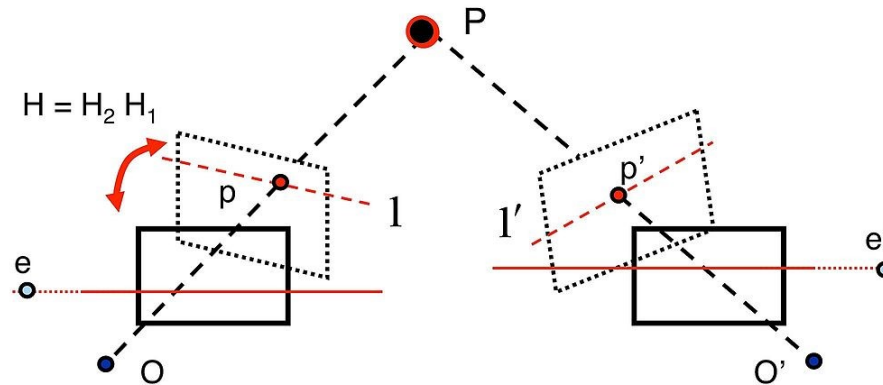


Figure 2-6 The general setup of the stereo rectification problem. Two homographies are computed to virtually transform the images such that epipolar lines become horizontal and image planes become parallel.⁶

⁶ Image credit: Stanford university CS231A course notes 3 by Kenji Hata and Silvio Savarese. URL: http://web.stanford.edu/class/cs231a/course_notes.html (Accessed Aug. 30,2019).

2.2.2 Triangulation

Triangulation is the process of computing the 3D point location given its projections in two or more images. It is a fundamental task in multiple view geometry. In Figure 2-5 we saw how by knowing the projected points p and p' in the left and right images and by knowing the two cameras' parameters, then two rays $\overrightarrow{O_1p}$ and $\overrightarrow{O_2p'}$ must intersect at the scene point P and thus its 3D location is found. This sounds straightforward and easy in theory. However, in practice due to noise in measuring the projected points p and p' , and imperfections in camera calibration processes, those two rays $\overrightarrow{O_1p}$ and $\overrightarrow{O_2p'}$ rarely intersect in the real world. One solution to this problem is by finding the scene point P that minimizes the reprojection error in both views. A discussion of this problem and possible solution is by Hartley and Sturm [9].

2.2.3 Structure From Motion

Structure from motion (SfM) can be defined as the process of simultaneously estimating the 3D structure of the scene and cameras' parameters using a set of images of the scene taken by the cameras. In the most general scenario, the set of images could be unordered, and each is taken by potentially a different camera, thus have a different intrinsics. An example of this is scene reconstruction from community photos found on the internet [10]. Structure from motion also has important applications in robotics and augmented reality (AR) [11]. In this scenario, the problem is slightly simplified as the images are taken from the same single camera and they are consecutive. An illustration of general setup of the structure from motion problem is shown in Figure 2-7. Notice that if monocular cameras were used, the scene will be reconstructed up to a scale factor. That is, the reconstructed scene and the actual real-world scene are related by a similarity transformation. In order to obtain true scale reconstruction, another source of information is needed such as the physical dimensions of an object in the scene.

A more formal definition of the structure from motion problem is as follows, suppose we have m cameras each with a projective transform M_i which encompasses both intrinsic and extrinsic parameters of the camera. Assume the scene consist of n 3D points each is denoted as X_j . Each 3D scene point X_j is projected into the 2D point x_{ij} in the image planes of the cameras in which it is visible using their projective transformation M_i . The objective of the structure from motion problem is then to recover both the *structure* of the scene, the locations of the n 3D

points, and the *motion* of the cameras, the m projective transformations, using the measurements x_{ij} .

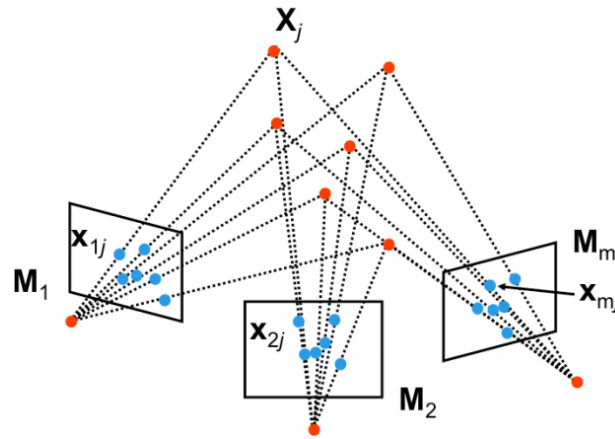


Figure 2-7 The general setup of structure from motion. ⁷

Bundle adjustment is one of the fundamental methods to solve the structure from motion problem. It was initially proposed in the photogrammetry field and later introduced in the computer vision community [12]. Bundle adjustment is formalized as a non-linear optimization problem with the objective to minimize the reprojection errors. Reprojection error is defined as distance between the projection of each reconstructed 3D point on the images where it is visible and its corresponding observations there. This minimization problem is usually solved using the non-linear iterative methods Gauss-Newton or Levenberg-Marquardt. Bundle adjustment powers most recent feature-based visual SLAM (simultaneous localization and mapping) systems [11] [13]. Bundle adjustment can handle a large number of cameras and scenarios where not every 3D point is observed by every camera. However, it can result in a large optimization problem to be solved. Moreover, as it is solved using non-linear iterative optimization algorithms, it requires good initial conditions.

2.3 Linear Filtering

Linear filtering is one of the elementary operations in image processing. It involves using a *neighborhood* or *local operator* which utilizes the pixel values in the vicinity of the target pixel to determine its final output value. This is in contrast to *point operators* that map each input pixel

⁷ Image credit: Stanford university CS231A course notes 4 by Kenji Hata and Silvio Savarese. URL: http://web.stanford.edu/class/cs231a/course_notes.html (Accessed Aug. 30,2019).

to an output pixel disregarding any other pixel value. An example of point operators is color space conversion whereas an example of local operators is gaussian blurring of an image.

This filtering is performed by a mathematical operation known as *convolution*. In image processing, convolution is performed by convolving the input image with a small matrix called the *kernel* as in the following equation:

$$g(x, y) = w * f(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x - s, y - t) \quad (2.5)$$

where $g(x, y)$ is the filtered output image, $f(x, y)$ is the original input image, w is the kernel. This basically means that the kernel slides over the original image pixel-by-pixel. At each pixel location, each element of the kernel is multiplied by its corresponding element from the input image. Multiplications results are then summed, and this summed result will become the value of the corresponding output pixel.

Kernels can be engineered to accomplish different image processing tasks, such as image blurring, sharpening, blob detection, among many others. An example of an edge detection kernel is shown in Figure 2-8. Image convolution is the underlying operation of *convolutional neural networks* (CNNs) where rather than manually hand-crafting the kernels, they are learned from the data instead.



Figure 2-8 Example of image convolution where the kernel w was engineered to detect the outline of the input image.⁸

2.4 Image Features

A captured image is a dense array of intensity samples. Depending on the task at hand, this can be cumbersome and inefficient to work with. An approach is to extract image features from the image, therefore reducing this dense array of pixels to a smaller set of distinctive features.

⁸ The online tool by Victor Powell was used to compute the convolution. URL: <http://setosa.io/ev/image-kernels/> (Accessed Aug. 31, 2019).

The extracted features can then be used to perform different computer vision tasks such as 3D reconstruction, object recognition, image stitching, among others. Detected image feature types include point or corner-like, lines, and uniform regions (blobs) features. A sample showing the different feature types is shown in Figure 2-9.

Point or corner features usually are the fastest to detect and require the least amount of computation, that is why they are more common in real-time high frame rate applications such as visual odometry. On the other hand, blob features enjoy the greatest level of invariance but they are slower to compute. Blob features are usually used for visual search, object recognition, place recognition and similar applications. Other than the required computation and the memory footprint of features, one important performance metric is *repeatability*.

Matching between features can be done using *feature descriptors*. A feature descriptor acts as a signature for that feature, it is some function that is applied to the image patch around the feature in order to describe it in a way that makes it *invariant* to different image changes of interest applicable to our application.



Figure 2-9 Common feature types: [TOP] Points or corners, [MIDDLE] Lines, [BOTTOM] Blob or region features [14].

2.5 Deep Neural Networks

Deep learning [15] has recently resulted in many breakthroughs in different domains including computer vision, speech processing, robotics, bioinformatics, among many others. Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction [3]. Deep learning models are data-driven computing models where they learn by feeding them examples rather than explicitly programming the domain-specific rules. An *artificial neural network* (ANN) is a computing model inspired from biological brains. They only took their initial inspiration from biological brains but then their development diverged, and they no longer mimic biological brains anymore.

A simple example of a neural network is shown in Figure 2-10. The blue circles represent *neurons*, or more accurately, *cells* or *units* as they no longer accurately mimic biological neurons. These cells are arranged in layers with connections between cells in consecutive layers. The first layer is called the *input* layer as the data to the network is fed through it. Then, there could be multiple *hidden* layers. And the final layer is called the *output* layer. Each connection would have a weight value associated with it. The input signals from the input units pass-through connections and their values get multiplied by the weight values of the connections they encounter. The results of these multiplications will become the activation values of the next layer neurons. This process continues for each subsequent layer until the values of the final activation of the output layer are computed.

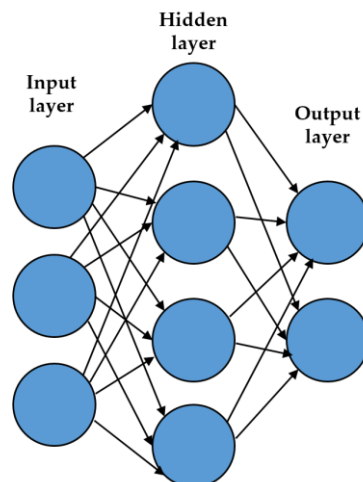


Figure 2-10 A simple illustration of a feedforward artificial neural network.

Note how this simplified model is still linear, values are only multiplied and added as they are propagating forward to the output. In order to enable the neural network to approximate non-

linearities, we need to add *activation functions* after units. These activation functions will transform their input values in a non-linear way. For example, the sigmoid activation function is defined as:

$$\phi(z) = \frac{1}{1 + e^{-z}} \quad (2.6)$$

This has the effect of non-linearly transforming the input values to the [0,1] output range.

The process of *training* the neural network; i.e., finding the weight values of all the units, involves first defining a *loss function*. This loss function quantifies the error between the outputs from the neural network and the ground truth values. Once we have this error estimate, we can update the weights of all the units with the objective of minimizing this error. This neural network training and weight update are performed using an algorithm called *backpropagation*. Backpropagation essentially relies on the recursive application of the chain rule from Calculus to break up the computations and on the gradient descent optimization algorithm. The most important finding of artificial neural networks is that any feedforward neural network with a single hidden layer can approximate any function, this is called the *universal approximation theorem* [16] [17].

An important class of deep neural networks is *convolution neural networks*, often abbreviated as *CNNs* or *ConvNets*. These networks sit at the heart of most modern computer vision methods. Modern ConvNets in visual applications were introduced by LeCun et al. [18]. As the name implies, the convolution operation is the basis of at least one layer of these networks. In section 2.3, I discussed how image kernels or filters can be engineered to extract desired features from the image by convolving the kernel with the image. The operational concept of ConvNets is to *learn* these filters from data using backpropagation. An example of a typical ConvNet architecture is shown in Figure 2-11.

Each convolutional layer consists of a set of filters with learnable parameters. These filters are small in width and height dimensions (e.g., 3×3) but extend to the full depth of their input. The resultant output response of each filter, when convolved with its input, is called a *feature map*. All the resultant feature maps from all the filters in a given convolutional layer are stacked together as the final output of that layer. These stacked feature maps are then passed to the next layer in the network.

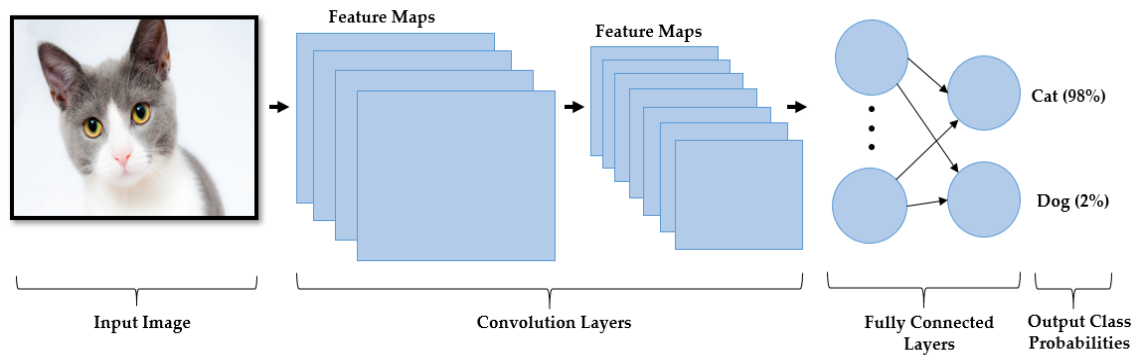


Figure 2-11 Typical convolutional neural network architecture. This example network consists of two convolutional layers and an output fully connected layer to classify input images as either cats or dogs.

The filters in the first convolutional layer of the network which operates directly on the input image would typically learn simple features, such as blobs and edges oriented in different angles. However, as subsequent convolutional layers will operate on the feature maps produced from their predecessor layer, their filters would learn higher-level complex features. This contributed to the success of convolution neural networks as powerful feature detectors for visual recognition applications. However, this also makes it difficult to interpret the meaning of these learned filters. There have been many attempts at visualizing and understanding the intrinsics of convolutional neural networks [19] [20] [21].

Chapter 3

A Real-Time Visual Odometry Algorithm

Real-time motion estimation is a fundamental task for a mobile robot. Low-latency motion estimates are important for stable high-speed control and decision making. Despite being well studied, it remains challenging to provide timely and accurate robot motion estimates for applications where the mobile robot is operating in uncontrolled and previously unknown environments. Examples of such applications include *micro-aerial vehicles* (MAVs) exploring a new environment, *advanced driver-assistance systems* (ADAS) in modern automobiles, and autonomous self-driving vehicles. Visual odometry is an effective method to provide said motion estimates in real-time. Visual odometry can be defined as the process of estimating the pose (position and orientation) of an agent (e.g. a mobile robot) using only the visual input from a camera attached to it [22]. My work in this chapter was originally published in Ref. [23] [24].

3.1 Introduction

Estimating the mobile robot's position robustly and accurately for long-term operation is a difficult problem, and it is an active area of research. Dead-reckoning approach based on a combination of *Inertial Measurement Units* (IMUs) and wheel odometers might be acceptable for short-term tasks. However, continuous integration of noisy measurements from such sensors results in the accumulation of error and permanent drift in estimated position, which makes them unsuitable for long-term sustained operation. Moreover, wheel odometry suffers from wheel slippage on uneven terrains or slippery surfaces, such as those facing Mars rovers [25].

The *Global Positioning System* (GPS) is widely available and is capable of providing geolocation to the receiver. This absolute location measurement can be used to observe the drift in dead-reckoned position estimate and hence correct it properly. However, GPS can only provide said location information when it has unobstructed line-of-sight to at least four GPS satellites. This means GPS sensors lose satellite lock in tunnels and urban canyons. Other commonly used sensors include *Light Detection and Ranging* (LIDAR) which fail in rain, Ultra-

sonic sensors which have a short-range, and Radar which has low angular resolution and may not detect less massive objects well, such as pedestrians or animals. Each sensor has its advantages and its share of downsides, which makes it necessary to employ a suite of sensors when designing navigation systems to overcome the limitations of each sensor alone.

Cameras are an attractive sensor to perform motion estimation because of their affordability and the richness of information that can be gathered. Vision-based motion estimation is capable of providing accurate estimates in conditions where dead-reckoning or GPS fails. Moreover, they can be fused with other sensors as well, such as GPS [26], or Inertial Measurement Units (IMUs) in what is called visual-inertial odometry as in Ref. [27].

In this chapter, I will present my innovative visual odometry system called *lightweight visual tracking* (LVT) [23]. Unlike typical visual odometry approaches where features are tracked and motion is estimated between consecutive frames only, my system tracks features for as long as possible. This results in a system that is approaching full visual simultaneous localization and mapping (V-SLAM) systems in terms of accuracy while still maintaining low computational overhead. Moreover, the system supports both stereo and RGB-D cameras.

3.2 Related Work

Visual odometry is considered an active area of research where many different methods have been developed over the years. A detailed review of the field of visual odometry was published by Scaramuzza and Fraunhofer [22]. The problem of estimating vehicle motion from visual input was first approached by Moravec [28] in the early 1980s. Moravec established the first motion-estimation pipeline, whose main functional blocks are still used today. However, the term visual odometry was first coined by Nister et al. in their landmark paper [29]. Their paper was the first to demonstrate a real-time long-run implementation with a robust outlier rejection scheme.

Kitt et al. presented a visual odometry algorithm based directly on the trifocal geometry between image triples [30]. Howard has presented a visual odometry system in which inliers are detected based on geometric constraints rather than on performing the more commonly used outlier rejection schemes [31]. However, those approaches follow the typical visual odometry scheme of matching or tracking features between consecutive frames and using these features for ego-motion estimation. The first to propose using the whole history of tracked features is the work by Badino et al. [32]. They do so by computing integrated features, which are the sample mean of all previous measured positions of each feature. Integrated features are then used in the

motion computation between the two consecutive frames. My approach differs in that I employ a transient local 3D map for tracking. It consists of a sparse set of 3D points (features) that are used for tracking and, therefore, motion estimation. This local map is internal to the system and is not an attempt to build a global map of the environment. As long as a feature is useful and can be utilized for motion estimation, it is kept alive in this local map.

Related to visual odometry is the simultaneous localization and mapping (SLAM) problem, where the goal is to build an accurate map of the environment while simultaneously localizing within this map. Visual SLAM systems employ sophisticated techniques to improve their accuracy, such as detecting loop-closures, where the system detects a previously visited location and uses this information to correct the map. Early work of bringing computer vision into SLAM was Davison's MonoSLAM [33], which was a filter-based method utilizing an Extended Kalman Filter (EKF). Filter-based methods were dominant until the development of the Parallel Tracking And Mapping (PTAM) system by Klein and Murray [11]. PTAM separated and parallelized the motion estimation and mapping tasks through a keyframe-based architecture while employing bundle adjustment [12]. An excellent introduction and survey of keyframe-based visual SLAM is by Younes et al. [34]. Unlike typical visual SLAM systems, by employing my local map approach my system is able to achieve high accuracy while maintaining low memory and computation requirements.

Additionally, Visual SLAM and odometry systems can be classified as either *direct* or *feature-based* methods. Feature-based methods aim to extract distinctive interest points and track them in subsequent frames to estimate camera ego-motion. In contrast, direct methods operate on the whole image directly. That is, the camera is localized by optimizing directly over image pixel intensities. Key representatives of feature-based visual SLAM systems are S-PTAM [35] and ORB-SLAM [13], whereas a representative of the direct visual SLAM systems is LSD-SLAM [36]. My system is based on point or corner-like features as they enable real-time performance while running completely on a CPU.

With the introduction of commodity depth sensors, RGB-D cameras have become popular in robotics. Huang et al. have introduced a visual odometry system based on RGB-D cameras called Fovis [37]. Their system extracts point features from the image, and then each feature's depth is extracted from the depth image. They follow an inlier detection approach similar to Howard's [31] and track features between consecutive frames. On the other hand, DVO by Kerl et al. [38]

is a dense visual odometry method that aims to exploit both the intensity and depth information of RGB-D cameras. Yet other approaches use depth information alone for ego-motion estimation. One common approach is by 3D point-cloud registration, which is commonly performed using an *iterative closest point* (ICP) algorithm [39] [40]. KinectFusion by Newcombe et al. [41] is one of the earliest and most well-known RGB-D SLAM systems. It fuses depth data into a volumetric dense model that is used for tracking camera motion. My system supports RGB-D data as well, where features are detected from the RGB image and depth information is extracted from the depth image. The rest of the system remains intact and performs the same operations regardless of the used sensor.

That was for classical model-based methods. On the other hand, initial efforts in data-driven visual odometry include the work of Ciarfuglia et al [42] in which they regressed visual odometry using *support vector machines* (SVMs). However, traditional machine learning approaches are limited when it comes to learning large and highly non-linear data such as RGB images encountered in this problem. Thus, only data-driven methods based on deep learning started achieving good results. For this, Wang et al. have developed DeepVO [43] which is based on a recurrent convolutional neural network (RCNN) architecture. Li et al. proposed UnDeepVO [44] which is an unsupervised learning approach. ESP-VO [45] is an end-to-end, sequence-to-sequence probabilistic visual odometry framework that infers poses along with uncertainties. However, as the authors state: “*Although the proposed DL-based VO method shows some potential in this area, we stress that it is not expected as a replacement to the classic geometry-based approaches. On the contrary, it would be a viable complement, i.e. incorporating geometry-based approaches with the representation, knowledge and models learned by the DNNs to further improve the VO systems in terms of accuracy and, more importantly, robustness.*” [45]. Hence, I resort to a classical geometric solution to visual odometry.

3.3 Method

A high-level overview of the visual odometry pipeline is shown in Figure 3-1 and the algorithm is presented in Algorithm 1. In the rest of the chapter, camera pose is understood to encompass both position and orientation, i.e., the complete 6 degrees-of-freedom transformation. The world reference coordinate frame is set at the pose of the first frame of the sequence.

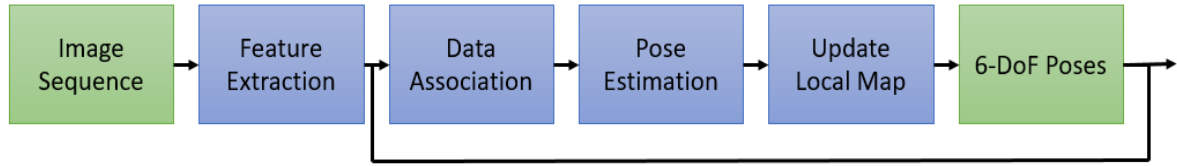


Figure 3-1 A high-level demonstration of the main loop of the visual odometry pipeline. Green boxes are inputs and outputs, whereas the blue ones are main processing stages.

Algorithm 1 Visual Odometry Algorithm Overview

```

for each frame (stereo or RGB-D)
  Extract features (AGAST, BRIEF)
  if first frame
    Initialize local map
    Set world reference coordinate frame
  else
    Predict new pose
    Track local map
    Estimate pose using tracked measurements
    Update staged map points
    Perform new triangulations if necessary
    Clear no longer trackable map points
  end if
end for
  
```

The details of the different components are illustrated in the subsections below.

3.3.1 Image Sequence

The first step is to feed the system a frame that is acquired either from a stereo or RGB-D sensor. In the stereo camera case, the retrieved stereo frame consists of the synchronized images from the left and the right cameras. The stereo frame is assumed to be stereo-rectified. Stereo rectification is the process of virtually transforming the stereo frame so that it appears as if the two cameras of the stereo rig have their image planes aligned to be coplanar. Consequently, epipolar lines are now parallel to the stereo baseline between the cameras. This reduces the stereo correspondence problem to a one-dimensional search, as matching features between the two cameras will lie on the same pixel row, assuming a horizontal stereo rig [8]. In the RGB-D sensor case, the RGB image is converted into a grayscale one, and then it is fed along with the depth image to the system.

3.3.2 Feature Extraction

As a feature-based method, the most important decision to be made is the type of feature to be used. The type of feature and its detection strategy will affect the overall performance of the system, as this feature will be used for finding correspondences between left and right images of the stereo camera for the purpose of triangulation of new 3D points. These features will also be used for tracking the local map and thus finally performing the motion estimation. In this algorithm, the chosen feature type is a point or corner-like features, as they are fast to compute and the availability of good corner detection methods.

In this algorithm, the Adaptive and Generic Accelerated Segment Test (AGAST) [46] corner detector is used. AGAST is based on the Accelerated Segment Test (AST) developed by Edward Rosten and Tom Drummond in their Features from Accelerated Segment Test (FAST) corner detector [47]. Upon its introduction, FAST outperformed conventional corner detectors in terms of both performance and repeatability, which made it a common choice for real-time tasks in computer vision. The basic steps of FAST corner detector can be summarized as:

1. Select a pixel p in the image which is to be tested if it is a corner or not. Let its intensity be I_p .
2. Choose an appropriate threshold value t .
3. Consider a circle of 16 pixels around the pixel under test.
4. The pixel p is a corner if there exists a set of n contiguous pixels in the 16-pixel circle, and all of them are either brighter than $I_p + t$, or darker than $I_p - t$.

In the accelerated segment test, n is chosen to be twelve because it enables a high-speed test to exclude a large number of non-corners: first examine only the four pixels at 1, 5, 9, 13 (four compass directions), if p is a corner then three of those pixels must satisfy the corner condition mentioned in item 4 above. Otherwise, it cannot be a corner. If this test passes, then we can proceed to test the full 16 pixels in the circle. Despite high performance, this approach still suffers from several weaknesses including that the high-speed test does not generalize well for n less than twelve, among others. The proposed solution to such weaknesses was to use a machine learning approach where the detector is trained from a selected set of images, preferably from the target application domain in order to compute a decision tree. The full details can be found in the original paper [47].

This approach still has some shortcomings⁹:

- If the camera is rotated, the pixel configuration of a corner may change significantly.
- Some corner configurations may be missing in the training set which leads to false positive and false negative responses of the corner detector.
- The decision tree has to be learned for every new environment from scratch.
- FAST builds a ternary decision tree for a binary target machine - a binary tree would be more efficient.
- ID3 (decision tree classifier) is used to build the decision tree, which is a greedy algorithm and, therefore, the result can be quite suboptimal.

AGAST proposes a technique to compute a binary decision tree (corner detector) which is generic and does not have to be adapted to new environments. By combining two trees, the corner detector adapts to the environment automatically. Hence, it results in a corner detector, which is faster and does not have to be trained while preserving the same corner response and repeatability as the complete FAST corner detector. Refer to the paper for full details [46].

One final important note about feature extraction, in this VO algorithm, no scale image pyramid of the target image is built. Corners are extracted and used from the full-size images directly.

After the corners are extracted, the next step is to compute a descriptor for the image patch around each corner. In this work, *Binary Robust Independent Elementary Features* (BRIEF) [48] descriptors are used. BRIEF is a binary descriptor where the descriptor vector is in the form of a binary string; in fact, BRIEF was the first binary descriptor developed. Binary descriptors offer many advantages over Histogram of Oriented Gradients (HOG) based patch descriptors such as SIFT [4] for constrained and real-time systems. The computations involved to arrive at these HOG descriptors are costly for many real-time and embedded systems. Furthermore, SIFT uses a 128-dimension vector for its descriptor, and it uses floating-point numbers, which result in a descriptor size of 512 bytes. Now, assuming hundreds or thousands of such feature descriptors, this will require a significant amount of memory which is potentially not feasible for constrained systems. Moreover, SIFT is patent-protected and thus cannot be freely used in applications.

⁹ <http://www6.in.tum.de/Main/ResearchAgast>

A binary descriptor, on the other hand, is very fast to compute. For example, the chosen BRIEF algorithm computes the binary descriptor string by following the basic steps outlined below:

1. Smooth the target image patch.
2. Select a set of location pairs in the image patch in a unique way. The original paper describes five different sampling strategies.
3. For each selected location pair, compare the pixel intensity at the first point with that of the second one. If the intensity is larger, then append 1 to the descriptor string; otherwise, append 0.

That is, the number of selected pairs will determine the length of the binary string. In this work, 256 pairs are used, resulting in a descriptor of size of 32 bytes. This, with the fact that descriptor computation has reduced to simple pixel intensity comparisons, offers a significant advantage over HOG descriptors both in terms of computation and memory requirement.

As for comparing two descriptors with each other, *hamming distance* is used. Hamming distance is defined as the number of positions at which the corresponding bits are different. Hamming distance is simple and fast to compute because it is just an Exclusive-OR (XOR) operation and a bit count. That is, $\text{sum}(\text{xor}(\text{descriptor1}, \text{descriptor2}))$. This operation is especially fast on modern CPUs where *Single Instruction Multiple Data* (SIMD) instruction sets are employed.

An important problem to consider is the distribution of the detected features across the image. Poor distribution where detected features are concentrated in one region of the image can lead to poor results. We follow a two-step process to overcome this problem. First, the image is divided into cells where features will be detected in each cell separately. Then, a technique known as adaptive non-maximal suppression, as described by Brown et al. [49], is performed in each cell. Adaptive non-maximal suppression aims to limit the maximum number of features extracted while at the same time ensuring good distribution across the image. Features are suppressed based on corner strength, and only ones that are local maxima in the neighborhood are retained. The implemented adaptive non-maximal suppression is presented in Algorithm 2.

Algorithm 2 Adaptive non-maximal suppression.

```
Adaptive-Non-Maximal-Suppression(Corners, NumToKeep)
  Sort-Descending(Corners) // Sort the corners in descending order based on corner response.
  radii.length = 0 // Initialize empty radii array
  for i = 1 to Corners.length
    response = Corners[i].response * ROBUST_COEFFICIENT
    radius =  $\infty$ 
    j = 1
    while j < i and Corners[j].response > response
      current_radius = L2-Distance(Corners[i], Corners[j])
      radius = Minimum(radius, current_radius)
    end while
    radii.append(radius)
  end for
  anms_points.length = 0
  decision_radius = Largest-Element-At(radii, NumToKeep) // NumToKeep is passed as index.
  for i = 1 to radii.length
    if radii[i]  $\geq$  decision_radius
      anms_points.append(Corners[i])
    end if
  end for
  return anms_points
```

3.3.3 Pose Prediction

Before proceeding to the next step, a prediction of the current camera pose is performed. For ground vehicles, dead reckoning using wheel odometry can be used to perform such prediction since it is usually available. In this work, we will follow a simple motion model where velocity is assumed constant between frames, and then computed velocities are averaged over frames. In this simple motion model, a constant frame rate is assumed, i.e., time is not considered in the calculations. The motion is calculated as follows: assuming the pose at frame k to be C_k , which consists of orientation represented as a quaternion, $q_k \in SO(3)$, and position $t_k \in \mathbb{R}^3$. C_k is the pose to be predicted. The linear velocity at frame k is computed as:

$$v_k = t_{k-1} - t_{k-2} \quad (3.1)$$

This linear velocity is then averaged over time which is similar to the midpoint method:

$$v_k = (v_k + v_{k-1})/2 \quad (3.2)$$

A similar approach is followed for the rotational component:

$$w_k = q_{k-1} * q_{k-2}^{-1} \quad (3.3)$$

and then the rotational velocity is also averaged over time using the spherical linear interpolation (SLERP) operation:

$$w_k = \text{slerp}(w_k, w_{k-1}, 0.5) \quad (3.4)$$

After computing the new velocities, the predicted pose is easily computed as follows:

$$t_k = t_{k-1} + v_k \quad (3.5)$$

$$q_k = q_{k-1} * w_k \quad (3.6)$$

The predicted pose is then used as a guide in the next local map-tracking step.

3.3.4 Local Map Tracking (Data Association)

The goal of this step is to correctly associate visible 3D map points with 2D image features. This 3D-2D data association is used by the following pose estimation step. Note that the associated 2D image features here are, in the stereo camera case, the ones extracted from the left camera image only. The 3D map points are projected onto the image plane of the left camera in the stereo camera case and of the RGB camera in the RGB-D camera case using the pinhole camera model. The neighborhood of each projected 3D map point is then examined for the best matching 2D feature detected in the feature extraction step. In the current implementation, the search neighborhood is set to a 25-pixel radius around each projected feature. If no enough matches are found, the search radius is doubled, and the tracking step is performed again. This process is illustrated in Figure 3-2, where the local 3D map is shown to the left and the detected 2D image features in the current frame are shown to the right.

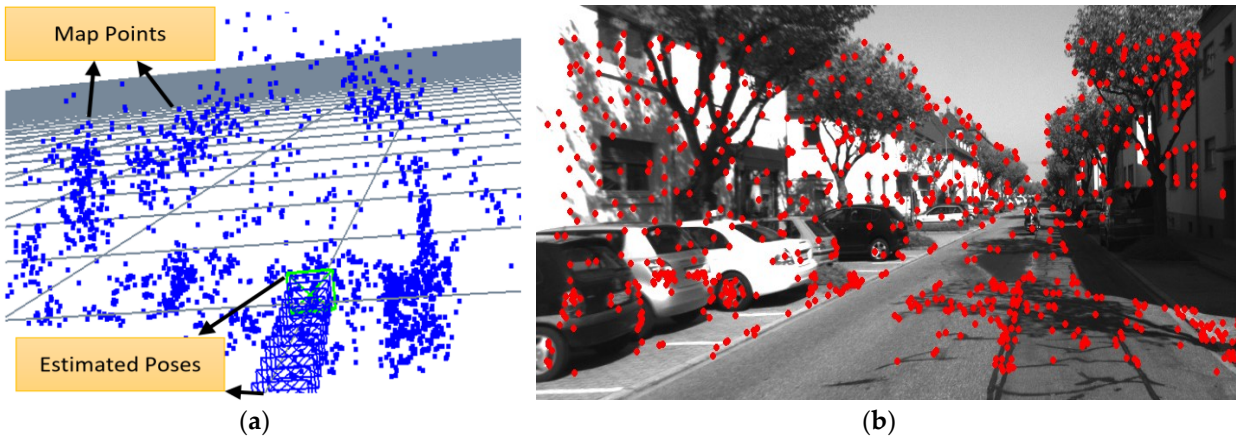


Figure 3-2 An illustration showing: (a) local 3D map, and (b) detected 2D image features. The goal is to find the correspondence between each 3D map point and detected 2D feature. Sample image from KITTI dataset. (Credit: [23]).

Neighborhood search is accelerated by means of *spatial hashing*. During the feature extraction step, the image plane is divided into a two-dimensional grid. Each cell of this grid is assigned the list of corners that happen to be within its boundaries. Now, the projected point's prospective grid cell is computed and thus the list of potential matches is readily available. Once candidate neighborhood features are identified, finding the best match of the projected point proceeds by using the widely accepted ratio test proposed by Lowe [4]. The ratio test works by comparing the distance of the closest neighbor to that of the second closest one. The nearest neighbor here is defined as the one with the minimum hamming distance for the BRIEF descriptor. If the nearest feature is much closer than the second nearest one, then it has a higher probability of being the correct match. The ratio test value is set to 0.80 in our implementation.

3.3.5 Pose Estimation

The found matches are then used for computing the camera pose. Camera pose consists of orientation $\mathbf{R} \in SO(3)$ and position $\mathbf{t} \in \mathbb{R}^3$. Finding the camera pose is formalized as an optimization problem to find the optimal \mathbf{R}, \mathbf{t} that minimizes the reprojection error between the matched 3D points and the image 2D features:

$$\{\mathbf{R}, \mathbf{t}\} = \operatorname{argmin}_{\mathbf{R}, \mathbf{t}} \sum_{i \in S} \rho \left(\|x^i - \pi(\mathbf{R}X^i + \mathbf{t})\|^2 \right) \quad (3.7)$$

where $x^i \in \mathbb{R}^2$ are image features, $X^i \in \mathbb{R}^3$ are world 3D points, for $i \in S$ the set of all matches. ρ is the Cauchy cost function. π is the projection function. This minimization problem is solved iteratively using the Levenberg–Marquardt algorithm. Levenberg–Marquardt algorithm is an optimization algorithm used to solve non-linear least-squares problems, it combines the Gauss-Newton and the gradient descent method. Furthermore, outliers are detected and excluded, and the optimization is run for a second time with the inlier set.

3.3.6 Local Map Maintenance

I maintain a secondary map of points that is referred to as staged points. These staged points are also tracked over time but are not used in motion estimation until they are found to be of high quality. After the camera pose is computed, staged map points are updated. When a new 3D-points triangulation occurs, these new points are initially stored in this staging area and are not added immediately to the local map. If a staged point is tracked successfully for a specified number of frames, then it is declared of good quality and added to the map. If it fails tracking,

then it is removed. However, if the number of map points drops below 1000 points, then staged points will be added immediately to the map in order to always maintain a minimum number of points in the map.

For the purpose of triggering a new points triangulation, the system will monitor the number of 2D-3D matches found in the current frame and in the previous two frames. If the number of matches is decreasing, a new triangulation is performed. New points are triangulated from features that were not tracked in the current frame. These newly triangulated points are added to the staging area as described previously. Additionally, map points that fail tracking for a specified number of frames are deleted from the map. Hence, the local map is kept fresh with good immediately useful points from the staging area, while no-longer-trackable points are removed.

When the system initially starts, the local map is empty, so the first frame is used to triangulate the initial set of 3D points, which are added immediately to the local map. This first frame also sets the world reference coordinate frame. All reported poses will be with respect to this world coordinate frame. In the stereo case, triangulations are performed using the Linear-LS method described by Hartley and Sturm [9]. As the stereo frame is rectified, matching corners between the left and right images is greatly simplified, as the search is restricted to the same row in both images. In the RGB-D case, triangulations are performed by extracting depth values directly from the depth image.

3.3.7 General Implementation Remarks

The algorithm is implemented using the C++ programming language. OpenCV library was used to perform image processing tasks. Corner detector and descriptor extractor implementations available as part of OpenCV library were used. The general graph optimization (g2o) library [50] was used to perform the Levenberg–Marquardt minimization in the pose estimation step.

The algorithm runs purely on the CPU, with no GPU acceleration used. Moreover, the algorithm runs primarily in one thread. However, a parallel thread is spawned to perform the features computation step on the right image in the stereo cameras case while the main thread is busy performing it on the left one.

3.4 Evaluation

In this subsection, results of the evaluation of the visual odometry system on three challenging publicly available datasets, namely, the KITTI dataset [51], the EuRoC MAV dataset [52], and the TUM RGB-D dataset [53] are presented. Each dataset has its unique characteristics, which will enable a comprehensive evaluation of the visual odometry system.

3.4.1 KITTI Dataset

The KITTI dataset is widely used for evaluating autonomous driving algorithms. The dataset was collected by driving in different traffic scenarios in the city of Karlsruhe, Germany. Some of the challenging aspects of the dataset are the presence of dynamic moving objects (vehicles, cyclists and pedestrians), the different lighting and shadow conditions as the vehicle is moving and the presence of foliage, which results in the detection of many non-stable and challenging-to-track corners.

The presented visual odometry system is also evaluated against two open-source systems, S-PTAM [35] and LIBVISO2 [54]. S-PTAM is a state-of-the-art full V-SLAM system. S-PTAM was compiled without loop-closing capability but all other operations remain intact. LIBVISO2 is a famous stereo visual odometry system that tracks features and estimates motion between consecutive frames only. With S-PTAM being a V-SLAM system and LIBVISO2 a frame-to-frame visual odometry system, they provide a good comparison ground for our presented system, which aims to reach V-SLAM systems accuracy while being lightweight like typical visual odometry systems.

Accuracy

For evaluating the accuracy, two error metrics will be reported. First is the average translation error E_t over all subsequences of length (100, ..., 800) meters as defined in the KITTI dataset paper [51]. We define the other metric ξ as:

$$\xi = RMSE(\mathbf{T}_{1:n}) = \left(\frac{1}{n} \sum_{i=1}^n \mathbf{T}_i^2 \right)^{1/2} \quad (3.8)$$

where T_i is the magnitude of the Euclidean distance along the horizontal plane between the estimated and ground truth pose at frame i .

The computed error metrics for KITTI sequences (00-10) except for sequence 01 are shown in Table 3-1. Sequence 01 is a challenging highway with unreliable far features. Although our

algorithm does not lose tracking, it drifts badly and fails to provide meaningful estimates. Hence, it was excluded. From the presented results, it can be seen how our proposed visual odometry algorithm comes very close to S-PTAM, which is a complete V-SLAM system and surpasses LIBVISO2. Plots of the estimated path against ground truth are shown in Figure 3-3.

Table 3-1 Visual odometry results on KITTI dataset compared to two other methods.

Sequence	Length [km]	LVT		S-PTAM		LIBVISO2	
		E_t [%]	ξ [m]	E_t [%]	ξ [m]	E_t [%]	ξ [m]
00	3.7223	1.25	11.01	0.84	8.81	2.74	47.03
02	5.0605	1.33	13.59	0.96	22.20	2.20	69.52
03	0.5590	1.04	2.48	1.14	3.43	2.27	4.66
04	0.3936	0.56	0.78	1.29	2.72	1.08	2.67
05	2.2046	0.89	4.27	0.91	3.01	2.26	19.80
06	1.2326	1.04	2.11	1.28	3.34	1.28	4.20
07	0.6944	0.98	3.17	0.88	2.97	2.34	5.74
08	3.2137	1.25	6.57	1.05	6.69	2.83	44.52
09	1.7025	1.76	9.23	1.21	9.28	2.84	24.32
10	0.9178	1.02	3.81	0.64	3.61	1.39	2.97
Total	19.701	1.23	9.05	0.97	11.73	2.45	43.98

Additionally, I submitted the system for evaluation on the KITTI sequences (11–21) which are test sequences that have no publicly available ground truth. The system achieved an average translation error of 5.8%. While running the analysis, I suspect the main source of error is sequence 21 which is a highway and suffers from the same issue as sequence 01 before. The problem with highway scenes is that features are far away relative to the stereo baseline causing the stereo cameras to degenerate into a monocular one. This results in the loss of scale information. Moreover, the KITTI evaluation server provides the path plots of a portion of the test sequences which are shown in Figure 3-3. From these path plots, we can see that our system is able to provide accurate estimates on the test sequences comparable to its estimates on the training ones. This means with the exclusion of the highway sequence 21, I expect our system to attain a comparable average translation error to the one achieved on the training sequences which is 1.23%. Properly handling the problematic far away features is a task for future work.

An important observation is that through our innovative approach of keeping features alive in a local 3D map and tracking them for as long as possible, we were able to greatly improve estimation accuracy compared to LIBVISO2. LIBVISO2 follows the traditional visual odometry approach of tracking features between consecutive frames, i.e., from frame to frame only. We will define feature age as the number of frames in which this feature was successfully tracked

and used in pose estimation. The average feature age in each frame for KITTI sequence 00 is shown in Figure 3-4.

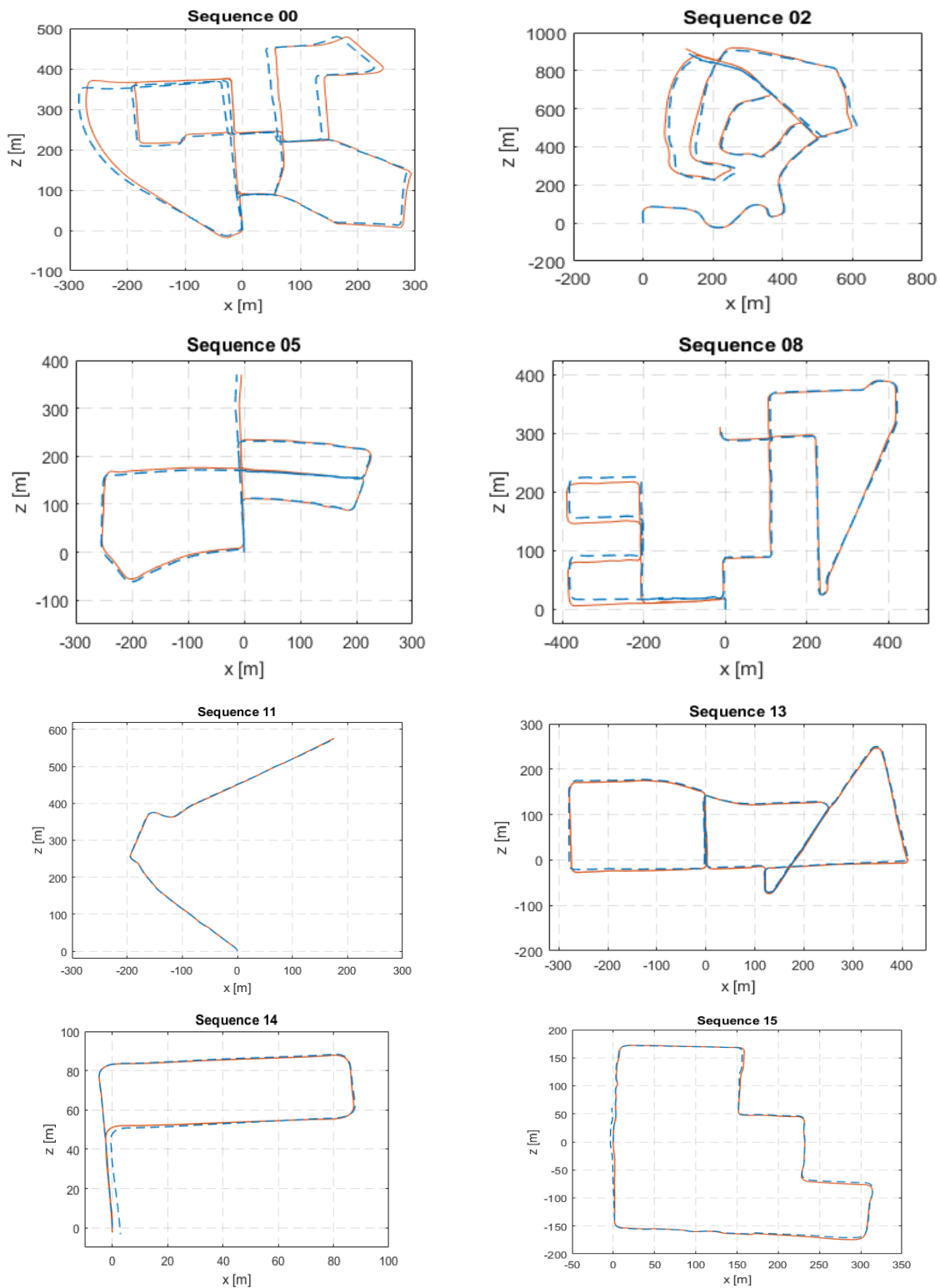


Figure 3-3 Estimated trajectory (dashed blue) from LVT against ground truth (solid red) in KITTI dataset. (Credit [23]).

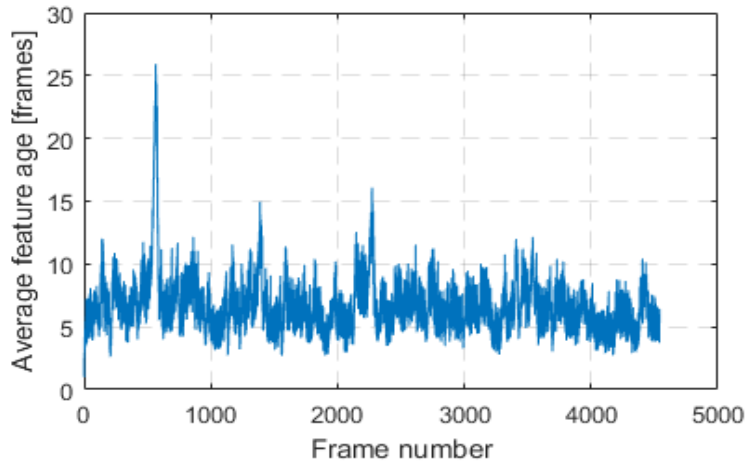


Figure 3-4 Average feature age in each frame of KITTI sequence 00.

Runtime Performance

Runtime performance evaluation experiments were performed on a laptop computer running the Ubuntu 16.04 operating system, with an Intel i7-7700HQ CPU and 16GB of memory (RAM). For evaluating the runtime speed, we have timed the processing time of each frame from KITTI sequence 00, excluding the portion where the system is retrieving the stereo image and with no visualization enabled. As for evaluating the memory requirement, memory is read from the System Monitor utility just before the last frame. This process was repeated five times to account for operating system loading variability, and the results are listed in Table 3-2.

Table 3-2 Runtime performance measurements.

	Mean \pm Std[ms]	Memory
LVT	13.82 ± 4.4	22.12 MiB
S-PTAM	36.7 ± 23.8	1.5 GiB
LIBVISO2	23.2 ± 4.5	41.82 MiB

I have timed the four main stages of the visual odometry system, and the result is shown in Figure 3-5.

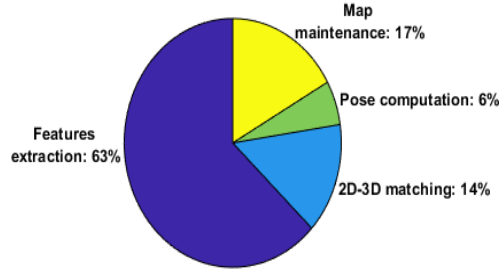


Figure 3-5 Relative runtime speed of the main stages of the visual odometry system. (Credit [23]).

The suitability of the presented visual odometry system for constrained real-time applications is evaluated on two embedded Linux single-board computers, namely, the Raspberry Pi 3 [55] and the ODROID XU4 [56]. The time to process each frame of KITTI sequence 00 is recorded and computational performance is listed in Table 3-3.

Table 3-3 Computational performance on embedded computers.

	Mean \pm Std[ms]
Raspberry Pi 3	162.37 \pm 42.38
ODROID XU4	87.96 \pm 20.35

3.4.2 EuRoC Dataset

The EuRoC datasets [52] were collected on-board a micro-aerial vehicle (MAV). Stereo images were collected at a rate of 20 Hz with a stereo camera that provides monochrome WVGA images. The different datasets vary in their level of difficulty based on the flight dynamics and illumination conditions. The fact that AGAST corners and BRIEF descriptors used in LVT are by their design not invariant to in-plane rotations will provide useful insights, given that the MAV is moving in all 6 degrees of freedom. We will use the five sequences collected in an industrial machine hall for evaluation. The VICON room sequences constitute primarily of white plain surfaces and there is not enough texture in the scene for the feature detector to detect corners, thus they were not used in the evaluation. A sample image from the first machine hall dataset showing the detected features is shown in Figure 3-6.



Figure 3-6 A sample image from Machine Hall 01 of the EuRoC datasets, where detected features are plotted (red dots). (Credit [23]).

Accuracy

For evaluating the accuracy, two error metrics, absolute trajectory error (ATE) and relative pose error (RPE), as defined by Sturm et al. [53], are used. The ATE is well suited for evaluating Visual SLAM systems, as it measures the global consistency of the estimated trajectory. On the other hand, the RPE is more suited for evaluating visual odometry systems, as it measures the local accuracy of a trajectory over a fixed time interval. That is, it measures the drift in a trajectory. In our evaluation, as the stereo images are recorded at a rate of 20 Hz, we will set the fixed time interval for RPE to be 0.05 s. Therefore, RPE will correspond to drift in units of meters per second. We have also run LIBVISO2 [54] on the same dataset. The results are reported in Table 3-4. Both visual odometry systems are achieving similar drift rates.

Table 3-4 RMSE values of absolute trajectory error (ATE) and relative pose error (RPE) on EuRoC datasets.

Sequence	LVT		LIBVISO2	
	ATE [m]	RPE [m/s]	ATE [m]	RPE [m/s]
MH_01_easy	0.232	0.028	0.234	0.028
MH_02_easy	0.129	0.028	0.284	0.028
MH_03_medium	1.347	0.070	0.86	0.069
MH_04_difficult	1.635	0.069	1.151	0.068
MH_05_difficult	1.768	0.060	0.818	0.060

Effect of Rotational-Invariant Features

AGAST corners and BRIEF descriptors used in our presented visual odometry system do not provide rotational invariance capability. When evaluated previously on the KITTI dataset, this did not pose a problem, as the vehicle is moving on a locally planar ground. However, in the

EuRoC datasets, the MAV is flying in all 6 degrees of freedom. In order to evaluate said effect, I have replaced the default AGAST/BRIEF with oriented fast and rotated brief (ORB) [57]. ORB provides rotational-invariant features. With everything else remaining fixed, I have re-run the evaluation and the results are reported in Table 3-5.

Table 3-5 RMSE values of ATE and RPE on EuRoC datasets using rotational-invariant oriented fast and rotated brief (ORB) features.

Sequence	ATE [m]	RPE [m/s]
MH_01_easy	0.292	0.029
MH_02_easy	x	x
MH_03_medium	1.328	0.071
MH_04_difficult	2.194	0.070
MH_05_difficult	1.515	0.061

From Table 3-5, with the same original parameters, LVT now fails to complete the MH_02_easy dataset. The cause is that the system fails to track enough ORB features in a motion-blurred frame. Drift rates are almost the same and no clear improvement in accuracy is attained by the new feature detector alone. I found from this experiment that simply replacing the feature detector with a rotationally-invariant one did not result in any major improvement in the results as was expected.

3.4.3 TUM RGB-D Dataset

The Technical University of Munich (TUM) RGB-D dataset [53] is a large dataset collected indoors using an RGB-D sensor under different illumination, texture and movement scenarios. The data was collected at a 30 Hz frame rate and a sensor resolution of 640×480 . The evaluation results on a subset that is suited for V-SLAM and visual odometry evaluation are reported in Table 3-6 along with evaluation against Fovis. Although my visual odometry system was initially designed for stereo cameras, it was, surprisingly, able to achieve low drift rates on RGB-D data. The main difference is that I found it necessary to trigger a new triangulation operation at every frame, unlike the default behavior as described in map maintenance before. We can see from Table 3-6 that my visual odometry system is able to achieve similar drift rates to Fovis which was originally designed for RGB-D cameras. However, a fast rotation while the camera faces a low-textured wall during the movement results in a large error in fr1_room for my system.

Table 3-6 RMSE values of ATE and RPE on TUM RGB-D dataset.

Sequence	LVT		Fovis	
	ATE [m]	RPE [m/s]	ATE [m]	RPE [m/s]
fr1_desk	0.109	0.010	0.259	0.009
fr1_desk2	0.116	0.012	0.125	0.009
fr1_room	4.138	0.077	0.184	0.007
fr1_xyz	0.035	0.006	0.051	0.006
fr2_desk	0.080	0.003	0.103	0.003
fr2_xyz	0.014	0.002	0.013	0.002
fr3_office	0.132	0.006	0.188	0.005

3.5 A Study on Visual Odometry Performance Under Low Illumination

Visual SLAM and visual odometry systems in the literature are typically evaluated at daytime under typical lighting conditions. This is mainly driven by the fact that common datasets and benchmarks as shown in the previous section are collected under such good lighting conditions. However, autonomous mobile systems such as self-driving cars should be expected to work at night and under low-light conditions as well. Utilizing active sensors and other methods (such as LIDARs) is necessary to provide functionality under these conditions. However, expanding the operation range of passive cameras through image enhancement approaches is compelling to maintain sensor redundancy under these conditions, which is a factor to functional safety. Prior work focusing on achieving operation of feature-based vision algorithms at night is limited, which motivated this work [24].

LIDARs are active sensors and can be used to perform the localization tasks at night, as in the work by Dong and Barfoot [58]. Maintaining a suite of sensors consisting of heterogeneous sensing modes is advantageous, as it protects against failure modes of a particular sensor. For example, being active sensors, LIDARs are expected to interfere with each other when they are in proximity, while passive vision systems do not have this problem. Moreover, LIDARs are more expensive than cameras. Brunner et al [59] combined visual and infrared imaging in a visual SLAM system to solve the localization problem under day and night conditions. Nelson et al [60] have presented a solution for localization at night by localizing with respect to artificial light sources that are expected to be present at night in urban environments. MacTavish et al [61] have developed a visual odometry system and relied on mounted headlights on the mobile robot platform as the main illumination source.

In this section, I investigate potential image preprocessing techniques to enhance night images for vision-based robotic perception. I introduce four different image preprocessing techniques and study their performance on three visual motion estimation systems:

- ORB-SLAM2 [62], which is a state-of-the-art visual SLAM system.
- LIBVISO2 [54], which is a well-known visual odometry system.
- LVT [23], my previously developed visual odometry system.

Low-light image enhancement affects feature-based algorithms at two main steps. First is feature detection. Feature or interest point detection depends on contrast/gradients in the image, and a better lit image provides more distinctive features. Second is matching detected interest points between images. Matching interest points can be performed by comparing the pixel intensities of the image patch around each interest point directly using some similarity measure. A descriptor that acts as a signature for that image patch can also be computed and different descriptors can be compared. In either case, image enhancement, in this case, can provide more unique descriptors and result in more reliable matching. Such algorithms can be operating normally with no preprocessing under typical lighting conditions. Once the mean scene luminance drops below a certain threshold, our preprocessing is activated to enhance the incoming low-light images.

3.5.1 Studied Image Preprocessing Techniques

In this section, I describe the four image preprocessing techniques, namely Gamma Correction, $L^*a^*b^*$ + CLAHE, RG + CLAHE, and Bioinspired Retina Model. CLAHE stands for *contrast limited adaptive histogram equalization* and will be explained below.

Gamma Correction

Gamma correction refers to the process of adjusting the luminance of an input image using a non-linear mapping of the form:

$$image_{output} = image_{input}^{1/\gamma} \quad (3.9)$$

where γ is the gamma value, and the input image pixel intensities must be in the range [0,1]. In my evaluation, a gamma value of 1.5 has been used.

Here we apply the same mapping to each pixel. Research has been done to develop more advanced adaptive enhancement techniques. For example, Huang et al. [63] have demonstrated an automatic transformation technique through gamma correction and probability distribution of

luminance values. Rahman et al. [64] demonstrate a technique where they classify the image content first, then apply adaptive gamma correction based on this image classification information. As we can see, there are many proposed methods, each with its own characteristics. All those methods will add to the complexity of the image enhancement step. Hence, I decided to use the base gamma correction which is fast and easy to implement.

L*a*b* + CLAHE

In this technique, I will convert the image from the traditional RGB color space into the L*a*b* one. This color space consists of the following three components, L* which represents lightness or intensity; a* and b* represents the color dimensions. We will extract the lightness channel, then distribute its intensity values as uniformly as possible. This distribution is performed using *contrast limited adaptive histogram equalization* (CLAHE) technique. In adaptive histogram equalization, multiple histograms, each corresponding to a different section of the image, are computed and then used to redistribute the intensity values across the image. However, this overamplifies the noise in relatively homogeneous regions of the image. CLAHE circumvents this problem by limiting the amplification. After the CLAHE is applied on the lightness channel, the L*a*b* image is converted back into the RGB color space then grayscale.

RG + CLAHE

In this technique, we will remove the blue channel from the RGB image and convert it to grayscale by averaging the remaining red and green channels. Blue light is not strong in artificial lights used in vehicle headlamps and streetlights, unlike during daytime under sunlight. We observed the blue channel to be very noisy, as have others in prior work [65] [66]. Excluding the blue channel results in a less noisy image. After that, the same limited adaptive histogram equalization (CLAHE) technique described in the L*a*b* color space technique section is applied to the resulting grayscale image.

Bioinspired Retina Model

Building models to simulate parts of the human visual system offers a valuable and attractive solution to several computer vision tasks. In this paper, we will use the retina model by Benoit et al. [67] which is available in the open-source OpenCV library. This retina model separates spatial and temporal information into two output channels, the parvocellular pathway (parvo) which is related to details extraction; and the magnocellular pathway (magno) which is related to detecting motion and events.

This retina model is capable of luminance and detail enhancement. It performs local logarithmic luminance compression which allows both very bright and very dark areas to be visible. It also performs spectral whitening which attenuates the mean luminance energy and enhances mid-frequencies, which correspond to details. We will be using the output of the parvo channel. The parameter values used as input to the retinal model implemented in OpenCV are listed in Table 3-7. Those values were set heuristically to achieve a smooth image.

Table 3-7 Parameter values used for the parvo channel in the retina model.

Parameter	Value
Photoreceptors Local Adaptation Sensitivity	0.29
Photoreceptors Temporal Constant	0.89
Photoreceptors Spatial Constant	0.53
Horizontal Cells Gain	0.3
Hcells Temporal Constant	0.5
Hcells Spatial Constant	4.0
Ganglion Cells Sensitivity	0.29

The result of applying the aforementioned preprocessing techniques on a sample test night image is shown in Figure 3-7.

3.5.2 Evaluation

For evaluation, a dataset has been collected by driving around the University of Michigan-Dearborn campus at night. The sequence is sequence 28 of our own previously collected adverse weather dataset¹⁰. The traveled loop is shown in Figure 3-8. This dataset was collected on November 17th, 2017; and has an approximate length of 1.49 miles. The dataset was collected by a stereo camera at a resolution of 1280×720 and at a rate of 20Hz.

In this evaluation, I will use sequence 3 of the previously mentioned dataset besides sequence 28. Sequence 3 was collected by traveling the same loop as sequence 28 but in daytime. This daytime dataset should be useful for comparison. The result estimated paths from the three evaluated systems, on the daytime, raw night, and preprocessed night sequence with the described techniques, are plotted on maps and shown in Table 3-8.

¹⁰ <http://sar-lab.net/adverse-weather-dataset/>



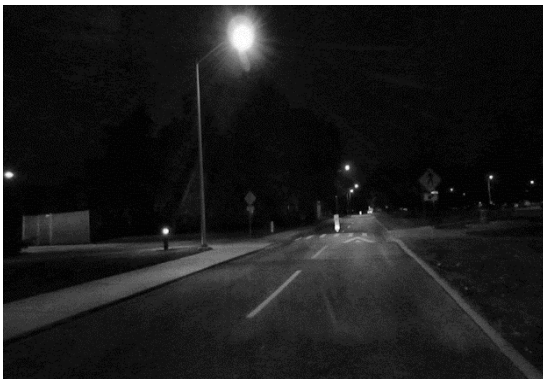
(a)



(b)



(c)



(d)



(e)

Figure 3-7 Results of the four image preprocessing techniques (a) Raw test image without any preprocessing. (b) Gamma correction (c) $L^*a^*b^*$ + CLAHE (d) RG + CLAHE (e) Bioinspired retina model. (Credit [24]).

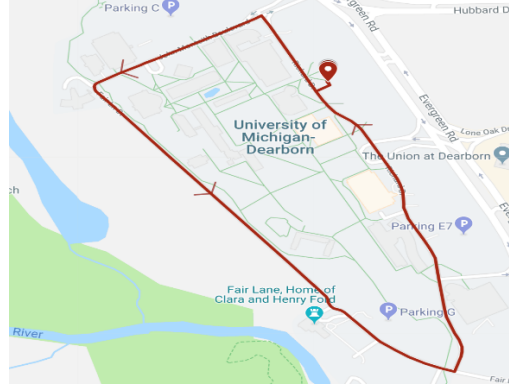
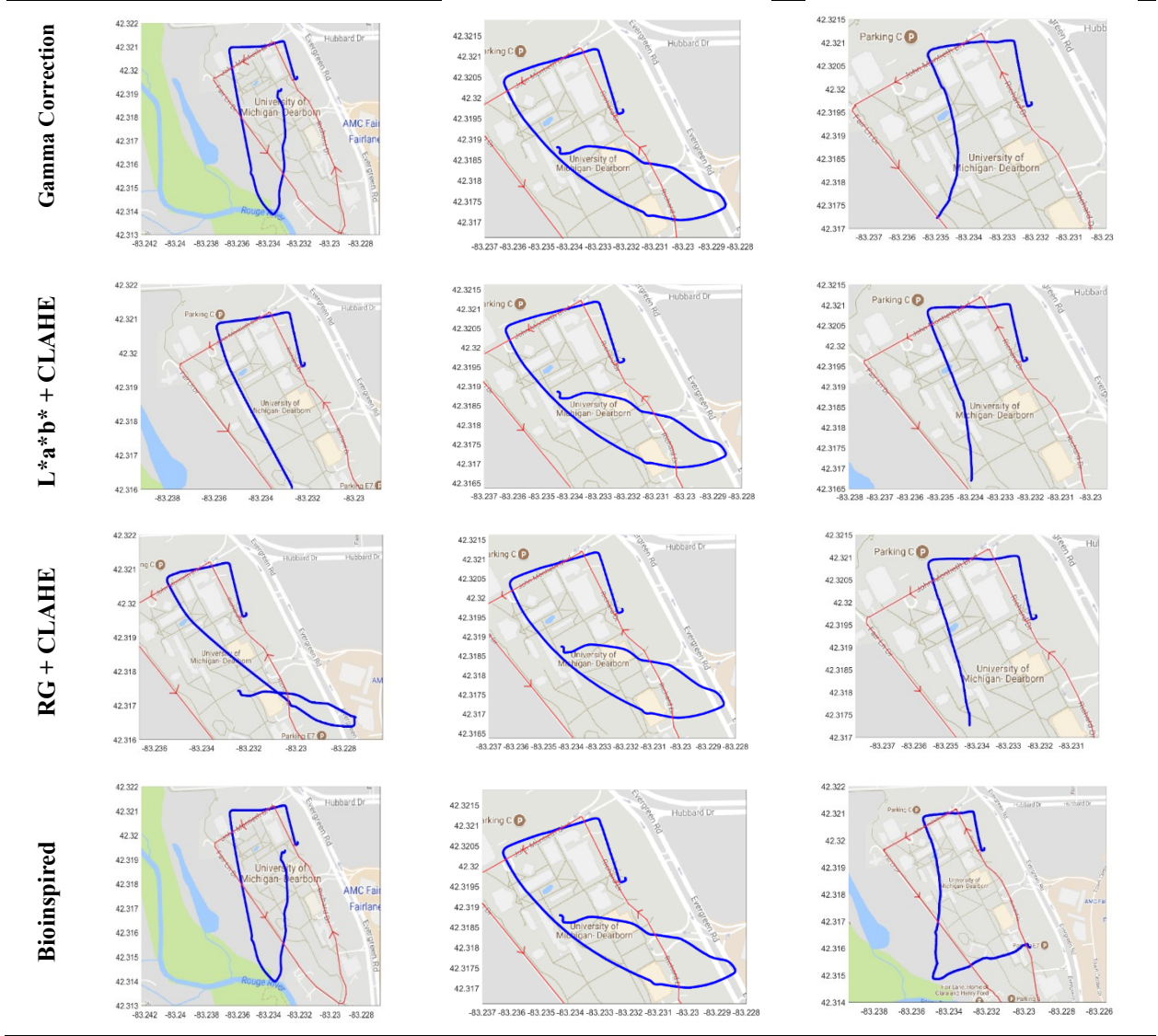


Figure 3-8 The loop traversed while collected the dataset for evaluation. (Attribution: Google Maps).

From Table 3-8, several observations can be made. First, all three systems succeed in completing the morning dataset. In the raw night dataset, only LIBVISO2 was able to keep track to the end. Gamma correction helped ORB-SLAM2 to complete the dataset and give good estimate of the traveled path. The Lab + CLAHE technique failed to help any of the systems in processing the dataset or improving the results. The RG + CLAHE technique helped ORB-SLAM2 to complete the dataset to the end without losing tracking but did not assist in improving the quality of trajectory estimation. Finally, the bioinspired retina method was the only one to help all the three systems to complete the dataset to the end without losing tracking, and it resulted in the best trajectory estimate for all three systems.

Table 3-8 Result of the three systems on sequence 3 (Daytime - unprocessed) and the night sequence 28 raw unprocessed and processed with presented techniques. Estimated paths are in blue while driven paths are in red. (Attribution: Google Maps; Credit [24])

	ORB-SLAM2	LIBVISO2	LVT
Daytime			
Night Raw (unprocessed)			



More insights can be obtained by extracting some statistics from my visual odometry system (LVT) running on all the preprocessed night datasets. I will extract three statistics:

1. *Map size*, this represents the number of 3D points that have been triangulated and currently reside in the local map.
2. *2D-3D Associations*, this is the number of 3D map points that was successfully associated with 2D image features in the local map tracking step, and thus was used for pose estimation.
3. *Inliers Count*, this is the number of found inliers after outlier rejection which is performed as part of pose estimation.

A plot of these statistics for a segment of 200 frames is shown in Figure 3-9. A quantitative presentation of these values is reported in Table 3-9 where the mean and standard deviation

values are computed over the number of successfully tracked frames. Table 3-9 also reports the number of tracked frames which shows how long the visual odometry was able to keep track before failing, mostly likely due to tracking too few points in the scene. From Figure 3-9 and Table 3-9, we note that daytime results are the best as expected. All preprocessing methods provide a substantial improvement over the raw unprocessed night sequence. Bioinspired technique has the best values generally. It is also interesting to observe that Lab + CLAHE technique gives high values for map size and data associations but low inlier count, which means that most of the associations are bad. Gamma correction method appears to give the second-best results after the bioinspired.

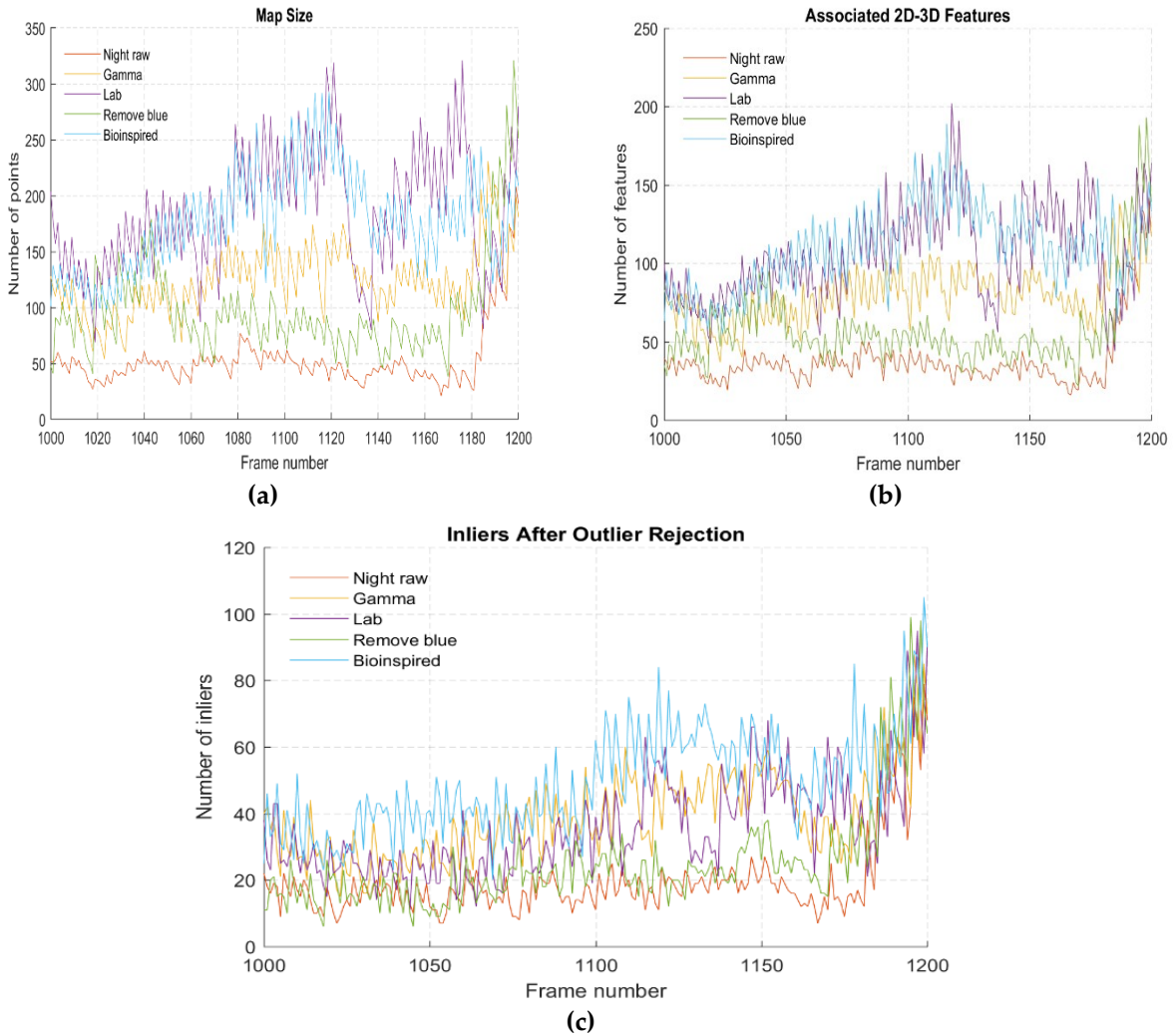


Figure 3-9 Status information from our visual odometry system on different datasets: (a) Number of points in the local map; (b) Number of matched 3D map points with 2D image features to be used in pose estimation; (c) Number of found inliers after pose estimation. (Credit [24]).

Table 3-9 Mean \pm standard deviation values of the extracted metric values from our visual odometry system for the different datasets, along with the number of successfully tracked frames.

	Map Size	2D-3D Associations	Inliers Count	Tracked Frames
Daytime	1103.3 \pm 249.3	649.8 \pm 159.8	244.0 \pm 100.0	8750/8750
Night raw	123.8 \pm 68.3	83.6 \pm 50.5	39.5 \pm 33.1	4042/8300
Gamma correction	200.9 \pm 77.0	123.6 \pm 52.6	52.5 \pm 33.5	4404/8300
L*a*b* + CLAHE	224.4 \pm 93.5	127.8 \pm 59.1	46.6 \pm 34.2	4577/8300
RG + CLAHE	208.2 \pm 99.6	123.5 \pm 66.3	49.7 \pm 37.7	4283/8300
Bioinspired	218.3 \pm 73.7	129.4 \pm 47.9	54.0 \pm 30.6	8300/8300

Runtime Performance

For runtime performance evaluation, I have measured the time to perform the preprocessing computation on each frame. The evaluation was performed on a laptop computer running the Ubuntu 16.04 operating system with an Intel i7-7700HQ CPU. It should be mentioned that the bioinspired model that is available in the OpenCV library which we used, has GPU acceleration support; however, I opted to use the CPU implementation instead so that all methods are evaluated on the CPU. The results are reported in Table 3-10. Gamma correction technique is the fastest because the mapping performed by Equation 3.9 described before, can be precomputed in a lookup table to be used at runtime.

Table 3-10 Computational performance of the four described preprocessing techniques.

	Mean \pm Std [ms]
Gamma	5.71 \pm 0.59
L*a*b*	39.37 \pm 3.23
Remove blue	15.61 \pm 2.56
Bioinspired	118.58 \pm 13.64

3.6 Summary

In this chapter, I have presented my feature-based visual odometry system called LVT, which is compatible with both stereo and RGB-D sensors. Its innovative usage of a transient local map enables it to approach the estimation accuracies common to full V-SLAM systems. The algorithm is designed for real-time operation with low computational overhead and memory requirements. The system was evaluated on KITTI autonomous driving datasets and compared with state-of-the-art V-SLAM and VO systems. Furthermore, it was also evaluated on the EuRoC MAV industrial machine hall and TUM RGB-D datasets as other use cases. This work

accompanies the release of the source code of the system under a permissive license and with support for the Robot Operating System (ROS)¹¹.

Loss of scale for highway scenes in my algorithm occurs because 3D point placement is based entirely on stereo disparity. Increasing the stereo camera baseline and/or resolution could provide better depth measurement and address this challenge. Another solution would be to use other sensor modalities to estimate scale, such as integrating inertial sensors, or using other odometry techniques based on wheel odometers or LIDAR.

Additionally, I have presented four different image preprocessing techniques with the goal of enhancing night and very low-light images. Such enhancement expands the operation range of passive cameras and is compelling in order to maintain sensor redundancy under these challenging low-light conditions. The presented image enhancements techniques are applicable to a wide variety of computer vision tasks such as image stitching, registration, and recognition. However, I have focused on assisting feature-based visual odometry and visual SLAM systems in operating in such challenging night conditions.

It can be observed that using a retina model inspired from human visual system resulted in best enhancement for visual odometry and visual SLAM systems. However, this comes at the cost of higher computational requirements. Depending on the application, this might be a bottleneck. That said, the authors of the Retina model provide a GPU implementation of their model with the OpenCV library, but I have used the CPU for fair comparison with the other techniques. Gamma correction results came in second after the bioinspired method, but it beats it in terms of computational performance significantly. In fact, gamma correction is the fastest method as it can be reduced to a simple table lookup operation at runtime.

¹¹ <https://github.com/SAR-Research-Lab/lvt>

Chapter 4

A Combined Visual Odometry and Multi-Object Detection and Tracking System

Detection and tracking of different moving objects in the surrounding environment is an essential capability for any robotic perception system. This enables the forecasting of objects trajectories and safe navigation of a mobile robot, such as a self-driving car, toward its goal. In this chapter, I demonstrate a multi-object tracker from image data in a tracking-by-detection paradigm. The tracking algorithm imposes no prior requirements on the tracked objects. This work demonstrates how with a simple and intuitive cost function, competitive and real-time performance is achieved. Furthermore, the multi-object tracker is combined with my previously developed stereo visual odometry system to obtain a more complete knowledge of the environment. Beside 2D object detections and tracking in the image domain, this provides robot motion estimates and 3D object trajectories in the world frame. Evaluation and discussion of the system are presented along with a study on the effect of removing features lying on tracked objects on visual odometry performance. This chapter is based on my prior published work [68].

4.1 Introduction

Autonomous mobile robots must perceive enough information of the surrounding environment to be able to navigate and operate safely toward their end goal. In structured or controlled environments, such as material delivery robots in warehouses, robot perception problem is easier and more tractable. This is true as well for situations where the observer is stationary such as in surveillance applications. However, for successful widespread deployment of self-driving cars and autonomous delivery robots in urban environments, robust perception algorithms must be developed. These algorithms must be able to handle the complexity and uncertainty in urban environments while the observer is moving using only the onboard sensory. Moreover, with other tasks such as localization, sensor fusion, trajectory planning and control [69], these algorithms must be computationally efficient.

Dynamic *Multi-Object Tracking* (MOT) refers to the ability of a robot perception system to identify moving or potentially moving objects in the surrounding environment and tracking their 3D motion trajectories in the world frame. This is a necessary requirement for safe navigation and obstacle avoidance. LIDAR sensor has been used successfully for this task of detection and tracking [70] [71]. LIDARs can provide precise measurements of the environment compared to other depth sensors. However, cameras are the focus AI sensor under study in this dissertation and my focus is on vision-related techniques.

Recent unprecedented success and increased accuracy of object detectors from image data based on convolutional neural networks (CNNs), [5] [72] [73], has enabled the *tracking-by-detection* approach. This approach is advantageous as it can easily identify new objects of interest in the scene, it can handle video captured by a moving robot, and it is robust to target drift [74]. The task of the tracker in this approach is to associate detections from the object detector across frames (*data association*). The output of this data association is multiple objects with their estimated trajectories across frames (i.e., through time). This association task is challenging and can be solved either *globally* or in an *online* manner.

In *global association*, the tracker has access to all frames and detections and thus can find optimal results. However, their utility is limited to situations where only analysis of pre-collected video sequences is required. On the other hand, *online association* acts only on data available till current frame and have no knowledge of future frames. For a tracking system that will be running in real-time on-board a mobile robot will thus need to run online. In order to achieve higher accuracy, some online methods will utilize batches of previous frames in their optimization routines. The more frames and complex these routines are, the more computational resources are required. A balance is required between runtime and tracking performance.

In this chapter, an online multi-object tracker with novel cost function in a tracking-by-detection framework, which enables real-time operation of the algorithm, is presented. This work will demonstrate how such an intuitive and simple tracking algorithm to implement with a modern deep learning-based object detector, competitive results on the level of more complicated state-of-the-art approaches could be easily achieved.

The presented tracker is generic and imposes no specific requirements on the targets of tracking. Additionally, as the tracker operates in the 2D image domain, it will be combined with the previously developed stereo visual odometry system in Chapter 3. This, coupled with

disparity maps computed from the stereo camera, will enable my system to estimate the tracking targets' trajectories in 3D world coordinates. As visual odometry is already providing camera pose (position and orientation) at each frame, the developed solution is indeed an indispensable building-block for safe navigation of autonomous mobile robots. A sample output from the system is shown in Figure 4-1.



Figure 4-1 Demonstration of the system output. [top] is estimated poses from visual odometry and 3D tracked objects in world coordinates. [bottom] is detected and tracked objects in the 2D image domain ([68] © 2019 IEEE).

4.2 Literature Survey

The majority of recent research on multi-object tracking from video data employs the tracking-by-detection strategy. Global or batch methods, [75] [76] [77] [78], assume knowledge of all frames and detections. They perform different optimization algorithms and result in more accurate results, but they have limited utility for autonomous mobile robotics where the multi-object tracker needs to operate in an online fashion.

As for online multi-object trackers, Gündüz and Acarman [79] solve this problem in a linear cost assignment of affinity features between detections and target tracked objects. Their affinity features include a combination of bounding boxes and color histograms where features are compared using pairwise cosine distance of bounding boxes whereas color histograms are compared using chi-squared distance. Those similarity distances are then set as the cost to be minimized. My tracker employs an even simpler set of costs. Moreover, my system is capable of estimating 3D trajectories besides tracking in the 2D image domain.

Sharma, Ansari, et al. [80] proposes a set of costs to solve the data association problem based on geometry and object shapes. Their set of costs includes 3D-2D, 3D-3D, appearance, and costs based on the shape and pose. They utilize the work in [81] to estimate the 3D shape of cars in the scene from monocular images. However, their system is designed around tracking cars. Tracking any new object class requires adding prior information about that class to the system. My system is generic and has no prior assumptions about the object being tracked.

Osep et al. [82] combine both 2D information from images and 3D information obtained by means of a stereo camera to generate an over-complete set of track hypotheses. A conditional random field (CRF) model is then used to score and select from the hypotheses. Selected hypotheses are then tracked in a coupled 2D-3D Kalman filter. Clearly, their pipeline is more sophisticated. I demonstrate a simpler system that provides comparable results. Geiger et al. [83] built a probabilistic generative model for scene understanding that can jointly estimate the 3D scene layout and object's poses in the environment.

Frossard and Urtasun [84] trained a CNN for detection and matching from both camera and LIDAR data. However, the association is solved as a linear program. Nam and Han [85] propose a CNN that learns targets tracking representations from different domains. Bergmann et al. [86] proposed to tackle multi-object tracking by exploiting the regression head of a detector to perform temporal realignment of object bounding boxes. Thus, they extended CNN object detectors to perform tracking as well.

4.3 Multi-Object Tracking Method

In this section, the main contribution of this work which is my multi-object tracker will be introduced. The multi-object tracker operates in the image domain, that is, it only requires monocular images to perform the data association. For each newly captured frame by the camera attached to the agent (e.g., mobile robot), an object detector is run to obtain a set of detections. The multi-object tracker would have a set of objects that are currently being tracked, its job then becomes to associate each new detection with the most likely tracked object and update its trajectory accordingly.

Object detectors based on convolutional neural networks have become the dominating approaches after the great success of AlexNet [5] in the ImageNet classification challenge [6]. In my work, I will use YOLOv3 object detector [87]. YOLO detector passes the image as a whole in a single pass through the network and outputs the predictions. This is unlike region proposal

object detectors such as Fast R-CNN [88] which perform detections on multiple region proposals and thus might require to look at the same region of the image multiple times. YOLO is accurate and much faster than Fast R-CNN. Although a very accurate but slow object detector to perform the detections could have been used, the goal was to use components ready to run in real-time on deployed robots. That said, the object detector is a swap-able component and it can be replaced by a different one that might suit the target application better.

At each new frame, we have object detections $D_n = \{D_1, D_2, \dots, D_n\}$, and active objects trackers $T_m = \{T_1, T_2, \dots, T_m\}$; that is the set of objects being tracked and their trajectory information. We use D_n and T_m to form a cost matrix C of size $m \times n$. Each entry C_{ij} represents the cost of assigning detection D_j to tracker T_i . The goal then is to find the optimal assignment that minimizes the total cost. Each cost C_{ij} is computed as a combination of two costs described below:

Keypoints Cost

For the first cost, distinctive keypoints or corner-like features are extracted. Keypoints are one of the most widely used visual cues in robotic vision. Shi-Tomasi feature detector [89] is used to extract the keypoints. Note that if the bounding rectangle of each detected object retrieved from the object detector is used as a mask to set the allowed region of the image where the feature detector is to operate, some keypoints might fall outside the physical object as shown in Figure 4-2 (a). In order to alleviate this issue, and to increase the number of actual keypoints that fall on the detected object, the ellipse inscribed in the bounding rectangle is calculated and set as the mask, as shown in Figure 4-2 (b).

Matches between different keypoints are found by means of keypoint descriptors. The BRIEF (binary robust independent elementary features) [48] is used. BRIEF descriptors are in the form of a binary string which means comparing them is fast using Hamming distance. Keypoints computed from each detection at the current frame t is compared to keypoints for each object tracker which were computed at frame $t-1$. Keypoint matches are determined through the ratio test [4]. This cost will then become the number of successful matches and the goal is to maximize their number. However, as the optimization objective is to minimize the total cost, this cost will be formulated as the equivalent of minimizing the unmatched keypoints.

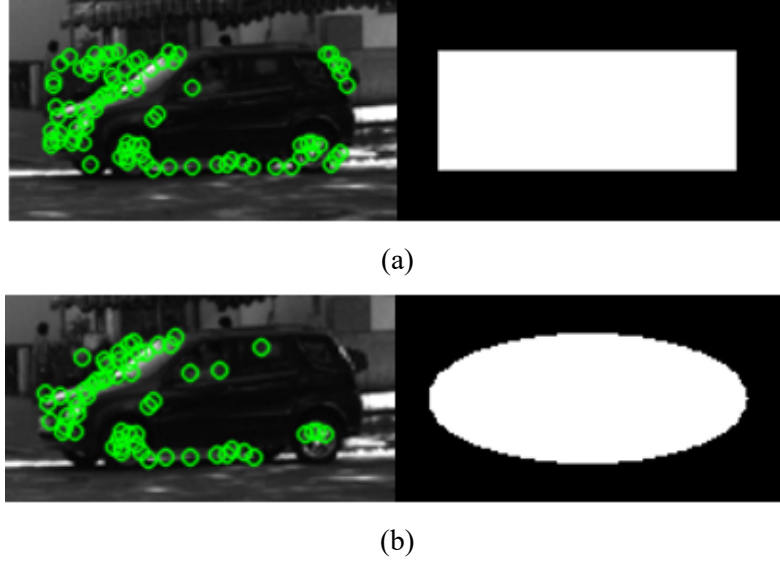


Figure 4-2 Keypoints extraction from detected objects. (a) shows that if the mask is set to extract keypoints exactly in the detection bounding box, some keypoints might fall outside the physical object. (b) here I modify the mask to extract keypoints only in the ellipse inscribed in the bounding rectangle. This reduces the number of erroneous keypoints ([68] © 2019 IEEE).

IoU Cost

This cost measures the overlap between the bounding box of a detected object $B_{D,j}$ and the last measured bounding box of an object tracker $B_{T,i}$ which was measured at time $t - 1$. The cost based on *Intersection-over-Union* (IoU) measure of overlap is then defined as:

$$C_{ij}^{IoU} = \frac{B_{D,j} \cap B_{T,i}}{B_{D,j} \cup B_{T,i}} \quad (4.1)$$

Despite its simplicity, such a measure of overlap works well in practice as objects typically do not move much between consecutive frames. Moreover, such a cost is necessary to complement the first keypoints cost because depending on factors such as illumination conditions and the size of the detected objects, the keypoint detector might not detect enough keypoints per object if not at all. In situations like this, this IoU cost will be the only reliable measure for data association.

Once both costs for each detection/tracker pair are computed, they are linearly mapped to the same integer range so both will have equal contribution to the total cost. They are then added together, and the result of the addition will become an entry C_{ij} of the cost matrix. The optimal

assignment that minimizes the total cost is then solved using the Hungarian-Munkres method [90] [91].

After assignments are found, and as a final filtering attempt against wrong associations, if an object tracker and its assigned detection do not have their bounding boxes intersect, or if the assigned detection has a different class than that of the object tracker, then this assignment is declared wrong. The first filtering condition ensures that no too far away objects are associated. The second condition comes as an advantage of using a CNN object detector that recognizes object classes (e.g., “Car”, “Person”, ...etc.). If an object of type “Car” was being tracked and then suddenly it is associated with a detection of class “Person” then something wrong has happened. Additionally, at some frames we might have fewer detections than active objects trackers. All excess trackers and trackers that were wrongfully assigned are immediately removed from the list of active trackers. Objects trackers update their trajectory with successfully assigned detections. And if at a frame there were more detections than trackers, then these excess detections, as well as any wrongfully assigned detections, are initialized as new active objects trackers.

4.4 Combined System

In this section, the multi-object tracker introduced in the previous section will be combined with my visual odometry algorithm developed in Chapter 3. Moreover, since the visual odometry algorithm relies on a stereo camera, the stereo images will be used to compute disparity (depth) maps as well and used along with pose information computed from visual odometry to estimate the object trackers in 3D world frame. The main components showing dependencies and information flow is shown in Figure 4-3. Note that the system components are loosely coupled, and each component can easily be switched. The components are discussed below.

Using the input from the stereo camera, the disparity (depth) map can be computed. The stereo method by Hirschmuller [92] is used to compute this disparity map. I relied on its implementation found in the open-source computer vision library (OpenCV). This depth map is then passed to the 3D tracking estimation component.

The 3D Tracking Estimation component utilizes the information from visual odometry, depth map, and 2D multi-object tracker in order to estimate tracked objects bounding boxes in the 3D world coordinate frame. This is a necessary knowledge to enable safe navigation and obstacle avoidance in real-world scenarios.

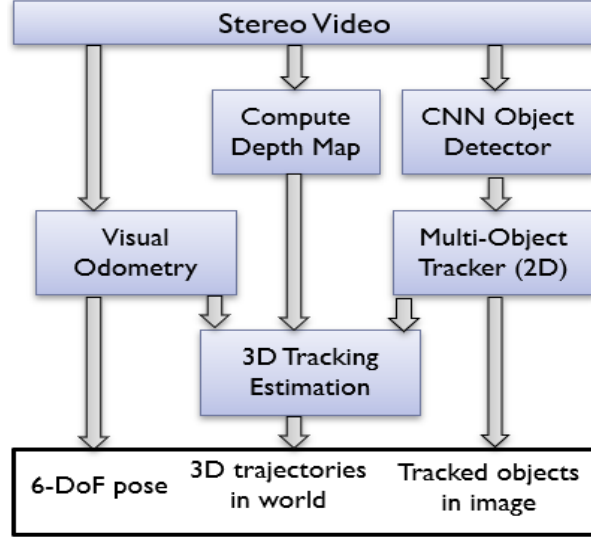


Figure 4-3 An overview of the complete combined system and its components. Arrows show the direction of data flow ([68] © 2019 IEEE).

For each tracked object, a set of 3D points lying on that object is triangulated as a first step. To do that, the same elliptical mask shown in Figure 4-2 before is used in order to obtain the disparity values from the computed disparity map corresponding to each tracked object. For each pixel at location (u, v) , a 3D point $(X, Y, Z)^T$ is triangulated:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} Z(u - c_x)/f \\ Z(v - c_y)/f \\ b * f/d \end{pmatrix} \quad (4.2)$$

where b is the stereo baseline, d is the disparity value at that pixel location, and f is the focal length. After that, the triangulated points are sorted by depth value (Z) and the ones closer and further than the first and third quartiles are removed. Those removed points are considered outliers. The remaining valid points are then placed in the global world frame by multiplying with the pose computed from visual odometry. A bounding box enclosing these transformed 3D points is now computed; this is the bounding box enclosing the tracked object in the 3D world at the current frame.

4.5 Evaluation

In this section, the introduced system will be evaluated. The well-known KITTI dataset [51] is used for evaluation. This dataset is comprehensive and targeted for evaluating different autonomous driving perception algorithms. Images were collected at a rate of 10Hz from cameras mounted on a moving vehicle.

The Tracking benchmark of the KITTI dataset consists of 21 training sequences that have the ground truth publicly available and 29 testing sequences. The benchmark has three tracking targets, Car, Pedestrian, and Cyclist, in an urban environment. As is expected to face in an urban environment, the sequences are crowded with tracking targets which result in occlusions making KITTI a challenging dataset. Evaluation criteria adopted are the CLEAR MOT [93] metrics and the Mostly-Tracked/Partly-Tracked/Mostly-Lost (MT/PT/ML) [94] metrics.

Recall, Precision, True Positives (TP), and False Positives (FP) metrics evaluate the object detector. Multiple Object Tracking Accuracy (MOTA) can be viewed as the ratio of false positives, misses, and mismatches computed over all ground-truth objects. Multiple Object Tracking Precision (MOTP) measures the ability of the tracker to estimate precise objects' positions. MT is the percentage of ground truth trajectories that are covered by tracker output for more than 80% in length. ML is the percentage of ground truth trajectories that are covered by tracker output for less than 20% in length. PT is $1.0 - MT - ML$. IDS is the number of ID switches or the number of times a tracked trajectory changed its ground truth one. FRAG (fragments) is the number of times the ground truth trajectory was interrupted [94]. Smaller values of ML, IDS, and FRAG is better. The metrics are summarized in Table 4-1.

Unlike other work ([74] [80] [82]) which splits the training set into training and validation subsets in order to tune their algorithms' hyperparameters or specifically train the object detector, I test directly on the training set and do not learn anything from it. That is, I do not use the training sequences to train or tune the object detector or do any sort of hyperparameters search. My motivation for this is that I wanted to build a system from generic components that can be easily deployed in different scenarios and still provides a good expected performance. That is, without being biased toward any specific dataset.

Table 4-1 Metrics used to evaluate multi-object detection and tracking. Based on Ref. [93] [94]. ([68] © 2019 IEEE).

Metric	Definition
Recall	Correctly matched objects / total groundtruth objects.
Precision	Correctly matched objects / total output objects.
TP	True Positives: Positive instances that are correctly classified as such.
FP	False Positives: Negative instances that are wrongfully classified as positive.
MT%	Mostly tracked: Percentage of GT trajectories which are covered by tracker output for more than 80% in length.
ML%	Mostly lost: Percentage of GT trajectories which are covered by tracker output for less than 20% in length. The smaller the better
PL%	Partially tracked: 1.0-MT-ML
Frag	Fragments: The total of No. of times that a GT trajectory is interrupted in tracking result. The smaller the better.
IDS	ID switches: The total of No. of times that a tracked trajectory changes its matched GT identity. The smaller the better.
MOTP	The total error in estimated position for matched object-hypothesis pairs over all frames, averaged by the total number of matches made. It evaluates tracker's ability to estimate precise object positions.
MOTA	This accounts for all object configuration errors made by the tracker, false positives, misses, mismatches, over all frames. It measures tracker's performance at detecting objects and keeping their trajectories, independent of the precision with which the object locations are estimated.

In the current implementation and evaluation, the YOLOv3 [87] object detector has been used with the original weights distributed with the paper. This detector was pre-trained on the MS COCO [95] dataset. COCO dataset has objects in 80 different categories. I will report the “Car” class detected by YOLO and “Person” as “Pedestrian” detection. YOLO expects input images to be square (input set to 608×608 in our evaluation) but the KITTI dataset has wide images of size 1242×375. Object detector confidence is set to 50% and non-maximal suppression is applied on the detections.

The results on KITTI training sequences of the “Car” tracking benchmark of my algorithm compared to two other state-of-the-art ones (BeyondPixels [80] and CIWT [82]) are reported in Table 4-2. The results of the compared algorithms are the ones reported in their original papers. We can see that my method even with the standard YOLO detector and without any specific prior training or tuning on the KITTI sequences, was able to come very close to CIWT in terms

of tracking accuracy metrics. However, CIWT excels in maintaining the least number of ID switches and fragmentations. This is primarily because of the coupled 2D-3D Kalman filter that the authors propose which helps their method to maintain longer trajectories.

Table 4-2 “CAR” tracking results on KITTI training sequences of my algorithm with two detectors (YOLO and RRC) compared to two different methods ([68] © 2019 IEEE).

	My MOT w/ YOLO	My MOT w/ RRC	Beyond Pixels [80]	CIWT [82]
MOTA	69.2	90.8	91.4	74.38
MOTP	77.5	89.9	89.83	82.85
Recall	78.6	94.62	94.65	-
Precision	94.4	98.47	98.47	-
MT	50.2	87.76	87.76	49.59
PT	42.2	10.64	10.63	40.68
ML	7.62	1.60	1.59	9.80
TP	21235	25290	25309	-
FP	1266	392	392	-
IDS	368	396	232	26
Frag	908	586	423	131

On the other hand, BeyondPixels is a two-frame greedy association tracker. Thus, similar to my method, they suffer from ID switches and fragmentations. The authors employ a set of costs and use the KITTI training set to fine-tune the weight of each cost. Furthermore, they use the training sequences to train Recurrent Rolling Convolution (RRC) [96] object detector. Additionally, BeyondPixels is designed and tuned mainly with tracking “Car” objects in mind. Hence, we can see it achieved the best tracking accuracy metrics. Nevertheless, the authors of BeyondPixels generously make their reference RRC detections for “Car” class on KITTI sequences publicly available. Once I tested my algorithm with RRC detections (see Table 4-2), I was able to achieve comparable performance to the state-of-the-art BeyondPixels algorithm. This result is particularly interesting because even though my method utilizes simple and straightforward costs, it was able to achieve performance comparable to state-of-the-art online trackers with an accurate set of detections. This leads me to believe that with the right set of detections, even an online two-frame tracker with simple cost function could achieve competitive results.

The effect of each of the two costs acting alone with each of the two detectors (YOLO and RRC) on the “Car” benchmark is reported in Table 4-3. We can see that even a simple cost such

as the IoU is able to achieve good tracking accuracy metrics. Keypoints cost is however needed as it helps in reducing ID switches and fragmentations.

Table 4-3 Results of each cost alone on KITTI ‘Car’ benchmark ([68] © 2019 IEEE).

Detector	Metric	Full	Keypoints	IoU
YOLO	MOTA	69.2	69.1	68.4
	MOTP	77.5	77.5	77.5
	IDS	368	392	554
	Frag	908	935	1087
RRC	MOTA	90.8	90.8	89.9
	MOTP	89.9	89.9	89.9
	IDS	396	396	596
	Frag	586	586	788

The next evaluation is on the “Pedestrian” benchmark. Tracking moving pedestrians is more challenging as they occupy a smaller region of the image, they move in less predicted ways, and they generally move in crowds causing occlusion and clutter. The results of my algorithm using YOLO and the Regionlets [97] detections provided by KITTI benchmark are reported in Table 4-4. The results of CIWT [82] as reported in the original paper are provided as well in Table 4-4 for comparison. We can see that although the more sophisticated CIWT algorithm surpasses mine in terms of MOTA/MOTP metrics, my algorithm achieves better results in terms of the MT/PT/ML ones.

Table 4-4 “Pedestrian” tracking results on KITTI training sequences of my algorithm with two detectors ([68] © 2019 IEEE).

	My MOT /w YOLO	My MOT w/ Regionlets	CIWT
MOTA	35.6	46.7	61.87
MOTP	71.0	75.5	78.85
Recall	66.5	78.9	-
Precision	70.1	73.6	-
MT	40.7	62.2	55.95
PT	49.7	35.3	33.33
ML	9.6	2.4	10.71
TP	7459	8877	-
FP	3187	3178	-
IDS	213	366	41
Frag	680	761	164

Qualitative results of the multi-object tracker on a selected number of frames are shown in Figure 4-4, while Figure 4-5 shows the results of 3D estimations. As for computational performance, the time spent by the tracker itself, that is, only solving for and associating new detections to current objects trajectories, and terminating and starting new trajectories; was measured by running KITTI training sequence 20 for 3 times and collecting time measurements. The experiment was performed on a laptop computer with Intel Core i7-7700HQ CPU and 16GB of RAM, running Linux Ubuntu 16.04. The mean elapsed time on tracking was measured to be 53.36ms with a standard deviation of 27.54ms.

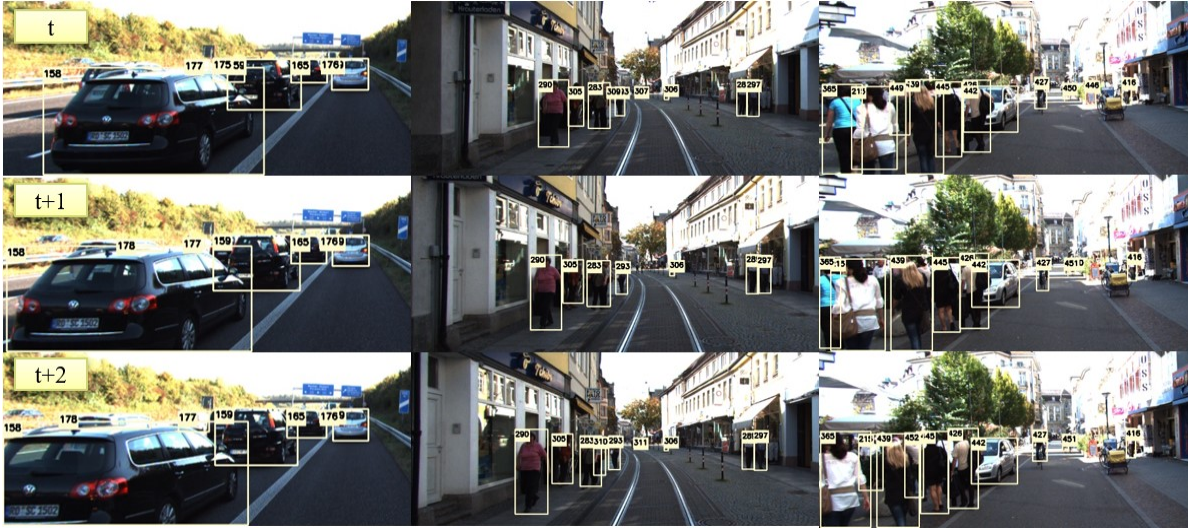


Figure 4-4 Qualitative evaluation of our algorithm on selected frames from KITTI dataset ([68] © 2019 IEEE).



Figure 4-5 Sample frames showing traffic participants being tracked in 3D world frame ([68] © 2019 IEEE).

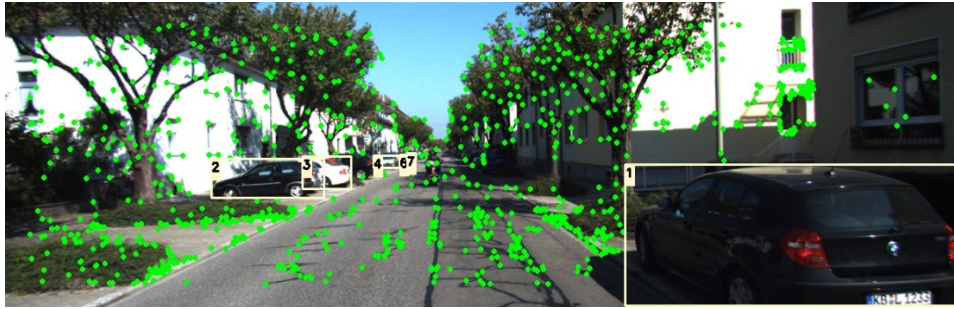
Moving on from multi-object tracking evaluation, I would like to evaluate a fundamental assumption of the visual odometry algorithm. Feature-based visual odometry systems work by detecting features (corners) in the scene and by detecting the same corners in subsequent frames, the movement relative to those corners can be estimated. The assumption here is that the corners lie on a static structure. That is, the corners observed in the scene across frames are stationary and any change in the pixel location where they are observed is due to the camera self-movement. Needless to say, some of the detected corners will lie on moving objects such as cars or pedestrians. Visual odometry systems typically work to spread out the detected corners across the image in order to reduce the probability of any corner being on a moving object. Moreover, outlier rejection techniques are utilized to deal with such corners.

In this experiment, the previously developed visual odometry algorithm in Chapter 3 was used to evaluate the effect of removing keypoints detected on tracked objects. The only differences from the standard algorithm and published parameters when evaluated originally in Chapter 3 are: staging points is disabled (staging threshold set to zero), untracked threshold of map points is set to 3, and triangulating new map points is performed once the number of the points in the local map drops below 1000. KITTI visual odometry training sequences are used in this evaluation and the same translation error metric is reported. The results are reported in Table 4-5.

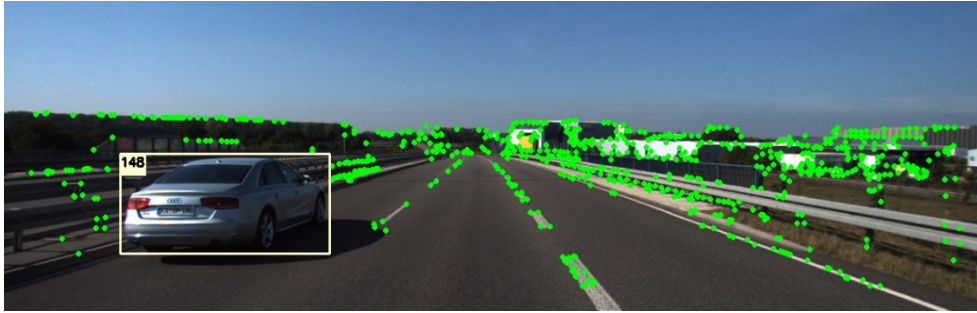
Table 4-5 Visual odometry translation error on KITTI sequences with and without removing features on tracked objects ([68] © 2019 IEEE).

Sequence	Et [%] with removing features	Et [%] without removing features
00	1.56	1.34
01	3.50	4.61
02	1.53	1.49
03	2.72	2.67
04	1.50	1.47
05	0.95	0.91
06	1.34	1.25
07	1.13	1.04
08	1.43	1.32
09	1.73	1.64
10	1.07	1.04
Total	1.53	1.49

To my surprise, not removing the detected features on tracked objects resulted in better results in all KITTI odometry training sequences except sequence 01. I believe the reason to be that most tracked objects (mainly cars) are stationary for the most part as can be seen from Figure 4-6 (a). So, by removing those features, we are reducing the total number of useful features that can be used successfully for motion estimation. This is in contrast to sequence 01 which is a highway with moving vehicles in the scene as can be seen from Figure 4-6 (b), and thus removing those features actually helps.



(a)



(b)

Figure 4-6 Demonstration of the effect of removing keypoints lying on tracked objects on visual odometry. (a) in most KITTI odometry sequences, cars are stationary. (b) sequence 01 is captured on a highway with moving cars in the scene, so removing those keypoints helps VO estimation ([68] © 2019 IEEE).

4.6 Summary

In this chapter, an online multi-object tracker that operates in a tracking-by-detection framework has been introduced. Competitive and real-time performance achieved by the simple and intuitive two costs used in association has been demonstrated. Moreover, this algorithm has been combined with the previously developed stereo visual odometry system to obtain motion estimates and 3D trajectories as well. This is an essential requirement for autonomy.

The most interesting finding in my evaluation was that with the right set of detections, even a simple online tracker with simple costs can achieve competitive results compared to more

elaborate methods. We saw that on KITTI “Car” tracking benchmark where I used a better detector that was specifically trained on KITTI dataset. Another interesting finding is that the effect of removing keypoints lying on tracked objects on visual odometry highly depends on the scene and might not be necessary.

Chapter 5

A Single-Stream CNN For Semantic Segmentation and Monocular Depth Prediction

Two essential visual perception tasks are semantic segmentation and depth prediction. Semantic segmentation refers to the task of classifying each pixel of an image. That is, assigning a class (e.g., “Car”, “Person”, “Building”, “Street”, ...etc.) to each pixel of the captured image. This is an essential first step toward scene understanding and interpretation. Depth prediction is the second important visual perception task that I will tackle in this chapter. The ability to infer scene depth, that is, being able to locate objects in 3D world frame from image data is an important safety requirement. It enables object collision avoidance and generating motion trajectories in 3D world frame. In this chapter, I will present an innovative convolutional neural network to perform both semantic segmentation and monocular depth prediction tasks simultaneously. This chapter is based on my work in Ref. [98].

5.1 Introduction

Visual perception refers to the brain’s ability to interpret the observed surrounding environment from signals coming from our eyes. Great efforts continue to be invested toward developing computer vision techniques that can -ideally- give a computer the same capability of interpreting information incoming from visual sensors. One of the main perception tasks is depth estimation of the perceived scene. This depth estimation enables tasks such as localization, obstacle avoidance, safe interaction and manipulation with objects in the environment, among many others.

Traditionally, specialized vision sensors are used to obtain depth information along with color images. Stereo cameras, which apply the principles of binocular vision, are one of the first vision sensors used to obtain depth information. The literature on stereo vision is rich and well-developed [8]. Other specialized vision sensors are ones based on structured light (RGB-D) that can measure depth directly. This type of sensors has been used successfully in mapping, reconstruction and tracking applications [41].

Despite their successes, these specialized vision sensors (stereo and RGB-D) have their share of limitations as well. Stereo cameras, for example, require tricky and complicated calibration procedures. They also require shutter synchronization. RGB-D sensors are active ones and thus are highly affected by other illumination sources in the environment. Moreover, RGB-D sensors have a limited range in which they can measure depth, which limits their utility to indoor environments. In light of such challenges and with the omnipresence of traditional monocular cameras, it would be most beneficial to extend their operational range with the capability to infer depth. This is one target objective of the work in this chapter.

Computer vision techniques based on deep learning methods have then successfully solved many challenging perception problems facing autonomous robots and self-driving cars. This is particularly true with the rise of deep learning-based methods since 2012 when a convolutional neural network (CNN) called AlexNet [5] won the ImageNet large-scale image classification challenge [6] beating the best classical methods with a significant margin. Example of these visual perception tasks include: object detection [99], semantic segmentation [100], optical flow estimation [101], road and road boundary detection [102], monocular depth prediction [103], among others.

The common factor in these methods is that a sophisticated model -usually based on CNNs- is built to solve one task and one task only. This, however, is potentially a limiting factor for deployment onboard mobile robots with limited memory, compute, and power resources. Each new model added to the robot perception algorithms stack will add such overhead. This led to an increasing interest in multi-task learning [104] where a single model can learn to perform multiple tasks.

In visual perception in autonomous driving and mobile robots' context, multi-task learning relies on convolutional neural networks. They tend to follow an encoder-decoder architecture [105]. The encoder extracts useful features common to all target tasks and is typically the shared part. A decoder specific to each task then takes the features extracted from the encoder and transforms them into their target output. This significantly reduces the computational requirements compared to multiple single-task networks. Moreover, this improves the generalization capacity as the encoder is learning better feature representations that are useful for different tasks. However, the main challenge of such multi-task methods is task weighing. That

is, how to properly tune the contribution of each task loss for backpropagation such that no one task is making progress learning while others stagnate.

In this chapter, I will demonstrate an innovative approach to building a single-stream CNN similar to single task networks but with the capability of performing two tasks at the same time similar to multi-task ones. The two tasks my network will be performing are semantic segmentation and monocular depth prediction. Hence, the model is called *One-stream Semantic Segmentation and Depth prediction Network* (OSD-Net). Once the model is built, it is trained using standard backpropagation.

5.2 Literature Survey

For monocular depth prediction, there have been many attempts at predicting depth from single images. For example, Saxena et al. [106] predicted depth from image features using linear regression and a Markov Random Field (MRF). However, such methods relying on traditional machine learning approaches and hand-crafted features are limited. Only after the rise of modern CNNs that the learning to predict depth from a single image became a feasible option. One of the first attempts in this matter is the work by Eigen et al. [107]. After that, many methods have been developed in the literature addressing monocular depth prediction using some form of a CNN [108] [109] [110].

On the other hand, regarding semantic segmentation, before the recent rise in deep learning approaches, it was difficult to achieve reasonable accuracy with classical methods. For example, a random forest classifier was used in [111], whereas semantic segmentation was formulated as a CRF problem in [112]. However, the progress of classical methods was always bounded by the hand-crafted features used [113]. This limitation was overcome by convolutional neural networks that can learn their own feature representations. The most notable CNN ones for semantic segmentation are FCN [114], SegNet [100], and UNet [115]. A survey of deep learning-based methods for semantic segmentation is by Siam et al. [113].

In these methods for monocular depth prediction and semantic segmentation, a single network is built and trained to perform its target task which could cause significant overhead for deployment on embedded devices onboard as previously discussed in the introduction. Multi-task learning has emerged as a promising solution to overcome such increasing computational requirements associated with performing more tasks. Multi-task learning has a history and foundations in machine learning [104] [116]; however, my focus is CNN based approaches used

in visual perception. Beside reduced computational complexity, learning multiple supervisory tasks by the same model improves generalization as the model is learning better feature representations that are useful to different tasks [117].

The common architecture is to have a single fixed and shared encoder network and multiple decoder networks for the different desired tasks [118] [119] [105]. As each decoder has its own task-specific loss function, the challenge becomes how to combine these losses to find the final loss value that will be used for training the whole network using backpropagation. Losses need to be properly balanced so that every task is making progress training. UberNet [118] and MultiNet [119] used weighted summation of the individual losses. MultiNet++ [105] used a geometric loss strategy. Guo et al. [120] proposed a dynamic task prioritization strategy where more difficult tasks are given higher priorities during training. Sener and Koltun [121] solve multi-task learning as a multi-objective optimization problem.

In this work, I propose to solve the two tasks (semantic segmentation and monocular depth prediction) in one network similar to multi-task learning ones. However, I do so in one single-stream network similar to single task networks. Thus, I avoid the need for tricky loss balancing while achieving the benefits of multi-task learning.

5.3 OSD-Net

My model titled *One-stream Semantic Segmentation and Depth prediction Network* (OSD-Net) follows the popular encoder-decoder architecture as shown in Figure 5-1 The encoder extracts complex features from the input image which the decoder takes and transforms into the desired output. I employ a single straightforward encoder-decoder network without any special sub-networks or multiple branches.

For the encoder network, I have implemented and tested my model with two feature encoders: ResNet-50 [72] and MobileNet-v2 [122]. In both cases, the final fully-connected layers are removed and the feature extraction backbone with weights pre-trained on ImageNet [6] is retained and used as the feature encoder.

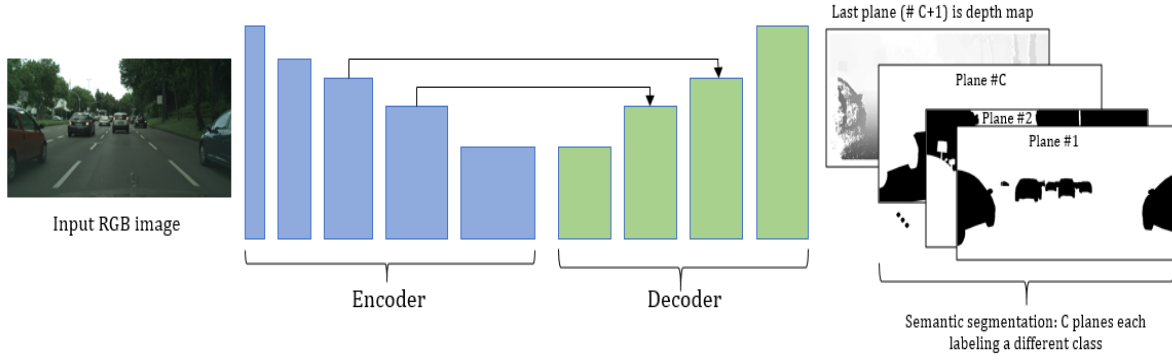


Figure 5-1 An overview of our network which consists of a single encoder-decoder. It can predict both semantic segmentation and depth in a single output. Assuming the input image has spatial dimensions of $H \times W$, and C classes for semantic segmentation, output has dimensions of $H \times W \times (C+1)$. Credit [98].

The decoder is fully convolutional and constructed similar to FCN [114]. It consists of two transposed convolution layers with a kernel size of 4 and stride of 2, those will upsample their input by a factor of 2. An additional transposed convolution layer with a stride of 8 and a kernel size of 16 is used to perform the final upsampling to obtain the end output. All the mentioned transposed convolution layers have an input and output depth (i.e., number of filters) equal to the depth of the final output from the network.

Two skip connections send the output of two layers from the encoder to the two 2x upsampling layers in the decoder. These skip connections are connected from layers that have the same spatial size as their target layers in the decoder. The data sent through these skip connections, as well as, from the final layer of the encoder are passed through feature aggregation layers before passing to the corresponding decoder layer. These feature aggregation layers are simply convolutions with a kernel size of 1 and an output depth equals the final output depth of the network. The values incoming from the aggregation layers through the skip connections are combined with their corresponding decoder layers through an element-wise addition. Leaky ReLu [123] activation is used after each aggregation and upsampling. A forward pass through the decoder is shown in Algorithm 3.

The method that enables outputting both semantic segmentation and monocular depth prediction from this single-stream network is to solve semantic segmentation as a regression problem. Semantic segmentation is typically solved as a classification problem using the cross-entropy loss function. Instead, I propose to stack semantic segmentation and depth map and solve them together as a regression problem.

Algorithm 3 A forward pass through the decoder where the input x is the output from the encoder.

Decoder Forward(x):

```
x1 ← Encoder[skip1]
x2 ← Encoder[skip2]
x ← LeakyReLU(Aggregate(x))
x ← LeakyReLU(TransposedConvolution_2x(x))
x ← x + LeakyReLU(Aggregate(x1))
x ← LeakyReLU(TransposedConvolution_2x(x))
x ← x + LeakyReLU(Aggregate(x2))
x ← LeakyReLU(TransposedConvolution_8x(x))
output ← x
```

Assuming an input RGB image of dimensions $H \times W \times 3$, and C classes (including background) for semantic segmentation. I first perform one-hot-encoding of the target labels creating labels of dimensions $H \times W \times C$ in which each pixel would have a value of 1 along the depth axis at the plane corresponding to its class label and a value of 0 otherwise. Afterward, the disparity map (which is related to depth map by means of the constant camera intrinsics) is normalized to the range $[0,1]$. It is then appended to this volume along the depth axis to have a final target output of size $H \times W \times (C+1)$. The network would produce a prediction of the same dimensions, and the loss function used is the smooth L1 loss which is also known as Huber loss. This loss function is typically used for regression problems and it is less sensitive to outliers compared to mean square error loss. This loss function is defined in Equation 5.1.

$$loss(x, y) = \begin{cases} 0.5(x - y)^2, & |x - y| < 1 \\ |x - y| - 0.5, & otherwise \end{cases} \quad (5.1)$$

5.4 Evaluation and Discussion

For training and evaluation, I will use the CityScapes dataset [124]. This dataset was collected by a vehicle driving in 50 cities throughout the period of several months. The dataset is split into 2975 RGB images that will be used for training and another 500 validation images that will be used to report performance. For semantic segmentation, the dataset provides accurate labels for 19 classes marked for evaluation. Disparity maps provided are computed using the two cameras installed in a stereo setup on the vehicle. Original images are of size 2048×1024 so they are resized to 1024×512 in order to control the memory requirement. Images are further pre-processed by standardizing them using mean of (0.485, 0.456, 0.406) and standard deviation of

(0.229, 0.224, 0.225) which are the same as ImageNet values. No further preprocessing or augmentations are performed on the images while training the models.

For a fair comparison, I have implemented two models each performing a single task (semantic segmentation and monocular disparity prediction) separately, and an additional multi-task network performing the two tasks jointly. The built models are thus:

1. OSD-: My single-stream model presented in the previous section.
2. 1-Seg-: A single task network performing semantic segmentation. The network architecture is constructed exactly as OSD with the same skip connections. However, the cross-entropy loss is used. This is the typical loss function used for multi-class classification problems.
3. 1-Disp-: A single task network performing monocular disparity map prediction. This is constructed exactly similar to OSD as well. As this network is solving a regression problem, the same Huber loss used in OSD is used here.
4. MSD-: A typical multi-task network architecture with a shared feature encoder and a decoder for each of the two tasks as shown in Figure 5-2. Each of the decoders is constructed exactly as its single-task counterpart and with the same loss function. The two losses are combined using the arithmetic mean to obtain the final network loss. The name MSD stands for *Multi-stream Semantic Segmentation and Depth prediction*.

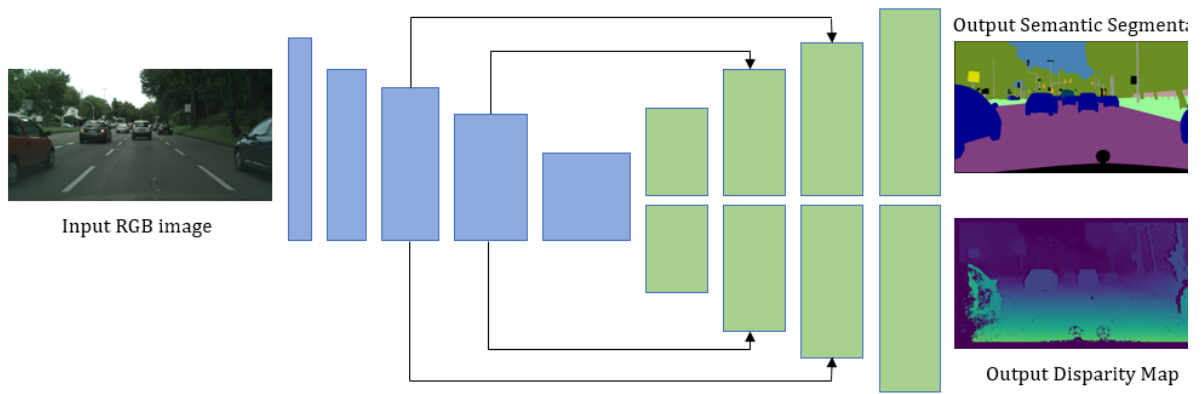


Figure 5-2 Vanilla Multi-Task Network with one shared encoder and two task-specific decoders. This network was implemented for comparison and evaluation. The two tasks' losses are combined using the arithmetic mean.

Each of these four architectures has two variations, one with ResNet50 [72] as a backbone feature encoder, and one with MobileNet-v2 [122]. Thus, we train a total of eight networks. All the models are built using PyTorch 1.1.0 and trained on a desktop computer with a single Nvidia

GTX 1080Ti GPU. I used Adam optimizer [125] with a learning rate of 5×10^{-5} and a batch size of 4 for training. Each model was trained for 50 epochs. Filters' weights are randomly initialized using He normal initializer [126].

The computed loss of the four models with the ResNet-50 encoder averaged over the validation set at each training epoch is shown in Figure 5-3. Note that the validation loss axis is shown in logarithmic scale. We can observe how the 1-Seg network which uses cross-entropy loss function has the highest loss values and this, in turn, drives the MSD network loss to be high. On the other hand, OSD and 1-Disp models which rely on a regression loss function, have lower loss values. This sort of discrepancy in the scale of different losses is a major challenge for multi-task networks such as MSD.

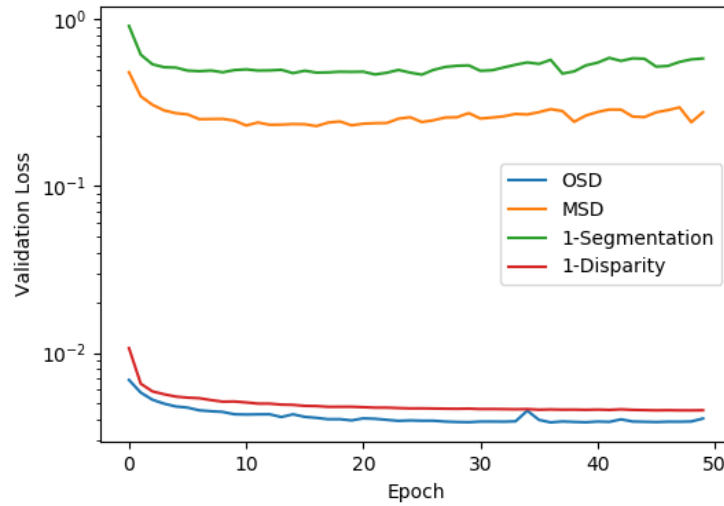


Figure 5-3 Average validation loss of each model with every training epoch. Credit [98].

Learning semantic segmentation with each epoch is shown in Figure 5-4. The mean intersection-over-union (IoU) metric reported in this figure is defined as the ratio of correctly classified pixels to the total number of pixels assigned to that class by both the ground truth and predicted by the model; then averaged over classes. We can observe from Fig. 4 that OSD was able to achieve the best mean IoU from the first epoch and it was able to learn faster than the other two models.

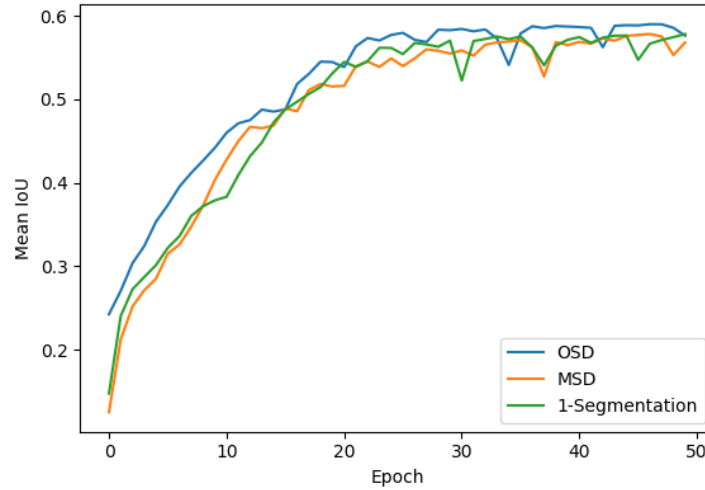


Figure 5-4 Achieved mean intersection over union (IoU) for semantic segmentation by each model at every training epoch. Credit [98].

Likewise, Figure 5-5 shows how each model is learning disparity with each epoch. We can observe from this figure how the multi-task network MSD is converging slower than the others. This is because segmentation task loss is dominating the final model loss causing disparity task to make slower progress. Note that Disparity Accuracy reported in Figure 5-5 is defined as the average number of disparity values predicted to be within 0.025 of their ground truths.

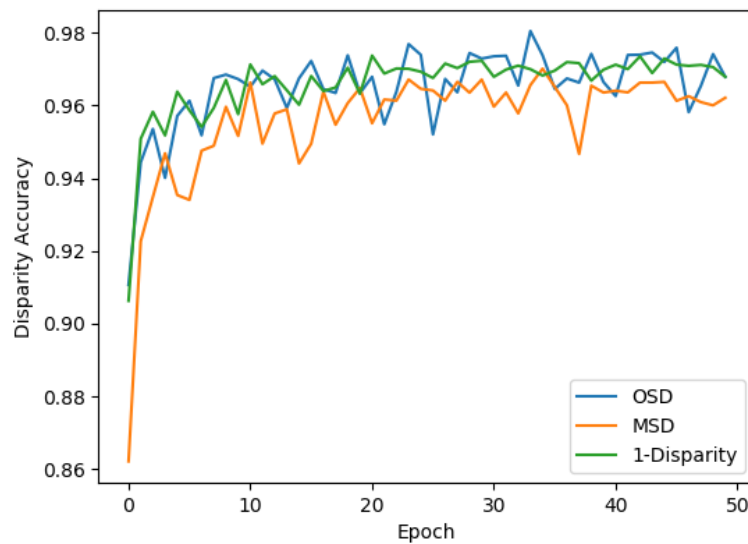


Figure 5-5 Disparity prediction accuracy by each model at each training epoch. Credit [98].

From both Figure 5-4 and Figure 5-5, it can be seen that my model -OSD- was able to achieve better results right from the first training epoch and converge faster than other methods. I

believe the reason to be as the same filters are being trained to perform the two tasks, each task is acting as a regularizer for the other.

A quantitative evaluation of the eight trained models is presented in Table 5-1. The four semantic segmentation evaluation metrics reported are defined as:

- Pixel accuracy is the number of correctly classified pixels to the total number of pixels.
- Mean accuracy is the mean of correctly classified pixels in each class to the total number of pixels in that class.
- Mean IoU was defined previously as the ratio of correctly classified pixels to the total number of pixels assigned to that class by both the ground truth and predicted by the model and averaged over classes.
- Frequency weighted (FW) accuracy is similar to Mean IoU but the mean is weighted by the number of pixels in each class.

Disparity is evaluated using the standard root-mean-squared-error (RMSE). Because the disparity values are already normalized to $[0,1]$ range, RMSE values are normalized.

Table 5-1 Quantitative evaluation of our method compared to standard baseline ones. Credit [98].




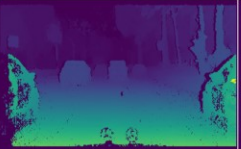
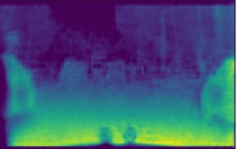



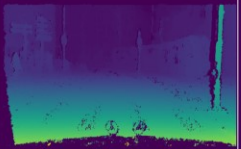
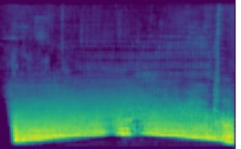

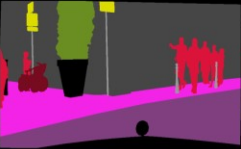
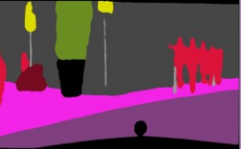
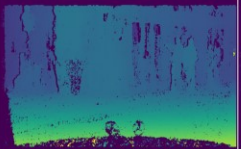
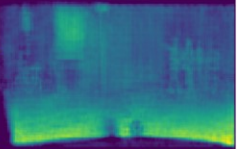

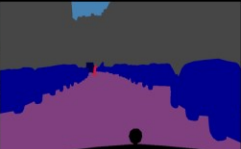
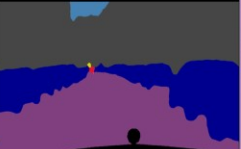
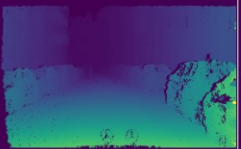
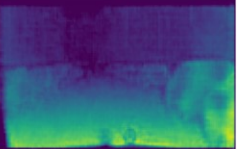
Model	Semantic Segmentation				Disparity
	Pixel acc.	Mean acc.	Mean IoU	FW. acc.	NRMSE
OSD-ResNet	90.13	68.70	57.62	82.87	9.45
MSD-ResNet	90.10	66.12	56.81	82.69	10.65
1-Seg-ResNet	90.26	68.05	57.82	82.85	-
1-Disp-ResNet	-	-	-	-	9.52
OSD-MobileNet2	89.24	65.12	54.67	81.28	9.69
MSD-MobileNet2	89.15	64.06	53.56	81.12	11.88
1-Seg-MobileNet2	89.08	62.42	52.84	81.04	-
1-Disp-MobileNet2	-	-	-	-	9.75

It can be observed that my proposed model -OSD- performed better than MSD on all semantic segmentation evaluation metrics and achieved comparable accuracy to the 1-Seg-network. It also achieved lower disparity prediction error than MSD and a comparable error value to the 1-Disp single task model. These results are very exciting as it means that my method resulted in learning a better model to perform both tasks simultaneously without us having to explicitly tune the combined loss function as in the MSD case.

The backbone feature encoders tested are ResNet-50 and MobileNet-v2. ResNet-50 is a large model that provides better features representation, whereas MobileNet-v2 is designed for deployment on mobile devices with tighter resources. For reference, ResNet-50 encoder standing

by itself has 23,508,032 parameters, whereas MobileNet-v2 has 2,223,872. This is a 90.5% reduction in parameters. With all hyperparameters remaining constant, we can observe from Table I that switching the feature encoder from ResNet-50 to MobileNet-v2 resulted in a slight drop in semantic segmentation accuracy for all of the three models. However, my approach - OSD- was able to achieve the best semantic segmentation results even compared to the dedicated 1-Seg- model. Qualitative results of OSD-ResNet50 are shown in Table 5-2.

Table 5-2 Qualitative results on random frames from the CityScapes dataset. Credit [98].

RGB Image	Semantic Segmentation		Disparity Prediction	
	Ground Truth	Prediction	Ground Truth	Prediction
				
				
				
				

5.5 Summary

In this chapter, I have presented an innovative approach for extending typical single-task networks to perform multiple tasks. This reduces the complexity associated with using multiple networks each performing a single task which is a limiting factor for deployment on embedded devices with limited resources. The traditional way of tackling this issue was to use a multi-task network with a shared encoder network and multiple decoders each for a specific task. The challenge then becomes in balancing different tasks' losses which I avoid altogether with my approach. The presented model outputs semantic segmentation and depth prediction from a single image. Both are essential capabilities for any visual perception system in autonomous driving. My approach of combining both tasks into a single tensor then solving as a regression

problem, is applicable to other network architectures as well. This is because the trick lies in the loss function and output shape and not in any architectural element of the network.

Chapter 6

Conclusion and Future Work

Robotics, Autonomy, and Artificial Intelligence are among the main defining technologies of the times to come. These technologies have been playing a central role in specific domains and in specialized applications for a while now, such as in factories. However, applications of such technologies have started to play a role in our everyday life. Advanced driver assistance systems (ADAS) and fully self-driving vehicles are among the most important applications of mobile robotics and AI.

6.1 Dissertation Summary

This dissertation focused on developing robust real-time perception algorithms with focus on autonomous driving as the main target area. Real-time and robust perception is important as it enables timely and proper decision making and control. Visual perception algorithms were tackled in this work given the advantages of cameras as a perception sensor. Solutions using both classical and deep-learning-based approaches have been investigated. The contributions of this dissertation include:

The Development Of A Real-Time Visual Odometry Algorithm.

Visual odometry endows a mobile robot with the capability of inferring its own motion in the environment. The developed method runs in real-time on CPU. The innovative usage of a transient local 3D map enabled low-drift rates over long-traveled distances while running in real-time. The developed algorithm was tested with both stereo and RGB-D cameras and evaluated on different datasets spanning autonomous driving in urban environments, a drone flying in all 6 DoF in an industrial plant, and a handheld camera in an office environment. Assessment of the real-time performance on embedded Linux-based single-board computers were performed and low-overhead of the algorithm was demonstrated. Such a low-overhead visual motion estimation algorithm is an essential component in any robotic visual perception system.

A Study On Visual Odometry Performance Under Low Illumination

Visual odometry systems are typically evaluated on standard datasets which are collected at daytime with good illumination conditions. However, visual odometry systems are expected to operate under challenging low illumination conditions as well. This was one motivation of collecting the adverse weather dataset by my colleagues in the lab. I have then used one challenging sequence collected at night to evaluate the performance of multiple feature-based visual odometry and visual SLAM systems. Four different image preprocessing techniques were investigated with the goal of enhancing captured images under low illumination for sustained tracking of the studied visual motion estimation systems. This study constitutes a first small step toward this important and challenging problem. The collected datasets are publicly available¹² and hopefully would help more research.

A Combined Visual Odometry And Multi-Object Detection And Tracking System

Detecting and tracking other traffic participants is an essential visual perception task which enables safe navigation for autonomous mobile robots. In order to address this problem, I have developed a two-frame multi-object tracker operating in a tracking-by-detection paradigm. An object detector based on a convolutional neural network is run on each frame then the tracker associates these detections through time. This multi-object tracker was combined with my visual odometry system and an innovative system that can estimate motion, detect and track objects both in 2D and 3D was developed. The performance of this system was evaluated for autonomous driving domain. The most interesting finding was that with an accurate set of detections, even a relatively simple association algorithm can achieve good tracking results.

A Single-Stream CNN For Semantic Segmentation And Monocular Depth Prediction

In this task, I have developed a single-stream convolutional neural network in an encoder-decoder architecture that can output semantic segmentation and monocular depth prediction simultaneously. Such multi-task networks are great for reducing the computational overhead associated with multiple single task ones. Moreover, by training a single model to perform multiple task, generalization is improved. The distinguishing feature of my method is that a single decoder is outputting the two tasks. That is, there is no separate task-specific decoders as can be found typically in multi-task CNNs for visual perception. The challenge with having one

¹² <http://sar-lab.net/adverse-weather-dataset/>

shared encoder and multiple task-specific decoders boils down to balancing the different tasks' loss functions. The developed model avoids this altogether. An interesting future investigation is studying non-standard loss functions and their utility in solving multiple tasks simultaneously in single-stream networks.

6.2 Future Work

Research opportunities in visual perception for autonomous driving and mobile robotics are rich and rewarding. There are still many challenging perception problems to tackle. One of the main focuses of my research is the real-timenes of visual perception algorithms. This is important as the perception system can be considered the bottleneck in the control loop. This can be seen considering that the actuation part of the loop is limited by the mechanical system dynamics. This means optimizing the perception part is very important for real-time reaction in safety-critical applications.

Traditional cameras are considered *frame-based* imagers. Their principle of operation is based on taking a series of snapshots (the frames) at a constant rate. The pixels are sampled redundantly, over and over again, even if they carry no new information. Bandwidth and dynamic range are limited by the identical sampling rate and integration time [127]. This causes transmission of large amounts of data which limits update rates. Additionally, frame-based imagers suffer from motion blur. This happens when the object moves while being recorded in a frame during a single exposure.

A potential solution to such challenges comes from the field of *Neuromorphic Engineering*. Novel vision sensor devices that adopt the neuromorphic architecture of redundancy-reduced address-event output have been developed. Those event cameras output a single asynchronous digital pulse (*event*) whenever the light intensity falling on a pixel changes above a certain threshold. This sensor gives unparalleled advantage in-terms of temporal resolution, low latency, and low bandwidth motion signals.

As a first step to investigate this novel sensing modality, I have implemented a multi-cluster tracking algorithm from the event stream [128]. This work is based on the work of Lagorce et al. [129]. The spatial distribution of events generated by a moving object can be approximated by a *bivariate gaussian distribution* $\mathcal{N}(\mu, \Sigma)$. The gaussian mean μ represents the gaussian cluster or

blob position, whereas the covariance Σ provides its shape and orientation. More details about the algorithm along with experimental evaluations can be found in Ref. [128].

This sensor is promising to unlock high-speed robotics perception and enable situations that were originally unattainable using traditional frame-based vision sensors. This novel sensor opens the doors to many exciting research problems. One thing is building sophisticated higher-level multi-object trackers based on the developed gaussian blob tracker. Events could be aggregated into higher-level geometric representation and tracked accordingly. Moreover, detectors could be trained to perform the detections from these geometric representations. Additionally, as blobs are tracked continuously through time, they might be promising to perform visual odometry because the data association problem is easily solved.

Bibliography

- [1] S. O.-R. A. V. S. Committee, "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems," *SAE Standard*, 2014.
- [2] "NHSTA Automated Vehicles for Safety," [Online]. Available: <https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety>. [Accessed 26 August 2019].
- [3] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
- [4] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91-110, 2004.
- [5] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, pp. 1097-1105, 2012.
- [6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248-255, 2009.
- [7] I. E. Lopez, "Large scale semantic 3D modeling of the urban landscape," *PhD dissertation, Universiteit van Amsterdam*, 2012.
- [8] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2004.
- [9] R. Hartley and P. Sturm, "Triangulation," *Computer vision and image understanding*, vol. 68, no. 2, pp. 146-157, 1997.
- [10] N. Snavely, S. M. Seitz and R. Szeliski, "Modeling the world from internet photo collections," *International journal of computer vision*, vol. 80, no. 2, pp. 189-210, 2008.
- [11] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR), Nara, Japan*, p. 225-234, November 2007.

- [12] B. Triggs, P. F. McLauchlan, R. I. Hartley and A. W. Fitzgibbon, "Bundle Adjustment — A Modern Synthesis," *Vision Algorithms: Theory and Practice*, pp. 298-372, 1999.
- [13] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147-1163, October 2015.
- [14] M. Aladem, "Robust Real-Time Visual Odometry for Autonomous Ground Vehicles," 2017.
- [15] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning*, MIT press, 2016.
- [16] G. Cybenko, "Approximations by superpositions of sigmoidal functions," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, p. 303–314, 1989.
- [17] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251-257, 1991.
- [18] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541-551, 1989.
- [19] K. Simonyan, A. Vedaldi and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.
- [20] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *European conference on computer vision*, pp. 818-833, 2014.
- [21] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580-587, 2014.
- [22] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80-92, 2011.
- [23] M. Aladem and S. A. Rawashdeh, "Lightweight Visual Odometry for Autonomous Mobile Robots," *Sensors*, vol. 18, no. 9, p. 2837, 2018.
- [24] M. Aladem, S. Baek and S. A. Rawashdeh, "Evaluation of Image Enhancement Techniques for Vision-Based Navigation under Low Illumination," *Journal of Robotics*, 2019.

- [25] M. Maimone, Y. Cheng and L. Matthies, "Two years of visual odometry on the mars exploration rovers," *Journal of Field Robotics*, vol. 24, no. 3, pp. 169-186, 2007.
- [26] M. Pollefeys, D. Nist'er, J. M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stew'enius, R. Yang, G. Welch and H. Towles, "Detailed real-time urban 3d reconstruction from video," *International Journal of Computer Vision*, vol. 78, no. 2-3, pp. 143-167, 2008.
- [27] E. S. Jones and S. Soatto, "Visual-inertial navigation, mapping and localization: A scalable real-time causal approach," *International Journal of Robotics Research*, vol. 30, no. 4, pp. 407-430, April 2011.
- [28] H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover (No. STAN-CS-80-813)," STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1980.
- [29] D. Nister, O. Naroditsky and J. Bergen, "Visual Odometry," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2004.
- [30] B. Kitt, A. Geiger and H. Lategahn, "Visual odometry based on stereo image sequences with RANSAC-based outlier rejection scheme," *Intelligent Vehicles Symposium (IV)*, 2010.
- [31] A. Howard, "Real-time stereo visual odometry for autonomous ground vehicles," *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS.*, 2008.
- [32] H. Badino, A. Yamamoto and T. Kanade, "Visual Odometry by Multi-frame Feature Integration," *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 222-229, 2013.
- [33] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," *Ninth IEEE International Conference on Computer Vision Proceedings.*, 2003.
- [34] G. Younes, D. Asmar, E. Shammass and J. Zelek, "Keyframe-based monocular SLAM: design, survey, and future directions," *Robotics and Autonomous Systems*, vol. 98, pp. 67-88, 2017.
- [35] T. Pire, T. Fischer, G. Castro, P. De Cristóforis, J. Civera and J. J. Berlles, "S-PTAM: Stereo Parallel Tracking and Mapping," *Robotics and Autonomous Systems*, 2017.
- [36] J. Engel, T. Schöps and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM," *European Conference on Computer Vision (ECCV)*, pp. 834-849, 2014.

- [37] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox and N. Roy, "Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera," *Robotics Research. Springer, Cham.*, vol. 100, pp. 235-252, 2017.
- [38] C. Kerl, J. Sturm and D. Cremers, "Robust odometry estimation for RGB-D cameras," *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.
- [39] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239 - 256, 1992.
- [40] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, 2001.
- [41] R. A. Newcombe, S. Izadi and O. Hilliges, "KinectFusion: Real-time dense surface mapping and tracking," *10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.
- [42] T. A. Ciarfuglia, G. Costante, P. Valigi and E. Ricci, "Evaluation of non-geometric methods for visual odometry," *Robotics and Autonomous Systems*, vol. 62, no. 12, pp. 1717-1730, 2014.
- [43] S. Wang, R. Clark, H. Wen and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2043-2050, 2017.
- [44] R. Li, S. Wang, Z. Long and D. Gu, "Undeepvo: Monocular visual odometry through unsupervised deep learning," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7286-7291, 2018.
- [45] S. Wang, R. Clark, H. Wen and N. Trigoni, "End-to-end, sequence-to-sequence probabilistic visual odometry through deep neural networks," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 513-542, 2018.
- [46] E. Mair, G. D. Hager, D. Burschka, M. Suppa and G. Hirzinger, "Adaptive and generic corner detection based on the accelerated segment test," *Proceedings of the European Conference on Computer Vision (ECCV'10)*, 2010.
- [47] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *European conference on computer vision*, pp. 430-443, 2006.
- [48] M. Calonder, V. Lepetit, C. Strecha and P. Fua, "Brief: Binary robust independent elementary features," *European conference on computer vision*, pp. 778-792, 2010.

- [49] M. Brown, R. Szeliski and S. Winder, "Multi-image matching using multi-scale oriented patches," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 510-517, 2005.
- [50] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige and W. Burgard, "g2o: A general framework for graph optimization," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3607-3613, 2011.
- [51] A. Geiger, P. Lenz and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3354-3361, 2012.
- [52] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *International Journal of Robotic Research*, vol. 35, no. 10, pp. 1157-1163, 2016.
- [53] J. Sturm, N. Engelhard, F. Endres, W. Burgard and D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems," *Proceedings of the International Conference on Intelligent Robot Systems (IROS)*, October 2012.
- [54] A. Geiger, J. Ziegler and C. Stiller, "StereoScan: Dense 3D Reconstruction in Real-time," *Intelligent Vehicles Symposium (IV)*, 2011.
- [55] "Raspberry Pi," [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.
- [56] "Odroid XU4," [Online]. Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825.
- [57] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *IEEE International Conference on Computer Vision (ICCV)*, 2011.
- [58] H. Dong and T. D. Barfoot, "Lighting-Invariant Visual Odometry using Lidar Intensity Imagery and Pose Interpolation," *Field and Service Robotics*, pp. 327-342, 2014.
- [59] C. Brunner, T. Peynot, T. Vidal-Calleja and J. Underwood, "Selective Combination of Visual and Thermal Imaging for Resilient Localization in Adverse Conditions: Day and Night, Smoke and Fire," *Journal of Field Robotics*, vol. 30, no. 4, pp. 641-666, 2013.
- [60] P. Nelson, W. Churchill, I. Posner and P. Newman, "From Dusk till Dawn: Localisation at Night using Artificial Light Sources," *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

- [61] K. MacTavish, M. Paton and T. D. Barfoot, "Night Rider: Visual Odometry Using Headlights," *Computer and Robot Vision (CRV), 2017 14th Conference on*.
- [62] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras," *arXiv preprint* , 2016.
- [63] S.-C. Huang, F.-C. Cheng and Y.-S. Chiu, "Efficient contrast enhancement using adaptive gamma correction with weighting distribution," *IEEE Transactions on Image Processing*, vol. 22, no. 3, pp. 1032-1041, 2013.
- [64] S. Rahman, M. M. Rahman, M. Abdullah-Al-Wadud, G. D. Al-Quaderi and M. Shoyaib, "An adaptive gamma correction for image enhancement," *EURASIP Journal on Image and Video Processing*, vol. 2016, no. 1, p. 35, 2016.
- [65] X. Zhang, P. Shen, L. Luo, L. Zhang and J. Song, "Enhancement and Noise Reduction of Very Low Light Level Images," in *International Conference on Pattern Recognition (ICPR 2012)*, Tsukuba, Japan, 2012.
- [66] H. Malm, M. Oskarsson, E. Warrant, P. Clarberg, J. Hasselgren and C. Lejdfors, "Adaptive enhancement and noise reduction in very low light-level video," in *IEEE International Conference on Computer Vision (ICCV)*, 2007.
- [67] A. Benoit, A. Caplier, B. Durette and J. Hérault, "Using Human Visual System modeling for bio-inspired low level image processing," *Computer Vision and Image Understanding*, vol. 114, no. 7, pp. 758-773, 2010.
- [68] M. Aladem and S. A. Rawashdeh, "A Combined Vision-Based Multiple Object Tracking and Visual Odometry System," *IEEE Sensors Journal*, 2019.
- [69] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling and S. Thrun, "Towards fully autonomous driving: Systems and algorithms," *2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 163-168, 2011.
- [70] A. Dewan, T. Caselitz, G. D. Tipaldi and W. Burgard, "Motion-based detection and tracking in 3d lidar scans," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4508-4513, 2016.
- [71] R. Kaestner, J. Maye, Y. Pilat and R. Siegwart, "Generative object detection and tracking in 3d range data," *2012 IEEE International Conference on Robotics and Automation*, pp. 3075-3081, 2012.

- [72] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778, 2016.
- [73] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely connected convolutional networks," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700-4708, 2017.
- [74] W. Choi, "Near-Online Multi-target Tracking with Aggregated Local Flow Descriptor," *Proceedings of the IEEE international conference on computer vision*, pp. 3029-3037, 2015.
- [75] J. Berclaz, F. Fleuret, E. Turetken and P. Fua, "Multiple object tracking using k-shortest paths optimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 9, pp. 1806-1819, 2011.
- [76] A. A. Butt and R. T. Collins, "Multi-target tracking by lagrangian relaxation to min-cost network flow," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1846-1853, 2013.
- [77] L. Zhang, Y. Li and R. Nevatia, "Global data association for multi-object tracking using network flows," *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8, 2008.
- [78] A. Dehghan, S. Modiri Assari and M. Shah, "Gmmcp tracker: Globally optimal generalized maximum multi clique problem for multiple object tracking," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4091-4099, 2015.
- [79] G. Gündüz and T. Acarman, "A Lightweight Online Multiple Object Vehicle Tracking Method," *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 427-432, 2018.
- [80] S. Sharma, J. A. Ansari, J. K. Murthy and K. M. Krishna, "Beyond pixels: Leveraging geometry and shape cues for online multi-object tracking," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3508-3515, 2018.
- [81] J. K. Murthy, G. S. Krishna, F. Chhaya and K. M. Krishna, "Reconstructing vehicles from a single image: Shape priors for road scene understanding," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 724-731, 2017.
- [82] A. Osep, W. Mehner, M. Mathias and B. Leibe, "Combined image- and world-space tracking in traffic scenes," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1988-1995, 2017.

- [83] A. Geiger, M. Lauer, C. Wojek, C. Stiller and R. Urtasun, "3d traffic scene understanding from movable platforms," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 5, pp. 1012-1025, 2014.
- [84] D. Frossard and R. Urtasun, "End-to-end learning of multi-sensor 3d tracking by detection," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 635-642, 2018.
- [85] H. Nam and B. Han, "Learning multi-domain convolutional neural networks for visual tracking," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4293-4302, 2016.
- [86] P. Bergmann, T. Meinhardt and L. Leal-Taixe, "Tracking without bells and whistles," *arXiv preprint arXiv:1903.05625*, 2019.
- [87] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv*, 2018.
- [88] R. Girshick, "Fast R-CNN," *International Conference on Computer Vision (ICCV)*, 2015.
- [89] J. Shi and C. Tomasi, "Good features to track," 1993.
- [90] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83-97, 1955.
- [91] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32--38, 1957.
- [92] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328-341, 2008.
- [93] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the CLEAR MOT metrics," *Journal on Image and Video Processing*, vol. 2008, p. 1, 2008.
- [94] Y. Li, C. Huang and R. Nevatia, "Learning to associate: Hybridboosted multi-target tracker for crowded scene," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2953-2960, 2009.
- [95] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, "Microsoft coco: Common objects in context," *European conference on computer vision*, pp. 740-755, 2014.

- [96] J. Ren, X. Chen, J. Liu, W. Sun, J. Pang, Q. Yan, Y.-W. Tai and L. Xu, "Accurate single stage detector using recurrent rolling convolution," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5420-5428, 2017.
- [97] X. Wang, M. Yang, S. Zhu and Y. Lin, "Regionlets for generic object detection," *Proceedings of the IEEE international conference on computer vision*, pp. 17-24, 2013.
- [98] M. Aladem and S. A. Rawashdeh, "A Single-Stream Segmentation and Depth Prediction CNN for Autonomous Driving," *IEEE Intelligent Systems*, to be published.
- [99] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "Ssd: Single shot multibox detector," *European conference on computer vision*, pp. 21-37, 2016.
- [100] V. Badrinarayanan, A. Kendall and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481-2495, 2017.
- [101] A. Ranjan and M. J. Black, "Optical flow estimation using a spatial pyramid network," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4161-4170, 2017.
- [102] Z. Chen and Z. Chen, "Rbnet: A deep neural network for unified road and road boundary detection," *International Conference on Neural Information Processing*, pp. 677-687, 2017.
- [103] H. Fu, M. Gong, C. Wang, K. Batmanghelich and D. Tao, "Deep ordinal regression network for monocular depth estimation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2002-2011, 2018.
- [104] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41-75, 1997.
- [105] S. Chennupati, G. Sistu, S. Yogamani and S. A Rawashdeh, "MultiNet++: Multi-Stream Feature Aggregation and Geometric Loss Strategy for Multi-Task Learning," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019.
- [106] A. Saxena, S. H. Chung and A. Y. Ng, "Learning depth from single monocular images," *Advances in neural information processing systems*, pp. 1161-1168, 2006.
- [107] D. Eigen, C. Puhrsch and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," *Advances in neural information processing systems*, pp. 2366-2374, 2014.

- [108] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy and T. Brox, "Demon: Depth and motion network for learning monocular stereo," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5038-5047, 2017.
- [109] F. Liu, C. Shen, G. Lin and I. Reid, "Learning depth from single monocular images using deep convolutional neural fields," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 2024-2039, 2016.
- [110] I. Alhashim and P. Wonka, "High Quality Monocular Depth Estimation via Transfer Learning," *arXiv preprint arXiv:1812.11941*, 2018.
- [111] J. Shotton, M. Johnson and R. Cipolla, "Semantic texton forests for image categorization and segmentation," *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8, 2008.
- [112] P. Sturgess, K. Alahari, L. Ladicky and P. H. Torr, "Combining appearance and structure from motion features for road scene understanding," 2009.
- [113] M. Siam, S. Elkerdawy, M. Jagersand and S. Yogamani, "Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges," *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1-8, 2017.
- [114] J. Long, E. Shelhamer and T. Darrell, "Fully convolutional networks for semantic segmentation," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431-3440, 2015.
- [115] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox and O. Ronneberger, "3D U-Net: learning dense volumetric segmentation from sparse annotation," *International conference on medical image computing and computer-assisted intervention*, pp. 424-432, 2016.
- [116] Y. Zhang and Q. Yang, "A survey on multi-task learning," *arXiv preprint arXiv:1707.08114*, 2017.
- [117] H. Bilen and A. Vedaldi, "Universal representations: The missing link between faces, text, planktons, and cat breeds," *arXiv preprint arXiv:1701.07275*, 2017.
- [118] I. Kokkinos, "Ubertnet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6129-6138, 2017.
- [119] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla and R. Urtasun, "Multinet: Real-time joint semantic reasoning for autonomous driving," *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1013-1020, 2018.

- [120] M. Guo, A. Haque, D.-A. Huang, S. Yeung and L. Fei-Fei, "Dynamic task prioritization for multitask learning," *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 270-287, 2018.
- [121] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *Advances in Neural Information Processing Systems*, pp. 527-538, 2018.
- [122] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510-4520, 2018.
- [123] A. L. Maas, A. Y. Hannun and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," *ICML*, vol. 30, no. 1, 2013.
- [124] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213-3223, 2016.
- [125] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [126] K. He, X. Zhang, S. Ren and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *Proceedings of the IEEE international conference on computer vision*, pp. 1026-1034, 2015.
- [127] T. Delbruck and P. Lichtsteiner, "Freeing vision from frames," *The Neuromorphic Engineer*, vol. 3, no. 1, pp. 3-4, 2006.
- [128] M. Aladem and S. A. Rawashdeh, "A Multi-Cluster Tracking Algorithm with an Event Camera," *2019 IEEE National Aerospace and Electronics Conference (NAECON)*, 2019.
- [129] X. Lagorce, C. Meyer, S.-H. Ieng, D. Filliat and R. Benosman, "Asynchronous event-based multikernel algorithm for high-speed visual features tracking," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 8, pp. 1710-1720, 2015.