Mobility21

A USDOT NATIONAL UNIVERSITY TRANSPORTATION CENTER

Carnegie Mellon University

Penn
University of Pennsylvania

The Ohio State University

CCAC

Image Processing Approaches to Traffic Situation Understanding, Risk Assessment, and Safety

Keith Redmill (PI) (https://orcid.org/0000-0003-1332-1332)
Ekim Yurtsever (https://orcid.org/0000-0002-3103-6052)
Dongfang Yang (https://orcid.org/0000-0001-9212-6804)
Linda Capito (https://orcid.org/0000-0002-9871-1243)
Haolin Zhang (https://orcid.org/0000-0002-8549-1974)
Ümit Özgüner (https://orcid.org/0000-0003-2241-7547)

FINAL RESEARCH REPORT - July 28, 2023

Contract # 69A3551747111

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Increasing the mobility of people and goods via automated transportation requires robust and safe intelligent vehicle systems. Computer vision is one field that can create new frontiers towards this end. This project explored several potential applications of image processing, including both traditional and neural network and deep learning image processing technologies, to automated vehicle sensing and control, traffic scene analysis, pedestrian detection and intention estimation, and improved image synthesis for automated vehicle simulation and testing.

Initially, we proposed three specific potential applications:

1. Investigating the safety and robustness of end-to-end vision-based automated driving systems,
2. Exploring optical flow techniques for automated vehicle control, and
3. Methodologies for extracting information from transit vehicle video for traffic count and flow estimation.

As the project progressed, we pursued additional applications including

4. Applying vision-based sensing to improve safety during COVID-19 by monitoring spacing and crowd densities,
5. Fusing image and lidar processing to improve the detection of pedestrians at longer distances,
6. Improving the prediction of pedestrian crossing intention estimation, and
7. Synthesizing and rendering more realistic imagery in driving simulators using a generative approach.

This report details each of these activities, describing the specific problem addressed, providing a survey of relevant literature and current best practices, exploring the proposed solution and its implementation, and presenting the results and outcomes of each study. This chapter will provide an introduction and overview of each of the seven problems explored in this project, and future chapters will individually describe each activity in detail. Information regarding available source code or datasets is given in the concluding chapter.

1.1 Integrating Deep Reinforcement Learning with Model-based Path Planners for Automated Driving

Automated driving in urban settings is challenging. Human participant behavior is difficult to model, and conventional, rule-based Automated Driving Systems (ADSs) tend to fail when they

face unmodeled dynamics. On the other hand, the more recent, end-to-end Deep Reinforcement Learning (DRL) based model-free ADSs have shown promising results. However, pure learning-based approaches lack the hard-coded safety measures of model-based controllers. Here we proposed a hybrid approach for integrating a path planning pipe into a vision based DRL framework to alleviate the shortcomings of both worlds. Our overall research objectives were to develop novel algorithms to integrate planning into deep reinforcement learning frameworks.

In summary, the DRL agent is trained to follow the path planner's waypoints as close as possible. The agent learns this policy by interacting with the environment. The reward function contains two major terms: the penalty of straying away from the path planner and the penalty of having a collision. The latter has precedence in the form of having a significantly greater numerical value. Experimental results show that the proposed method can plan its path and navigate between randomly chosen origin-destination points in CARLA, a dynamic urban simulation environment. Our code is open-source and available online at: `https://github.com/Ekim-Yurtsever/Hybrid-DeepRL-Automated-Driving`.

This activity was originally published in [1].

1.2 Optical Flow Based Visual Potential Fields for Automated Driving

Monocular vision based navigation for automated driving is a challenging task due to the lack of sufficient information to compute temporal relationships among objects on the road. Optical flow is an option to obtain temporal information from monocular camera images, and has been used widely with the purpose of identifying objects and their relative motion. This work proposed to generate an artificial potential field, i.e. visual potential field, from a sequence of images using sparse optical flow, which was used together with a gradient tracking sliding mode controller to navigate the vehicle to its destination without collision with obstacles. The angular reference for the vehicle is computed online.

Topics of interest included optical flow and focus of expansion, road and road boundary potential field design, and gradient tracking sliding mode controller for lateral and longitudinal control. This work assumed that the vehicle does did not have *a priori* information from the map or obstacles to navigate successfully. The proposed technique is tested both in synthetic and real images.

This activity was originally published in [2].

1.3 Automated Traffic Surveillance using Existing Cameras on Transit Buses

Millions of commuters face congestion as a part of their daily routines. Mitigating traffic congestion requires effective transportation planning, design, and management. Accurate traffic data are needed for informed decision making. As such, operating agencies deploy fixed-location and often temporary detectors on public roads to count passing vehicles. This traffic flow measurement is key to estimating demand throughout the network. However, fixed-location detectors are spatially sparse and do not cover the entirety of the road network, and temporary detectors are temporally sparse, providing often only a few days of measurements every few years.

Transit vehicles cover a nontrivial fraction of the road network as they follow their assigned routes and have the potential to act as traffic probes providing observations distributed over significant periods of time and space. Many transit systems have installed cameras and video capture

systems that, as a byproduct of their primary monitoring and safety functions, also capture traffic scenes and the presence and motion of vehicles, bicycles, and pedestrians surrounding the transit vehicle.

Against this backdrop, previous studies proposed that public transit bus fleets could be used as surveillance agents if additional sensors were installed, and the viability and accuracy of this methodology was established by manually processing video imagery recorded by cameras mounted on transit buses. In this work, we proposed to operationalize this traffic surveillance methodology for practical applications, leveraging the perception and localization sensors already deployed on these vehicles. We present an automatic, vision-based vehicle counting method applied to the video imagery recorded by cameras mounted on transit buses. Video data from in-service transit vehicles was provided by the Ohio State University (OSU) Campus Area Bus Service (CABS) operated by the OSU Traffic and Transportation Management (TTM) department. We expect the developments and applications of this activity to allow us to deliver useful information to our stakeholder (OSU TTM) and to demonstrate the potential for large scale implementation.

First, a state-of-the-art 2D deep learning model detects objects frame by frame. Then, detected objects are tracked with the commonly used SORT method. The proposed counting logic converts tracking results to vehicle counts and real-world birds-eye-view trajectories. Using multiple hours of real-world video imagery obtained from in-service transit buses, we demonstrate that the proposed system can detect and track vehicles, distinguish parked vehicles from traffic participants, and count vehicles bidirectionally. Through an exhaustive ablation study and analysis under various weather conditions, it is shown that the proposed method can achieve high-accuracy vehicle counts.

This activity was originally published in [3].

1.4 A Vision based Social Distancing and Critical Density Detection System for COVID-19

Social distancing (SD) is an effective measure to prevent the spread of the infectious Coronavirus Disease 2019 (COVID-19). However, a lack of spatial awareness may cause unintentional violations of this new measure. Against this backdrop, we proposed an active surveillance system to slow the spread of COVID-19 by warning individuals in a region-of-interest.

Our contribution is twofold. First, we introduce a vision-based real-time system that can detect SD violations and send non-intrusive audio-visual cues using state-of-the-art deep-learning models. Second, we define a novel critical social density value and show that the chance of SD violation occurrence can be held near zero if the pedestrian density is kept under this value. The proposed system is also ethically fair: it does not record data nor target individuals, and no human supervisor is present during the operation. The proposed system was evaluated across real-world datasets.

This activity was originally published in [4].

1.5 Faraway-Frustum: Dealing with Lidar Sparsity for 3D Object Detection using Fusion with Vision

Learned pointcloud representations do not generalize well with an increase in distance to the sensor. For example, at a range greater than 60 meters, the sparsity of lidar pointclouds reaches a point where even humans cannot discern object shapes from each other. However, this distance should not be considered very far for fast-moving vehicles: a vehicle can traverse 60 meters in under

two seconds while moving at 70 mph. For safe and robust driving automation, acute 3D object detection at these ranges is indispensable.

Against this backdrop, we introduced faraway-frustum: a novel fusion strategy for detecting faraway objects. The main strategy is to depend solely on the 2D vision sensor for recognizing and classifying an object, as object shape does not change drastically with an increase in depth, and use pointcloud data for object localization in the 3D space for faraway objects. For closer objects, we use learned pointcloud representations instead, following state-of-the-art practices. This strategy alleviates the main shortcoming of object detection with learned pointcloud representations. Experiments on the KITTI dataset demonstrate that our method outperforms state-of-the-art methods by a considerable margin for faraway object detection in bird's-eye view and 3D. Our code is open-source and publicly available: `https://github.com/dongfang-steven-yang/faraway-frustum`.

This activity was originally published in [5].

1.6 Predicting Pedestrian Crossing Intention With Feature Fusion and Spatio-Temporal Attention

Predicting vulnerable road user behavior is an essential prerequisite for deploying Automated Driving Systems (ADS) in the real-world. Pedestrian crossing intention should be recognized in real-time, especially for urban driving. Recent works have shown the potential of using vision-based deep neural network models for this task. However, these models are not robust and certain issues still need to be resolved. First, the global spatio-temporal context that accounts for the interaction between the target pedestrian and the scene has not been properly utilized. Second, the optimal strategy for fusing different sensor data has not been thoroughly investigated.

This work addressed the above limitations by introducing a novel neural network architecture to fuse inherently different spatio-temporal features for pedestrian crossing intention prediction. We fuse different phenomena such as sequences of RGB imagery, semantic segmentation masks, and ego-vehicle speed in an optimal way using attention mechanisms and a stack of recurrent neural networks. The optimal architecture was obtained through exhaustive ablation and comparison studies. Extensive comparative experiments on the JAAD and PIE pedestrian action prediction benchmarks demonstrate the effectiveness of the proposed method, where state-of-the-art performance was achieved. Our code is open-source and publicly available: `https://github.com/OSU-Haolin/Pedestrian_Crossing_Intention_Prediction`.

This activity was originally published in [6].

1.7 Photorealism in Driving Simulations: Blending Generative Adversarial Image Synthesis With Rendering

Driving simulators play a large role in developing and testing new intelligent vehicle systems. The visual fidelity of the simulation is critical for building vision-based algorithms and conducting human driver experiments. Low visual fidelity breaks immersion for human-in-the-loop driving experiments. Conventional computer graphics pipelines use detailed 3D models, meshes, textures, and rendering engines to generate 2D images from 3D scenes. These processes are labor-intensive, and they do not generate photorealistic imagery.

In this work we developed a hybrid generative neural graphics pipeline for improving the visual fidelity of driving simulations. Given a 3D scene, we partially-render only important objects of interest, such as vehicles, and use generative adversarial processes to synthesize the background

and the rest of the image. To this end, we proposed a novel image formation strategy to form 2D semantic images from 3D scenery consisting of simple object models without textures. These semantic images are then converted into photorealistic RGB images with a state-of-the-art Generative Adversarial Network (GAN) trained on real-world driving scenes. This replaces repetitiveness with randomly generated but photorealistic surfaces. Finally, the partially-rendered and GAN synthesized images are blended with a blending GAN.

We show that the photorealism of images generated with the proposed method is more similar to real-world driving datasets such as Cityscapes and KITTI than conventional approaches. This comparison is made using semantic retention analysis and Frechet Inception Distance (FID) measurements.

This activity was originally published in [7].

1.8 Acknowledgements

In addition to the name authors of this report, we wish to acknowledge the contributions of Ibrahim Mert Koc and Vishnu Renganathan, former graduate students in the Department of Electrical and Computer Engineering at The Ohio State University, and Rabi Mishalani, Benjamin Coifman, and Mark McCord, faculty members in the Department of Civil, Environmental, and Geodetic Sciences at The Ohio State University.

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, under award number 69A3551747111 for Mobility21: the National University Transportation Center for Improving Mobility, a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

Chapter 2

Integrated Deep Reinforcement Learning with Model-Based Path Planners for Automated Driving

2.1 Introduction

Automated Driving Systems (ADSs) promise a decisive answer to the ever-increasing transportation demands. However, widespread deployment is not on the horizon as state-of-the-art is not robust enough for urban driving. The recent Uber accident [8] is an unfortunate precursor: the technology is not ready yet.

There are two common ADS design choices [9]. The first one is the more conventional, model-based, modular pipeline approach [10, 11, 12, 13, 14, 15, 16, 17]. A typical pipe starts with a perception module. Robustness of perception modules has been increased greatly due to the recent advent of deep Convolutional Neural Networks (CNN) [18]. The pipe usually continues with scene understanding [19], assessment [20], planning [21] and finally ends with motor control. The major shortcomings of modular model-based planners can be summarized as complexity, error propagation, and lack of generalization outside pre-postulated model dynamics.

The alternative end-to-end approaches [22, 23, 24, 25, 26, 27, 28, 29, 30, 31] eliminated the complexity of conventional modular systems. With the recent developments in the machine learning field, sensory inputs now can directly be mapped to an action space. Deep Reinforcement Learning (DRL) based frameworks can learn to drive from front-facing monocular camera images directly [28]. However, the lack of hard-coded safety measures, interpretability, and direct control over path constraints limit the usefulness of these methods.

We propose a hybrid methodology to mitigate the drawbacks of both approaches. In summary, the proposed method integrates a short pipeline of localization and path planning modules into a DRL driving agent. The training goal is to teach the DRL agent to oversee the planner and follow it if it is safe to follow. The proposed method was implemented with a Deep Q Network (DQN) [32] based RL agent and the A* [33] path planner. First, the localization module outputs the ego-vehicle position. With a given destination point, the path planner uses the A* algorithm [33] to generate a set of waypoints. The distance to the closest waypoint, along with monocular camera images and ego-vehicle dynamics, are then fed into the DQN based RL agent to select discretized steering and acceleration actions. During training, the driving agent is penalized for making collisions and being far from the closest waypoint asymmetrically, with the former term having precedence. We believe this can make the agent prone to follow waypoints during free driving but have enough flexibility to

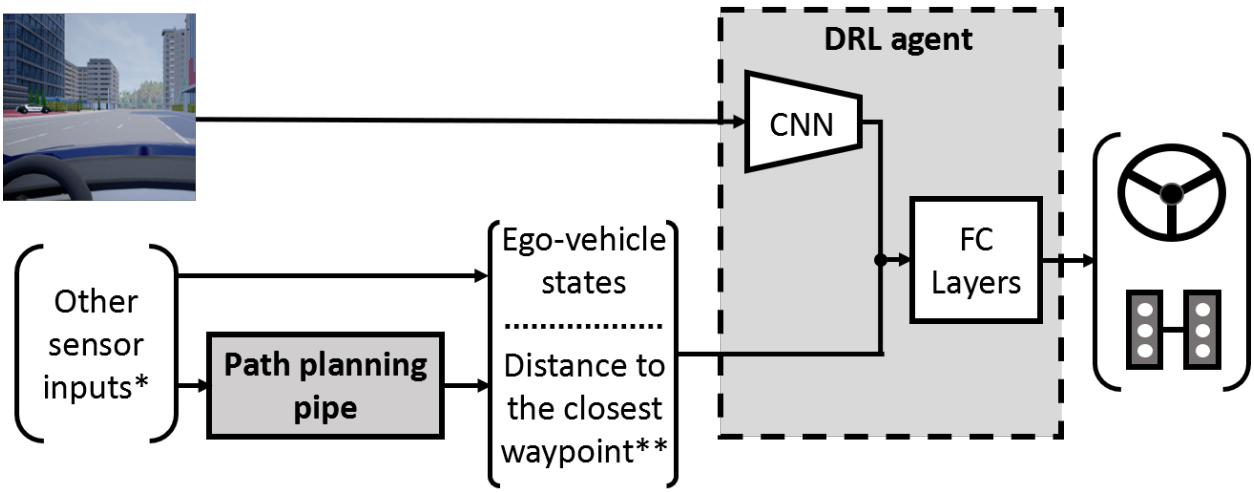


Figure 2.1: An overview of our framework. FC stands for Fully Connected layers. The proposed system is a hybrid of a model-based planner and a model-free DRL agent. *Other sensor inputs can be anything the conventional pipe needs. ** We integrate planning into the DRL agent by adding 'distance to the closest waypoint' into our state-space, where the path planner gives the closest waypoint. Any kind of path planner can be integrated into the DRL agent with the proposed method.

stray from the path for collision avoidance using visual cues. An overview of the proposed approach is shown in Figure 2.1.

The major contributions of this work can be summarized as follows:

- A general framework for integrating path planners into model-free DRL based driving agents
- Implementation of the proposed method with an A* planner and a DQN RL agent. Our code is open-source and available online at https://github.com/Ekim-Yurtsever/Hybrid-DeepRL-Automated-Driving.

The remainder of this chapter is organized in five sections. A brief literature survey is given in Section 2.2. Section 2.3 explains the proposed methodology and is followed by experimental details in Section 2.4. Results are discussed in Section 2.5 and a short conclusion is given in Section 2.6.

2.2 Related Works

End-to-end driving systems use a single algorithm/module to map sensory inputs to an action space. ALVINN [23] was the first end-to-end driving system and utilized a shallow, fully connected neural network to map image and laser range inputs to a discretized direction space. The network was trained in a supervised fashion with labeled simulation data. More recent studies employed real-world driving data and used convolutional layers to increase performance [25]. However, real-world urban driving has not been realized with an end-to-end system yet.

A CNN based partial end-to-end system was introduced to map the image space to a finite set of intermediary "affordance indicators" [22]. A simple controller logic was then used to generate driving actions from these affordance indicators. Chauffer Net [34] is another example of a mid-to-mid system. These systems benefit from robust perception modules on the one end, and rule-based controllers with hard-coded safety measures on the other end.

All the methods mentioned above suffer from shortcomings of supervised learning—namely, a significant dependency on labeled data, overfitting, and lack of interpretability. Deep Reinforcement Learning (DRL) based automated driving agents [27, 28] replaced the need for huge amounts of labeled data with online interaction. DRL agents try to learn the optimum way of driving instead of imitating a target human driver. However, the need for interaction raises a significant issue. Since failures cannot be tolerated for safety-critical applications, in almost all cases, the agent must be trained in a virtual environment. This adds the additional virtual-to-real transfer learning problem to the task. In addition, DRL still suffers from a lack of interpretability and hard-coded safety measures.

A very recent study [35] focused on general tactical decision making for automated driving using the AlphaGo Zero algorithm [36]. AlphaGo Zero combines tree-search with neural networks in a reinforcement learning framework, and its implementation to the automated driving domain is promising. However, this study [35] was limited to only high-level tactical driving actions such as staying on a lane or making a lane change.

Against this backdrop, here we propose a hybrid DRL-based driving automation framework. The primary motivation is to integrate path-planning into DRL frameworks for achieving a more robust driving experience and a faster learning process.

2.3 Proposed Method

2.3.1 Problem formulation

In this study, automated driving is defined as a Markov Decision Process (MDP) with the tuple of (S, A, P, r). We integrate path-planning into the MDP by adding d, distance to the closest waypoint, to the state-space.

S: A set of states. We associate observations made at time t with the state s_t as $s_t \simeq (z_t, e_t, d_t)$ where; 1) $z_t = f_{cnn}(I_t)$ is a visual feature vector which is extracted using a deep CNN from a single image I_t captured by a front-facing monocular camera. 2) e_t is a vector of ego-vehicle states including speed and location 3) d_t is the distance to the closest waypoint obtained from the model-based path planner. d_t is the key observation which links model-based path planners to the MDP.

A: A set of discrete driving actions illustrated in Figure 2.3. Actions consist of discretized steering angle and acceleration values. The agent executes actions to change states.

P: The transition probability $P_t = \Pr(s_{t+1}|s_t, a_t)$. Which is the probability of reaching state s_{t+1} after executing action a_t in state s_t.

r: A reward function $r(s_{t+1}, s_t, a_t)$. Which gives the instant reward of going from state s_t to s_{t+1} with a_t.

The goal is to find a policy function $\pi(s_t) = a_t$ that will select an action given a state such that it will maximize the following expectation of cumulative future rewards where s_{t+1} is taken from P_t.

$$\mathbb{E} \quad \sum_{t=0}^{\infty} \left(\gamma^t r(s_t, s_{t+1}, a_t) \right) \left(\right. \tag{2.1}$$

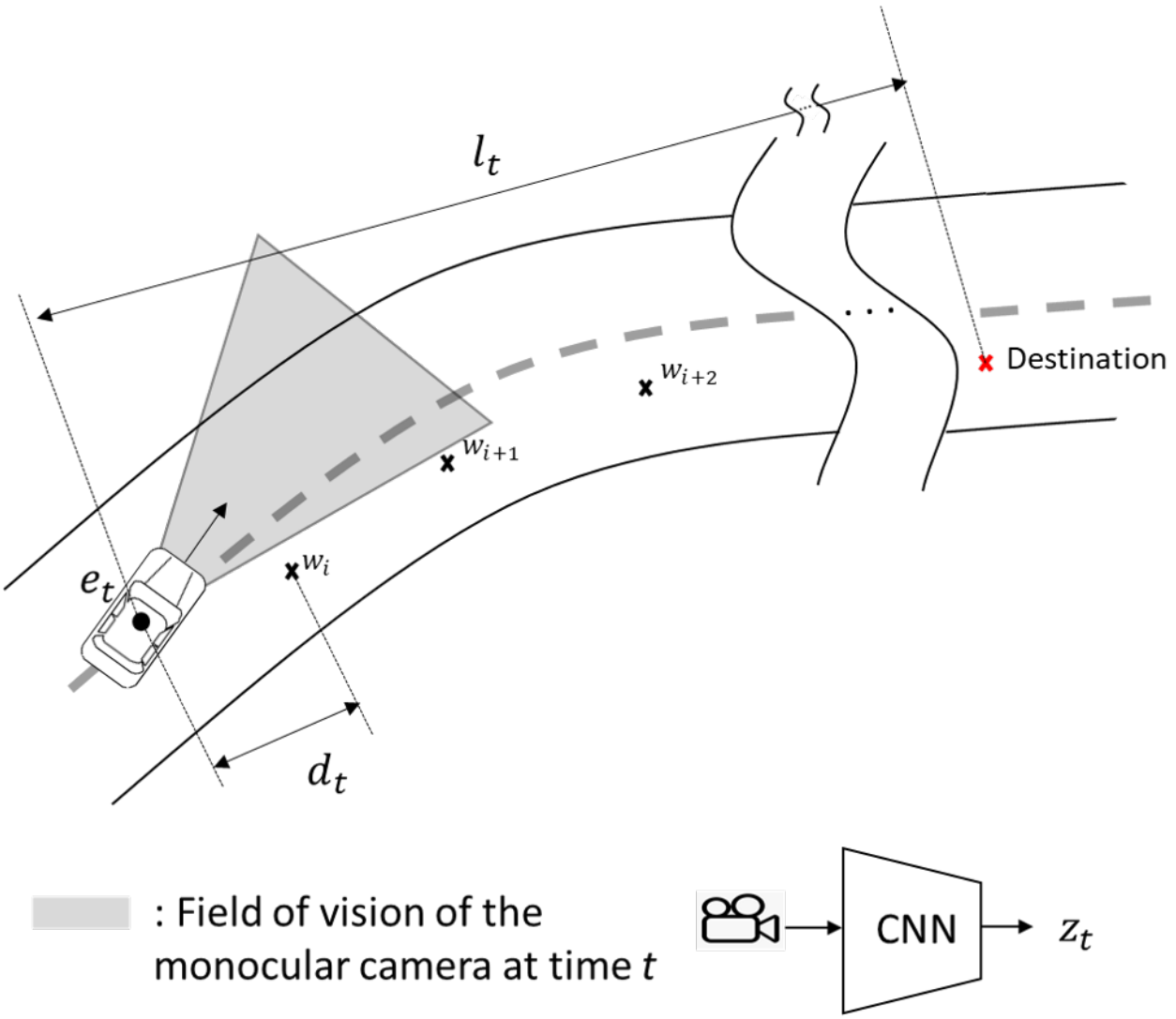


Figure 2.2: Illustration of state $s_t \simeq (z_t, e_t, d_t)$ and distance to the final destination l_t at time t. Waypoints $w \in W$ are to be obtained from the path planner.

Where γ is the discount factor, which is a scalar between $0 \leq \gamma \leq 1$ that determines the relative importance of later rewards with respect to previous rewards. We fix the horizon for this expectation with a finite value in practice.

Our problem formulation is similar to a previous study [28], the critical difference being the addition of d_t to the state space and the reward function. An illustration of our formulation is shown in Figure 2.2.

2.3.2 Reinforcement Learning

Reinforcement learning is an umbrella term for a large number of algorithms derived for solving the Markov Decision Problems (MDP) [28].

In our framework, the objective of reinforcement learning is to train a driving agent who can execute 'good' actions so that the new state *and* possible state transitions until a finite expectation horizon will yield a high cumulative reward. The overall goal is quite straightforward for driving:

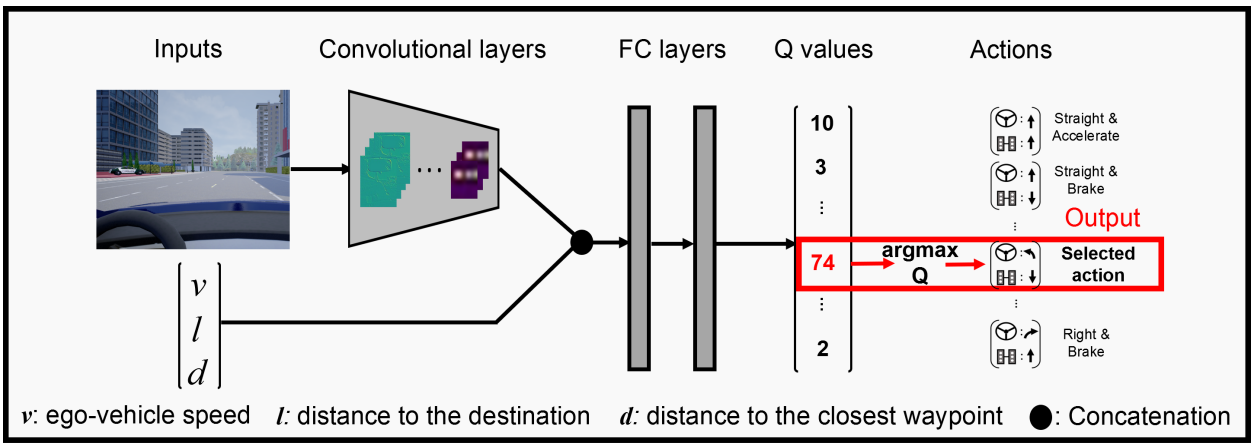


Figure 2.3: The DQN based DRL agent. FC stands for fully connected. After training, the agent selects the best action by taking the argmax of predicted Q values.

not making collisions and reaching the destination should yield a good reward and vice versa. It must be noted that RL frameworks are not greedy unless $\gamma = 0$. In other words, when an action is chosen, not only the immediate reward but the cumulative rewards of all the expected future state transitions are considered.

Here we employ DQN [32] to solve the MDP problem described above. The main idea of DQN is to use neural networks to approximate the optimal action-value function $Q(s, a)$. This Q function maps the state-action space to $\mathbb{R}$. $Q : S \times A \rightarrow \mathbb{R}$ while maximizing equation 2.1. The problem comes down to approximiate or to learn this Q function. The following loss function is used for Q-learning at iteration i.

$$L_i(\theta) = \mathbb{E}_{(s,a,r)}\left[\left(r + \gamma \max_{a_{t+1}} Q^{\theta_i^-}(s_{t+1}, a_{t+1}) - Q^{\theta_i}(s_t, a_t)\right)^2\right] \tag{2.2}$$

Where Q-Learning updates are applied on samples $(s, a, r) \sim U(D)$. $U(D)$ draws random samples from the data batch D. θ_i is the Q-network parameters and θ_i^- is the target network parameters at iteration i. Details of DQN can be found in [32].

2.3.3 Integrating path planning into model-free DRL frameworks

The main contribution of this work is the integration of path planning into DRL frameworks. We achieve this by modifying the state-space with the addition of d. Also, the reward function is changed to include a new reward term r_w, which rewards being close to the nearest waypoint obtained from the model-based path planner, i.e. a small d. Utilizing waypoints to evaluate a DRL framework were suggested in a very recent work [37], but their approach does not consider integrating the waypoint generator into the model.

The proposed reward function is as follows.

$$r = \beta_c r_c + \beta_v r_v + \beta_l r_l + \beta_w r_w \tag{2.3}$$

Where r_c is the no-collision reward, r_v is the not driving very slow reward, r_l is being-close to the destination reward, and r_w is the proposed being-close to the nearest waypoint reward. The distance to the nearest waypoint d is shown in Figure 2.2. The weights of these rewards, $\beta_c, \beta_v \beta_l, \beta_w$, are parameters defining the relative importance of rewards. These parameters are

determined heuristically. In the special case of $\beta_c = \beta_v = \beta_l = 0$, the integrated model should mimic the model-based planner.

Please note that any planner, from the naive A* to more complicated algorithms with complete obstacle avoidance capabilities, can be integrated into this framework as long as they provide a waypoint.

2.4 Experiments

As in all RL frameworks, the agent needs to interact with the environment and fail a lot to learn the desired policies. This makes training RL driving agents in real-world extremely challenging as failed attempts cannot be tolerated. As such, we focused only on simulations in this study. Real-world adaptation is outside of the scope of this work.

The proposed method was implemented in Python based on an open-source RL framework [38] and CARLA [39] was used as the simulation environment. The commonly used A* algorithm [33] was employed as the model-based path planner, and the recently proposed DQN [32] was chosen as the model-free DRL.

2.4.1 Details of the reward function

The general form of r was given in the previous Section in equation 2.3. Here, the special case and numerical values used throughout the experiments are explained.

$$r = \begin{cases} r_v + r_l + r_w & , r_c = 0 \ \& \ l \geq \epsilon \\ 100 & , r_c = 0 \ \& \ l < \epsilon \\ r_c & , r_c \neq 0 \end{cases} \tag{2.4}$$

$$r_c = \begin{cases} 0 & , \text{no collision} \\ -1 & , \text{there is a collsion} \end{cases} \tag{2.5}$$

$$r_v = \frac{1}{v_0} v - 1 \tag{2.6}$$

$$r_l = 1 - \frac{l}{l_{\text{previous}}} \tag{2.7}$$

$$r_w = 1 - \frac{d}{d_0} \tag{2.8}$$

Where $\epsilon = 5m$, the desired speed $v_0 = 50km/h$, and $d_0 = 8m$. In summary, r_w rewards keeping a distance less than d_0 to the closest waypoint at every time step, and r_l rewards decreasing l over l_{previous}, distance to the destination in the the previous time step. The last term of r_l allows to continuously penalize/reward the agent for getting further/closer to the final destination.

If there is a collision, the episode is over and the reward gets a penalty equal to -1. If the vehicle reaches its destination $\exists \epsilon > 0 : l < \epsilon$, a reward of 100 is sent back. Otherwise, the reward consists of the sum of the other terms. d_0 was selected as $8m$ because the average distance between waypoints of the A* equals to this value.

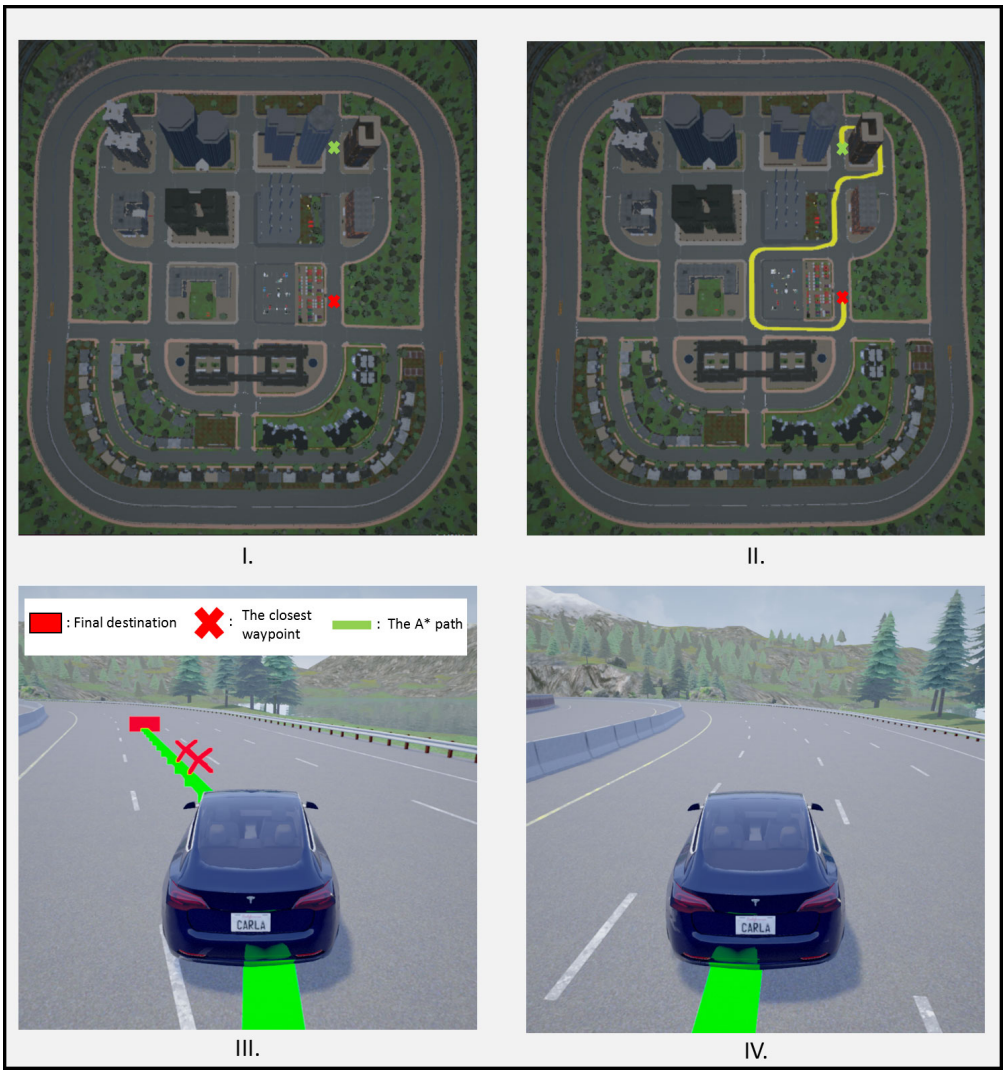


Figure 2.4: The experimental process: I. A random origin-destination pair was selected. II. The A* algorithm was used to generate a path. III. The hybrid DRL agent starts to take action with the incoming state stream. IV. The end of the episode.

2.4.2 DQN architecture and hyperparameters

The deep neural network architecture employed in the DQN is shown in Figure 2.3. The CNN consisted of three identical convolutional layers with 64 filters and a 3×3 window. Each convolutional layer was followed by average pooling. After flattening, the output of the final convolutional layer, ego-vehicle speed and distance to the closest waypoint were concatenated and fed into a stack of two fully connected layers with 256 hidden units. All but the last layer had rectifier activation functions. The final layer had a linear activation function and outputed the predicted Q values, which were used to choose the optimum action by taking argmax.
Q

2.4.3 Experimental process & training

The experimental process is shown in Figure 2.4. The following steps were carried repeatedly until the agent learned to drive.

1. Select two random points on the map as an origin-destination pair for each episode
2. Use A* path planner to generate a path between origin-destination using the road topology graph of CARLA.
3. Start feeding the stream of states, including distance to the closest waypoint, into the DRL agent. DRL agent starts to take actions at this point. If this is the first episode, initialize the DQN with random weights.
4. End the episode if a collision is detected, or the goal is reached.
5. Update the weights of the DQN after each episode with the loss function given in equation 2.2.
6. Repeat the above steps sixty thousand times

2.4.4 Comparision and evaluation

The proposed hybrid approach was compared against a complete end-to-end DQN agent. The complete end-to-end agent took only monocular camera images and ego-vehicle speed as input. The same network architecture was employed for both methods.

A human driving experiment was also conducted to serve as a baseline. The same reward function that was used to train the DRL agent was used as the evaluation metric. Four adults aging between 25 to 30 years old participated in the experiments. The participants drove a virtual car in CARLA using a keyboard and were told to follow the on-screen path (marked by a green line). The participants did not see their scores. Every participant drove each of the seven predefined routes five times. The average cumulative reward of each route was accepted as the "average human score."

2.5 Results

Figure 2.5 illustrates the training process. The result is clear and evident: The proposed hybrid approach learned to drive much faster than its complete end-to-end counterpart. It should be noted that the proposed approach made a quick jump at the beginning of the training. We believe the waypoints acted as a 'guide' and made the algorithm learn faster that way. Our method can be used for spooling up the training process of a complete end-to-end variant with transfer learning. Qualitative analysis of the driving performance can be done by watching the simulation videos on our repository[2.1].

The proposed method outperformed the end-to-end DQN, however, it is still not good as the average human driver as can be seen in Table 2.1.

Even though promising results were obtained, the experiments at this stage can only be considered as proof of concepts, rather than an exhaustive evaluation. The proposed method needs to consider other integration options, be compared against other state-of-the-art agents, and eventually should be deployed to the real-world and tested there.

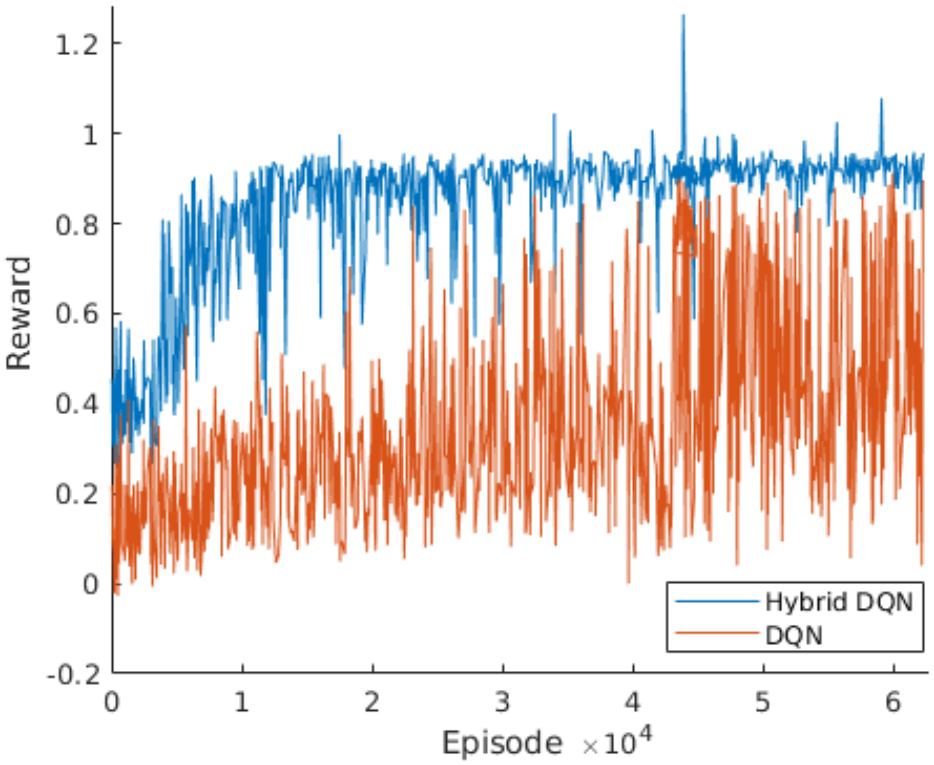


Figure 2.5: Normalized reward versus episode number. The proposed hybrid approach learned to drive faster than its complete end-to-end counterpart.

The model-based path planner tested here is also very naive. In addition, the obstacle avoidance capabilities of the proposed method was not evaluated. Future experiments should focus on this aspect. The integration of more complete path planners with full obstacle avoidance capabilities can yield better results.

Table 2.1: Average reward scores for five runs in each route type.

Route type	**Hybrid-DQN**	**Human average**
Straight (highway)	21.1	43.4
Straight (urban)	27.6	38.1
Straight (under bridge)	31.6	45.2
Slight curve	30.4	49.5
Sharp curve	-74.4	-8.9
Right turn in intersection	-136.9	-12.1
Left turn in intersection	-385.9	-25.5

2.6 Conclusions

In this study, a novel hybrid approach for integrating path planning into model-free DRL frameworks was proposed. A proof-of-concept implementation and experiments in a virtual environment showed that the proposed method is capable of learning to drive.

The proposed integration strategy is not limited to path planning. Potentially, the same state-space modification and reward strategy can be applied for integrating vehicle control and trajectory planning modules into model-free DRL agents.

Finally, the current implementation was limited to output only discretized actions. Future work will focus on enabling continuous control and real-world testing.

Chapter 3

Optical Flow Based Potential Field for Autonomous Driving

3.1 Introduction

Vision based vehicular navigation with a single camera has been shown to be a challenging problem within the field of computer vision, mainly because it is hard to obtain a force to control the vehicle just from monocular images [40, 41]. To obtain this force, we can compute a so called visual potential field, which is an approximation of the projection of the potential field in the 3D world to the image plane. This technique has been used with success for Unmanned Aerial Vehicles (UAVs) [42] and it has also been applied to robots, in environments which are basically restriction free, i.e. the robot can move to whichever direction it finds suitable [43].

Navigation for cars is different from UAVs and mobile robots in the sense that a car has additional constraints from the road, like lane boundaries. Then, it is necessary to create those restrictions from the available visual information. One way to get the visual potential field is to adequately estimate the optical flow from the scene.

Optical flow is a vector field that consists of the direction and magnitude of color intensity changes from the movement of objects with the same brightness value or feature pattern between two consequent images, obtained from the projection of an object in a 3D space onto an image plane. So, when an object moves, its projection will change position in the image plane and generate several vectors (direction, magnitude) [44]. This allows to infer temporal information about the current scene, while being less computationally expensive than processing full raw images, which makes it an attractive method for online navigation controllers.

Optical flow has been widely used for obstacle avoidance and vehicular/mobile robot navigation. For instance, optical flow was used to recognize the lane boundaries of the road under adverse weather conditions as reported by [45]. In [46] this technique was used within a road following algorithm that allowed to identify the road without making assumptions about its structure or appearance. It was used to get a steering angle through getting the optical flow between subsequent images in [47]. More recently, optical flow has been used as a mean to model human behavior, as in [48], where an optical flow automatic steering system for the vehicle is presented.

Referring specifically to direct control of the vehicle, the most common idea is to estimate the time to collision (TTC) from the optical flow vector field and use it to steer away from static or dynamic obstacles as seen in [44]. There is also the balance strategy approach, which consists of a set of behavior rules to be applied when the average of the vector field is above/below certain predefined thresholds [49, 50]. In [41], the authors get an visual potential field from the optical

flow vector field by identifying the dominant plane on the image. Then apply a balance strategy for the robot navigation.

We propose to compute an Artificial Potential Field APF (Visual Potential Field) from the optical flow vector field information, that contains both the information from the obstacles as the restrictions of the road. In general, an APF generates a surface where the target is a global minimum and the obstacles are local maxima. The target generates an attractive force on the vehicle while obstacles create a repulsive force. The gradient lines from the optical flow are used to generate this field and get the navigation reference [51, 52].

To include the constraints from the road, a *road potential field* can be used, such as in [53], where the field provides a steep slope when it is closer to the boundaries of the road and has local minima along the center of the lane. Other approaches are introduced in [54, 55], where the distance to the lane boundaries and obstacles are used to compute a total risk field U_{risk}. These contributions consider certain knowledge about the geometry of the road.

Unlike [41, 42], the presented approach works for automated vehicles that have to navigate in a restricted environment (i.e. follow the road), hence the contribution of this work lies in combining previous techniques proven in mobile/aerial robots and extending them to work with cars, and is based on [56].

This chapter is organized as follows: Section II presents the problem formulation, then Section III introduces Optical Flow and focus of expansion (FOE) definitions. Section IV presents the computation of the vision potential field based on optical flow and Section V introduces the controller design. Section VI presents evaluation of the proposed method under different weather conditions in two sets of images. Finally, conclusions are presented in Section VII.

3.2 Problem formulation

The problem that we solve is the visual based navigation of an autonomous vehicle using a monocular camera. Fig.3.1 presents the modules considered for the implementation of the solution.

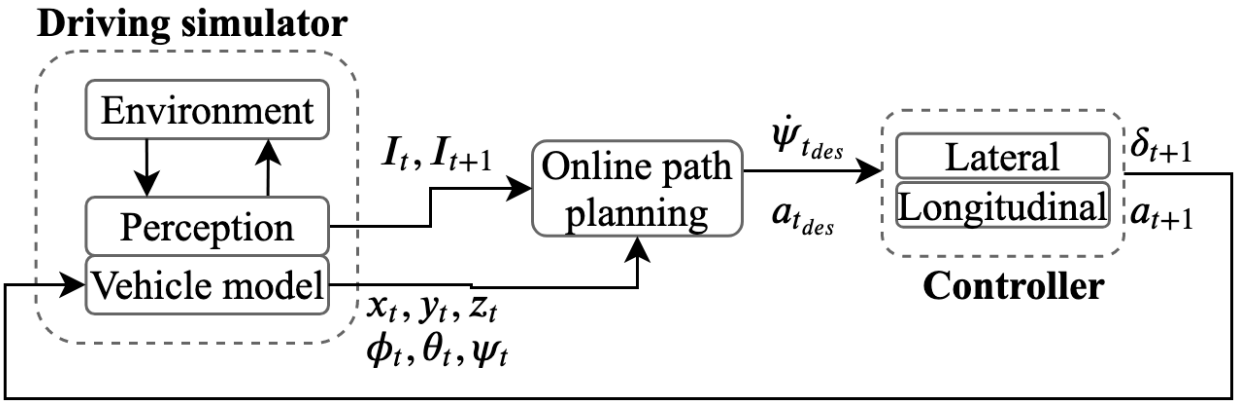


Figure 3.1: Proposed control framework

It is considered that we have access to all the ground truth data from the vehicle, position (x, y, z) and orientation (roll ϕ, pitch θ, yaw ψ). The vehicle has a monocular RGB camera mounted on the windshield, at a fixed position (x_c, y_c, z_c) with respect to the center of mass of the vehicle. The camera provides images (I_t) at each timestep. The *Online Path Planning* block takes pairs of images (I_t, I_{t+1}) and computes the optical flow vector field, the Focus of Expansion (FOE) and the visual potential field, which allows to generate the speed and heading reference, which are

sent to the *Controller* block. This last block computes the acceleration (a) and steering (δ) control actions using a gradient tracking sliding mode controller [51]. The control actions are then sent to the vehicle.

3.3 Optical Flow

Given a set of pixels on an image I_t, the optical flow vector field can find those same pixels in the subsequent image I_{t+1}.

Given two subsequent images, they are first transformed to grayscale, then the corners are detected using the Shi-Tomasi algorithm [57], and finally, the algorithm Lucas-Kanade (LK) with Pyramidal implementation [58] is employed to obtain the optical flow vector field.

3.3.1 Focus of Expansion (FOE)

The Focus of Expansion (FOE) point shows the direction of the vehicle motion, and is computed as the intersection of all the optical flow vectors [48].

The FOE is obtained through a Least-Squares formulation [48], where we minimize the error with respect to the gradient of the optical flow. Consider image I has pixels $p_i(x, y), i = 1, ..., n$, where (x, y) is the position of the pixel p_i in the image plane. We can build the matrices A and b using the spatial gradient $\nabla I(p_i)$ and the temporal gradient $I_t(p_i)$ respectively:

$$A = \begin{bmatrix} \nabla I(p_1) \\ \nabla I(p_2) \\ \vdots \\ \nabla I(p_n) \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ \vdots & \vdots \\ a_{n0} & a_{n1} \end{bmatrix} \tag{3.1}$$

$$b = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ \vdots \\ -I_t(p_n) \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \tag{3.2}$$

For each pixel $p_i(x, y)$ the obtained flow vector (v_x, v_y) provides $a_{i0} = v_y$, $a_{i1} = -v_x$ and $b_i = xv_y - yv_x$. Then the location of the FOE (x_{FOE}, y_{FOE}) on the image plane is given by:

$$\begin{aligned} FOE &= (A^T A)^{-1} A^T b \\ &= \begin{bmatrix} \sum a_{i0} b_i \sum a_{j1}^2 - \sum a_{i1} b_i \sum a_{j0} a_{j1} \\ -\sum a_{i0} b_i \sum a_{j0} a_{j1} + \sum a_{i1} b_i \sum a_{j0}^2 \end{bmatrix} - \frac{1}{\sum a_{j0}^2 a_{j1}^2 - (\sum a_{i0} a_{i1})^2} \end{aligned}$$

3.3.2 Depth from Optical Flow

To recover depth from the most important pixels on the image is possible [56] but very inaccurate, as the optical flow values are noisy near the FOE. Hence, we use instead the time to contact (TTC), which is depth in terms of time and provides enough information about the vehicle motion. The TTC for each pixel $p_i(x, y)$ with optical flow vector field (v_x, v_y) is computed as follows:

$$TTC_i = \frac{\sqrt{(x - x_{FOE})^2 + (y - y_{FOE})^2}}{\sqrt{v_x^2 + v_y^2}} \tag{3.3}$$

3.3.3 Obstacle detection

The optical flow vector field presents a disturbance when an obstacle is present in the image. While the disturbance may be heuristically computed, a feasible method to distinguish the background from the obstacles using the optical flow field is to use the Otsu threshold segmentation method [59].

Let $I(x, y, t) := I_t(x, y)$ be the image at time t. Let $O(x, y, t) := O_t(x, y)$ be the plane that shows only the obstacles in the image obtained by the LK algorithm, i.e. a binary image [41]. Then to get the gradient, we use a smoothing Gaussian function $G(x, y)$:

$$G(x, y) = \frac{1}{2\pi\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.4}$$

The parameter σ can be chosen as half the image width/length. Then, we get the convolution between $O(x, y, t)$ and (3.4):

$$G * O(x, y, t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(u - x, v - y) O(x, y, t) du dv \tag{3.5}$$

Which allows to obtain a function $g(x, y, t)$:

$$g(x, y, t) = \nabla(G * O(x, y, t)) = \begin{pmatrix} \frac{\partial}{\partial x}(G * O(x, y, t)) \\ \frac{\partial}{\partial y}(G * O(x, y, t)) \end{pmatrix} \begin{pmatrix} \end{pmatrix} \tag{3.6}$$

Where Eq. (3.6) represents the gradient vector of $O_t(x, y)$.

3.4 Visual Potential Field

We compute separately the potential fields for the target, the obstacles, and for the road, and then sum them up to get the total potential field at a certain time t.

3.4.1 Target Potential Field

This potential field is directly proportional to the Euclidean distance away from the vehicle, so can be obtained by:

$$U_{att} = \frac{1}{2}\alpha \quad \sqrt{(x - x_{goal})^2 + (y - y_{goal})^2} \tag{3.7}$$

The goal term pulls the vehicle towards the goal. Its strength increases proportionally with the distance to the goal and its adjusted through the constant α. The module of the attraction force is the gradient of the potential field:

$$F_{att} = \nabla U_{att} = \alpha \sqrt{(x - x_{goal})^2 + (y - y_{goal})^2} \tag{3.8}$$

The angle that this force makes with the image plane is equal to the goal angle, i.e. $\theta_{goal} = atan\frac{y - y_{goal}}{x - x_{goal}}$.

3.4.2 Obstacle Potential Field

The obstacle potential field is an approximation of the projection of the 3D potential field onto the image plane, hence the gradient vector $g(x, y, t)$ of $O_t(x, y)$ found in Section 3.3.3. The TTC for each pixel in the image is also used for the definition of the repulsive force [41, 42]:

$$F_{rep} = \begin{pmatrix} F_x \\ F_y \end{pmatrix} = \gamma \frac{1}{|R|} \begin{pmatrix} \int_{(x,y)\in A} g(x, y, t)dx \\ \sum_{(x_i,y_i)\in A} TTC_i \end{pmatrix} \tag{3.9}$$

With A defined as in (3.1), $|R|$ is the region of interest in the image and γ is a gain that modules the strength of the repulsive force.

3.4.3 Road Potential Field

Around the vehicle, the road boundaries should act like a barrier that prevents the car from departing from the lane. Then, the road potential field is designed in a way that it presents local maxima at the road boundaries and local minima in the center of the road, as that is the preferred position of the vehicle.

The road edges are computed using the sparse optical flow. Then, a modified version of the Morse Potential Field is used for the design ([53, 55]). Now, we will consider the local coordinates of the vehicle instead of the image plane, as seen in 3.2. The local coordinates form the plane XY, which differ from the image plane coordinates xy.

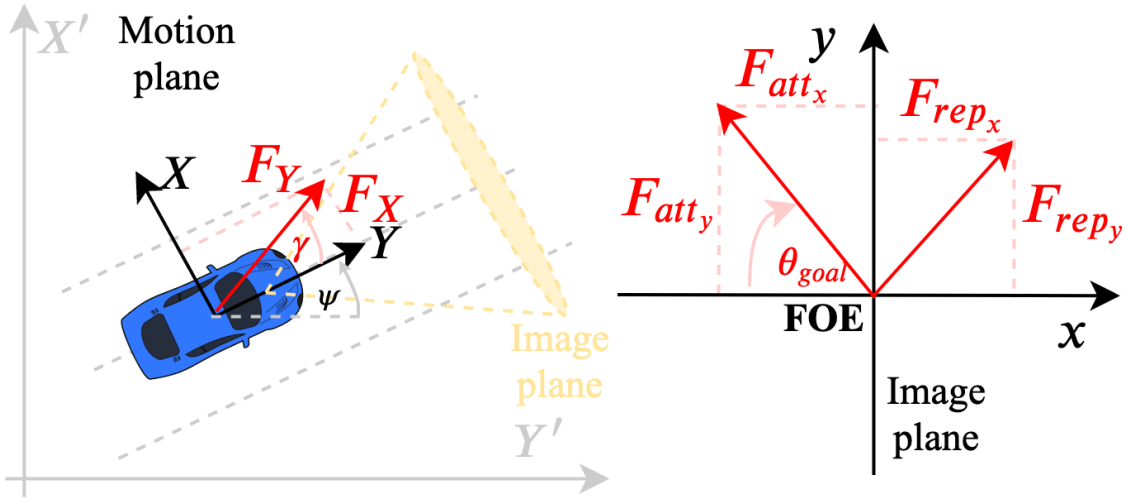


Figure 3.2: Motion plane coordinates are (X, Y), image plane coordinates lie in the vertical plane (x, y)

Eq. (3.10) shows the expression for the total potential field (U_s) used for the straight segments of the road as the sum of the right (U_{s_r}) and left (U_{s_l}) lane potential fields.

$$U_s = U_{s_r} + U_{s_l} \tag{3.10}$$

Where:

$$U_{s_r} = A\left(1 - e^{-b(y-y_r)}\right)^2, \quad U_{s_l} = A\left(1 - e^{b(y-y_l)}\right)^2 \tag{3.11}$$

In the above expressions, A and b are the depth and a parameter based on the variance of the repulsive potential field respectively. y is the current position of the vehicle (considering the vehicle moves parallel to the X-axis), y_r and y_l are the relative distances to the right and left lane boundaries respectively. We consider that the vehicle moves in a four lane road, then $y_r = 5.25(m)$ and $y_l = -8.75(m)$ for a total road width of $14(m)$ and a preferred position of the vehicle in the second lane from the right.

The computation of the right and left potential fields is designed as follows [53]:

$$U_{s_r} = A \quad 1 - e^{-bsign(y-y_r)\sqrt{\left(\frac{y-b_y}{m}-(x+\delta x)\right)^2+(y_r-y)^2}}\Bigg)^2 \tag{3.12}$$

$$U_{s_l} = A \quad 1 - e^{bsign(y-y_l)\sqrt{\left(\frac{y-b_y}{m}-(x+\delta x)\right)^2+(y_l-y)^2}}\Bigg)^2 \tag{3.13}$$

Where x is the current x position, b_y is the y-intercept, $\delta x > 0$ is a small number, m is the slope of the line perpendicular to the lane center, and the rest of the variables are the same as before. In this case, both y_r and y_l are computed as follows:

$$y_r = c_2(x+\delta x)^n + c_1(x+\delta x) + c_{0_r} \tag{3.14}$$

$$y_l = c_2(x+\delta x)^n + c_1(x+\delta x) + c_{0_l} \tag{3.15}$$

In the previous equations,c_{0_r} and c_{0_l} represent the distance to the right lane marking from the center of the lane to the right and left respectively. The parameter c_1 is chosen as zero, and c_2 and n will be chosen accordingly if the road is straight or curved.

To decide the curvature of the road, the position of the FOE is taken into account. If it is located in the center of the captured frame, then it is considered that the road is still straight for the next few seconds. In that case, $c_2 = 0.005$ and $n = 1$. If the FOE is located to the right or to the left, it is considered that the road is curved for the next few seconds. In that case, the choice of parameters $c_2 = \pm 5e-6$ (for left and right curve respectively) and $n = 2$ have given good results.

Then, to compute the slope:

$$m = -\frac{1}{2c_2(x+\delta x)+c_1} \tag{3.16}$$

As seen in Eq. (3.16), δx is needed to avoid m to go to infinity when $x = 0$. Then, to compute the y-intercept of the line perpendicular to the lane center b_y:

$$b_y = y_r - m(x+\delta x) \tag{3.17}$$

Table 3.1 shows the chosen values for the parameters.

The designed potential field is shown in Fig. 3.3 for a one lane only left-curved road.

Finally, define the repulsive road force: $\mathbf{F}_{rep_{rd}} = (F_{rep_{rd_X}}, F_{rep_{rd_Y}})$:

$$\mathbf{F}_{rep_{rd}} = \nabla(U_{s_r} + U_{s_l}) \tag{3.18}$$

Table 3.1: Parameters used for APF implementation

Description	Symbol	Value	Unit
APF depth	A	0.5	-
Parameter based on the variance	b	1	-
Parameter (± curved road left/right)	c_2	5e-6	-
Parameter (straight road)	c_2	0.005	-
Parameter	c_1	0	-
Parameter (left,right side APF)	(c_{0_l}, c_{0_r})	$(-8.75, 5.25)$	(m)
Small longitudinal offset	δx	1.0e-10	(m)

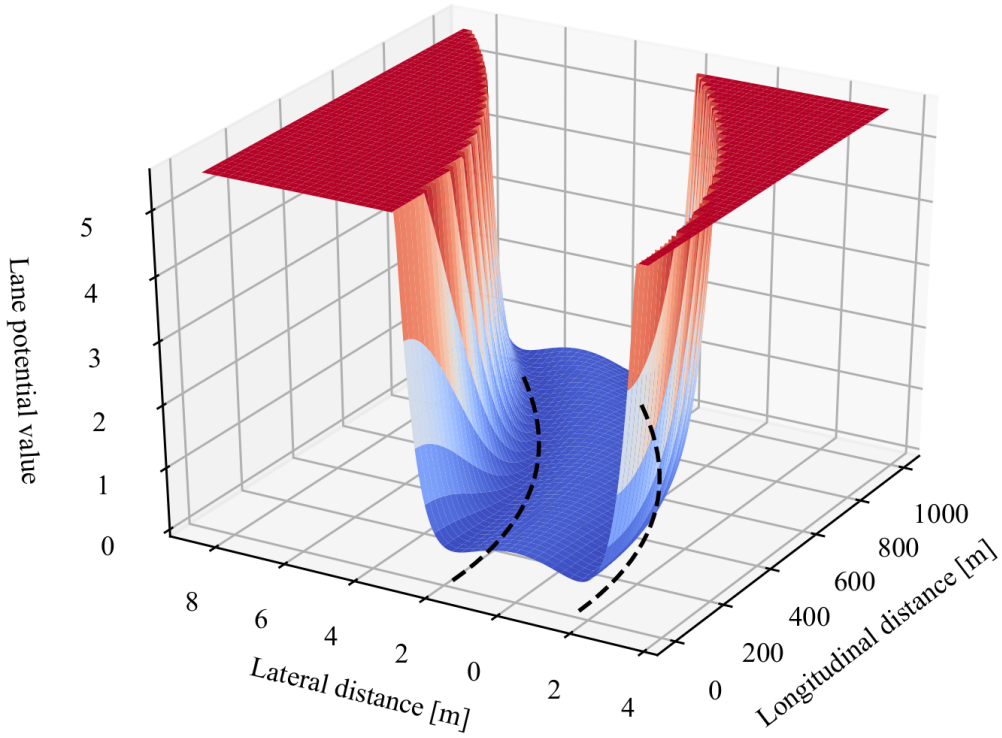


Figure 3.3: Designed potential field for a one lane curved road (angled view)

3.5 Total Potential Field

We consider the forces in the image plane and in the motion plane, seen in Fig. 3.2 and developed in Sections 3.4.1, 3.4.2 and 3.4.3.

$$F_{X_T} = F_{att_x} - F_{rep_x} - \lambda_X F_X \tag{3.19}$$
$$F_{Y_T} = F_{att_y} - F_{rep_y} - \lambda_Y F_Y \tag{3.20}$$

Where $\lambda_X, \lambda_Y > 0$ are chosen appropriately to weigh the potential field from the motion plane with respect to the image plane. Ultimately, to find the force in the global coordinates:

$$\begin{pmatrix} F_{X'} \\ F_{Y'} \end{pmatrix} = \begin{pmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} F_{X_T} \\ F_{Y_T} \end{pmatrix} \tag{3.21}$$

Where ψ is the orientation of the vehicle (or yaw angle) with respect to the global coordinates.

3.6 Gradient Tracking Sliding Mode Controller (GTSMC)

3.6.1 Model of the vehicle

The simplified bicycle kinematic model [60] of the vehicle is used for the design of the controller. Considering that only the front wheels can be steered, the model is:

$$\dot{x} = v\cos(\psi + \beta) \tag{3.22}$$
$$\dot{y} = v\sin(\psi + \beta) \tag{3.23}$$
$$\dot{\psi} = \frac{1}{l_f + l_r} v \cos\beta \tan\delta_f \tag{3.24}$$
$$\beta = arctan\left(\frac{l_r \tan\delta_f}{l_f + l_r}\right) \tag{3.25}$$
$$\dot{v} = a \tag{3.26}$$

Where x, y are the global X, Y-axis coordinates respectively, the yaw angle is represented by ψ is the yaw angle (orientation of the vehicle with respect to the global X-axis and β is the vehicle slip angle. The speed is $v : |\mathbf{v}|$ and acceleration is a, in the same direction of the velocity $\mathbf{v}$. The control inputs are the front steering angle δ_f and the acceleration a. This model is shown in Fig. 3.4.

3.6.2 Controller design

A gradient tracking sliding mode controller is chosen for the lateral controller [51, 52]. The objective of the controller is to force the motion of the system $\dot{X} = f(X, u)$ to stay within a "sliding manifold" [61]. Since there is no preset trajectory, the gradient of the visual potential field is used to obtain an orientation reference [52].

Let $p = (x, y)$ be the position of the CG of the vehicle. The motion direction against the gradient of the artificial potential field were already obtained in Section 3.4.1. Then, for each point

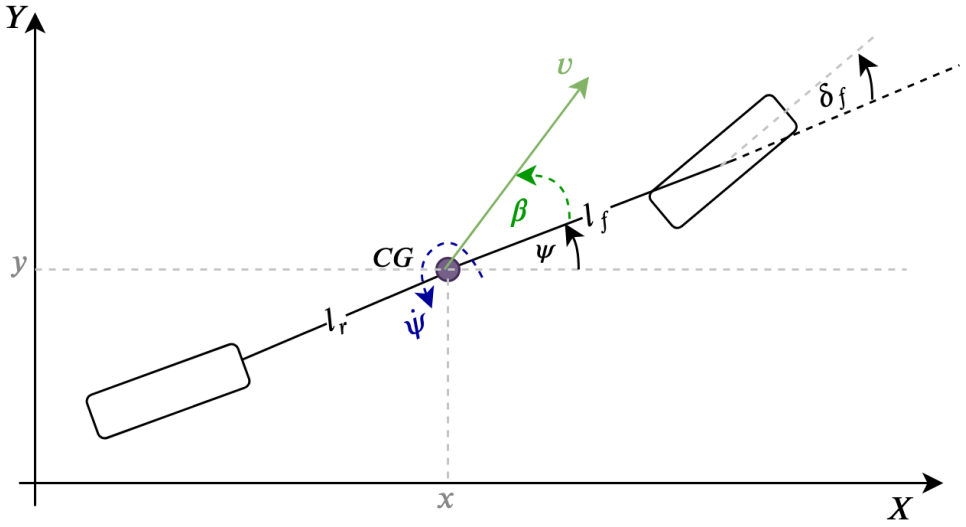


Figure 3.4: Kinematic bicycle model of a vehicle

p a continuous trajectory, called *gradient line* is obtained. Thus, the desired orientation of the vehicle corresponds to:

$$\psi_d(p) = atan\frac{F_{Y'}(p)}{F_{X'}(p)} \tag{3.27}$$

Where $\psi_d(p) \in [-\pi, \pi[$. Then, we can find the rotational manifold s_r:

$$s_r(p,t) = c_r\psi_e(p,t) + \dot{\psi}_e(p,t) \tag{3.28}$$

Where $c_r > 0$ is a constant, $\psi_e(p,t) = \psi(t) - \psi_d(p)$, with $\psi_d(p)$ from (3.27).

Using the rotational manifold, we model the front wheel steering actuator as an integrator with constraints $|\delta| \leq \delta_0$, $|u| \leq u_0$, and: [51]:

$$\dot{\delta_f} = u \tag{3.29}$$

$$u = -u_0 sign(s_r) \tag{3.30}$$

When the amplitudes chosen are large enough it has been proven that the controller reaches the sliding manifold in finite time. Details are in [61].

A longitudinal manifold s_t is used for the longitudinal controller with a constant $c_l > 0$:

$$s_l(p,t) = c_l v(p,t) - v_d(p,t) \tag{3.31}$$

For the longitudinal control, the following simple sliding mode controller is chosen:

$$a = -a_0 sign(s_l) \tag{3.32}$$

3.7 Experiment Setup

The experiment consists of two parts: first we use a driving simulator to extract images and obtain the proposed control and compare its performance against a PID. Then, we use a set of real images collected along with their control, and use our approach to predict the steering and throttle.

Table 3.2: Simulator setup

Parameter	Value
Steering angle limits	$[-40°, 40°]$
Reference speed	˜5.55m/s (20km/h)
Weather	Clear Noon, Hard Rain
Scenario	Highway Town04
FPS	˜60
Window simulation size	640×480 pixels

3.7.1 Synthetic images

Synthetic images were extracted from Carla driving simulator [39] because of its realistic 3D environments.

The optical flow pyramidal implementation uses a size of the search window equal to $(25, 25)$ pixels and the accuracy threshold is $\epsilon = 0.03$. The input image I has the size 640×480 pixels, and three levels of the pyramid are used.

Table 3.2 shows the selected simulation parameters. The controlled variables are the throttle a and steering angle δ. These parameters accept normalized values: $a \in [0, 1]$ for acceleration, $a \in [-1, 0]$ for deceleration and $\delta \in [-1, 1]$. The vehicle stops when it reaches certain predefined location in the map.

Two different weathers were tested in the same route, with and without obstacles on the road. This comparison is interesting because the rain makes the road look wet, almost like a mirror that reflects the objects in the scene, as seen in Fig. 3.5. We evaluate how the vehicle behaves in both environments and present the path in terms of the global coordinates.

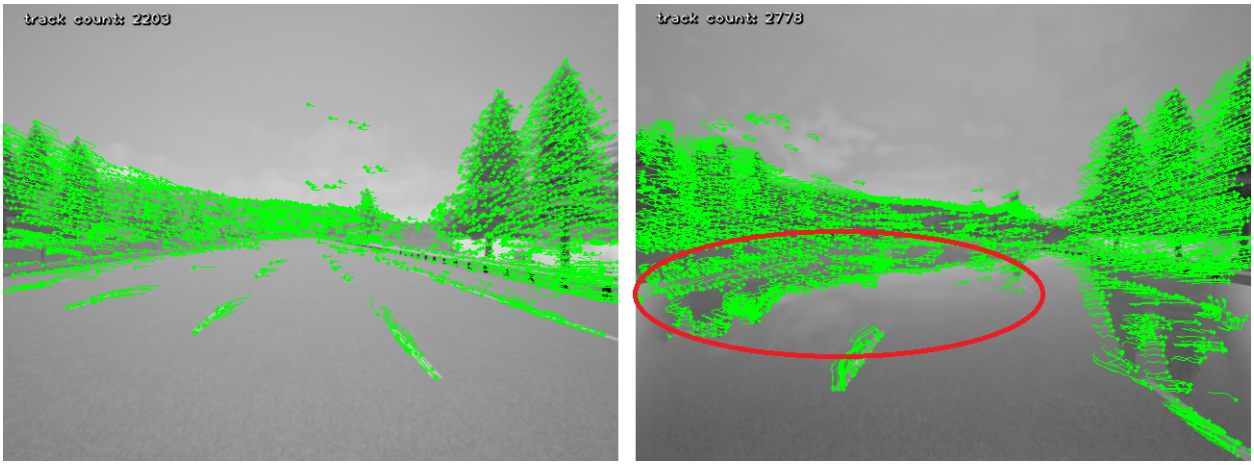


Figure 3.5: Sparse optical flow in clear (left) and rainy (right) weather

The "ideal" route is defined via waypoints available in the simulator and it is expected that the vehicle be able to recreate a similar path. Fig. 3.6 shows the path comparison when there are no obstacles and the weather is good. In the zoomed out plot it looks that the vehicle has followed perfectly the path, but once we zoom in, it is noticeable that there are deviations from the ideal path. Since the restrictions of the road considered that the whole 4 lanes of the road are allowable path, this behavior is expected.

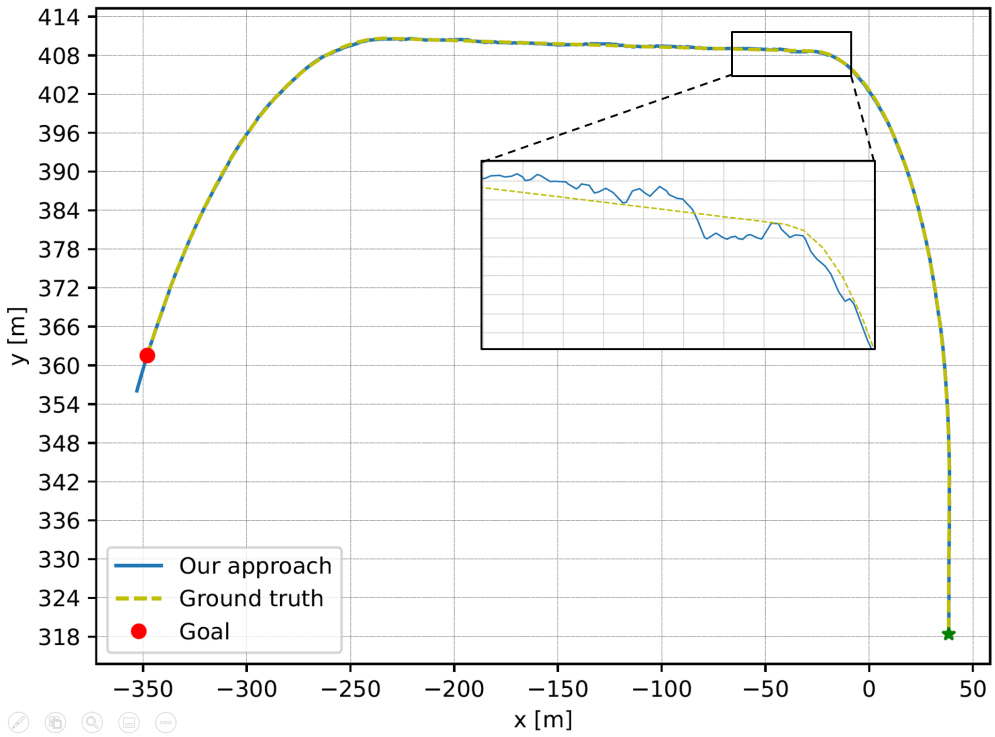


Figure 3.6: Vehicle path with no obstacles in clear weather

By inserting obstacles in the previous weather, we get a disturbed path from the controlled vehicle, as seen in Fig. 3.7. The visual potential field makes the vehicle change its direction to avoid both the obstacle and the lane boundaries.

When there is rain in the scene, the reflection on the road causes disturbances in obtaining the obstacle map ($O_t(x, y)$), hence the path described by the vehicle is more jerky than when the weather is clear. Fig. 3.8 shows this motion.

Inserting obstacles in the last situation makes it more challenging for the vehicle. We observe a very jerky motion in Fig. 3.9. This behavior is expected, but nonetheless, the vehicle reaches its destination.

It is worth noting that the vehicle was tested in a curved and a straight road, for which we designed precisely the road potential field. In case of having more complicated types of roads, it is necessary to generalize our method. So far, as long as we are taking a horizon of $t = 5$ seconds ($\approx 27.5m$ at $20km/h$) we are able to generate an estimated road potential field using the position of the FOE, as explained in Section 3.4.3.

It is expected that the vehicle motion with obstacles would be jerkier. The disturbance in the obtained flow and potential field is reflected in the motion of the vehicle, seen in Fig. 3.9. Here it is more noticeable that the vehicle cannot stay on its own lane, due in part to the fact that it is avoiding only the edges of the whole road, not only of its lane. This change was necessary to provide enough room to perform lane change (obstacle evasion).

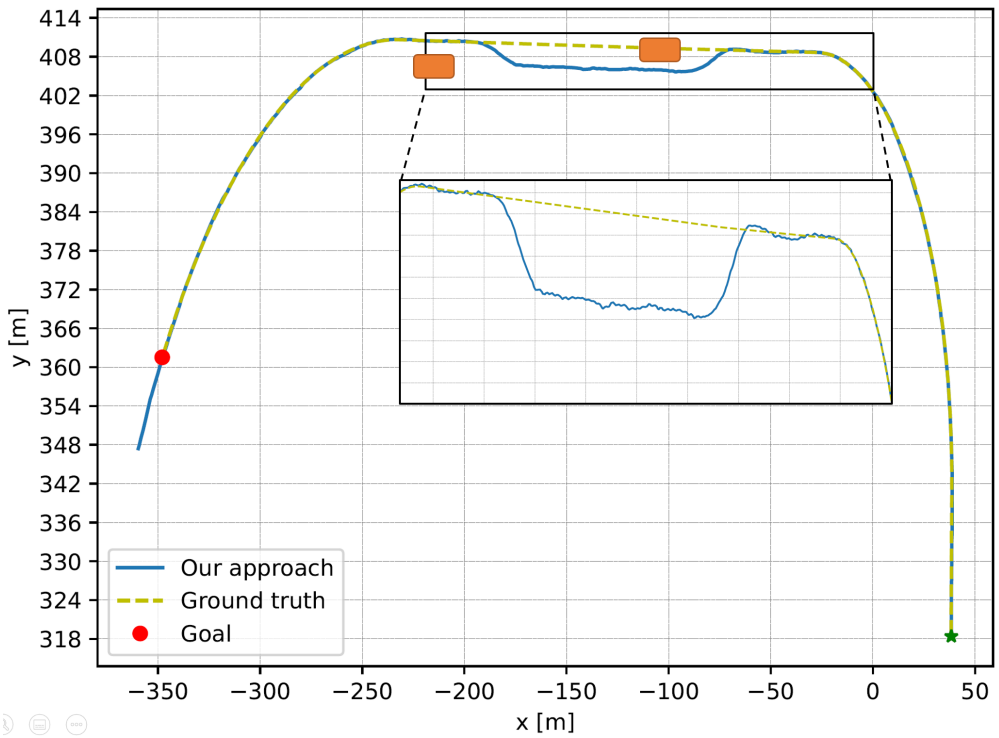


Figure 3.7: Vehicle path with obstacles in clear weather, obstacles at (-188.3, 406.4) and (-95.6, 409.4.4)

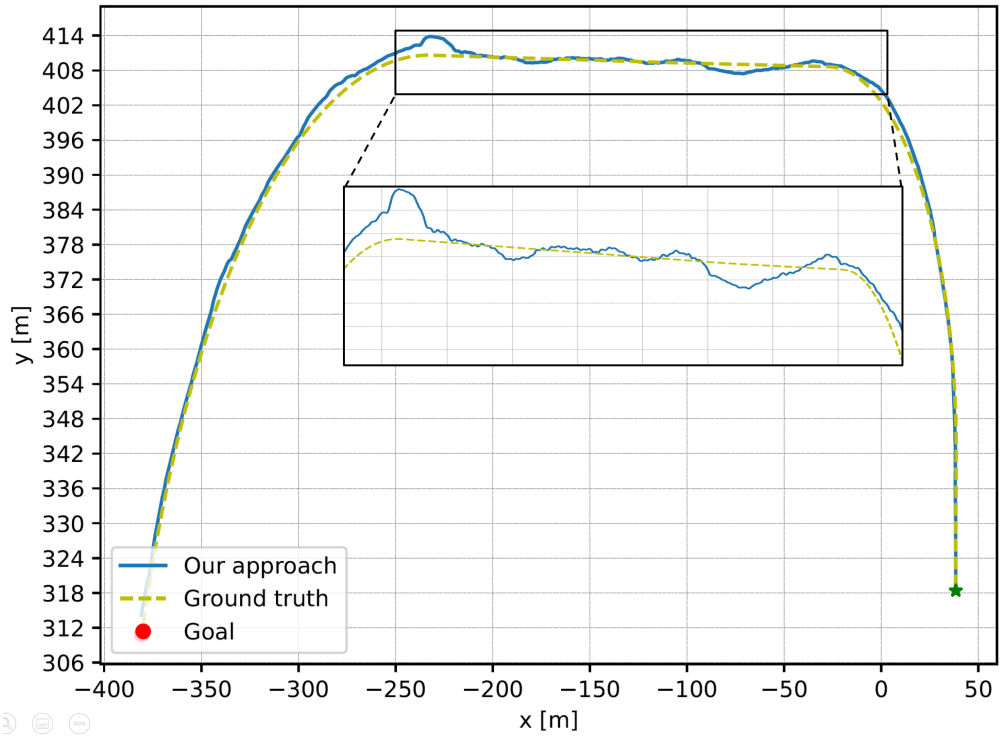


Figure 3.8: Vehicle path with no obstacles in rainy weather

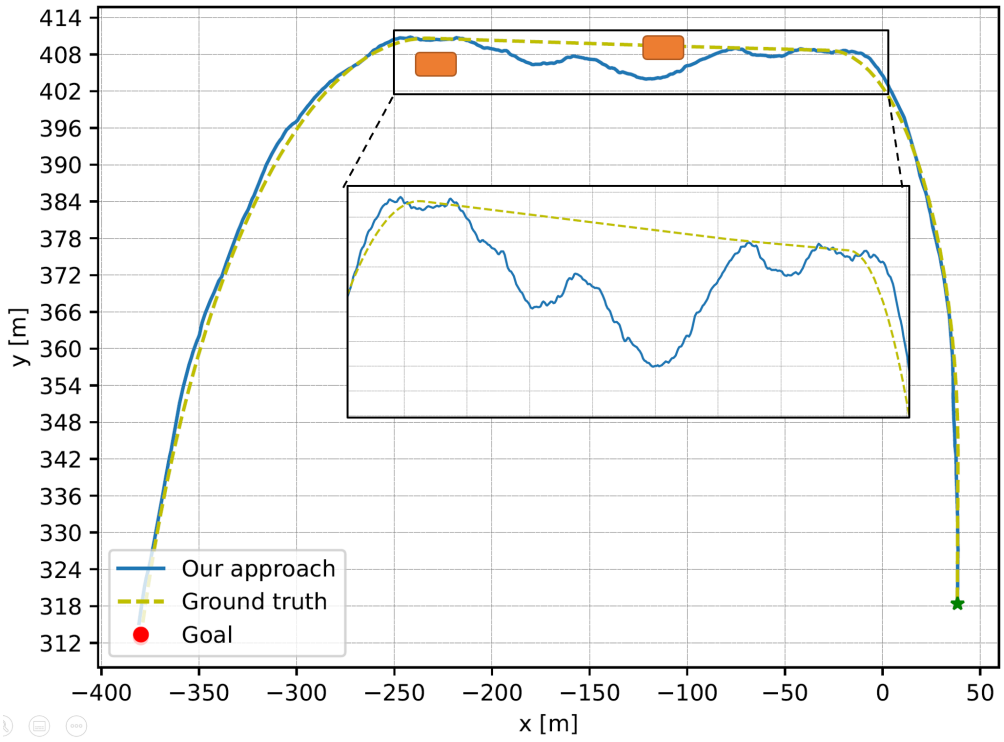


Figure 3.9: Vehicle path with obstacles in rainy weather, obstacles at (-188.3, 406.4) and (-95.6, 409.4)

Table 3.3: Simulator results comparison table, prediction accuracy (%) for throttle a and steering δ with respect to PID, mean square error for desired trajectory.

Weather	**No obstacles**			**With obstacles**		
	$a\%$	$\delta\%$	$\overline{MSE}_{XY}(m)$	$a\%$	$\delta\%$	$\overline{MSE}_{XY}(m)$
Cloudy	82.34	72.14	0.42	78.51	71.67	0.74
Rain	78.45	67.12	0.97	70.76	65.45	1.34

Next, we compute a feasible path from the starting point to the end point using waypoints over the road, and use a longitudinal and lateral PID controller for the path tracking. We use the output of these controllers to compare the throttle and steering values obtained by our method. We divide the route into obstacle and no obstacle section, and summarize the results from this experiment in Table 3.3.

Whereas the prediction results for throttle and steering are not perfect, they are a good approximation that allow the average mean square error ($\overline{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + (x_i - \hat{x}_i)^2$) with respect to the original planned trajectory to be less than $1(m)$ in average, which is one-third of the average US highway lane width ($\sim 3.7(m)$).

3.7.2 Real Images Dataset

Our approach is also tested on a set of real images that were collected alongside with their GPS, control and state information. The images show the front camera view of a vehicle in three weathers: cloudy, light rainy and rainy.

Analyzing the images, we notice that whenever it starts raining, our optical flow algorithm starts tracking the water droplets in the windshield. Furthermore, feature tracking is interrupted when the wipers are activated, see Fig. 3.10. Identifying and removing these objects is important for this method to work, and is posed as a possible future contribution. However, we used the exact same pipeline as for the simulator synthetic images to evaluate its performance *as is* .

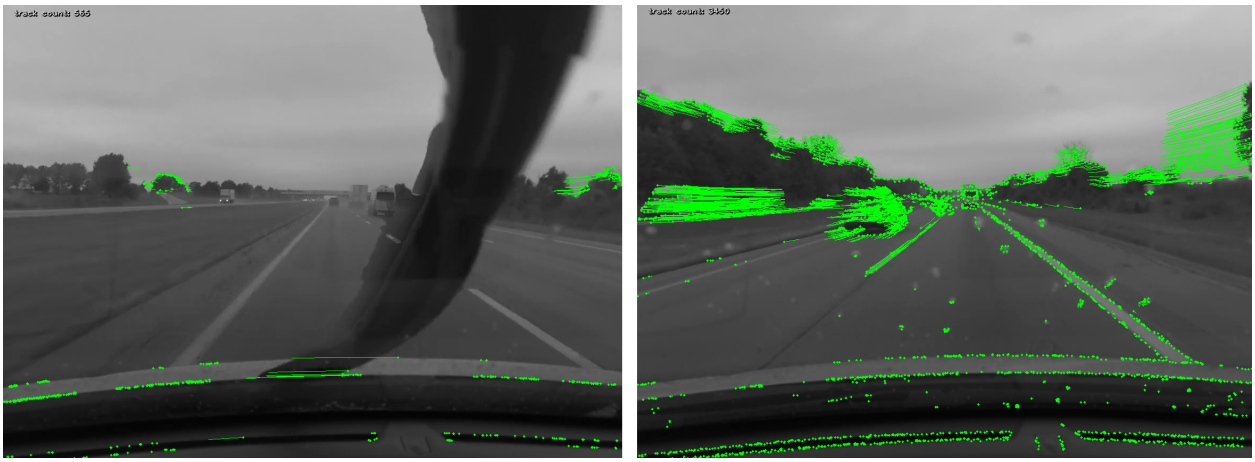


Figure 3.10: Sparse optical flow when using wipers (left) and when droplets fall on the windshield (right)

We have analyzed each given route and divided them into obstacle free routes and routes with obstacles, to do a similar comparison as in Section 3.7.1. Then for each route, and by using the GPS

Table 3.4: Results comparison table for real dataset, prediction accuracy (%) for throttle a and steering δ.

Weather	**No obstacles**		**With obstacles**	
	$a\%$	$\delta\%$	$a\%$	$\delta\%$
Cloudy	69.43	64.92	67.12	62.34
Light rain	58.27	55.87	55.31	54.23
Rain	56.12	48.51	50.97	46.12

coordinates information, we have selected an end point for our algorithm to represent the global minimum. The initial speed for the vehicle model is set according to the information provided by the dataset for each route segment.

We ran the images through the proposed pipeline and predicted the throttle a and the steering angle δ. We compare these results against the real throttle and steering angle from the dataset, and summarize in Table 3.4.

As would be expected, the prediction accuracy for real images drops ($\sim 50 - 60\%$), but still remains a promising result considering that the optical flow output was not modified. These results would increase with a tailored tuned optical flow with some removal methods to avoid the effect of droplets or wiper blades.

3.8 Conclusions

This work showed that it is possible to obtain a visual potential field from the optical flow information from a monocular camera. The novelty of this work consists on the formulation of the potential field for both the obstacles and the road boundaries and applying it to control a vehicle.

This visual based navigation method is less computationally expensive than learning based techniques, but at the same time, it allows to capture the features of dynamically changing environment. For this reason, it can serve as a baseline for comparison with both classical and learning based approaches.

Optical flow has its own limitations, such as being inaccurate for large motions, when there is occlusion and for strong illumination changes. Even when optical flow methods can be highly accurate in synthetic scenes, it is well known that its estimation on natural scenes can be a problem and require computationally expensive techniques to be solved, which contrasts with the benefits of low computation expense for the optical flow itself. This was shown in the set of real data presented in the experimental section.

Future work includes the comparison of this technique with learning based and classical ones, not only in performance, but in formulation complexity and execution time. Additionally, the method should be tested in more extensive sets of real data and ideally in a drive-by-wire vehicle to test not only the prediction, but actual performance.

Chapter 4

Automated Traffic Surveillance Using Existing Cameras on Transit Buses

4.1 Introduction

Ever-increasing demand and congestion on existing road networks necessitates effective transportation planning, design, and management. This informed decision making requires accurate traffic data. Most medium- to long-term traffic planning and modeling is focused on traffic demand, as quantified by traffic counts or flow. Traffic flow is generally measured by fixed-location counting, which can be accomplished using permanent detectors or, in many cases, temporary detectors or human observers. Many traditional traffic studies use temporary pneumatic tube or inductive loop technologies that are deployed for only a few days at each location, and each location is revisited only every three to five years. So, while traffic flow measurement is key to estimating demand throughout the network, the available data are spatially and temporally sparse and often out of date.

A large body of work has focused on automatic vision-based traffic surveillance with stationary cameras [62, 63, 64, 65, 66, 67, 68]. More recently, state-of-the-art vehicle detection performance has appreciably improved with Convolutional Neural Network (CNN)-based mature 2D object detectors such as Mask-RCNN [69], Yolo v3 [70], and Yolo v4 [71]. These models are feasible to deploy in real time. For most surveillance tasks, a processing rate of 4–5 frames per second is sufficient [72], which is achievable with these detectors. However, stationary, fixed-location cameras cannot cover the entirety of the road network.

Unmanned Aerial Vehicle (UAV) based automatic traffic surveillance [73, 74, 75, 76] is a good alternative to stationary cameras. Drones can monitor a larger part of the network and track vehicles across multiple segments. However, UAV operation carries its own technical and regulatory limitations. Adverse climate affects flight dynamics substantially [76], and vehicle detection suffers from the lower resolution of camera views at higher flight elevations.

Recent studies proposed and developed the concept that public transit bus fleets could be used as surveillance agents [77, 78, 79]. Transit buses tend to operate on roadways with a greater concentration of businesses, residences, and people, and so they often cover many of the more heavily used streets that would be selected for a traditional traffic study. As discussed in the above noted studies, this method could also be deployed on other municipal service vehicles that might already be equipped with cameras; however, one advantage of using transit buses is that their large size allows cameras to be mounted higher above the ground, thus reducing the occurrence of occlusions.

Prior work on measuring traffic flow from moving vehicles has relied on manual counts or used dedicated probe vehicles equipped with lidar sensors [77, 78]. Lidar sensors were used because the automatic processing of lidar data is readily achievable. However, unlike lidar, video sensors are already deployed on transit buses for liability, safety, and security purposes. The viability, validity, and accuracy of the use of video imagery recorded by cameras mounted on transit buses was established by manually counting vehicles captured in this imagery with the aid of a graphical user interface (GUI) [79]. However, for practical large-scale applications, the processing of video imagery and the counting of vehicles must be fully automated.

In this work, we propose to operationalize this traffic surveillance methodology for practical applications, leveraging the perception and localization sensors already deployed on these vehicles. We develop a new, automatic, vision-based vehicle counting method applied to the video imagery recorded by cameras mounted on transit buses. To the best of our knowledge, this work is the first effort to apply automated vision-based vehicle detection and counting techniques to video imagery recorded from cameras mounted on bus-transit vehicles. Figure 4.1 shows an example deployment of cameras on a transit bus, including example video imagery. This configuration is used in the experiments presented in this chapter. By fully automating the vehicle counting process, this approach offers a potentially low-cost method to extend surveillance to considerably more of the traffic network than fixed-location sensors currently provide, while also providing more timely updates than temporary traffic study sensor deployments.

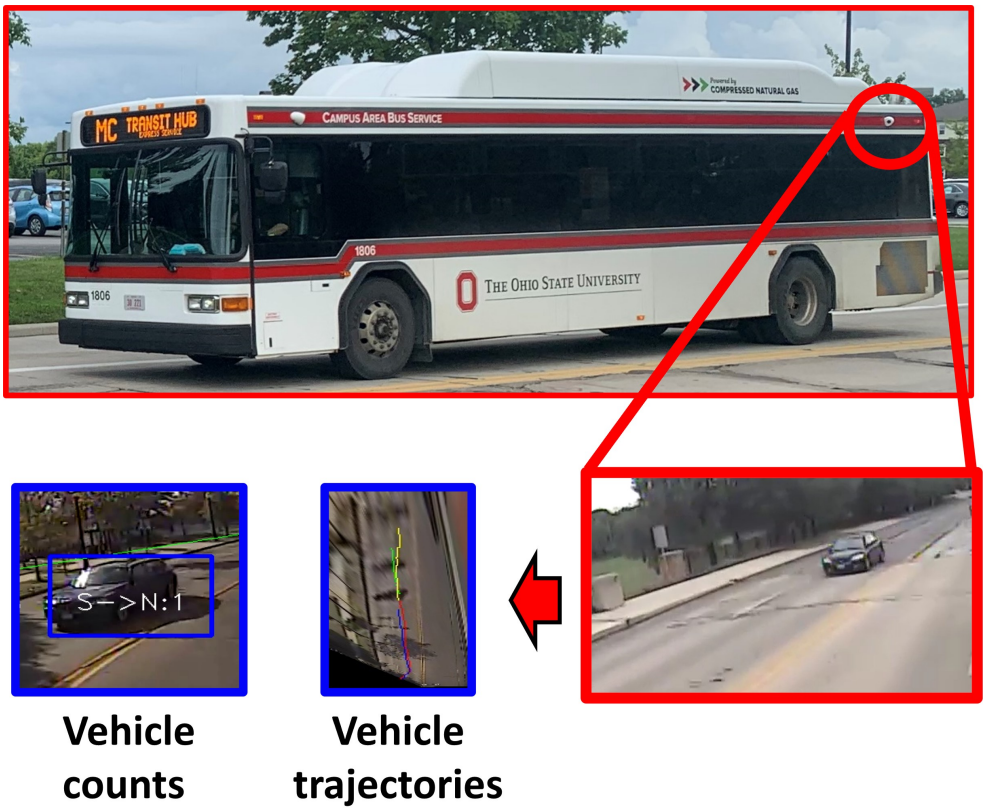


Figure 4.1: The proposed vehicle counting and trajectory extraction framework utilizes video cameras already mounted on existing transit buses for purposes other than traffic surveillance. Using transit buses as intelligent surveillance agents would thus be cost-effective and could increase traffic network coverage.

The main contributions of this study are the following:

- An automatic vision-based vehicle counting and trajectory extraction method using video imagery recorded by cameras mounted on transit buses.
- A real-world demonstration of the automated system and its validation using video from in-service buses. An extensive ablation study was conducted to determine the best components of the pipeline, including comparisons of state-of-the-art CNN object detectors, Kalman filtering, deep-learning-based trackers, and region-of-interest (ROI) strategies.

The main challenges in this endeavor are as follows: estimating vehicle trajectories from a limited observation window, robust vehicle detection and tracking while moving, localizing the sensor platform, and differentiating parked vehicles from traffic participants. Figure 4.2 shows an overview of the processing pipeline. First, images obtained from a monocular camera stream are processed with the automatic vehicle counter developed in this study. A Yolo v4 [71] deep CNN trained on the MS COCO dataset [80] detects vehicles, while SORT [81], a Kalman filter and Hungarian algorithm based tracker which solves an optimal assignment problem, generates unique tracking IDs for each detected vehicle. Next, using subsequent frames, the trajectory of each detected vehicle is projected onto the ground plane with a homographic perspective transformation. Then, each trajectory inside a predefined, geo-referenced region-of-interest (ROI) is counted with a direction indicator. The automatic vehicle counts are compared against human-annotated ground-truth counts for validation. In addition, an exhaustive ablation study is conducted to determine the best selection of 2D detectors, trackers, and ROI strategies.

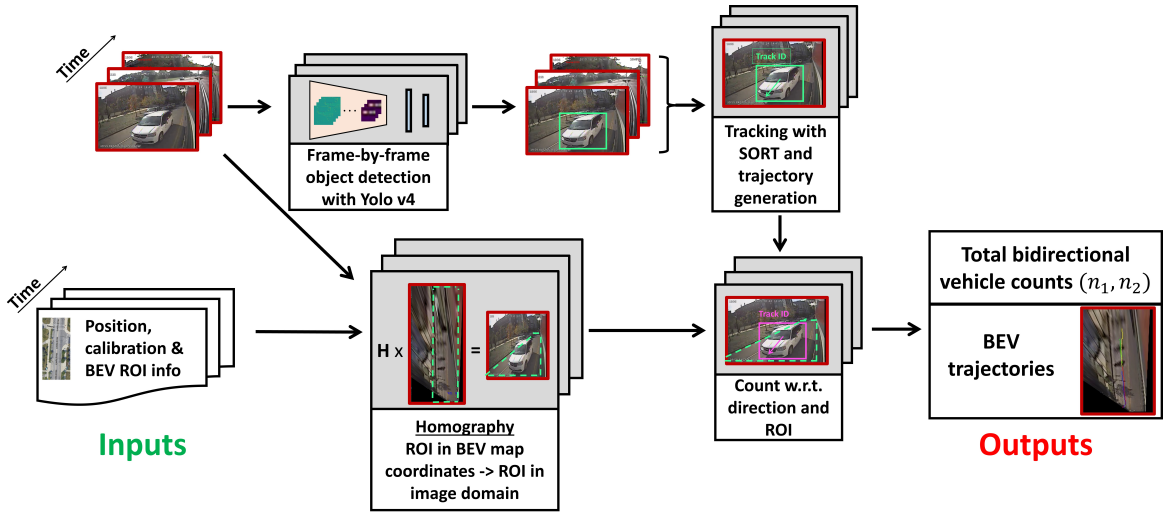


Figure 4.2: Overview of the proposed automated vehicle counting system. The objective is twofold: obtaining total bidirectional vehicle counts and extracting trajectories in bird's-eye-view (BEV) coordinates. Our method uses streams of images, GNSS measurements, and predefined ROI information on a 2D map as inputs. The detection and tracking branch does not require geo-referencing, but it is necessary for counting and projecting trajectories in BEV world coordinates. The homography calibration needs to be performed only once. Only four point correspondences are required for the calibration process.

4.2 Related work

4.2.1 Traffic Surveillance with Stationary Cameras

Automating vehicle detection and counting is crucial for increasing the efficiency of image-based surveillance technologies. A popular approach has been to decompose the problem into subproblems: detecting, tracking, and counting [63].

Detection: Conventional methods employ background–foreground difference to detect vehicles. Once a background frame, such as an empty road stretch, is recorded, the background can be subtracted from the query frames to detect vehicles on the road [64].

More recently, state-of-the-art deep-learning-based 2D object detectors such as Mask-RCNN [69], Yolo v3 [70], and Yolo v4 [71] have achieved remarkable scores on challenging object detection benchmarks [80, 18]. Deep CNNs completely remove the background frame requirement. As such, pretrained, off-the-shelf deep CNNs can be deployed directly for vehicle detection and, if needed, vehicle classification tasks [67, 68]. However, most vehicle detectors work on a single image frame, hence the need for tracking.

Tracking: Tracking is a critical step for associating detected objects across multiple frames and assigning a unique tracking ID to each vehicle. This step is essential for counting, as the same vehicle across multiple image frames should not be counted more than once. Once the vehicle is detected, a common approach is to track the bounding box with a Kalman filter [82].

SORT [81] is a more recent Kalman filter and Hungarian algorithm based tracker. Deep SORT [83] is an extension of SORT with a deep association metric, using a CNN-based feature extractor to associate objects with similar features across frames. Deep-SORT-based vehicle detectors [84] can achieve state-of-the-art performance.

Counting: The final component of the processing pipeline is counting. Usually, lane semantics with ROI reasoning [65] are used to define a counting area. This area is then used to decide which vehicles to count as traffic participants.

4.2.2 Traffic Surveillance with UAVs

Automated vehicle detection has progressed significantly over the years. However, stationary cameras cannot cover the entirety of the road network. The shortcomings of fixed-location, stationary cameras can be circumvented by using a UAV as a sensor platform [73, 74, 75, 76]. Operating regulations have somewhat relaxed, and the cost of UAVs has fallen significantly over the past decade, allowing them to become a more cost-efficient solution for airborne surveillance.

Similar trends such as using Yolo [85] and Faster RCNN [86] are prevalent for UAV-based traffic surveillance. An important design criterion is flight altitude. There are three flight categories: low (up to 70 m) [87], mid-range (80–120 m) [88], and high (100–150 m) [89] altitude flight.

UAV-based surveillance systems suffer from lower resolutions and smaller observed vehicle sizes. In addition, since the viewing angle is top-down, vehicles' distinguishing features cannot be observed. As such, off-the-shelf object detectors cannot be deployed directly. The deep learning models must be trained with a top-down object detection dataset [90].

UAV-based traffic surveillance is promising. However, adverse weather affects flight dynamics negatively [76], and safety concerns are a limiting factor for the usability of airborne drones.

4.2.3 Traffic Surveillance with Probe Ground Vehicles

The general concept of using moving vehicles to determine traffic variables is not new. It is common practice to use the floating car method to measure travel times and delays for arterial

traffic studies [91, 92] and to use probe vehicles to measure travel times in real time [93, 94, 95, 96, 97, 98, 99, 100]. However, these approaches focus only on measuring the probe vehicle's travel time.

This study focuses on the estimation of traffic flow rather than travel times. Traffic flow estimation requires counting other vehicles. Although the theoretical foundations of such a framework were laid decades ago [101], an operational fully automated system has not yet been realized and deployed in a practical large-scale setting. While there are recent efforts to extend cellphone tracking to estimate traffic volumes [102, 103], these approaches are limited by the fact that there is not a one-to-one match between cellphones and vehicles.

4.2.4 Traffic Flow Using Moving Observers

The main idea behind the moving observer method is to estimate hourly traffic volume on a finite road section while the observer is traversing it. Recent studies have made progress in this regard. Specifically, one study used an instrumented vehicle equipped with lidar sensors to emulate a transit bus [77, 78]. Lidar sensors and data were used because they are readily processed, with limited computing power, to automatically detect vehicles. The detected vehicles were then converted to counts, which in turn were transformed into estimates of traffic flows. However, modern transit buses are already equipped with video cameras for purposes other than traffic flow estimation, rather than with lidar sensors. Therefore, in a follow-on study [79], counts from video imagery recorded by cameras mounted on transit buses, which had been extracted manually using a GUI, were used to show and establish the viability and validity of estimating accurate traffic flows from the such imagery.

Another recent study focused on applying moving observer techniques to estimate traffic flow using automated vehicles and a data-driven approach [104]. However, a practical large-scale implementation of such a moving observer surveillance system is not possible without a method that can entirely automatically extract vehicle trajectories and count vehicles from a limited observation window.

We aim to fill this gap in the literature with the proposed automatic vehicle trajectory extraction and counting methodology. In principle, the methodology developed herein for counting vehicles from existing cameras on transit buses could be transferred to existing cameras on public service, maintenance, or safety vehicles and even private automobiles equipped with cameras already being used for driver assist, adaptive cruise control, or vehicle automation, e.g., [104].

4.3 Method

As mentioned previously, our work focuses solely on automatically obtaining vehicle counts and extracting trajectories using computer vision. What follows are detailed descriptions of the problem and the various steps of the developed methodology.

4.3.1 Problem Formulation

Given an observation at time t as $O_t = (\mathbf{I}_t, p_t^{\text{bus}})$, where $\mathbf{I} \in \mathbb{Z}^{H \times W \times 3}$ is an image captured by a monocular camera mounted on the bus and $p \in \mathbb{R}^2$ is the position of the bus on a top-down 2D map, the first objective is to find $f_1 : S \to (n_1, n_2)$, a function that maps a sequence of observations $S = (O_1, \cdots, O_N)$ to the total number of vehicles counted traveling in each direction (n_1, n_2). The second objective is to find $f_2 : S \to \{T_i\}_m$, which provides the m tuples of detected vehicles' trajectories. The trajectory of detected vehicle i is given with $T_i = \{(p_{i,j}, t_{i,j}) | j \in (1, 2 \cdots, J)\}$,

Algorithm 1: Detect-Track-Project-Count: DTPC($S, \mathbf{H}, \mathbf{R}$)

Input:
$S = (O_1, \cdots, O_N)$, $O_t = (\mathbf{I}_t, p_t^{\text{bus}})$: a sequence of observation tuples. Each tuple O consists of an RGB image and a 2D real-word pose vector.
$\mathbf{H}$: the homography calibration matrix
$\mathbf{R}$: the ROI info in real-world coordinates
Output:
(n_1, n_2): the bidirectional vehicle count.
$\{T\}_k, T_i = \{(p_{i,j}, t_{i,j}) | j \in (1, 2 \cdots, J)\}$: the trajectories of detected vehicles.
Main algorithm:
initialize tracker states $\mathbf{X}$;
initialize counted vehicles memory $\mathbf{M}$;
preallocate trajectories $\{T\}_k$;
foreach t **do**
  $\text{ROI}_{image} \leftarrow$ update_ROI($\mathbf{I}_t, p_t^{\text{bus}}, \mathbf{H}, \mathbf{R}$);
  $\{D\}_l = f_{\text{CNN}}(\mathbf{I}_t)$;
  $\mathbf{X} \leftarrow$ tracker_update($\mathbf{X}, \{D\}_l$);
  $\{T\}_k \leftarrow$ traj_project_and_update($\mathbf{X}, \{\overline{T}\}_k, \mathbf{H}^{-1}$);
  foreach *tracking object s in* $\boldsymbol{X}$ **do**
    if *is_not_counted(s,* $\boldsymbol{M}$*) & inROI(s)* **then**
      if *direction(s) is north2south* **then**
        $n_1 \leftarrow n_1 + 1$;
      else if *direction(s) is south2north* **then**
        $n_2 \leftarrow n_2 + 1$;
    end
  end
end

where J is the total number of frames in which vehicle i was observed by the bus and $p_{i,j}$ is the bird's-eye-view real-world position vector of vehicle i.

The overview of the solution is shown in Figure 4.2. The proposed algorithm, Detect-Track-Project-Count (DTPC), is given in Algorithm 1. The implementation details of the system are presented in Section 4.4.

4.3.2 Two-Dimensional Detection

A vehicle detection D is defined as $D = (\mathbf{b}, l, c)$, where $\mathbf{b} = ((u_1, v_1), (u_2, v_2), (u_3, v_3), (u_4, v_4))$ contains the bounding box corners in pixel coordinates, l is the class label (e.g., car, bus, or truck), and c is the confidence of the detection. A 2D CNN $f_{\text{CNN}} : \mathbf{I} \rightarrow \{D\}_q$ maps each image $\mathbf{I}$ captured by the bus camera to q tuples of detections D. This processing is performed on individual image frames. The object classification information is used to avoid counting pedestrians and bicycles but could also be used in other studies.

The networks commonly known as Mask-RCNN [69], Yolo v3 [70], and Yolo v4 [71], pretrained on the MS COCO dataset [80], are employed for object detection. Based on the ablation study presented subsequently, the best-performing detector was identified as Yolo v4.

4.3.3 Tracking

The goal of tracking is to associate independent frame-by-frame detection results across time. This step is essential for trajectory extraction, which is needed in order to avoid counting the same vehicle more than once.

SORT [81], a fast and reliable algorithm that uses Kalman filtering and the Hungarian algorithm to solve the assignment problem, is employed for the tracking task. First, define the state vector as

$$\mathbf{x} = (u_c, v_c, s, r, \dot{u}, \dot{v}, \dot{s})^\intercal \tag{4.1}$$

where $u_c = u_1 + (u_3 - u_1)/2$ and $v_c = v_2 + (v_4 - v_2)/2$ are the bounding box center coordinates derived from the resultant 2D detection bounding boxes, $s = (u_3 - u_1) \times (v_4 - v_2)$ is its area, $r = (u_3 - u_1)/(v_4 - v_2)$ is its aspect ratio, and $\dot{u}, \dot{v}$, and $\dot{s}$ are the corresponding first derivatives with respect to time.

The next step consists of associating each detection to already existing target boxes with unique tracking IDs. This subproblem can be formulated as an optimal assignment matching problem where the matching cost is the Intersection-over-Union (IoU) value between a detection box and a target box i.e.,

$$\text{IoU} = \frac{Area\,of\,Intersection}{Area\,of\,Union}. \tag{4.2}$$

This problem can be solved with the Hungarian algorithm [105].

After each detection is assigned to a target, the target state is updated with a Kalman filter[106]. The Kalman filter assumes the following dynamical model:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k. \tag{4.3}$$

where, $\mathbf{F}_k$ is the state transition matrix, $\mathbf{B}_k$ is the control input matrix, $\mathbf{u}_k$ is the control vector, and $\mathbf{w}_k$ is normally distributed noise. The Kalman filter recursively estimates the current state from the previous state and the current actual observation as follows:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k \tag{4.4}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\intercal + \mathbf{Q}_k \tag{4.5}$$

where $\mathbf{P}$ is the predicted covariance matrix and $\mathbf{Q}$ is the covariance of the multivariate normal noise distribution. Details of the update step can be found in the original Kalman filter paper [106].

Deep SORT [83] was also considered as an alternative to SORT [81]; however, SORT gave better results. An example of a detected, tracked, and counted vehicle is shown in Figure 4.3.

Figure 4.3: An example of a detected, tracked, and counted vehicle. The proposed system outputs bidirectional traffic counts. The segment ID indicates the current road segment occupied by the bus.

4.3.4 Geo-Referencing and Homography

The proposed method utilizes geo-referencing and homographic calibration to count vehicles in the desired ROI and transforms the detected vehicles' trajectories from pixel coordinates to real-world coordinates.

Global Navigation Satellite System (GNSS) measurement data are used to localize the bus with a pose vector $\mathbf{p}_t^{\text{bus}}$ at time t. A predefined database contains geo-referenced map information about the road network and divides each road into road segments with a predefined geometry, including the lane widths for each segment. Road segments tend to run from one intersection to another but can also be divided at points where the road topology changes significantly, for example, when a lane divides. This information is used to build the ROI in BEV real-world coordinates for each road segment.

The pixel coordinates of each detection can be transformed into real-world coordinates with planar homography using

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.6}$$

since planar homography is up-to-scale with a scaling factor. $\mathbf{H}$, the homography matrix, has eight degrees of freedom. Hence, $\mathbf{H}$ can be determined from four real-world image point correspondences [107]. Finding four point correspondences is fairly straightforward for urban road scenes. Standard road markings and the width of a lane are used to estimate $\mathbf{H}$. The homographic calibration process is shown in Figure 4.4.

Once $\mathbf{H}$ is obtained, the inverse projection can be easily achieved with the inverse homography matrix $\mathbf{H}^{-1}$. Inverse homography is used to convert a BEV real-world ROI to a perspective ROI for the image plane. After obtaining the perspective ROI, vehicles on the corresponding regions can be counted.

4.3.5 Counting

The objective of counting is to count each unique tracked vehicle once if it is a traffic participant and not a parked vehicle, travels in the corresponding travel direction, and is within the ROI.

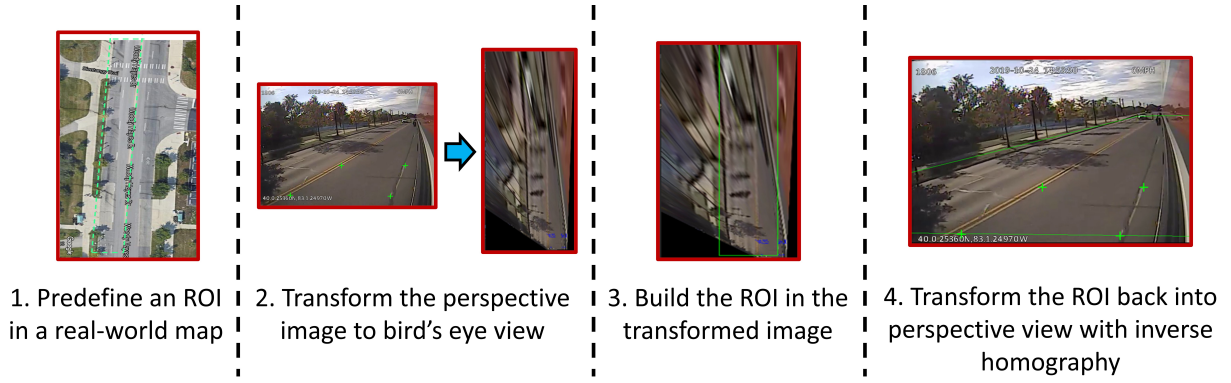


Figure 4.4: ROI alignment with homography. A predefined ROI in a 2D BEV world-map can be utilized for any camera angle with a planar homography transformation. The camera only needs to be calibrated once with four point correspondences.

The 2D object detector's output is considered only if the inferred class $l = \{car, bus, truck\}$ and the bounding box center is within the ROI. Thus, at each counting time t, using $u_{c,i}$ and $v_{c,i}$, the bounding box center coordinates of each tracked vehicle i, the count in the top-to-bottom direction (i.e., opposite the direction of travel of the bus) is updated incrementally $n_1 \leftarrow n_1 + 1$ if the one-step sequence $(u_{c,i,t-1}, v_{c,i,t-1}), (u_{c,i,t}, v_{c,i,t})$ of the image domain trajectory of vehicle i satisfies $\text{sgn}(u_{c,i,t} - u_{c,i,t-1}) = 1$ and vehicle i was not counted before. In a similar fashion, the count in the bottom-to-top direction (i.e., in the direction of travel of the bus) is updated incrementally $n_2 \leftarrow n_2 + 1$ if $\text{sgn}(u_{c,i,t} - u_{c,i,t-1}) = -1$ and vehicle i was not counted before. The ROI alignment for counting purposes is shown in steps 3 and 4 of Figure 4.4.

Distinguishing parked vehicles from traffic flow participants is achieved by the ROI. This distinction and the difference between detecting and counting vehicles is illustrated in Figure 4.5.

4.3.6 Trajectory Extraction in BEV Real-World Coordinates

Finally, for each tracked unique vehicle, a trajectory in BEV real-world coordinates relative to the bus is built by transforming the image-domain coordinates using the inverse homography projection. The image domain trajectory of vehicle i, $\overline{T}_i = \{(u_{c,i,t}, v_{c,i,t}) | t \in (t_1, t_2 \cdots, t_K)\}$, is transformed into T_i, with inverse homography, $(x_{c,i,t}, y_{c,i,t}, 1)^\intercal = \mathbf{H}^{-1}(u_{c,i,t}, v_{c,i,t}, 1)^\intercal$. An example of extracted trajectories is illustrated in Figure 4.6.

Trajectory extraction in BEV real-world coordinates is not strictly necessary for the purpose of counting unique vehicles, which could be achieved without such transformation. However, it is quite valuable for understanding the continuous behavior of traffic participants, including their speed, and therefore could be used as input to other studies.

4.4 Experimental Evaluation

4.4.1 Transit Bus

The Ohio State University (OSU) owns and operates the Campus Area Bus Service (CABS) system consisting of a fleet of about 50 40-foot transit buses that operate on and in the vicinity of the OSU campus, serving around five million passengers annually (pre-COVID-19). As is the case for many transit systems, the buses are equipped with an Automatic Vehicle Location (AVL)

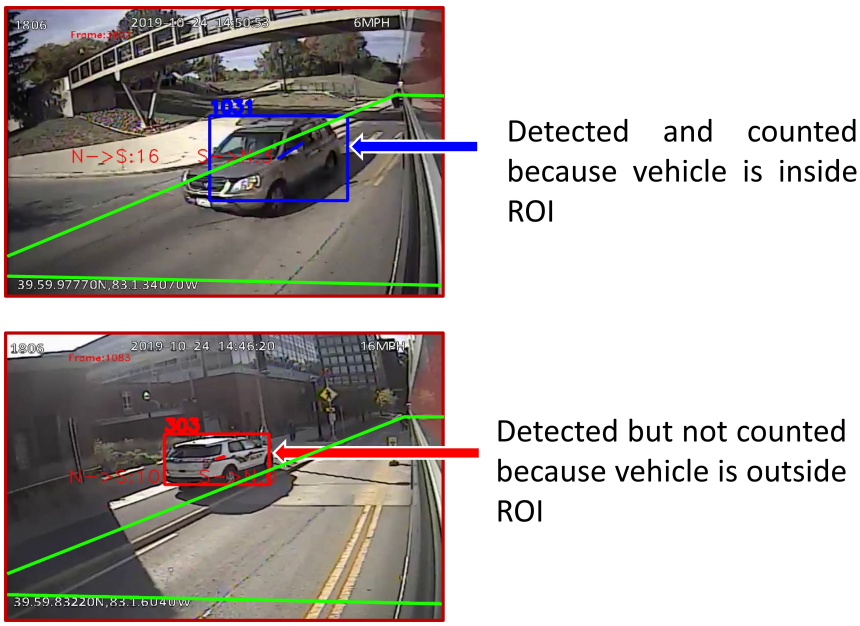


Figure 4.5: Distinguishing parked vehicles from traffic participants. Since the sensor platform is moving, excluding parked vehicles from the total count is not trivial. Using an ROI alleviates this issue.

system that includes GNSS sensors for operational and real-time information purposes and several interior- and exterior-facing monocular cameras that were installed for liability, safety, and security purposes. The proposed method depends on having an external view angle wide enough to capture the motion pattern of the surrounding vehicles. Figures 4.3–4.5 show sample image frames recorded by these cameras.

Video imagery collected from CABS buses while in service are used to implement and test the developed method. We chose to use the left side-view camera in this study because it is mounted higher than the front-view camera, which both reduces the potential for occlusions caused by other vehicles and improves the view of multiple traffic lanes to the side of the bus, particularly on wider multilane roads or divided roads with a median. Moreover, video from these cameras is captured at a lower resolution, which significantly reduces the size of the video files that needed to be offloaded, transferred, and stored. The video footage was recorded at 10 frames per second with a resolution of 696 $\times$ 478.

4.4.2 Implementation Details

The proposed algorithm was implemented in Python using OpenCV, Tensorflow, and Pytorch libraries. For 2D detectors, the original implementations of Mask-RCNN [69], Yolo v3 [70], and Yolo v4 [71] were used. All models were pretrained on the MS COCO dataset [80]. Experiments were conducted with an NVIDIA RTX 2080 GPU.

High-speed real-time applications are not within the scope of this study. For surface road traffic surveillance purposes, low frame rates are sufficient. The video files used in this study were downloaded from the buses after they complete their trips. However, it would be possible to perform the required calculations using edge computing installed on the bus or even integrated with the camera. This is a topic for future study.

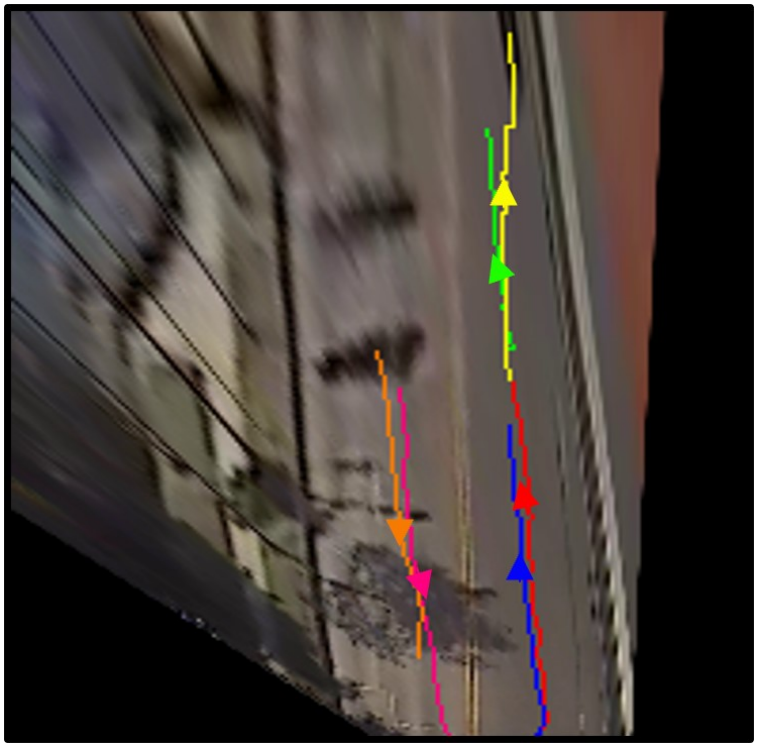


Figure 4.6: Extracted trajectories in real-world BEV coordinates for an observation window of approximately 5 s are shown. Six vehicles were detected, tracked, and have been projected onto a BEV plane. The trajectory of each vehicle is shown in a distinct color.

4.4.3 Data Collection and Annotation

A total of 3.5 hours of video footage, collected during October 2019, was used for the experimental evaluation of the ablation study. An additional 3 hours of video footage, collected during March 2022, was used for the evaluation of the impacts of adverse weather. The West Campus CABS route on which the footage was recorded traverses four-lane and two-lane public roads. The route is shown in Figure 4.7. Details of the route can be found in [79]. As described above, this route was divided into road segments which tend to run from one intersection to another but may be divided at other points, such as when the road topology changes significantly. This can include points at which the number of lanes change or, in the case of this route, areas in which a tree-lined median strip obscures the camera view. Using this map database, sections with a tree-lined median strip and occlusions were excluded from the study. For each road segment, an ROI was defined with respect to road topology and lane semantics to specify the counting area. As a result, a total of 55 bus pass and roadway segment combinations were used in the ablation study evaluation analysis.

We note that should the bus not follow its prescribed route or the route changes, the algorithm can detect and report this due to the map database, as well as remove or ignore those portions of traveled road that are off-route or for which the road geometry information needed to form an ROI is not available.

The video footage was processed by human annotators in order to extract ground-truth vehicle counts. Annotators used a GUI to incrementally increase the total vehicle count for each oncoming unique vehicle and to capture the corresponding video frame number. Annotators counted vehicles once and only if the vehicle passed a virtual horizontal line drawn on the screen of the GUI while

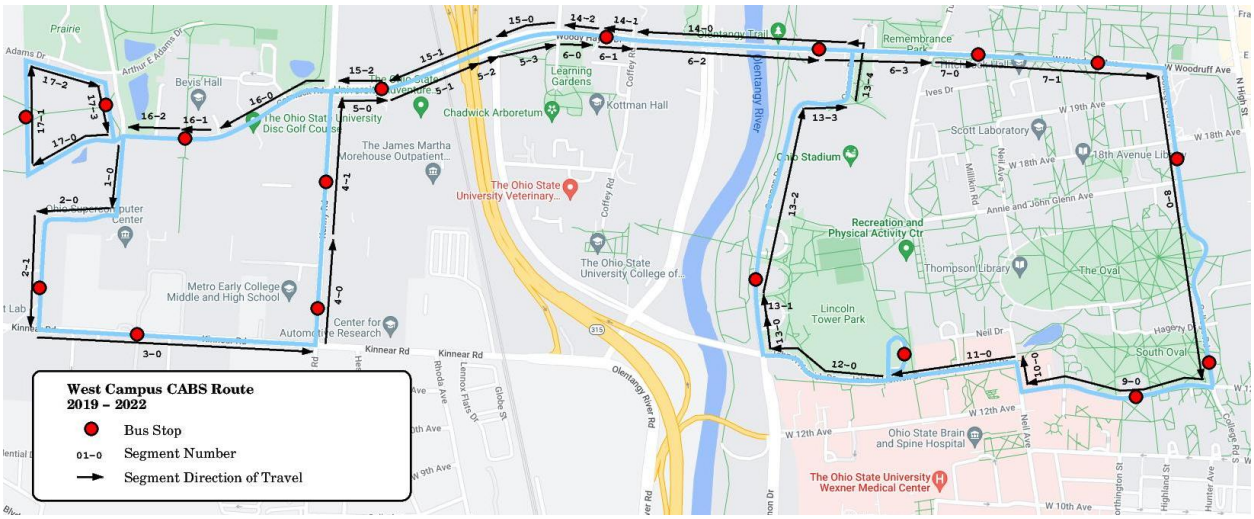


Figure 4.7: Data were collected in the main campus area of the Ohio State University in Columbus, OH. The route contains 4-lane and 2-lane public roads and on-campus streets. Some streets were divided by a median strip. The direction of travel arrows indicates the travel direction of the bus.

traveling in the direction opposite to that of the bus.

After annotation, the video frames, counts, and GNSS coordinates were synchronized for comparison with the results of the developed image-processing-based fully automated method.

4.4.4 Evaluation Metrics

First, for each of the 55 bus pass and segment combination, ground-truth counts and inferred counts using the developed automatic counting method were compared with one another. In addition to examining a scatter plot that depicts this comparison, four metrics were considered, namely difference $c_{gt} - c_i$, absolute difference $|c_{gt} - c_i|$, absolute relative difference $|c_{gt} - c_i|/c_{gt}$, and relative difference $(c_{gt} - c_i)/c_{gt}$, where c_{gt} is the ground-truth count, and c_i is the inferred count.

The sample mean, sample median, and empirical Cumulative Distribution Function (eCDF) were calculated for the proposed and alternative automated baseline methods.

4.4.5 Ablation Study

An extensive ablation study was conducted to identify the best alternative among multiple methods that could be applied to each phase of the counting system. All the combinations of modules from the following list were compared with one another using the sample mean and sample median of the evaluation metrics to find the better performing combinations:

- 2D Detectors: Mask-RCNN [69], Yolo v3 [70], and Yolo v4. [71]
- Tracking option: SORT [81] and Deep SORT [83].
- ROI strategy: No ROI, Generic (single) ROI, and Dynamic Homography ROI.

The Generic ROI was defined based on a standard US two-lane road in perspective view. Each combination option is denoted by the sequence of uppercase first letters of the name of the modules. For example, Y4SGR stands for the Yolo 4 detector, SORT tracker, and Generic ROI combination.

The proposed method is denoted with "Proposed" and stands for the Yolo 4 detector, SORT tracker, and Dynamic ROI combination.

4.5 Results and Discussion

4.5.1 Ablation study

Considering all combinations of tools, the proposed method, consisting of the Yolo v4 object detector, SORT tracker, and Dynamic Homography ROI, was found to be the best based on the sample mean, sample median, and eCDF of the four evaluation metrics. Figure 4.8 shows pairs of inferred counts plotted against ground-truth counts for each of the 55 road segment passes for select combinations of modules (including the best-performing one among all combinations) that reflect a wide range of performance. For a perfect automatic counting system, the scatter plot of inferred counts versus ground truth should be on the identity line (y = x). A regression line was estimated for each combination. The estimated lines are shown in Figure 4.8, along with their confidence limits. Clearly, the proposed Y4SDR outperforms the other four combinations shown in Figure 4.8 by a substantial margin.

As expected, the proposed homography-derived ROI performs better than the No ROI and the Generic ROI modules. The latter two ROI modules lead to over counts because of their limitations in distinguishing traffic participants from irrelevant vehicles. In contrast, the dynamic ROI allows for counting vehicles only in the pertinent parts of the road network, thus omitting irrelevant vehicles, which in this study consisted mostly of vehicles parked on the side of the roads or in adjacent parking lots. This result validates the developed birdâ€™s-eye-view inverse projection approach to defining dynamic ROIs suitable to each roadway segment.

Figure 4.9 shows the eCDF plots of the four different functions from the four different trackers. In addition to confirming the overall superiority of the Y4SDR combination, these plots also indicate that this combination is highly reliable, as is evident from the thin tails of the distributions.

Summary results for all combinations of modules are shown in Table 4.1. Specifically, the sample mean and sample median of absolute differences and absolute relative differences are given for each considered detector, tracker, and ROI combination. Across all combinations of detector and ROI options, SORT outperformed Deep SORT consistently. This result indicates that the deep association metric used by Deep SORT needs more training to be efficient. In addition, across all combinations of tracker and ROI options, the detectors Yolo v3 and Yolo v4 performed similarly and better than Mask-RCNN. This result indicates that two-stage detectors, such as Mask-RCNN, are prone to overfitting to the training data more than single-stage detectors. Moreover, it can be seen that for all detector and tracker combinations, the proposed dynamic ROI option outperforms the other two ROI options by a large margin, as one might expect. Finally, from the results in Table 4.1, the Y4SDR combination is seen to perform the best among all combinations.

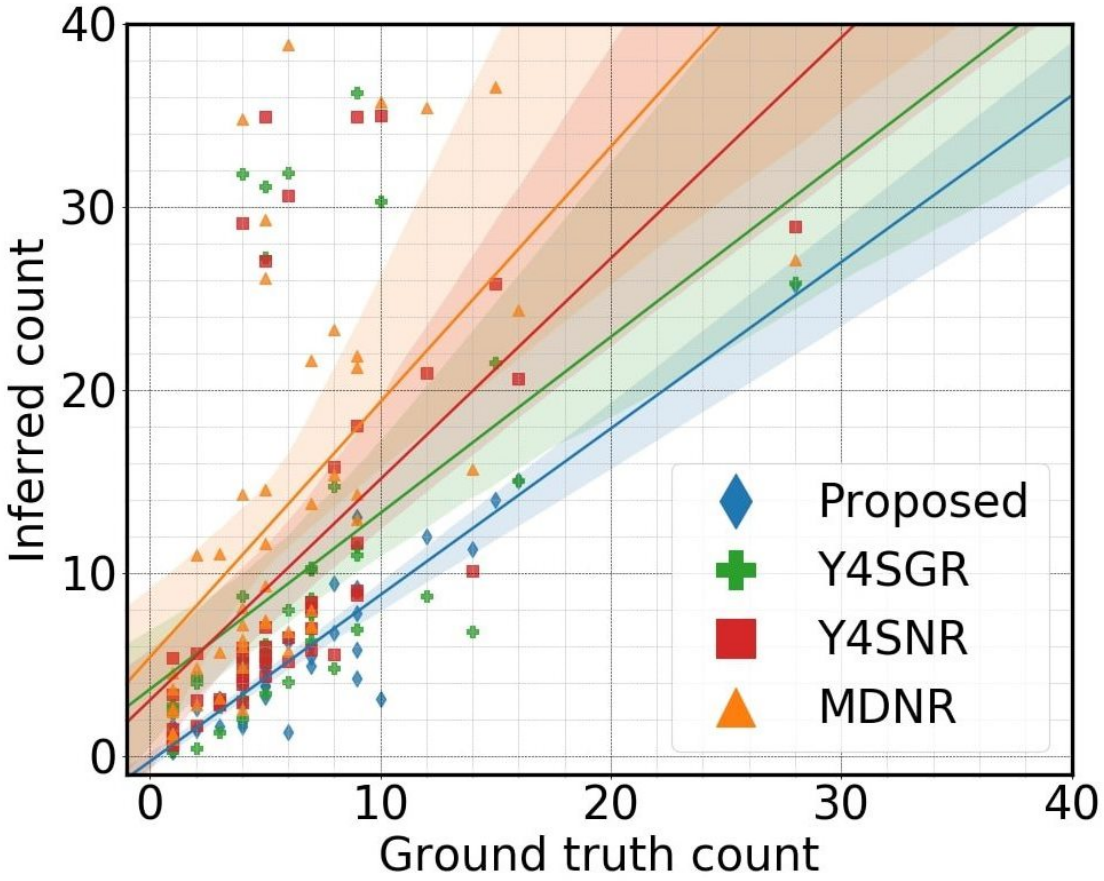


Figure 4.8: Inferred count versus ground-truth count: Y4SDR (Proposed) indicates Yolo 4 detector, SORT tracker, and Dynamic ROI; Y4SGR indicates Yolo 4 detector, SORT tracker, and Generic ROI; Y4SNR indicates Yolo 4 detector, SORT tracker, and No ROI; MDNR indicates Mask-RCNN detector, Deep SORT tracker, and No ROI. An ideal counter should be on the identity line ($y = x$). The proposed method is close to ideal, while other methods overcount.

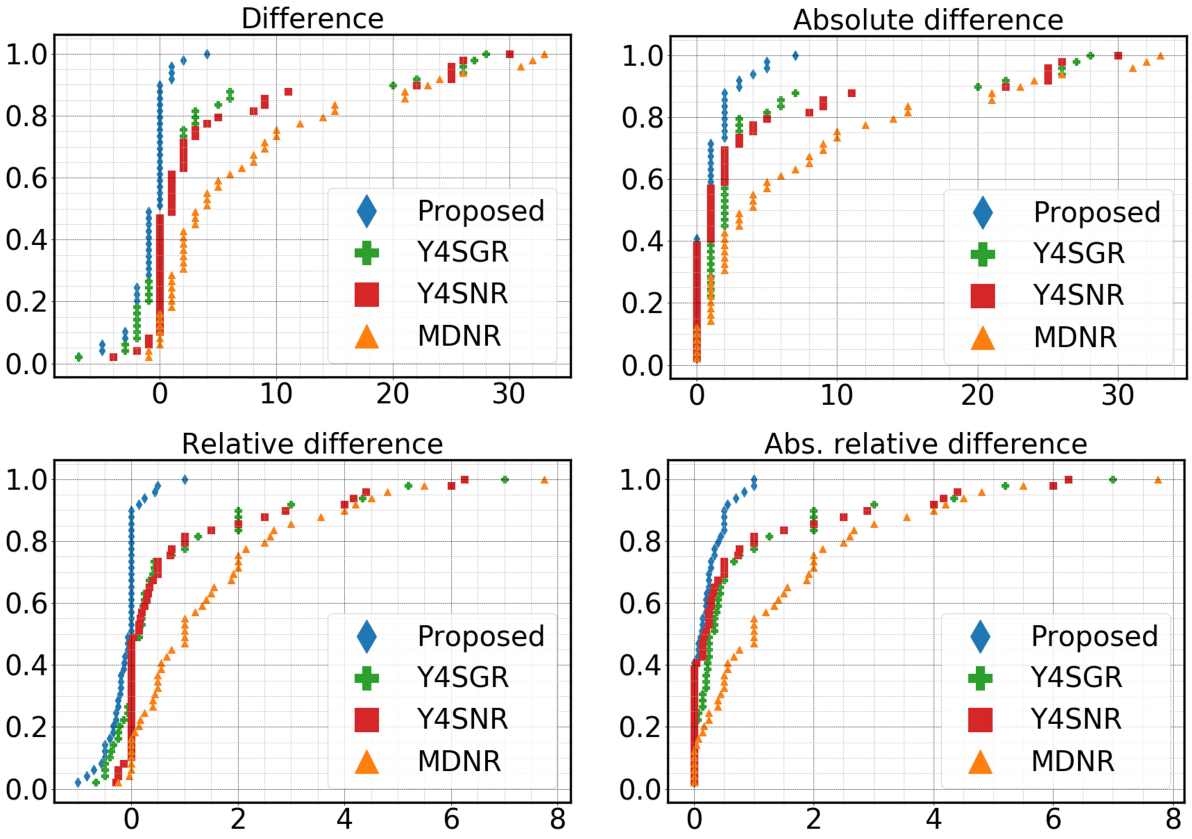


Figure 4.9: Empirical Cumulative Distribution Functions (eCDFs) of the proposed method and three other alternative combinations of modules. The proposed method consistently makes fewer errors (difference from the ground-truth count) across multiple road segments.

Table 4.1: Comparison of alternative configurations of modules. The proposed method with a Yolo 4 detector, SORT tracker, and Dynamic ROI has the lowest difference (error) from the ground-truth counts. A good ROI ensures the exclusion of parked and irrelevant vehicles from the count.

		Absolute Error		**Abs. Relative Error**	
		Tracker Option		**Tracker Option**	
ROI Option	**Detector Option**	**SORT [81]**	**Deep SORT [83]**	**SORT [81]**	**Deep SORT [83]**
Dynamic ROI (Proposed)		μ (M)	μ (M)	μ (M)	μ (M)
	Yolo v3 [70]	1.24 (1)	2.02 (1)	0.22 (0.14)	0.32 (0.28)
	Yolo v4 [71]	1.18 (1)	1.67 (1)	0.21 (0.12)	(0.32, 0.25)
	Mask-RCNN [69]	2.34 (1)	2.32 (1)	0.39 (0.25)	0.46 (0.26)
Generic ROI					
	Yolo v3 [70]	4.16 (1)	4.48 (2)	0.75 (0.25)	0.84 (0.33)
	Yolo v4 [71]	4.30 (1)	4.59 (2)	0.75 (0.25)	0.89 (0.33)
	Mask-RCNN [69]	5.40 (2)	6.24 (4)	0.94 (0.5)	1.26 (0.86)
No ROI					
	Yolo v3 [70]	4.44 (1)	5.04 (1)	0.85 (0.25)	0.98 (0.28)
	Yolo v4 [71]	4.69 (1)	5.14 (2)	0.85 (0.2)	1.02 (0.35)
	Mask-RCNN [69]	6.02 (3)	7.89 (4)	1.05 (0.5)	1.53 (1)

μ: mean, M: median.

4.5.2 Impacts of Adverse Weather and Lighting

Having determined the best algorithm to implement from the results of the ablation study, one may also consider the performance of both the image processing and the overall counting system in the presence of inclement weather, including rain or post-rain conditions with puddles and irregular lighting effects. Inclement weather exposes several issues with an image-processing-based system, including darker and more varied lighting conditions, reflective puddles that can be misidentified as vehicles (Figure 4.10a), roadway markings misidentified as vehicles (Figure 4.10b), water droplets and smudges formed from dust or dirt and water partially obscuring some portions of the image (Figure 4.10c), and finally, blowing heavy rain, or possibly snow, with falling drops that can be seen in the images (Figure 4.10d).

(a) puddle in partial sunlight misidentified as car

(b) crosswalk stripes misidentifed as car

(c) drops, smudges, and fog on camera after rain

(d) blowing heavy rain with visible falling drops

Figure 4.10: Examples of challenges for image processing results during and after heavy rain.

We compared human-extracted ground truth with the automated extraction and counting results from videos of several loops of the West Campus route both on dry, cloudless, sunny days and days with rainfall varying from a light drizzle to heavy thunderstorms, which also included post-rain periods of time with breaking clouds that provided both darker and occasionally sunlit conditions. All the videos were acquired from actual in-service transit vehicles during the middle of the day.

Qualitatively, rainfall and wet conditions caused more transient—lasting only for one to two frames— inaccurate detection events to occur in the base image processing portion of the algorithm,

including vehicles not detected, misclassified vehicles and other objects, and double detections when a vehicle is split into two overlapping objects, along with incorrectly identifying a greater number of background elements (e.g., puddles and road markings) as vehicles or objects. However, these transient events generally make little difference in the final vehicle counts, as the overall algorithm employs both filtering to eliminate unreasonable image processing results and tracking within the region of interest, such that a vehicle is declared present and counted only after a fairly complete track is established from the point it enters to the point it departs the camera's field of view. This approach, in general terms, imposes continuity requirements, causing transient random events to be discarded unless they are so severe as to make it impossible to match and track a vehicle as it passes through the images, thereby improving the robustness of the approach.

More problematic, however, are persistent artifacts such as water droplets or smudges formed by wet dust and dirt, which often appear on the camera lens or enclosure cover after the rain stops and the lens begins to dry. For example, in one five-minute period of one loop, there was a large smudge on the left side at the vertical center of the lens, which obscured the region of the image where vehicles several lanes to the left of the transit bus tended to cross through and leave the image frame, causing them to not be counted due to incomplete tracking.

We note that these are general problems that can affect most video and image processing systems—if the lens is occluded you cannot see anything in that region of the image. It would be possible in future work to implement a method to dynamically detect when the lens is occluded and note the temporary exclusion of those regions from counts while the occlusion persists. As a final note, heavy rain was also observed to clean the lens.

Quantitatively, we present the results of the automated extraction and counting experiments in Table 4.2 for both the dry, clear, sunny loops and the rainy and post-rain loops. The table presents the percentage of correctly counted vehicles, the percentage of vehicles missed due to not being detected at all or detected too infrequently to build a sufficient track, the percentage of vehicles not detected due to being substantially occluded by another vehicle (this is not actually a weather-related event but is included for completeness), and the percentage of vehicles not counted due to a smudge or water droplet covering part of the camera lens. The final columns of Table 4.2 indicate the percentage of double-counted vehicles and the percentage of false detections or identifications that persisted long enough to be tracked and incorrectly counted. These are two impacts of transient image processing failures that are not always detected, at present, by our filtering and tracking algorithms.

As can be seen in Table 4.2, the overall effects of poor weather conditions result in only a minor increase in the errors committed by this system.

Table 4.2: Comparison of vehicle detection and counting results for clear/dry versus rain/post-rain weather conditions.

Weather	**True Vehicle Counts**	**Missed Vehicles**	**Missed Vehicles Due to Occlusion by other Vehicles**	**Missed Vehicles Due to Smudge or Drops on Lens**	**Double-Counted Vehicles**	**Falsely Counted as Vehicle**
	%	%	%	%	%	%
Dry/Clear/Sunny	90.7%	4.6%	4.6%	-	-	0.9%
Rain/Post-Rain	88.9%	5.9%	2.8%	2.3%	2.0%	1.7%

4.6 Conclusions

This chapter introduced and evaluated a fully automatic vision-based method for counting and tracking vehicles captured in video imagery from cameras mounted on buses for the purpose of estimating traffic flows on roadway segments using a previously developed moving observer methodology. The proposed method was implemented and tested using imagery from in-service transit buses, and its feasibility and accuracy was shown through experimental validation. Ablation studies were conducted to identify the best selection of alternative modules for the automated method.

The proposed method can be directly integrated into existing and future ground-vehicle-based traffic surveillance approaches. Furthermore, since cameras are ubiquitous, the proposed method can be utilized for different applications.

Reimagining public transit buses as data collection platforms has great promise. With widespread deployment of the previously developed moving observer methodology facilitated by the full automation of vehicle counting proposed in this effort, a new dimension can be added to intelligent traffic surveillance. Combined with more conventional methods, such as fixed location and the emerging possibilities of UAV-based surveillance, spatial and temporal coverage of roadway networks can be increased and made more comprehensive. This three-pronged approach has the potential of achieving close to full-coverage traffic surveillance in the future.

Future work could focus on further comprehensive evaluation of the method presented here under more varied conditions, subsequent refinements, and the use of edge computing technologies to perform the image processing and automatic counting onboard the buses in real time. Another potential extension would involve coordinated tracking of vehicles across multiple buses, although this raises certain social and political privacy issues that would need to be addressed. Finally, there could be significant uses and value in vehicle motion and classification information, potential extensions to include tracking and counting bicycles, motorcycles, and pedestrians, and the eventual integration into smart city infrastructure deployments.

4.7 Data Availability

The original video data used in this study as well as the manually extracted ground truth records are available in the Zenodo repository at DOI 10.5281/zenodo.7955464.

4.8 Acknowledgements

The authors are grateful to The Ohio State University's Transportation and Traffic Management and the Campus Area Bus Service, notably Sean Roberts for providing video footage and AVL data and former Director Beth Snoke and current Director Tom Holman for their administrative support. The authors also wish to acknowledge the efforts of Diego Galdino of the OSU Transit Lab in identifying the video clips used in this study.

Chapter 5

A Vision-Based Social Distancing and Critical Density Detection System for COVID-19

5.1 Introduction

With the outbreak of the novel Coronavirus Disease 2019 (COVID-19) [108], social distancing (SD) emerged as an effective measure against it. Maintaining social distancing in public areas such as transit stations, shopping malls, and university campuses is crucial to prevent or slow the spread of the virus. The practice of social distancing (SD) may continue in the following years until the spread of the virus is completely phased out. However, social distancing is prone to be violated unwillingly, as populations are not accustomed to keeping the necessary 2-meter bubble around each individual. This work proposes a vision-based automatic warning system that can detect social distancing statuses and identify a critical pedestrian density threshold to modulate inflow to crowded areas. Besides being an automated monitoring and warning system, the proposed framework can serve as a tool to detect key variables and statistics for local and global virus control.

Vision-based automatic detection and control systems [109, 110, 111, 112, 113, 114, 115, 116] are economic and effective solutions to mitigate the spread of COVID-19 in public areas. Although the conceptualization is straightforward, the design and deployment of such systems require smart system design and serious ethical considerations.

First, the system must be fast and real-time. Only a real-time system can detect social distancing statuses immediately and send a warning. Privacy concerns [117, 118, 119] can be mitigated with a real-time system by not storing sensitive image data while only keeping aggregate statistics, such as the number of SD violations. With a real-time active surveillance system, appropriate measures can be taken as quickly as possible to reduce further spread of COVID-19.

The second design objective is that the system must be accurate and effective enough, but not discriminative. The safest way to achieve this is to build an AI-based detection system. AI-based vision detectors have become the state-of-the-art in people-detection tasks, achieving higher scores in most vision benchmarks than detectors with hand-crafted feature extractors. Furthermore, the latter may lead to maligned designs, whereas an end-to-end AI-based system, such as a deep neural network without any feature-based input space, is much fairer, with one caveat: the training data distribution must be fair.

The third objective aims to provide a more advanced measure than pure social distancing monitoring to further reduce the spread of COVID-19. This leads to our proposed approach of

critical pedestrian density identification. The critical density may serve as an indicator to inform the space manager to control the entry port to regulate the incoming pedestrian flow. An online warning is also possible, but it should be non-alarming. For example, the system can send a non-intrusive audio-visual cue to the vicinity of the social distancing violation. Individuals in this region can then make their own decisions with this cue.

We identify the fourth design objective as trust establishment. The whole system and its implementation must be open-sourced. Open-sourcing is crucial for establishing trust between the active surveillance system and society. In addition, researchers and developers can freely and quickly access relevant material and further improve their own designs according to their requirements. This can hasten the development and deployment of anti-COVID-19 technologies, leading to stopping of the spread of the deadly disease and saving lives.

Against this backdrop, we propose a non-intrusive, AI-based active surveillance system for social distancing detection, monitoring, analysis, and control. The overview of the system is shown in Figure 5.1. The proposed system first uses a pre-trained deep convolutional neural network (CNN) [120, 71] to detect individuals with bounding boxes in a given monocular camera frame. Then, detections in the image domain are transformed into real-world bird's-eye-view coordinates for social distancing detection. Once the social distancing is detected, information is passed to two branches for further processing. One branch is online monitoring and control. If a social distancing violation happens, the system emits a non-alarming audio-visual cue. Simultaneously, the system measures social (pedestrian) density. If the social density is larger than a critical threshold, the system sends an advisory inflow modulation signal to prevent overcrowding. The other branch is offline analysis, which provides necessary information for overcrowding prevention and policymaking. The main analysis is the identification of a critical social density. If the pedestrian density is regulated under this critical value, the probability of social distancing violations will be kept near zero. Finally, the regulator can receive both the offline aggregate statistics and the online status of social distancing control. If immediate action is required, the regulator can act as quickly as possible.

The overall system never stores personal information. Only the processed average results, such as the number of violations and pedestrian density, are stored. This is extremely important for privacy concerns. Our system is also open-sourced for further development.

Our main contributions are:

- A novel, vision-based, real-time social distancing and critical social density detection system.
- Definition of critical social density and a statistical approach to measuring it.
- Measurements of social distancing and critical density statistics of common crowded places, such as the New York Central Station, an indoor mall, and a busy town center in Oxford.
- Quantitative validation of the proposed approach to detect social distancing and critical density.

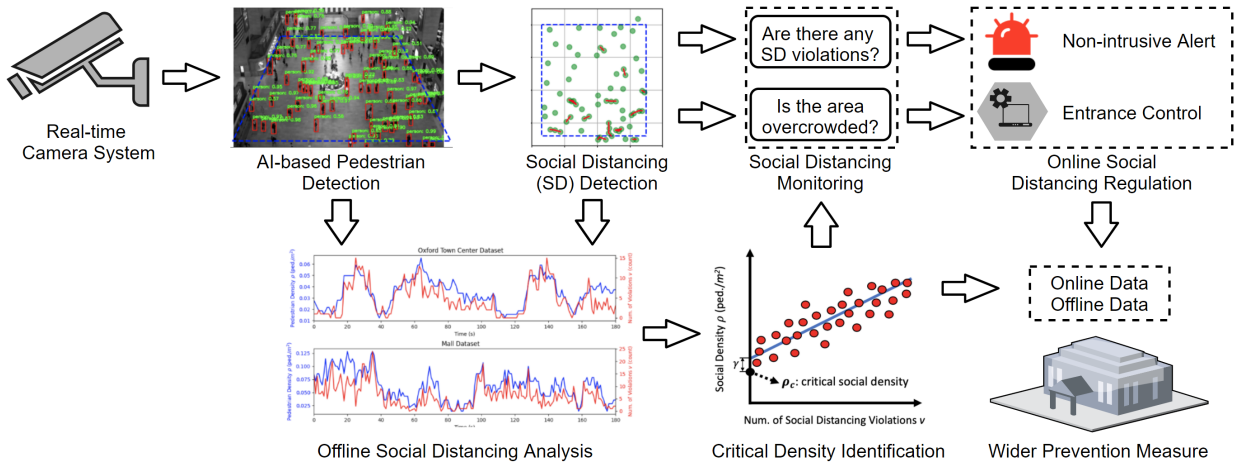


Figure 5.1: Overview of the proposed system. An audio-visual cue is emitted each time an individual breach of social distancing is detected. We also make a novel contribution by defining a critical social density value ρ_c for measuring overcrowding. Entrance into the region-of-interest can be modulated online with this value. The aggregated non-personal data can also be analyzed offline to provide more insights into the social distancing practice in different public areas. Based on both online and offline data, wider prevention measures can be taken as quickly as possible when necessary. Our system is real-time and does not record data.

5.2 Related Work

Social distancing for COVID-19. COVID-19 has caused severe acute respiratory syndromes around the world since December 2019 [121]. Social distancing is an effective measure to slow the spread of COVID-19 [108], which is defined as keeping a minimum of 2 meters (6 feet) apart from other individuals to avoid possible contact. Further analysis [122] also suggests that social distancing has substantial economic benefits. COVID-19 may not be completely eliminated in the short term, but an automated system that can help in the monitoring and analyzing social distancing measures can greatly benefit our society. Statistics from recent works [108] have demonstrated that strong social distancing measures can indeed reduce the growth rate of COVID-19.

The requirement of social distancing has shaped the development of IoT sensors and smart city technologies. The spread prevention and outbreak alerting of COVID-19 now must be considered in these areas. A recent work [123] reviews potential solutions and recent approaches, such as IoT sensors, social media, personal gadgets, and public agents for COVID-19 outbreak alerting. Another work [124] summarises IoT and associated sensor technologies for virus tracing, tracking, and spread mitigation, and highlights the challenges of deploying such sensor hardware.

With the help of the above technologies, social distancing can be better practiced, which will eventually alleviate the spread of the virus and "flatten the curve".

Social distancing monitoring. In public areas, social distancing is mostly monitored by vision-based IoT systems with pedestrian detection capabilities. Appropriate measures are subsequently taken on this basis.

Pedestrian detection can be viewed as a sub-task of generic object detection or as a specific task of detecting pedestrians only. A detailed survey of 2D object detectors and the corresponding datasets, metrics, and fundamentals can be found in [125]. Another survey [126] focuses on deep-learning-based approaches for both the generic object detectors and the pedestrian detec-

Table 5.1: Comparison of vision-based social distancing detection.

Work	CC	AVS	RPDE	SDDE	OC
Khandelwal et. al. [116]	Yes		Yes		
DeepSOCIAL [113]	Yes	Yes	Yes		
Ahmed et. al. [114]			Yes		
Cota [115]	Yes		Yes	Yes	
Ours	Yes	Yes	Yes	Yes	Yes

Acronyms: camera calibration (CC), applicable to various scenes (AVS), real-time pedestrian detection evaluation (RPDE), social distancing detection evaluation (SDDE), overcrowding control (OC). This table compares the features that are relevant to this work. Some works may provide additional features.

tors. Generally speaking, state-of-the-art detectors are divided into two categories. One category is two-stage detectors. Most of them are based on R-CNN [127, 120], which starts with region proposals and then performs the classification and bounding box regression. The other category is one-stage detectors. Prominent models include YOLO [128, 70, 71], SSD [129], and EfficientDet [130]. The detectors can also be classified as anchor-based [127, 120, 70, 71, 129, 130] or anchor-free approaches [131, 132]. The major difference between them is whether to use a set of predefined bounding boxes as candidate positions for the objects. Evaluating these approaches was usually done using the datasets of Pascal VOC [133] and MS COCO [80]. The accuracy and real-time performance of these approaches are good enough for deploying pre-trained models for social distancing detection.

There are several emerging technologies that assist in the practice of social distancing. A recent work [109] has identified how emerging technologies like wireless, networking, and artificial intelligence (AI) can enable or even enforce social distancing. The work discussed possible basic concepts, measurements, models, and practical scenarios for social distancing. Another work [110] has classified various emerging techniques as either human-centric or smart-space categories, along with the SWOT analysis of the discussed techniques. Social distancing monitoring is also defined as a visual social distancing (VSD) problem in [111]. The work introduced a skeleton-detection-based approach for inter-personal distance measuring. It also discussed the effect of social context on people's social distancing and raised the concern of privacy. The discussions are inspirational, but again, do not generate solid results for social distancing monitoring and leaves the question open. A specific social distancing monitoring approach [112] that utilizes YOLOv3 and Deepsort was proposed to detect and track pedestrians followed by calculating a violation index for non-social-distancing behaviors. The approach is interesting, but the results do not contain any statistical analysis. Furthermore, there is no implementation or privacy-related discussion other than the violation index. Another work [113] developed a DNN model called DeepSOCIAL for people detection, tracking, and distance estimation. In addition to social distancing monitoring, it also performed dynamic risk assessment. However, this work did not specifically consider the performance of pure violation detection and the solution to prevent overcrowding. More recently, [134] provides a data-driven deep-learning-based framework for the sustainable development of a smart city. Some other works [116, 114, 115] also proposed vision-based solutions. A comparison of the above methods with our proposed method can be found in Table 5.1.

Several prototypes utilizing machine learning and sensing technologies have already been developed. Landing AI [135] was almost the first one to introduce a social distancing detector using a surveillance camera to highlight people whose physical distance is below the recommended value. A similar system [116] was deployed to monitor worker activity and send real-time voice alerts in a

manufacturing plant. In addition to surveillance cameras, LiDAR-based [136] and stereo-camera-based [137] systems were also proposed, which demonstrated that different types of sensors besides surveillance cameras can also help.

The above systems are interesting, but recording data and sending intrusive alerts might be unacceptable by some people. On the contrary, we propose a non-intrusive warning system with softer omnidirectional audio-visual cues. In addition, our system evaluates critical social density and modulates inflow into a region-of-interest.

5.3 Preliminaries

Object detection with deep learning. Object detection in the image domain is a fundamental computer vision problem. The goal is to detect instances of semantic objects that belong to certain classes, such as humans, cars, and buildings. Recently, object detection benchmarks have been dominated by deep Convolutional Neural Network (CNN) models [127, 120, 70, 71, 129, 130]. For example, top scores on MS COCO [80], which has over 123K images and 896K objects in the training-validation set and 80K images in the testing set with 80 categories, have almost doubled thanks to the recent breakthrough in deep CNNs.

These models are usually trained by supervised learning, with techniques like data augmentation [138] to increase the variety of data.

Model generalization. The generalization capability [139] of the state-of-the-art is good enough for deploying pre-trained models to new environments. For 2D object detection, even with different camera models, angles, and illumination conditions, pre-trained models can still achieve good performance.

Therefore, a pre-trained state-of-the-art deep-learning-based pedestrian detector can be directly utilized for the task of social distancing monitoring.

5.4 Method

We propose to use a fixed monocular camera to detect individuals in a region of interest (ROI) and measure the inter-personal distances in real time without data recording. The proposed system sends a non-intrusive audio-visual cue to warn the crowd if any social distancing breach is detected. Furthermore, we define a novel critical social density metric and propose to advise not entering into the ROI if the density is higher than this value. The overview of our approach is given in Figure 5.1, and the formal description starts below.

5.4.1 Problem Formulation

We define a scene at time t as a sextuple $S = (\mathbf{I}, A_0, d_c, c_1, c_2, U_0)$, where $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$ is an RGB image captured from a fixed monocular camera with height H and width W. $A_0 \in \mathbb{R}$ is the area of the ROI on the ground plane in the real world and $d_c \in \mathbb{R}$ is the required minimum physical distance. c_1 is a binary control signal for sending a non-intrusive audio-visual cue if any inter-pedestrian distance is less than d_c. c_2 is another binary control signal for controlling the entrance to the ROI to prevent overcrowding. Overcrowding is detected with our novel definition of critical social density ρ_c. ρ_c ensures the social distancing violation occurrence probability stays lower than U_0. The threshold U_0 should be set as small as possible to reduce the probability of social distancing violation. For example, the threshold could be $U_0 = \frac{1-P_{\mathrm{CI}}}{2}$, where P_{CI} is the

cumulative probability of the 95% confidence interval of a normal distribution. Other choices of U_0 also work, depending on the specific requirement of social distancing monitoring.

Problem 1. Given S, we are interested in finding a list of pedestrian position vectors $P = (\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n)$, $\mathbf{p} \in \mathbb{R}^2$, in real-world coordinates on the ground plane and a corresponding list of inter-pedestrian distances $D = (d_{1,2}, \cdots, d_{1,n}, d_{2,3}, \cdots, d_{2,n}, \cdots, d_{n-1,n})$, $d \in \mathbb{R}^+$. n is the number of pedestrians in the ROI. Additionally, we are interested in finding a critical social density value ρ_c. ρ_c should ensure the probability $p(d > d_c | \rho < \rho_c)$ stays over $1 - U_0$, where we define social density as $\rho := n/A_0$.

Once Problem 1 is solved, the following control algorithm can be used to warn or advise the population in the ROI.

Algorithm 1. If $d \leq d_c$, then a non-intrusive audio-visual cue is activated with setting the control signal $c_1 = 1$, otherwise $c_1 = 0$. In addition, if $\rho > \rho_c$, then entering the area is not advised with setting $c_2 = 1$, otherwise $c_2 = 0$.

Our solution to Problem 1 starts below.

5.4.2 Pedestrian Detection in the Image Domain

First, pedestrians are detected in the image domain with a deep CNN model trained on a real-world dataset:

$$\{T_i\}_k = f_{\text{cnn}}(\mathbf{I}). \tag{5.1}$$

$f_{\text{cnn}} : \mathbf{I} \rightarrow \{T_i\}_n$ maps an image $\mathbf{I}$ into n tuples $T_i = (l_i \mathbf{b}_i, s_i), \forall i \in \{1, 2, \cdots, n\}$. n is the number of detected objects. $l_i \in L$ is the object class label, where L, the set of object labels, is defined in f_{cnn}. $\mathbf{b}_i = (\mathbf{b}_{i,1}, \mathbf{b}_{i,2}, \mathbf{b}_{i,3}, \mathbf{b}_{i,4})$ is the associated bounding box (BB) with four corners. $\mathbf{b}_{i,j} = (x_{i,j}, y_{i,j})$ gives pixel indices in the image domain. The second sub-index j indicates the corners at top-left, top-right, bottom-left, and bottom-right, respectively. s_i is the corresponding detection score. Implementation details of f_{cnn} is given in Section 5.5.

We are only interested in the case of $l =$ 'person'. We define $\mathbf{p}'_i$, the pixel pose vector of person i, by using the middle point of the bottom edge of the BB:

$$\mathbf{p}'_i := \frac{(\mathbf{b}_{i,3} + \mathbf{b}_{i,4})}{2}. \tag{5.2}$$

5.4.3 Image to Real-World Mapping

The next step is obtaining the second mapping function $h : \mathbf{p}' \rightarrow \mathbf{p}$. h is an inverse perspective transformation function that maps $\mathbf{p}'$ in image coordinates to $\mathbf{p} \in \mathbb{R}^2$ in real-world coordinates. $\mathbf{p}$ is in 2D bird's-eye-view (BEV) coordinates by assuming the ground plane $z = 0$. We use the following well-known inverse homography transformation [107] for this task:

$$\mathbf{p}^{\text{bev}} = \mathbf{M}^{-1} \mathbf{p}^{\text{im}}, \tag{5.3}$$

where $\mathbf{M} \in \mathbb{R}^{3 \times 3}$ is a transformation matrix describing the rotation and translation from world coordinates to image coordinates. $\mathbf{p}^{\text{im}} = [p'_x, p'_y, 1]$ is the homogeneous representation of $\mathbf{p}' = [p'_x, p'_y]$ in image coordinates, and $\mathbf{p}^{\text{bev}} = [p_x^{\text{bev}}, p_y^{\text{bev}}, 1]$ is the homogeneous representation of the mapped pose vector.

The transformation matrix $\mathbf{M}$ can be found by identifying the geometric relationship among some key points in both the real world and the image, respectively, and then calculating $\mathbf{M}$ based on homography [107]. More details on camera calibration in this particular work can be found in Section 5.5.

The world pose vector $\mathbf{p}$ is derived from $\mathbf{p}^{\text{bev}}$ with $\mathbf{p} = [p_x^{\text{bev}}, p_y^{\text{bev}}]$.

5.4.4 Social Distancing Detection

After getting $P = (\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n)$ in real-world coordinates, obtaining the corresponding list of inter-pedestrian distances D is straightforward. The distance $d_{i,j}$ for pedestrians i and j is obtained by taking the Euclidean distance between their pose vectors:

$$d_{i,j} = \|\mathbf{p}_i - \mathbf{p}_j\|. \tag{5.4}$$

The total number of social distancing violations v in a scene can be calculated by:

$$v = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbb{I}(d_{i,j}), \tag{5.5}$$

where $\mathbb{I}(d_{i,j}) = 1$ if $d_{i,j} < d_c$, otherwise 0.

5.4.5 Critical Social Density Estimation

Finally, we want to find a critical social density value ρ_{c} that can ensure the social distancing violation occurrence probability stays below U_0. It should be noted that a trivial solution of $\rho_{\text{c}} = 0$ will ensure $v = 0$, but it has no practical use. Instead, we want to find the maximum critical social density ρ_{c} that can still be considered safe.

To find ρ_{c}, we propose to conduct a simple linear regression using the social density ρ as the dependent variable and the total number of violations v as the independent variable:

$$\rho = \beta_0 + \beta_1 v + \epsilon, \tag{5.6}$$

where $\boldsymbol{\beta} = [\beta_0, \beta_1]$ is the regression parameter vector and ϵ is the error term which is assumed to be normal. The regression model is fitted with the ordinary least squares method. Fitting this model requires training data. However, once the model is learned, data are not required anymore. After deployment, the surveillance system operates without recording data.

Once the model is fitted, we can obtain the predicted social density $\hat{\rho}|_{v=0}$ when there is no social distancing violation ($v = 0$). To further reduce the probability of social distancing violation occurrence, instead of using $\hat{\rho}|_{v=0}$, we propose to determine the critical social density as:

$$\rho_{\text{c}} = \rho_{\text{lb}}^{\text{pred}}, \tag{5.7}$$

where $\rho_{\text{lb}}^{\text{pred}}$ is the lower bound of the 95% prediction interval $(\rho_{\text{lb}}^{\text{pred}}, \rho_{\text{ub}}^{\text{pred}})$ at $v = 0$, as illustrated in Figure 5.2.

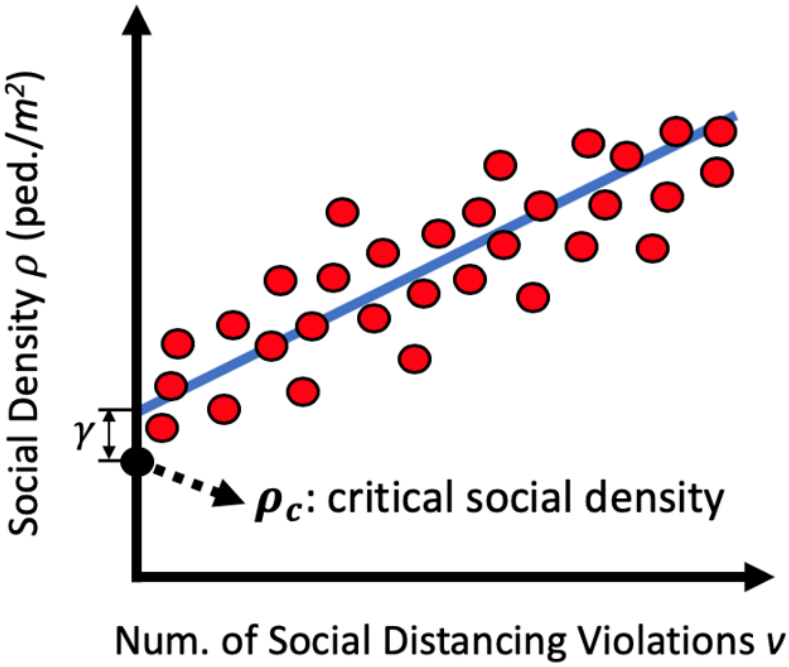


Figure 5.2: Obtaining the critical social density ρ_c. Keeping ρ under ρ_c will drive the number of social distancing violations v towards zero with the linear regression assumption.

If we keep the social density ρ of a scene to be smaller than the lower bound ρ_{lb} of the social density's 95% prediction interval at $v = 0$, the probability of social distancing violation occurrence can be pushed near zero. This is because under the linear regression assumption, the cumulative probability $P(\rho < \rho_{\text{lb}}^{\text{pred}}) = 0.05$, which is very small.

5.4.6 Broader Implementation

To further utilize the obtained critical social density ρ_c, subsequent measures must be taken to prevent the spread of COVID-19. There are two branches of post-processing mechanisms.

First, social distancing can be monitored and controlled online. Non-alarming audio-visual cues are sent to the people in the areas where the social density ρ is larger than the critical value ρ_c. In this way, people are immediately aware that they are violating the social distancing practice. The system can also send inflow modulation signals. Site managers can use these signals to keep the people density under ρ_c. This way, overcrowding is prevented.

Second, the critical density ρ_c, as well as the statistics, can be used for offline analysis. Analyzed offline data, such as averaged people densities of certain public areas or trends of people density of public events, can be utilized by regulators for better policymaking and large event organization.

Combining both the offline and online information provided by the proposed system, wider prevention measures can be taken as quickly as possible when necessary. The above procedures can be visualized in Figure 5.1.

5.5 Experiments

We conducted three case studies to evaluate the proposed method. Each case utilizes a different pedestrian crowd dataset. They are the Oxford Town Center Dataset (an urban street) [140], the

Mall Dataset (an indoor mall) [141], and the Train Station Dataset (New York City Grand Central Terminal) [142]. Table 5.2 shows detailed information about these datasets.

To validate the effectiveness of the proposed method in detecting social distancing violation, we conducted experiments over Oxford Town Center Dataset to determine the accuracy of the proposed method.

Table 5.2: Information of each pedestrian dataset.

	FPS	**Resolution**	**Duration**
Oxford Town Ctr.	25	1920×1080	5 mins
Mall	$\sim$1	640×480	33 mins
Train Station	25	720×480	33 mins

Implementation

The first step was finding the perspective transformation matrix $\mathbf{M}$ for the scene of each dataset. For the Oxford Town Center Dataset, we directly used the transformation matrix available on its official website. The other two datasets do not provide the transformation matrices, so we need to find them manually. We first identified the real distances among four key points in the scene and the corresponding coordinates of these points in the image. Then, these four points were used to identify the perspective transformation (homography) [107] so that the transformation matrix $\mathbf{M}$ can be calculated. For the Train Station Dataset, we found the floor plan of NYC Grand Central Terminal and measured the exact distances among the key points. For the Mall Dataset, we first estimated the size of a reference object in the image by comparing it with the width of detected pedestrians and then utilized the key points of the reference object.

The second step was applying the pedestrian detector on each dataset. The experiments were conducted on a regular PC with an Intel Core i7-4790 CPU, 32GB RAM, and an Nvidia GeForce GTX 1070Ti GPU running Ubuntu 16.04 LTS 64-bit operating system. Once the pedestrians were detected, their positions were converted from the image coordinates into the real-world coordinates.

The last step was conducting the social distancing measurement and finding the critical density ρ_c. Only the pedestrians within the ROI were considered. The statistics of the social density ρ, the inter-pedestrian distances $d_{i,j}$, and the number of violations v were recorded over time.

5.6 Results

5.6.1 Real-Time Pedestrian Detection

We experimented with two different deep-CNN-based object detectors: Faster R-CNN and YOLOv4. Figure 5.3 shows the qualitative results of pedestrian detection in the image using Faster R-CNN [120] and the corresponding social distancing in world coordinates. According to the qualitative results, there are a few missed detections. The reasons could be two-fold. First, occlusions can cause missed detections. This can be found in Mall Dataset, in which the shopping cart may affect the detection. Second, if the pedestrian size is too small, missed detections may also happen. This can be found in the Train Station Dataset. A limited number of missed detections do not affect the social distancing violation too much, as the first priority of the system is to detect whether there is any social distancing violation. For the number of violations and critical social density, as long as we can find a close enough estimation, it will satisfy our requirement.

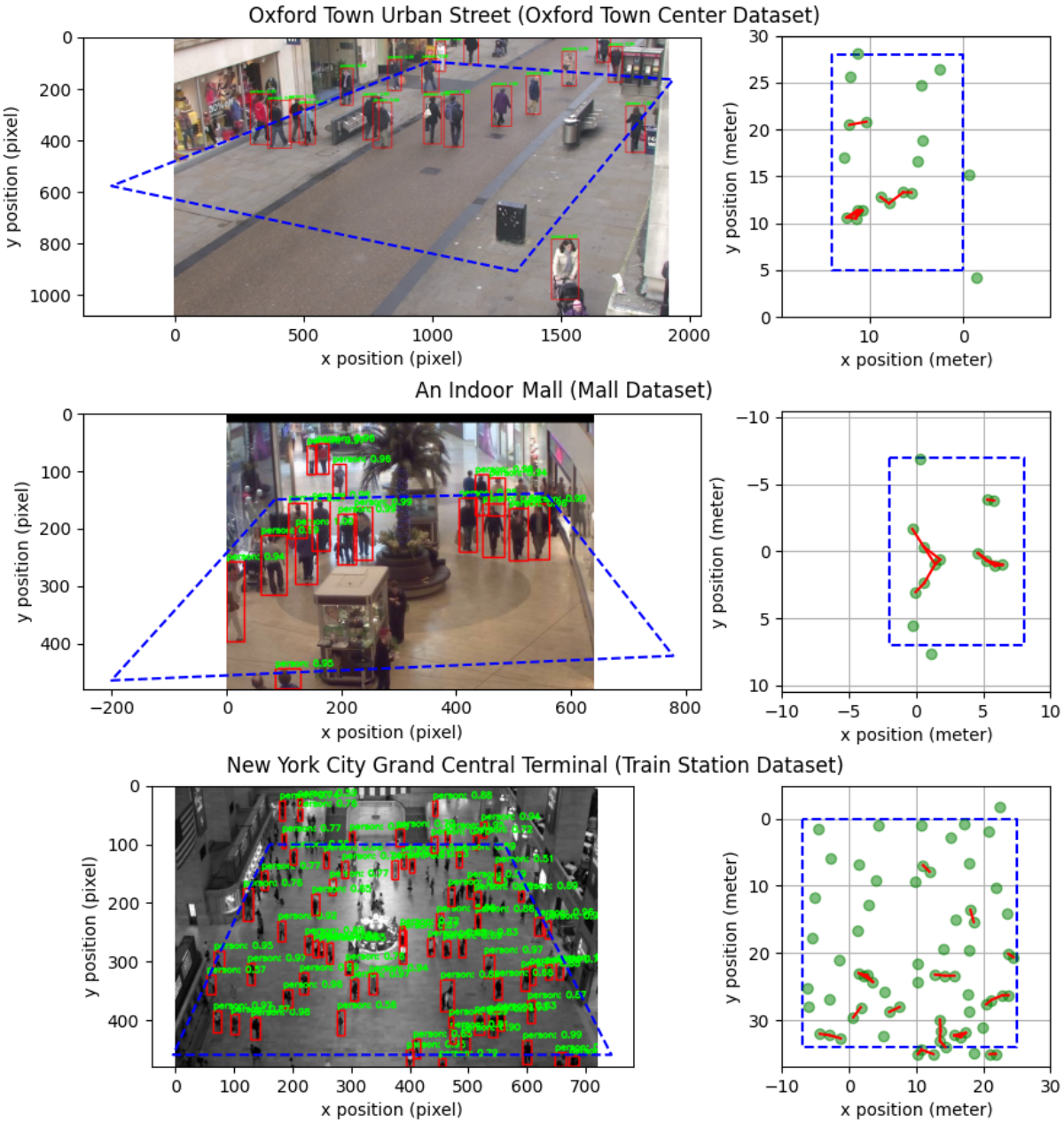


Figure 5.3: Illustration of pedestrian detection using Faster R-CNN and the corresponding social distancing.

The detector performances are given in Table 5.3. As can be seen in the Table, both detectors achieved an inference time of about $0.1s$ per frame. This is adequate to achieve real-time social distancing detection. For detection accuracy, we provide the results of MS COCO dataset from original works [120, 71].

Table 5.3: Real-time performance of pedestrian detectors.

Method	mAP (%)	Inference Time (s)
Faster R-CNN [120]	42.1–42.7	0.145/0.116/0.108
YOLOv4 [71]	41.2–43.5	0.048/0.050/0.050

The mAP indicates mean average precision. The inference time reports the mean inference time for Oxford Town Center/Train Station/Mall datasets, respectively.

5.6.2 Social Distancing Violation Detection

Figure 5.4 shows the change of pedestrian density ρ and the number of violations v as time evolves. The result indicates an obvious positive correlation between ρ and v. For example, at $t = 41$ s in the Oxford Town Center Dataset, $t = 84$ s in the Mall Dataset, and $t = 6$ s in the Train Station Dataset, when ρ is low, v is also relatively low. Figure 5.5 shows 2D histograms of this relationship. It further validates the observed positive correlation. This correlation leads to the subsequent proposed linear regression method to identify critical social density.

To validate our proposed methodology, we conducted the evaluation over the Oxford Town Center dataset, as it provides ground truth pedestrian detection. There are, in total, 4501 annotated frames. We split them into two parts, 2500 frames for training and 2001 frames for validation. The reason for splitting the dataset is to compare the proposed method with an end-to-end CNN model for social distancing violation detection, which was trained based on the training frames. All the evaluation results used the validation frames.

We first calculated the mean absolute error (MAE) of the average closest physical distance d_{avg} over all frames. d_{avg} is calculated by $d_{\text{avg}} = \frac{1}{n}\sum_{i=1}^{n} d_i^{\min}$, where $d_i^{\min} = \min(d_{i,j})$, $\forall j \neq i \in \{1, 2, \cdots n\}$ is the closest physical distance for a particular pedestrian i. We also calculated the MAE of the social distancing violation ratio $r_v = v/n$, where n is the number of pedestrians inside the ROI. The results were compared with a variant of our method which uses the center of the detected BB as the pedestrian position (BB-center method) instead of the middle point of the bottom edge, as described in Section 5.4.2 (BB-bottom method).

Table 5.4 reports the MAE of d_{avg} and the MAE of r_v. It quantifies the error in the detection of physical distance and social distancing violations. The proposed BB-bottom method has relatively low MAEs and is better than its variant BB-center method.

Table 5.4: Social distancing detection performance.

Method	MAE of d_{avg} (m)	MAE of r_v (Count)
BB-center	1.416	0.196
BB-bottom	**0.587**	**0.143**

Furthermore, the social distancing violation detection (the number of violations $v > 0$) was evaluated in terms of the precision, recall, and accuracy against the ground truth violation. In addition to the comparison with the variant method using the BB center, we also tried an end-to-end CNN model. Specifically, the CNN model inputs the image frame and outputs whether there is any social distancing violation or not. This new model employs ResNet50 as a backbone and has additional layers for social distancing violation detection. The loss is defined as weighted binary cross-entropy. The model was trained based on the first 2500 frames. The model performance was tested based on the remaining 2001 frames, which provides a fair comparison with the other two methods.

Table 5.5 shows the confusion matrix of social distancing violation detection using the BB-bottom method. In the Oxford Town Center Dataset, social distancing violation happens in the majority of the frames, so the number of true positives dominates the confusion matrix. Table 5.6 reports precision, recall, and accuracy of the violation detection and compares them over the methods of End-to-End CNN, BB-center, and BB-bottom. The result shows that the BB-bottom method performs better than the other two methods in all three metrics. The other two methods are not able to balance between the precision and recall metrics. End-to-end CNN has relatively high recall, but the precision is not good enough. BB-center has relatively high precision, but low recall. This further demonstrates the effectiveness of using pre-trained pedestrian detectors in the image domain and transforming the middle point of the BB bottom edge as the pedestrian position into world coordinates.

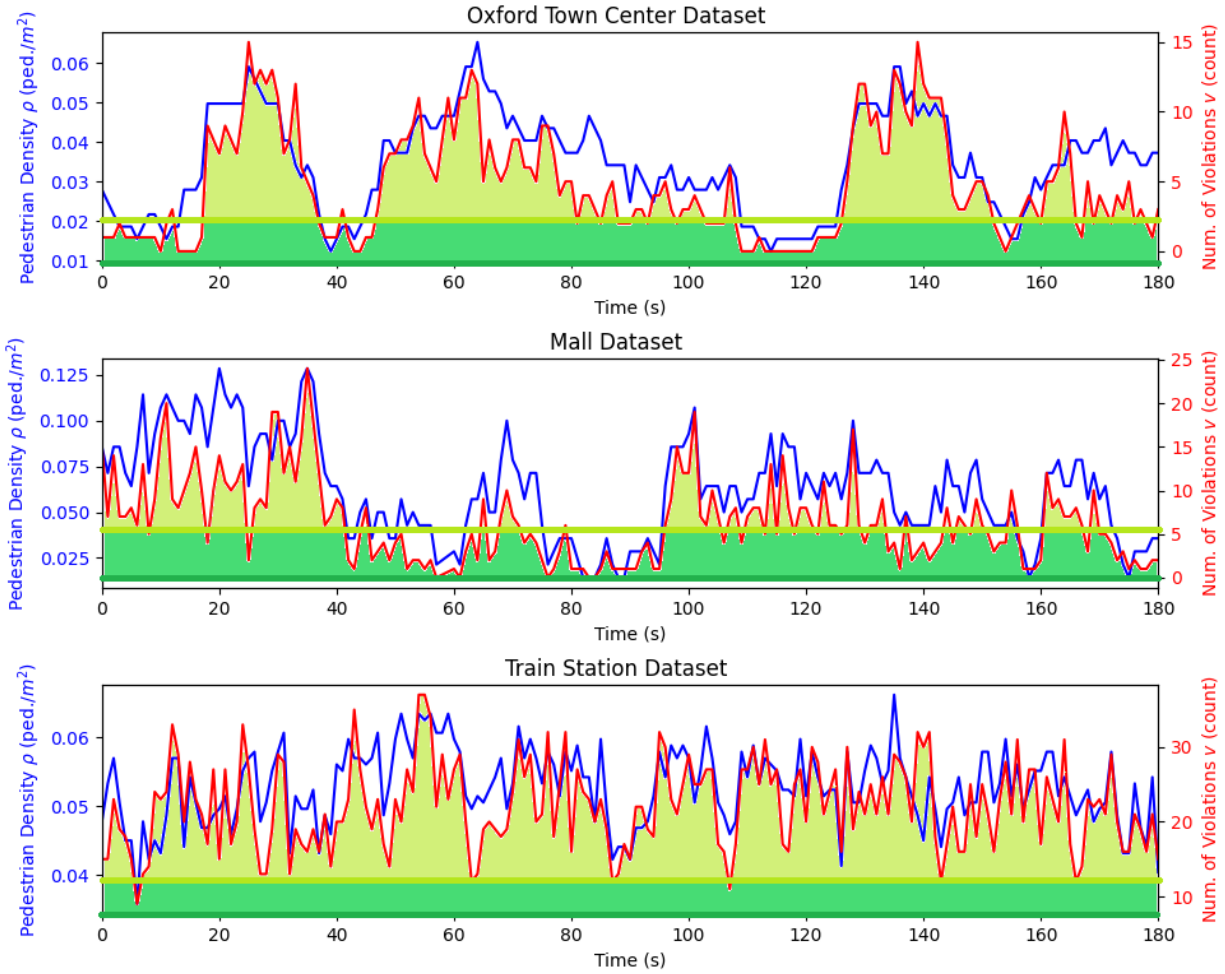


Figure 5.4: The change of pedestrian density ρ and the number of violations v over time. It shows an obvious positive correlation between ρ and v. The positive correlation is further illustrated in Figures 5.5 and 5.6, which show a linear relationship between ρ and v. The darker green horizontal line indicates the critical pedestrian density ρ_c and the lighter green line, the intercept density β_0. They are obtained by the proposed critical social density estimation methodology in Section 5.4.5. The shaded lighter green area shows that there will be more violations if the pedestrian density is above β_0. The shaded darker green area shows that more violations will be eliminated if the pedestrian density is further pushed below ρ_c, which is our critical social density.

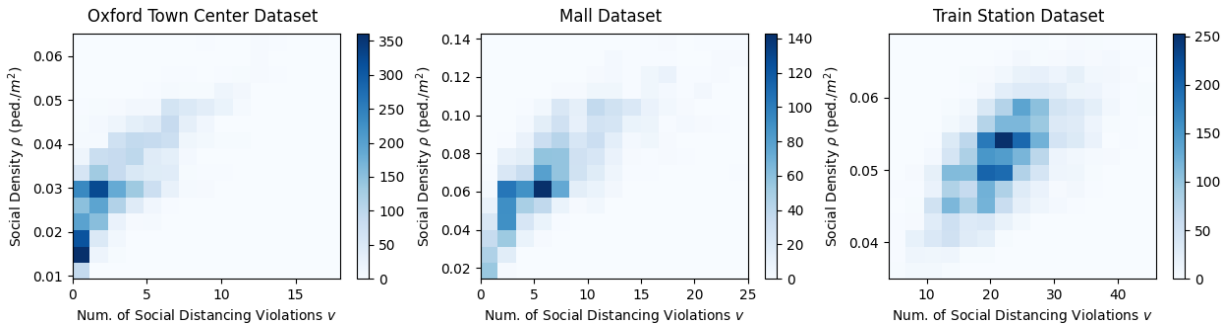


Figure 5.5: Two-dimensional histograms of the social density ρ versus the number of social distancing violations v. From the histograms we can see a linear relationship with positive correlation.

Table 5.5: Confusion matrix of social distancing violation detection.

		Ground Truth		**Total**
		Violation	**No Violation**	
Detected	Violation	1584	77	1661
	No Violation	67	273	340
	Total	1651	350	2001

The table reports the results of the BB-bottom method.

Table 5.6: Social distancing violation detection accuracy.

Method	**Precision (%)**	**Recall (%)**	**Accuracy (%)**
End-to-end CNN	83.27	94.37	79.71
BB-center	94.60	79.59	79.41
BB-bottom	**95.36**	**95.94**	**92.80**

5.6.3 Critical Social Density

To find the critical density ρ_c, we first investigated the relationship between the number of social distancing violations v and the social density ρ in 2D histograms, as shown in Figure 5.5. As can be seen in the Figure, v increases with an increase in ρ, which shows a linear relationship with a positive correlation. This indicates that the proposed linear regression can be used.

Then, we conducted the linear regression using the regression model of Equation (5.6), on the data points of v versus ρ. The skewness values of ρ for the Oxford Town Center Dataset, Mall Dataset, and Train Station Dataset are 0.32, 0.16, and -0.14, respectively, indicating the distributions of ρ are symmetric. This satisfies the normality assumption of the error term in linear regression. The regression result is displayed in Figure 5.6. The critical density ρ_c was identified as the lower bound of the prediction interval at $v = 0$. As can be seen from the Figure, for a social density value ρ that is smaller than the lower bound ρ_c, there are almost no data points. This means that according to the scene in the dataset, when $v = 0$, pedestrian density is hardly ever smaller than ρ_c. Since ρ_c is the lower bound of the 95% prediction interval, if we keep a social density $\rho < \rho_c$, we can push the probability of social distancing violation to almost zero.

Table 5.7 summarises the identified critical densities ρ_c, as well as the intercepts β_0 of the

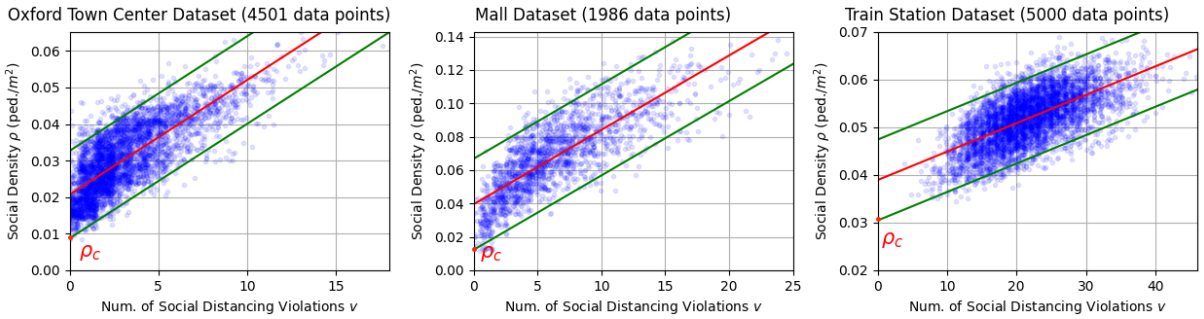


Figure 5.6: Linear regression (red line) of the social density ρ versus number of social distancing violations v data. Small random noise was added to each data point for better visualization. Green lines indicate the prediction intervals. The critical social densities ρ_c are the x-intercepts of the regression lines. Data points might overlap.

regression models. The obtained critical density values for all datasets are similar. They also follow the patterns of the data points as illustrated in Figure 5.6. This verified the effectiveness of our method.

To evaluate the effect of social distancing detection on determining the critical density, we also conducted the linear regression on the data of ground truth pedestrian positions in the Oxford Town Center Dataset. The obtained regression result over ground truth pedestrian positions are $\beta_{0,\mathrm{gt}} = 0.0217$ and $\rho_{c,\mathrm{gt}} = 0.0086$. The critical density ρ_c only has an error of 2%, which is very small. This further validated our proposed method of determining critical social density.

Table 5.7: Critical social density detection.

Dataset	**Intercept β_0**	**Critical Density ρ_c**
Oxford Town Ctr.	0.0207	0.0088
Mall	0.0396	0.0123
Train Station	0.0389	0.0305

The critical density was identified as the lower bound of the prediction interval at the number of social distancing violations $v = 0$.

5.7 Conclusions

This work proposed an AI- and monocular-camera-based real-time system to detect and monitor social distancing. In addition, our system utilized the proposed critical social density value to avoid overcrowding by modulating inflow to the ROI. The proposed approach was demonstrated using three different pedestrian crowd datasets. Quantitative validation was conducted over the Oxford Town Center Dataset that provides ground truth pedestrian detections.

There were some missed detections in the Mall Dataset and Train Station Dataset, as in some areas the pedestrian density is extremely high and occlusions occur. However, after our qualitative and quantitative analysis, most pedestrians were successfully captured and the missed detections have an minor effect on the proposed method. One future activity could be testing and verifying the proposed method over more datasets of various scenes.

Finally, in this work we did not consider that a group of people might belong to a single family or have some other connection that does not require social distancing. Understanding and addressing

this issue could be a futher direction of study. Nevertheless, one may argue that even individuals who have close relationships should still try to practice social distancing in public areas.

Our system is open-sourced. The implementation and the experiment data can be assessed via our GitHub repository:

`https://github.com/dongfang-steven-yang/social-distancing-monitoring`.

Chapter 6

Faraway-Frustum: Dealing with Lidar Sparsity for 3D Object Detection using Fusion

6.1 Introduction

3D/bird's-eye view (BEV) object detection is a critical task for many robotics applications. Existing lidar-based methods show good performance for close to medium range objects. However, a closer look at the state-of-the-art exposes an inherent problem: learned pointcloud representations do not generalize well with an increase in sparsity. This is not a surprising phenomenon. At a range greater than sixty meters, lidar pointcloud sparsity reaches a point where even humans cannot discern object shapes from each other. For example, in the KITTI 3D/BEV object detection benchmark [143], the state-of-the-art 3D object detection performance is remarkable. But when these high performing models face objects that are located at 60 meters and beyond, mean average precision drops to almost *zero*. We believe this is an important issue for automated driving. For instance, detecting faraway objects can offer more time for the automated vehicle to make better decisions.

3D/BEV object detection for faraway objects is challenging, and state-of-the-art (SOTA) lidar-based detectors [144, 145, 146, 147, 148] do not perform well for this task. We believe this is caused by sparsity and near-random scattering of the few points obtained from faraway objects. Learned representations from close to medium range objects do not generalize to faraway cases, and since SOTA approaches are primarily deep neural networks, they cannot learn the representations of faraway cases.

RGB-pointcloud fusion is a common strategy [149, 150, 151, 152, 153, 154] to increase 3D object detection performance. For example, some works [149], [154] focus on using 2D detection results to generate frustum-based search spaces in pointclouds. As shown in Fig. 6.1, a faraway object in the RGB image domain usually contains around 400 pixels, which can be easier to recognize with a mature 2D detector. As such, a fusion-based approach can be a good candidate for faraway object detection. However, even though the aforementioned studies use RGB imagery to boost detection performance, they still depend exclusively on learned pointcloud representations to localize objects in 3D.

In this work, we propose an alternative 3D/BEV detector, *Faraway-Frustum*, to address the problem of faraway object detection. We follow the idea of frustum generation but use clustering instead of a neural network to estimate an initial object location in the cropped pointcloud for

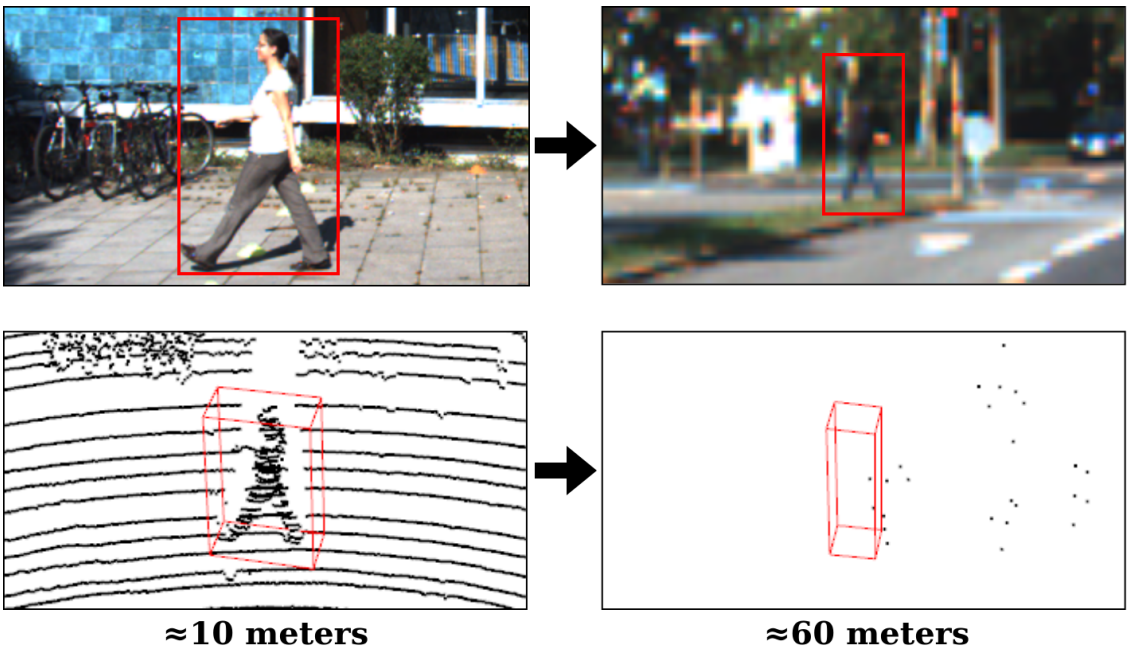


Figure 6.1: Learned pointcloud representations do not generalize well with an increase in sparsity. This problem does not translate to the 2D RGB image domain in the same fashion, as object shape does not change drastically with an increase in depth. However, sparse points in the target object's vicinity can still be used to estimate depth. Our method utilizes these sparse points to estimate depth while using 2D RGB information to recognize shape and object-class.

faraway objects. We still train a neural network to regress bounding box shape and refine depth. The overview of the proposed method is shown in Fig. 6.2. We first use 2D instance segmentation masks (or 2D bounding boxes) for each object in the RGB image space to generate frustums in the pointcloud space and find the corresponding lidar points for each object. Then, a pointcloud clustering technique is applied to estimate the 3D centroid of the object. By comparing the centroid distance with a faraway threshold, a decision is made to treat an object as either faraway or nearby. If faraway, a 3D bounding box is regressed by our Faraway Frustum Network (FF-Net) to the object based on the estimated centroid and the frustum pointcloud. Otherwise, instead of clustering the raw pointcloud, learned representations are directly used for 3D box fitting, following SOTA practices.

To evaluate the proposed method, we conducted bench-marking experiments using the KITTI dataset [143]. In KITTI, the average number of lidar points for each faraway object (e.g. pedestrians over 60 meters and cars over 75 meters) is ten or less, which supports our motivating assertion that an alternative approach is necessary for detecting faraway objects instead of directly using pointcloud-driven neural network approaches. The experimental results demonstrate that our method outperforms SOTA methods on faraway object detection, which indicates that our method effectively fuses the RGB data with a very sparse pointcloud. As shown in Fig. 6.6, our proposed method successfully detects faraway objects where SOTA methods (fusion or pointcloud only) fail. Our main contributions can be summarized as follows:

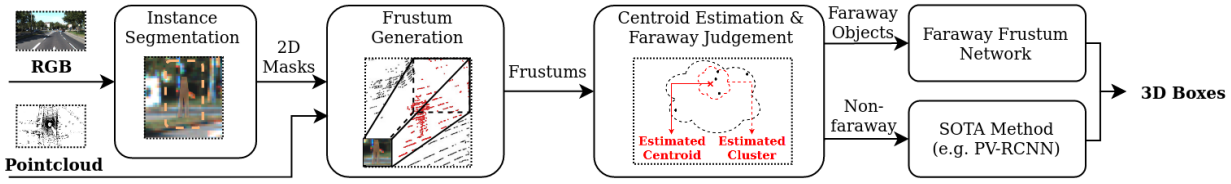


Figure 6.2: Overview of the 3D/BEV object detection system based on our proposed method (*Faraway-Frustum*). It contains three main stages: frustum generation, centroid estimation, and box regression. First, the 2D object information (classification and 2D semantic mask) is extracted from the image by conducting instance segmentation, and then the 3D frustum is shaped by extruding the 2D semantic mask to the 3D coordinate system. Second, lidar pointcloud (red) points in the frustum are collected and clustered, and then the 3D object centroid is estimated. Finally, depending on the faraway/nearby decision, the 3D bounding box is predicted by our Faraway Frustum Network or a state-of-the-art method.

6.1.1 Contributions

The main contributions of this work are:

- Introduction of a novel fusion strategy: depending solely on 2D vision sensing for object-class recognition and using frustum-cropped pointcloud data with clustering for 3D object localization.
- Showing that using clustering with cropped, very sparse raw pointcloud data is a better strategy than using learned representations for faraway 3D object detection. As shown in Fig. 6.1, within very sparse pointclouds, the shape of objects changes drastically and randomly. As such, using representations learned mostly from closer objects is not effective.
- Demonstrating the failure of state-of-the-art 3D object detectors with objects at a distance over sixty meters in the KITTI dataset. The proposed faraway-frustum approach outperforms SOTA methods with a significant margin.

6.2 Related Work

In this section, we briefly review state-of-the-art 3D/BEV object detection methods. We divide them into two main categories: pointcloud only methods and RGB-pointcloud fusion methods. In the second category, we mainly discuss feature-based fusion and frustum-based fusion. We also discuss their performance in detecting faraway objects.

Pointcloud only methods. One way of processing pointcloud data is based on voxels [145, 146, 144, 155]. Such methods first convert the pointcloud into voxel grids and then learn the representation of each voxel. 3D/BEV detection is achieved with the learned voxel representation. Alternatively, raw pointcloud data can be used for 3D/BEV object detection [156, 157, 148] by directly utilizing PointNet-based architectures [158]. These methods are robust for most objects. However, pointcloud only methods all have difficulty detecting faraway objects, because the lidar points of faraway objects are too sparse to be voxelized and learned, leading to no detection result in most cases.

Feature-based Fusion. Feature-based fusion methods try to make the pointcloud data and the RGB data complement each other. One way is to fuse the information from the pointcloud

into the RGB image. For example, [150] fuses the features from a Region of Interest (RoI) in both 2D image and 2D depth map, and then conducts 3D box regression. MV3D [151] projects the lidar pointcloud to two image representations (bird's-eye view and front view). The features and information extracted from these two image representations and the RGB image are then fed into a region-based fusion network for 3D object detection. AVOD [152] first generates a BEV map from a voxel grid representation of the lidar pointcloud. The features extracted from both the BEV map and the RGB image are fused for 3D object detection through a first-stage region proposal network and a second-stage detector network. The main problem of these methods is the loss of the 3D geometric information in the lidar pointcloud caused by using only the pointcloud's 2D representations, leading to some errors in locating small objects such as pedestrians. Feature-based fusion can also be achieved by fusing the information from the image space into the pointcloud. For example, [153] extracts the geometric features in 3D and color features in 2D from RGB-D images and then fuses them for 3D object detection. MVX-Net [159] fuses the RGB image and pointcloud point-wise or voxel-wise. The features extracted from the RGB image by a pre-trained 2D CNN are fused with the pointcloud in a voxel-based network to do 3D object detection. PointPainting [160] assigns the semantic feature to each lidar point by fusing the 2D detection result from the RGB image, thus achieving better results in pointcloud-based neural network detector. Since these approaches heavily rely on the pointcloud features, they still can not generate good results for faraway objects with sparse lidar points.

Frustum-based Fusion. Frustum-based fusion methods use the detection results from 2D image to generate frustums for the pointcloud, hence reducing the search space in 3D. An early and classic method is Frustum PointNets [149]. This method first generates a frustum for each object detected in 2D, then applies a PointNet-based approach to do instance segmentation and 3D box estimation in each frustum. Some work [161, 162] have improved the process of frustum generation by filtering out some background noise, and there is work focusing on changing the content of frustums. For example, Frustum ConvNet [154] generates a sequence of sub-frustums via sliding in the original 3D frustum. Frustum Voxnet [163] voxelizes parts of the frustum instead of using the whole frustum space, which offers more accurate representations around the area of interest. Some other researchers aimed to provide more fusion information for improving the accuracy of 3D detection. For example, one work [164] combines the pointcloud features in the frustum with the image features in the 2D bounding box as early-fusion and then applies a PointNet-based detector. Another work [165] fuses their own BEV detection results with 3D/BEV results from Frustum PointNets as late-fusion. These perform well for most objects. But unfortunately, for faraway objects with sparse lidar points, pure neural network based approaches cannot generalize well.

One recent work [166] achieved good 3D/BEV pedestrian detection results around 30 meters. In our work, we extend the range significantly, detecting pedestrians at 60 meters and beyond, where most SOTA approaches completely fail.

6.3 Proposed Method

An overview of our proposed method is shown in Fig. 6.2 and Algorithm 2. Our method takes both the RGB image and lidar pointcloud as input and outputs 3D/BEV bounding boxes $\mathbf{B}_i$ with class id c_i. There are three main stages in our method: frustum generation, centroid estimation, and depth-refinement with box regression. Each stage will be illustrated in detail in the following subsections.

Algorithm 2: Faraway-Frustum($\mathbf{P}$, $\mathbf{I}$, $\mathbf{T}$, z_{th})

Input:
Lidar pointcloud $\mathbf{P} \in \mathbb{R}^{N,3}$.
RGB image $\mathbf{I} \in \mathbb{R}^{H,W,3}$.
Calibration matrix $\mathbf{T} \in \mathbb{R}^{4,4}$.
Faraway object threshold $z_{\text{th}} \in \mathbb{R}$
Output:
3D object bounding box $\mathbf{B}_i \in \mathbb{R}^7$.
Class id c_i.
Main algorithm:
$\{c_i, \mathbf{b}_i, \mathbf{M}_i, s_i\} = f_{\text{Mask R-CNN}}(\mathbf{I})$ $(i = 1, 2...n)$;
foreach $i(1, 2...n)$ **do**
  $\mathbf{P}'_i = f_{\text{Mask-frustum}}(\mathbf{M}_i, \mathbf{T}, \mathbf{P})$;
  $(x_i, y_i, z_i) = f_{\text{clustering}}(\mathbf{P}'_i)$;
  if $z_i \geq z_{th}$ **then**
    $\mathbf{P}''_i = f_{\text{projection}}(\mathbf{P}'_i, x_i, y_i, z_i)$;
    $(z'_i, w_i, l_i, h_i, \alpha_i) = f_{\text{FF-Net}}(\mathbf{P}''_i, c_i)$;
    $\mathbf{B}_i = (x_i, y_i, z'_i, \alpha_i, w_i, l_i, h_i)$;
  else
    $\mathbf{B}_i, c_i = f_{\text{SOTA}}(\mathbf{P}, \mathbf{I})$;
  end
end

6.3.1 Frustum Generation

2D instance segmentation. 2D instance segmentation serves as the basis of frustum generation. It takes an image as input and outputs the 2D object detection results containing 2D bounding boxes and semantic masks.

In this work, we use the 2D instance segmentation framework Mask R-CNN [167] to obtain 2D object information $\{R_i\}$ from image $\mathbf{I}$:

$$\{R_i\} = f_{\text{Mask R-CNN}}(\mathbf{I}) \tag{6.1}$$

where $f_{\text{Mask R-CNN}}$ represents the Mask R-CNN framework. $R_i = (c_i, \mathbf{b}_i, \mathbf{M}_i, s_i)$ is the instance segmentation result, which is a 4-tuple consisting of class label c_i, 2D bounding box $\mathbf{b}_i$, 2D semantic mask $\mathbf{M}_i$, and confidence score s_i for object i.

Frustum generation. We use the set of 2D results $\{R_i\}$ to generate frustums and to further identify the lidar points that correspond to each object i. With the known transformation $\mathbf{T}$ between the camera and the lidar, we use the semantic mask $\mathbf{M}_i$ for 2D-to-3D projection as shown in Fig. 6.3(a). Then the corresponding "frustrum pointcloud" $\mathbf{P}'_i$ can be identified from the raw lidar pointcloud $\mathbf{P}$ based on the frustum:

$$\mathbf{P}'_i = f_{\text{Mask-frustum}}(\mathbf{M}_i, \mathbf{T}, \mathbf{P}) \tag{6.2}$$

The mask-frustum based projection $f_{\text{Mask-frustum}}$ is our main approach. As shown in Fig. 6.3(b), we believe that using the semantic mask can exclude some noise points that do not belong to the target object, e.g., the points from occluded objects or the background. As an alternative, we also tested the box-frustum based projection $\mathbf{P}'_i = f_{\text{Box-frustum}}(\mathbf{b}_i, \mathbf{T}, \mathbf{P})$, which is used as a comparison with our main approach.

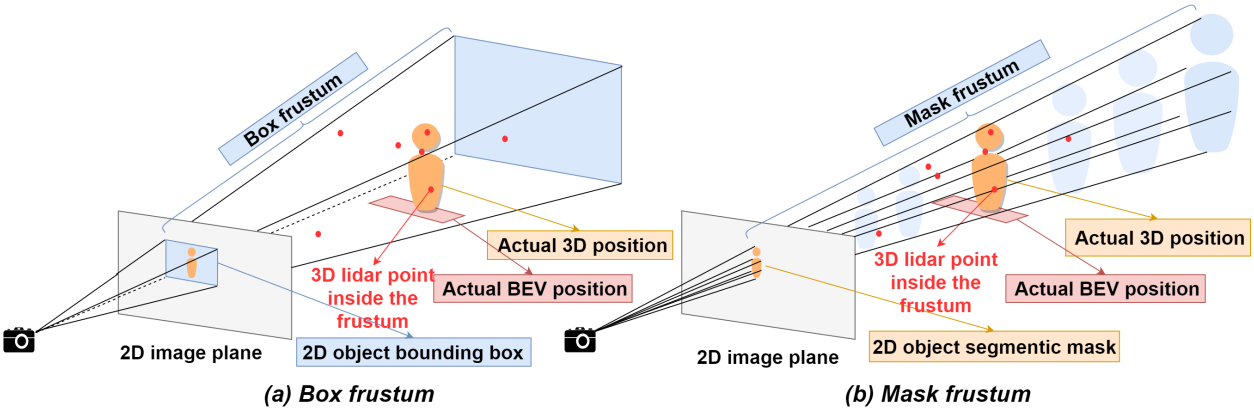


Figure 6.3: An illustration of frustum generation. The main difference between box frustum and mask frustum is that box frustum uses the 2D bounding box as the projection source, while mask frustum uses the 2D semantic mask. Mask frustum gives a more compact search space alongside the outline of the object, and thus excludes some noise points caused by potential occlusions.

6.3.2 Centroid Estimation

Centroid estimation. With $\mathbf{P}'_i$ obtained from the frustum, we then estimate the 3D centroid (x_i, y_i, z_i) for object i:

$$(x_i, y_i, z_i) = f_{\text{clustering}}(\mathbf{P}'_i) \tag{6.3}$$

The 3D object centroid plays two key roles in our method. One is to use the depth z_i to determine whether object i should be treated as a faraway object. The other is to further generate the 3D/BEV detection results for faraway objects.

Based on our observation in the KITTI dataset, regardless of whether the pointcloud in the frustum is dense or sparse, there are always some points on the object's surface. Thus, we adopt a fast clustering technique using histograms to estimate the 3D object centroid.

First, for all points in the pointcloud $\mathbf{P}'_i$, the histogram of all the coordinate values in each axis is generated (here we have 3 axes x, y, and z). For the histogram of each axis, we define the edges of every bin in the histogram as (e^l_j, e^r_j), $\forall j \in \{0, 1, \cdots, N\}$, and the count of values belonging to each bin as n_j, $\forall j \in \{0, 1, \cdots, N\}$, where N is the number of bins. Then, we identify the bin with the largest count value. This indicates that most of the points are concentrated within this bin. The corresponding index will be obtained by $j^* = \arg\max_j (n_j)$. Finally, the centroid value of an axis, for example, the centroid of x-axis, x_i, can be obtained by $x_i = \frac{1}{2}(e^l_{j^*} + e^r_{j^*})$. The centroid values y_i and z_i for the other two axes is estimated in the same way.

Faraway threshold. Using the estimated centroid (x_i, y_i, z_i), we determine whether each object i is considered faraway or nearby. We select different faraway thresholds z_{th} for different object classes based on the statistics of the number of ground truth lidar points in each object. As shown in Fig. 6.5, we first draw a line for the objects that have 10 lidar points, then we approximately select the z_{th} such that most objects of distance larger than z_{th} have less than 10 points. To determine whether object i should be treated as a faraway object, we compare the estimated distance z_i with z_{th} of the corresponding object class c_i. If $z_i > z_{\text{th}}$, then it is a faraway object, otherwise it is not.

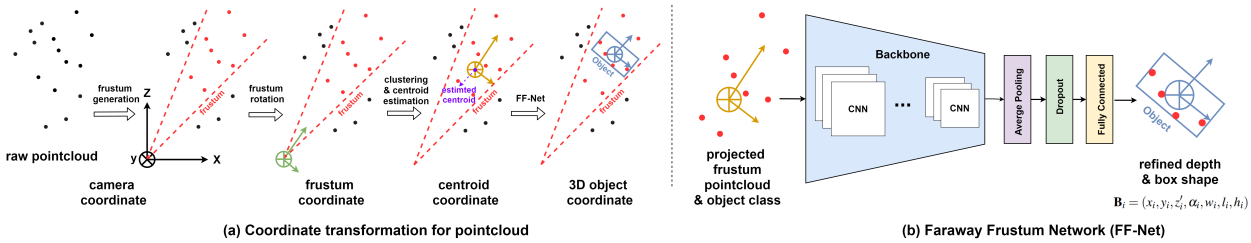


Figure 6.4: An illustration of coordinate transformation for pointcloud and Faraway Frustum Network (FF-Net). (a) Illustrates the process of projecting the pointcloud into different coordinate systems in our method. After carrying out frustum generation, frustum rotation, clustering, and centroid estimation, the frustum pointcloud is projected into the centroid coordinate system. Our goal is to further localize the 3D object by the 2D projection of the frustum pointcloud and the FF-Net. (b) The FF-Net is essential to refine the object center, regress the box size, and resolve certain issues that may occur while creating the frustum. For example, due to errors in detection or segmentation, the cluster centroid may not be aligned well with the object. The FF-Net is trained to deal with such issues and refine the object localization.

6.3.3 Box Regression

To obtain the 3D/BEV bounding box $\mathbf{B}_i$ for object i based on the estimated object centroid (x_i, y_i, z_i), we need to estimate the box shape: the length l_i, width w_i, height h_i, and orientation α_i. If object i is a faraway object, directly using learned representations from state-of-the-art models is not a good choice because they do not generalize well from dense pointclouds to very sparse pointclouds. Furthermore, the estimated object centroid (especially the depth) may still be quite far from the box center. As such, we propose to use a light model named Faraway Frustum Network (FF-Net) for faraway objects to refine the depth z_i' and regress the shape $(w_i, l_i, h_i, \alpha_i)$ of the 3D bounding box with the input of the object class c_i and a 2D projection $\mathbf{P}_i''$ of the frustum pointcloud, as shown in Fig. 6.4.

Pointcloud Projection. 2D projection $\mathbf{P}_i''$ of the frustum pointcloud is generated using a coordinate transformation as shown in Fig. 6.4(a). After conducting frustum generation for the raw pointcloud $\mathbf{P}$, the frustum pointcloud $\mathbf{P}_i'$ is obtained. First, the camera coordinate system is rotated to the center view of the frustum to build the frustum coordinate system. Second, after histogram-based clustering and centroid estimation, the frustum coordinate system is transformed to the centroid coordinate system with the estimated centroid at the origin. Finally, all lidar points inside of the frustum are projected into the centroid coordinate system in bird's-eye view. The projected frustum pointcloud in the centroid coordinate system is taken as the 2D projection $\mathbf{P}_i''$ of the frustum pointcloud.

Box Regression. FF-Net takes the object class c_i and a 2D projection

$$\mathbf{P}_i'' = f_{\text{projection}}(\mathbf{P}_i', x_i, y_i, z_i) \tag{6.4}$$

of the frustum pointcloud $\mathbf{P}_i'$ whose origin is the estimated centroid (x_i, y_i, z_i) as input and combines a MobileNet-based [168] backbone network with a multi-output regression head as shown in Fig. 6.4(b).

The estimated centroid (x_i, y_i, z_i) is considered as the origin of the input projection, and the goal of the FF-net is to shift this origin to the real center of the 3D bounding box, i.e. to transform a centroid coordinate to a 3D object coordinate as shown in Fig. 6.4(a). Furthermore, another

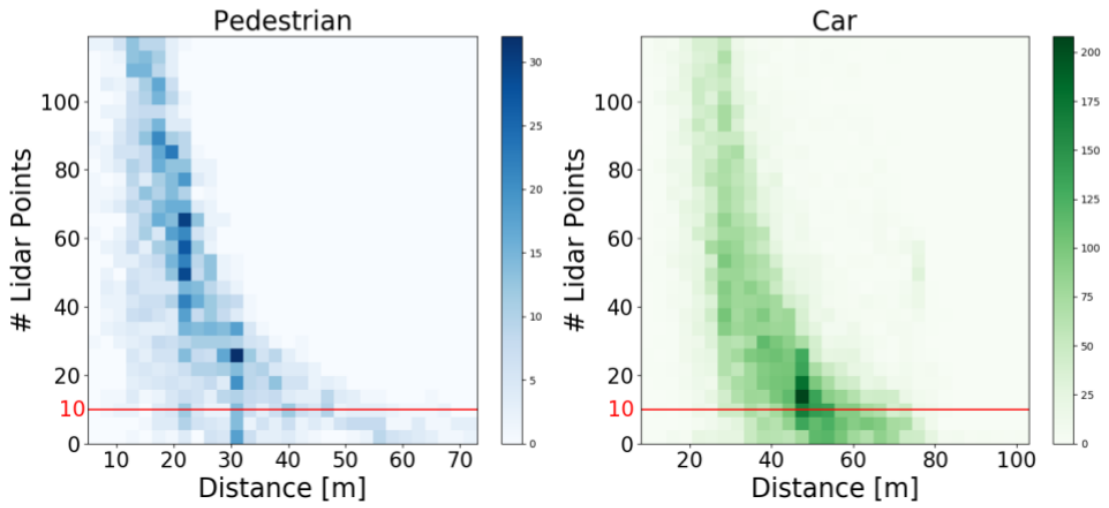


Figure 6.5: The number of points belonging to an object (pedestrians and cars) versus distance from the sensor in the KITTI dataset. As the distance (x-axis) increases, the number of lidar points in an object (y-axis) decreases drastically and the pointcloud of each faraway object is very sparse. When the number of lidar points is less than 10, the shape of objects cannot be recognized. Thus, objects with fewer than 10 points are considered faraway objects. We use this distribution to decide the faraway decision threshold.

goal is to regress the box shape, achieved by minimizing the loss of the regressed length, width and height of the box. We use mean absolute error (MAE) to compute the loss L_x,L_y,L_z,L_w,L_l,L_h,L_α of box centroid (x'_i, y'_i, z'_i) and shape $(w_i, l_i, h_i, \alpha_i)$ respectively. By summing these losses, FF-Net is trained and optimized with multi-task losses L_{FF-Net}.

Finally, for a faraway object, we take the shifted depth z'_i and the regressed 3D bounding box shape $(w_i, l_i, h_i, \alpha_i)$ from the output of FF-Net and combine them with (x_i, y_i) from the estimated centroid. We assign a 3D bounding box to the faraway object i as:

$$\mathbf{B}_i = (x_i, y_i, z'_i, \alpha_i, w_i, l_i, h_i). \tag{6.5}$$

It should be noted that the class id c_i is directly obtained with Mask R-CNN. If object i is not a faraway object, we switch to using learned representations following SOTA (e.g. Frustum-PointNets [149], PV-RCNN [144]) methods. In this case, the 3D bounding box and class id for a non-faraway object are obtained by $\mathbf{B}_i, c_i = f_{\text{SOTA}}(\mathbf{P}, \mathbf{I})$.

6.4 Experiments

We utilized the KITTI dataset [143] to conduct our experiments. We specifically extracted far-away objects in KITTI and investigated them separately. Details of dataset preparation, evaluation metrics, and implementation are described below.

Dataset preparation. First we analyzed the statistics of the original KITTI dataset by evaluating the distribution of the objects at different distances and having different numbers of lidar points, as shown in Fig. 6.5. It is obvious that as the distance increases, the number of lidar points in an object decreases. That is to say, the pointcloud is very sparse for faraway objects. We selected the faraway threshold z_{th} for cars as 75 meters and for pedestrians as 60 meters. We also split the KITTI dataset into the training set (3724 frames) and the validation set (3757 frames).

Evaluation metric. The major evaluation metric is the mean average precision (mAP) with a given IoU threshold, as suggested by the KITTI dataset. We use both the official benchmark and our specific benchmark for faraway objects. In the benchmark for faraway objects, we only evaluate faraway objects and we use a specific IoU threshold (0.1) for the mAP. We use a low IoU threshold because detecting an object with a small overlap with the ground truth is still better, at long distances, than no detection at all. The mAP results are computed with 11 recall positions which is the same as in [143].

We also evaluate the faraway objects using the average IoU ($aIoU$), which is defined as

$$aIoU = \frac{\sum_{i=1}^{n} \text{IoU}}{n} \tag{6.6}$$

where n is the total number of faraway objects and $\sum_{i=1}^{n} \text{IoU}$ is the sum of the IoU values calculated based on the ground truth and the predicted bounding box.

Implementation. Our method uses the instance-level semantic segmentation method (Mask R-CNN [167] pre-trained on the COCO dataset [169]) to generate 2D object information from the image space. Our first approach (ours1) uses 2D semantic masks to generate frustums in 3D. The second approach (ours2) uses 2D bounding boxes to generate frustums. As a baseline (ours3), we also use ground truth 2D bounding boxes provided by KITTI to generate frustums. All of our approaches are combined with PV-RCNN [144] for non-faraway objects. The Faraway Frustum Network (FF-Net) is trained using the Adam optimizer with early stopping. FF-Net is trained with the whole training set, but during inference only faraway objects are fed to the FF-Net.

We compared our proposed method with the following SOTA 3D/BEV object detectors: SECOND [145], PointPillars [146], PV-RCNN [144], and Frustum PointNets [149]. These SOTA methods are all trained from scratch using our data split, and we evaluated them with the same faraway metrics.

6.5 Results

Quantitative results. Table 6.1 shows the average IoU results for BEV detection of faraway objects in the KITTI validation dataset. All of our methods outperform SOTA methods with higher average IoU of at least 0.051 and at most 0.157. And surprisingly, none of the methods except ours can achieve an average of 0.1 IoU for both pedestrians and cars. This result not only demonstrates the effectiveness of our method, but also underlines an important shortcoming of SOTA methods. Furthermore, we believe finding the exact shape of faraway objects is not a priority. As long as we obtain the 3D/BEV detection result with even a small IoU (e.g. 0.1), it can still be very useful for certain applications such as automated driving. In other words, a 0.1 IoU detection is better than a false negative. As such, we set the IoU threshold to 0.1 for faraway objects in the mAP comparison.

The mAP results of faraway 3D/BEV detection over KITTI validation dataset for pedestrians and cars are shown in Table 6.2. For faraway pedestrians (over 60 meters), our methods (ours1 and ours2) outperform SOTA methods on 3D/BEV detection with large mAP margins (BEV: at least 22.14% and at most 45.45%, 3D: at least 9.55% and at most 44.54%). For faraway cars (over 75 meters), our methods (ours1 and ours2) outperform SOTA methods again with a higher mAP (BEV: at least 27.09% and at most 46.90%, 3D: at least 16.52% and at most 46.90%).

The mAP results of 3D/BEV detection over the KITTI validation dataset for pedestrians and cars are shown in Table 6.3. For the non-faraway official Easy/Mod/Hard benchmark, our method performs as well as the baseline SOTA method (PV-RCNN).

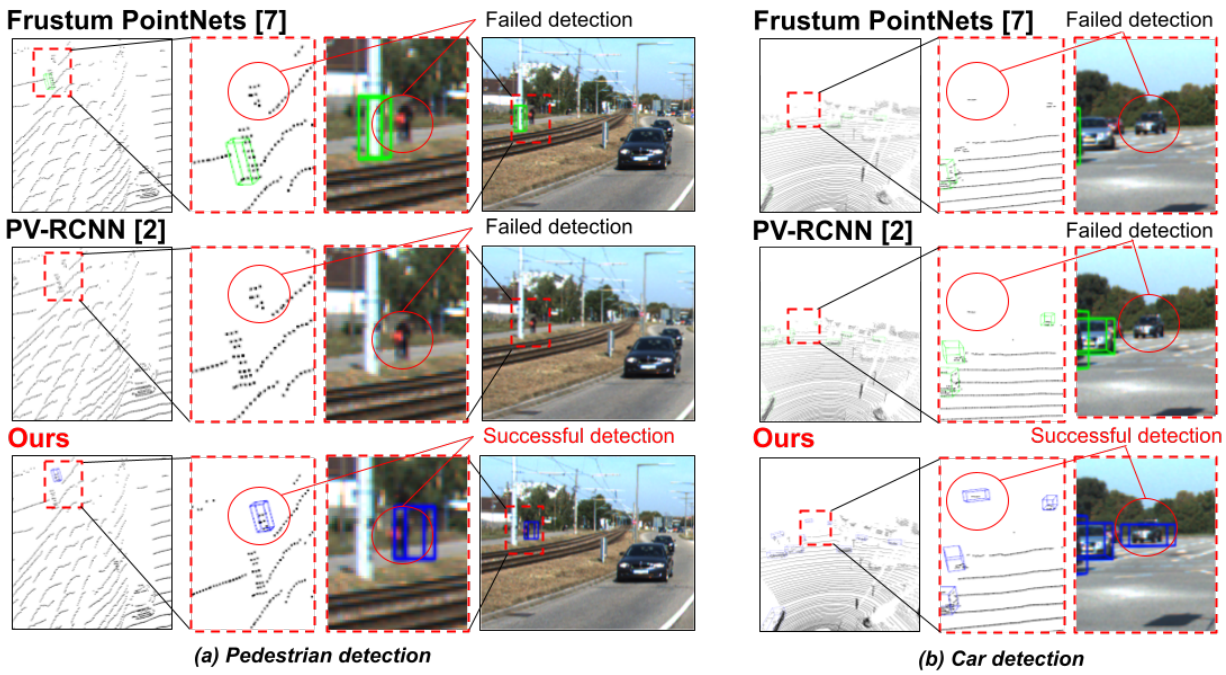


Figure 6.6: Example 3D detection results of faraway objects from the KITTI test set. (a) Pedestrian detection. *Top row*: Frustum PointNets, which is based on fusing multiple modalities (RGB and pointcloud). *Middle row*: PV-RCNN, which uses only the pointcloud. *Bottom row*: Our proposed method. (b) Car detection. Same arrangement as in (a). In these examples, for both the faraway pedestrian and the faraway car, our proposed method successfully detects the targets. However, state-of-the-art methods (Frustum PointNets and PV-RCNN all fail.

All the above results demonstrate that our method achieves better performance on faraway object detection without impairing the overall performance of SOTA methods.

Qualitative results. Fig. 6.6 shows a visual example of the results of different methods for faraway object detection. We compared our method with PV-RCNN [144] and Frustum PointNets [149]. In frame (a), Frustum PointNets mistakenly detects the pole as a pedestrian, while PV-RCNN provides no detection. However, for detecting the faraway pedestrian near the pole, only our detector succeeds. In frame (b), state-of-the-art methods all fail in detecting the faraway car. In contrast, our method successfully detects the car in 3D.

6.6 Conclusion

In this chapter, we proposed an alternative 3D/BEV detector, named *Faraway-Frustum*, to deal with lidar sparsity of faraway objects. Our method takes advantage of relatively dense image data to find faraway objects and circumvents the disadvantages of pointcloud-driven neural networks working on very sparse points. Moreover, our alternative detector can be flexibly combined with a state-of-the-art method to form an overall 3D/BEV object detection system via setting faraway thresholds.

The experiments demonstrated the feasibility of our approach, but they also exposed a significant shortcoming of state-of-the-art object detection methods: Relying on learned representations of very sparse lidar points to detect faraway objects is not a good strategy.

Table 6.1: Average IoU Comparison of Faraway BEV Object Detection on KITTI Val Dataset

Method	BEV Ped.	BEV Car
	> 60 m	> 75 m
Frustum PointNets [149]	0.000	0.000
SECOND [145]	0.036	0.009
PointPillars [146]	0.072	0.000
PV-RCNN [144]	0.051	0.018
Ours1 (FF-net mask)	0.123	**0.157**
Ours2 (FF-net box)	**0.124**	0.150

* Name explanation: Ped. (Pedestrian).
** The **bold** result indicate the best in all methods, and the blue result represents the second place.

Table 6.2: mAP Comparison of Faraway 3D/BEV Object Detection on KITTI Val Dataset

Method	3D Ped.	BEV Ped.	3D Car	BEV Car
	IoU threshold 0.1			
	Over 60 meters		Over 75 meters	
FP [149]	00.00	00.00	00.00	00.00
SE [145]	13.63	13.63	09.09	09.09
PP [146]	22.40	22.40	00.00	00.00
PV [144]	19.69	19.69	18.18	18.18
Ours1	**44.54**	44.54	34.70	45.27
Ours2	31.95	**45.45**	**46.90**	**46.90**

* Name explanation: FP (Frustum PointNets), SE (SECOND), PP (PointPillars), PV (PV-RCNN), Ours1 (FF-net mask), Ours2 (FF-net box), Ped. (Pedestrian).
** The **bold** result indicates the best in all methods, and the blue result represents the second place. We set the experimental IoU threshold as 0.1 for faraway pedestrians because in the current stage, it is extremely difficult to precisely locate faraway objects, while detecting faraway objects even with low IoU is still practical and useful.

Table 6.3: mAP Comparison of 3D/BEV Object Detection on KITTI Val Dataset

Method	3D/BEV Pedestrian			3D/BEV Car		
	IoU threshold 0.5			IoU threshold 0.7		
	Easy	Mod	Hard	Easy	Mod	Hard
PV-RCNN [144]	69.53/73.32	66.02/67.42	62.91/65.70	96.73/97.53	93.18/94.77	85.76/94.78
Ours1 (FF-net mask + PV-RCNN)	71.65/73.02	67.29/68.73	62.08/66.87	96.73/97.54	93.17/94.75	85.76/94.77
Ours2 (FF-net box + PV-RCNN)	71.74/73.11	67.29/68.73	62.08/66.88	96.73/97.54	93.17/94.75	85.76/94.77

For the non-faraway official Easy/Mod/Hard benchmark, our method performs as well as the baseline SOTA method (PV-RCNN). This result shows that the proposed method can be used to improve the faraway detection performance without sacrificing the non-faraway detection performance.

Chapter 7

Predicting Pedestrian Crossing Intention With Feature Fusion and Spatio-Temporal Attention

7.1 Introduction

Autonomous driving technology has progressed significantly in the past few years. However, to develop vehicle intelligence that is comparable to human drivers, understanding and predicting the behaviors of traffic agents is indispensable. This work aims to develop behavior understanding algorithms for vulnerable road users. Specifically, a vision-based pedestrian crossing intention prediction algorithm is proposed.

Behavior understanding plays an crucial role in autonomous driving. It establishes the trust between people and autonomous driving systems. By explicitly showing passengers how the system makes its decisions, people will be more willing to accept this technology.

In level 4 autonomy's driving, pedestrian crossing behavior is one of the most important behaviors that needs to be studied urgently. In urban scenarios, vehicles frequently interact with crossing pedestrians. If the autonomous system fails to handle vehicle-pedestrian interactions appropriately, casualties will most likely occur. With accurate intention prediction, the decision-making and planning modules in autonomous driving systems can access additional meaningful information, hence generating safer and more efficient maneuvers.

Nowadays, visual sensors such as front-facing cameras are becoming the standard configuration for autonomous driving systems. In the tasks of object detection and tracking, both the software and hardware of vision components are mature and ready for mass production. This provides a perfect platform on which vision-based behavior prediction algorithms can be deployed. Researchers and engineers in the prediction field can just focus on algorithm design. When the algorithm is ready, deployment becomes relatively trivial. The proposed algorithm is based on pure vision, it can be easily deployed. As long as the prediction algorithm is appropriately tested and verified, mass deployment becomes straightforward.

Vision-based pedestrian crossing intention prediction has been explored for several years. Early works [170] usually utilized a single frame as input to a convolutional neural network (CNN) based prediction system. This approach ignores the temporal aspect of image frames, which plays a critical role in the intention prediction task. Later on, with the maturity of recurrent neural networks (RNNs), pedestrian crossing intention was predicted by considering both the spatial and temporal information [171, 172, 173]. This led to different ways of fusing different features, e.g., the detected

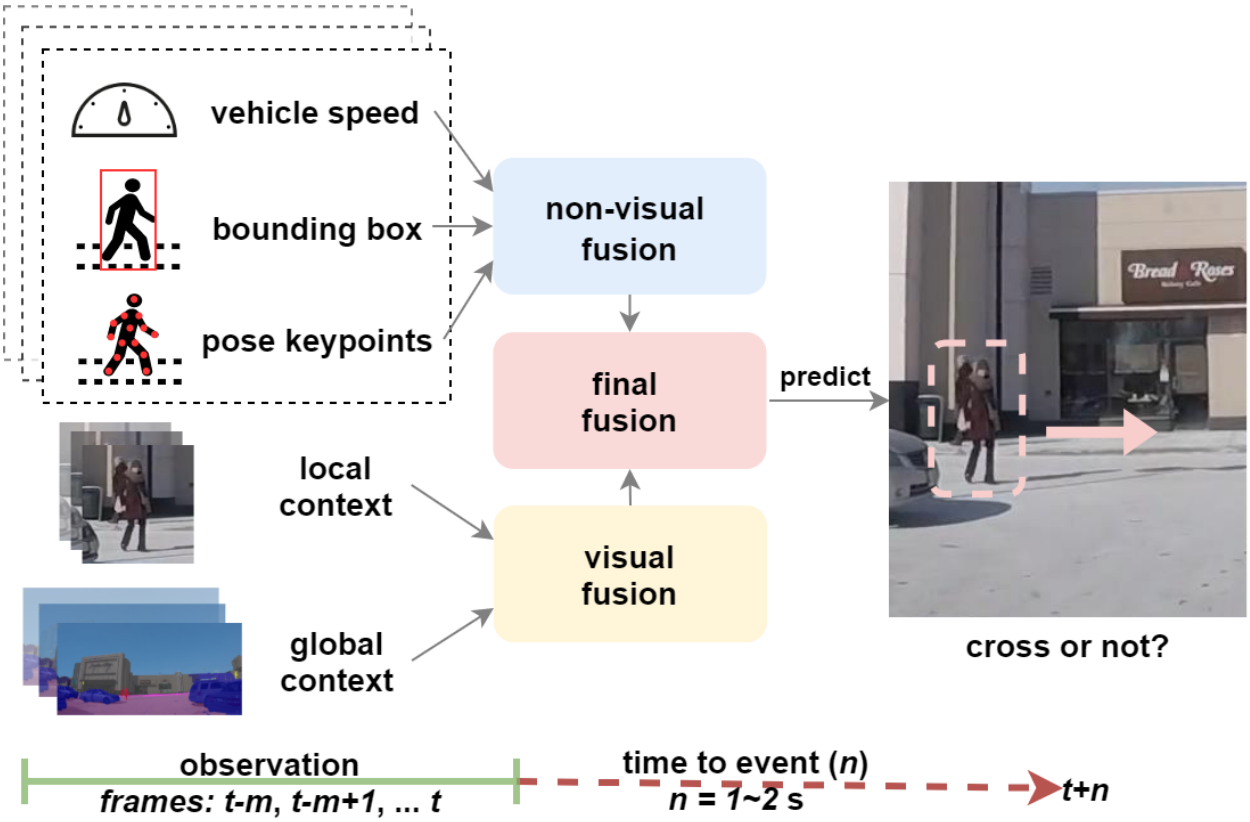


Figure 7.1: Predicting pedestrian crossing intention is a multi-modal spatio-temporal problem. Our method fuses inherently different spatio-temporal phenomena with CNN-based visual encoders, RNN stacks, and attention mechanisms to achieve state-of-the-art performance.

pedestrian bounding boxes, poses, appearance, and even the ego-vehicle information [174, 175, 176, 177, 178]. The most recent benchmark of pedestrian intention prediction was released by [179], in which the PCPA model achieved the state-of-the-art in the most popular dataset JAAD [170]. However, PCPA does not consider global contexts such as road geometry and other road users, factors we believe are nonnegligible in pedestrian crossing intention prediction. Furthermore, the existing fusion strategies may not be optimal.

In this work, we focus on improving the performance of vision-based prediction of pedestrian crossing intention, i.e., whether a pedestrian detected by a front-facing camera will cross the road or not in a short time horizon (1-2s). Our work leverages the power of deep neural networks and fuses the features from different channels. As shown in Figure 7.1, the proposed model considers both non-visual and visual information. They are extracted from a sequence of video frames 1-2s before the crossing / not crossing (C/NC) event. Non-visual information includes the pedestrian's bounding box, pose keypoints, and ego-vehicle speed. Visual information contains local context and global context. Local context is the enlarged pedestrian appearance based on the bounding box position. Global context is the semantic segmentation of the road, pedestrians (all pedestrians in the scene), and vehicles. They are used because they significantly affect the target pedestrian's crossing decision. We propose a hybrid method of fusing the the non-visual and visual features, which is justified by comparing different strategies of feature fusion.

Our main contributions are as follows:

- A novel vision-based pedestrian intention prediction framework for ADSs and ADASs. The proposed method employs a novel neural network architecture for utilizing different spatio-temporal features with a hybrid fusion strategy.
- Extensive ablation studies on different feature fusion strategies (early, later, hierarchical, or hybrid), input configurations (adding/removing input channels, using semantic segmentation masks as explicit global context), and visual encoder options (3D CNN or 2D convolution with RNN + attention) to identify the best model layout.
- Demonstrating the efficiency of the proposed method on the commonly used JAAD [170] and PIE [173] datasets, and achieving state-of-the-art performance on the most recent pedestrian action prediction benchmark [179].

7.2 Related Work

Vision-based pedestrian crossing prediction traces back to the works [180] that utilize the Caltech Pedestrian Detection Benchmark [181]. However, the Caltech dataset does not explicitly annotate the crossing behavior of the pedestrians. This gap was later filled by the introduction of the JAAD dataset [170] that offers high-resolution videos and explicit crossing behavior annotations. With the release of the JAAD dataset, a simple baseline was also created that uses a 2D convolutional neural network (CNN) to encode the features in a given previous frame and then uses a linear support vector machine (SVM) to predict the C/NC event.

Spatio-temporal modeling. Instead of using a single image, most recent works use image sequences as input to the prediction model due to the importance of temporal information in the prediction task. This leads to spatio-temporal modeling.

Spatio-temporal modeling can be achieved by first extracting visual (spatial) features per frame via 2D CNNs [182] or graph convolution networks (GCNs) [183], and then feeding these features into RNNs such as the long-short term memory (LSTM) model [184] and the gate recurrent unit (GRU) model [185]. For example, [171, 172, 173] use 2D convolution to extract the visual features from image sequence and RNNs to encode the temporal information among these features. The encoded sequential visual features are fed into a fully-connected layer to obtain the final intention prediction. [183] uses a graph representation to encode the spatial relationship among the target pedestrian and surrounding agents. The prediction task was evaluated from two different perspectives, a pedestrian-centric setting and a location-centric setting. However, ego vehicle motion and explicit visual features are not modeled in this work.

Another way of extracting the sequential visual features is utilizing a 3D CNN [186]. It directly captures the spatio-temporal features by replacing the 2D kernels of the convolution and the pooling layers in the 2D CNN with 3D counterparts. For example, [187, 188] use a 3D CNN based framework (3D DenseNet) to directly extract the sequential visual features from the pedestrian image sequence. The final prediction is achieved by using a fully-connected layer.

The crossing intention prediction task can also be combined with scene prediction. A couple of works [189, 190] attempted to decompose the prediction task into two stages. In the first stage, the model predicts a sequence of future scenes using an encoder/decoder network. Then, pedestrian actions are predicted based on the generated future scenes using a binary classifier.

Feature fusion. Instead of end-to-end modeling of visual features, information such as pedestrian's bounding box, body-pose keypoints, vehicle motion, and the explicit global scene context

can also be modeled as separate channels as inputs to the prediction model. This requires a proper way of fusing the above information.

For example, [174, 191, 192, 193, 175] introduced human poses/skeletons in pedestrian crossing prediction tasks since the human pose can be considered as a good indicator of human behaviors. By extracting the pose keypoints from cropped pedestrian images, crossing behavior classifiers were built based on the human pose feature vectors. Improvement in prediction accuracy shows the effectiveness of using pose features. However, these methods either only rely on human pose features without considering other important features or pay less attention to feature fusion.

Some other methods focused on novel fusion architectures. For instance, [176] proposed SF-GRU, a stacked RNN-based architecture, to hierarchically fuse five feature sources (pedestrian appearance, surrounding context, pose, bounding box, and ego-vehicle speed) for pedestrian crossing intention prediction. Nevertheless, this method does not take global context into account. [177] proposed a multi-modal based prediction system that integrates four feature sources (local scene, semantic map, pedestrian motion, and ego-motion). The global context (semantic map) is utilized, but it lacks other important features such as human pose. [194] proposed a multi-task based prediction framework to take advantages of feature sharing and multi-task learning. It fuses four feature sources (semantic map, pedestrians' trajectory, grid locations, and ego-motion). However, local context and human pose are not considered in the model.

Very recently, more datasets such as PIE [173] and PePScenes [195] provide annotations for fusing different features. A benchmark was also released with the PCPA model [179]. These create more room for researchers to explore the task of vision-based pedestrian crossing intention prediction.

7.3 Proposed Method

7.3.1 Problem formulation

The task of vision-based pedestrian crossing intention prediction is formulated as follows. Given a sequence of observed video frames from the vehicle's front view and the relevant information of ego-vehicle motion, the goal is to design a model that can estimate the probability of the target pedestrian i's action $A_i^{t+n} \in \{0, 1\}$ of crossing the road, where t is the specific time of the last observed frame and n is the number of frames from the last observed frame to the crossing / not crossing (C/NC) event.

It is worth noting that the existing literature usually uses the terms pedestrian action, behavior, and intention interchangeably. What is being predicted in this work is whether a pedestrian will cross or not in a short time horizon. Here we use pedestrian crossing intention as surrogates for crossing action or behavior. We assume that the action of crossing is equivalent to the intention of crossing.

In the proposed model, explicit features such as pedestrian's bounding box, pose keypoints, local context (cropped image around the pedestrian), and global context (semantic segmentation) are first extracted. They are then used together with the vehicle's speed as separate channels that serve as the input to the prediction model. Our model has the following inputs:

- The sequential local context around pedestrian i:

$$C_{li} = \{c_{li}^{t-m}, c_{li}^{t-m+1}, ..., c_{li}^{t}\};$$

- The 2D location trajectory of pedestrian i denoted by bounding box coordinates (top-left

points and bottom-right points):

$$L_i = \{l_i^{t-m}, l_i^{t-m+1}, ..., l_i^t\};$$

- Pose keypoints of pedestrian i:

$$P_i = \{p_i^{t-m}, p_i^{t-m+1}, ..., p_i^t\};$$

- Speed of ego-vehicle:

$$S = \{s^{t-m}, s^{t-m+1}, ..., s^t\};$$

- The sequential global context denoted by the mask of semantic segmentation:

$$C_g = \{c_g^{t-m}, c_g^{t-m+1}, ..., c_g^t\}.$$

Each source has a sequence of length $m+1$. The input sources are illustrated in Figure 7.2 and further described below.

7.3.2 Input acquisition

Local context and 2D location trajectory. The local context C_{li} provides visual features of the target pedestrian. The 2D location trajectory L_i gives the position change of the target

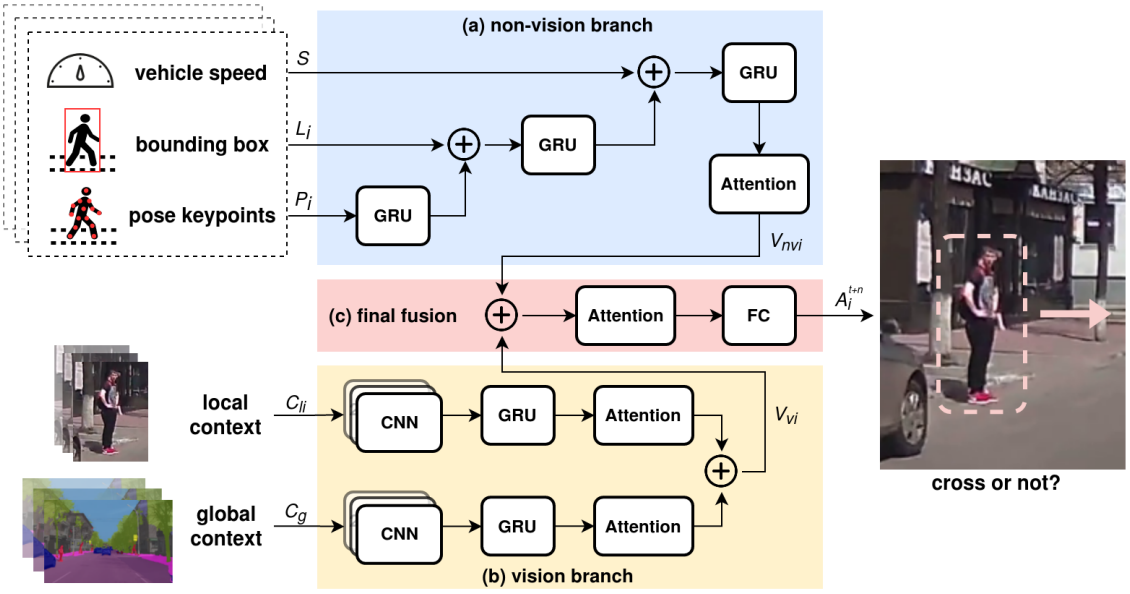


Figure 7.2: Overview of the proposed pedestrian crossing intention prediction model. The yellow part denotes the fusion of visual features. 2D convolutional features of local context and global context are encoded by GRUs and fed to the attention blocks respectively. The two outputs are concatenated as final visual features. The blue part denotes the fusion of local features (non-visual). These non-visual features are encoded by another GRU and fused hierarchically, and then fed to an attention block to obtain the final non-visual features. The red part denotes the final fusion. Final visual features and final non-visual features are concatenated and fed to an attention block. A fully-connected (FC) layer is then applied to make the final prediction.

pedestrian in the image. They can be extracted by a detection (e.g. YOLO [196]) and tracking (e.g. Deep-SORT [197]) system. At present, the detection and tracking algorithms are good enough to generate near ground-truth results. Therefore, in this work, we directly use the ground truth C_{li} and L_i from the dataset. The main reason is that pedestrian detection and tracking are not the primary focus of this work. We would like to focus on the model architecture design and remove the noise from the detection and tracking. This also follows the configurations in most related works. A small part of the work of [175] considers the impact of 2D detection in the prediction task. However, their innovation and focus are on how to build the overall pipeline. Another reason is that by using ground truth we can fairly compare our method with most related works. Specifically, the local context $C_{li} = \{c_{li}^{t-m}, c_{li}^{t-m+1}, ..., c_{li}^{t}\}$ consists of a sequence of RGB images of size $[224, 224]$ pixels around the target pedestrian. The 2D location trajectory $L_i = \{l_i^{t-m}, l_i^{t-m+1}, ..., l_i^{t}\}$ consists of target pedestrian's bounding box coordinates, i.e.,

$$l_i^{t-m} = \{x_{it}^{t-m}, y_{it}^{t-m}, x_{ib}^{t-m}, y_{ib}^{t-m}\},$$

where $x_{it}^{t-m}, y_{it}^{t-m}$ denotes the top-left point and $x_{ib}^{t-m}, y_{ib}^{t-m}$ bottom-right point.

Pedestrian pose keypoints. Pedestrian pose keypoints represent the target pedestrian's detailed motion, i.e., the posture at each frame while moving. They can be obtained by applying a pose estimation algorithm on the local context C_{li}. Since the applied JAAD dataset does not provide ground truth pose keypoints, we utilize the pre-trained OpenPose model [198] to extract the pedestrian pose keypoints $P_i = \{p_i^{t-m}, p_i^{t-m+1}, ..., p_i^{t}\}$, where p is a 36D vector of 2D coordinates that contain 18 pose joints, i.e.,

$$p_i^{t-m} = \{x_{i1}^{t-m}, y_{i1}^{t-m}, x_{i2}^{t-m}, y_{i2}^{t-m}, ..., x_{i18}^{t-m}, y_{i18}^{t-m}\}.$$

Ego-vehicle speed. Ego-vehicle speed S is a major factor that affects the pedestrian's crossing decision. It can be directly read from the ego-vehicle's system. Since the dataset contains the annotation of ego-vehicle's speed, we directly use the ground truth labels for the vehicle speed $S = \{s^{t-m}, s^{t-m+1}, ..., s^{t}\}$.

Global context. Global context $C_g = \{c_g^{t-m}, c_g^{t-m+1}, ..., c_g^{t}\}$ offers the visual features that account for multi-interactions between the road and road users, or among road users. In our work, we use pixel-level semantic masks to represent the global context. The semantic masks classify and localize different objects in the image by labeling all the pixels associated with the objects. Since the JAAD dataset does not have annotated ground truth of semantic masks, we use the DeepLabV3 model [199] pre-trained on the Cityscapes Dataset [200] to extract the semantic masks and select important objects (e.g. road, street, pedestrians and vehicles) as the global context. For the model to learn the interactions between the target pedestrian i and these objects, the target pedestrian is masked by an unique label. The mask area uses the target pedestrian i's bounding box (obtained from L_i). The semantic segmentation of all input frames are scaled to be of size $[224, 224]$ pixels, which is the same as the local context.

7.3.3 Model architecture

The overall architecture is shown in Figure 7.2. It consists of CNN modules, RNN modules, attention modules, and a novel way of fusing different features.

CNN module. We use the VGG19 [182] model pre-trained on the ImageNet dataset [201] to build the CNN module. Sequential RGB images are collected as a 4D array input with the dimensions of [number of observed frames, row, cols, channels] ($[16, 224, 224, 3]$ in this work), and then loaded by the CNN module. First, the feature map of every image from the fourth maxpooling

layer of VGG19 is extracted with size $[512, 14, 14]$. Second, every feature map is averaged by a pooling layer with a 14×14 kernel, and then flattened and concatenated, to obtain a final feature tensor with size $[16, 512]$, as sequential visual features.

RNN module. We use a gated recurrent unit (GRU) [185] to build the RNN module. The reason for choosing a GRU is that the GRU is more computationally efficient than its counterpart LSTM [184], which is older, and its architecture is relatively simple. The applied GRUs have 256 hidden units, which result in a feature tensor of size $[16, 256]$

Attention module. An attention module [202], by selectively focusing on parts of features, is used for better memorizing sequential sources. Sequential features (e.g. the output of RNN-based encoder) are represented as hidden states $h = \{h_1, h_2, ..., h_e\}$. The attention weight is computed as:

$$\alpha = \frac{exp(score(h_e, \tilde{h_s}))}{\sum_{s'} exp(score(h_e, \tilde{h_{s'}}))},$$

where $score(h_e, \tilde{h_s}) = h_e^T W_s \tilde{h_s}$ and W_s is a weight matrix. Such attention weight trades off the end hidden state h_e with each previous source hidden state $\tilde{h_s}$. The output vector of the attention module is produced as

$$V_{attention} = tanh(W_c[h_c; h_e]),$$

where W_c is a weight matrix, and h_c is the sum of all attention weighted hidden states as $h_c = \sum_{s'} \alpha \tilde{h_{s'}}$. The output of the attention module in our work is a feature tensor with size $[1, 256]$.

Hybrid fusion. We applied a hybrid way of fusing the features from different sources. The strategy is shown in Figure 7.2. The proposed architecture has two branches, one for non-visual features and one for visual features.

The non-vision branch fuses three non-visual features (bounding boxes, pose keypoints, and vehicle speed). They are hierarchically fused according to their complexity and level of abstraction. The later the fusion stage occurs, the more impact the fused features will have on the final prediction. This is illustrated in Figure 7.2(a). First, sequential pedestrian pose keypoints P_i are fed to an RNN-based encoder. Second, the output of the first stage is concatenated with 2D location trajectory L_i and fed to a new RNN-based encoder. Finally, the output of the second stage is concatenated with ego-vehicle speed S and fed to a final RNN-based encoder. The output of the final encoder is then fed to an attention block to obtain the final non-visual feature vectors V_{nvi}.

The vision branch fuses two visual features, consisting of local context (enlarged pedestrian appearance around the bounding box) and global context (semantic segmentation of important objects in the whole scene), as shown in Figure 7.2(b). Local context C_{li} is encoded by first extracting spatial features from the CNN module (as explained in the previous section) and then extracting temporal features from the GRU module. Global context C_g is encoded in the same way. Both local and global features are then fed into their attention modules, and finally, concatenated together to generate the final visual feature vectors V_{vi}.

Lastly, as shown in Figure 7.2(c), the final non-visual feature vectors V_{nvi} and the final visual feature vectors V_{vi} are concatenated and fed into another attention block, followed by a fully-connection (FC) layer to obtain the final predicted action:

$$A_i^{t+n} = f_{FC}(f_{attention}(V_{nvi}; V_{vi})).$$

7.4 Experiments

7.4.1 Dataset and Benchmark

The proposed model was evaluated using both the JAAD [170] and PIE [173] datasets. The JAAD dataset contains two subsets, JAAD behavioral data (JAAD_{beh}) and JAAD all data (JAAD_{all}). JAAD_{beh} contains pedestrians who are crossing (495 samples) or are about to cross (191 samples). JAAD_{all} has additional pedestrians (2100 samples) with non-crossing actions. To create a fair benchmark, the dataset configuration is the same as used in [179]. It uses a data sample overlap of 0.8 and a local context scale of 1.5.

The PIE dataset is a more comprehensive dataset compared to the JAAD dataset. It contains 1322 non-crossing samples and 512 crossing samples. Besides, the PIE dataset covers pedestrians with more different appearances and scenes with more different surroundings than those in the JAAD dataset.

The evaluation metrics use accuracy, AUC, F1 score, precision, and recall. These are the most recognized metrics and are used by most related works. False alarm rate is another important metric when deploying this algorithm in autonomous driving systems. False alarms may cause unnecessary brakes for autonomous cars, resulting in unpleasant experiences for the passengers. The above metrics inherently include the false alarm rate. They are more balanced metrics for evaluating a prediction system. Most related works and benchmarks adopt this metric system to report their results. Using this metric system, we can fairly compare our works with others.

7.4.2 Implementation

In the experiments, the proposed model was compared with the following methods: SingleRNN [171], SF-GRU [176] and PCPA[179]. We adopted the benchmark implementation released with the PCPA model [179]. This benchmark collects the implementations of most pedestrian intention prediction methods. Our model was developed based on this benchmark. We use a dropout of 0.5 in the attention module, L2 regularization of 0.001 in the FC layer, binary cross-entropy loss, and the Adam optimizer [203]. For the JAAD dataset, we use learning rate $= 5 \times 10^{-7}$, epochs $= 40$, and batch size $= 2$. For the PIE dataset, we use learning rate $= 5 \times 10^{-5}$, epochs $= 60$, and batch size $= 2$. ll models were trained and tested on the same split of the dataset, as suggested by the benchmark [179]. Note that the JAAD dataset does not provide explicit vehicle speed. Instead, the driver's action is recorded as an abstract encoding of the vehicle speed. The action contains [stopped (0), moving slow (1), moving fast (2), decelerating (3), accelerating (4)].

7.4.3 Ablation study

An ablation study was also conducted to compare different strategies of fusing different features. In addition to the baseline methods (SingleRNN [171], SF-GRU [176] and PCPA [179]) and the proposed model (Ours), a total of 7 variants of the proposed model (Ours1, Ours2, ... Ours7, as indicated in Table 7.5 and Table 7.6) were trained and compared with the proposed one. First, for the visual encoder, we tried (1) a 2D CNN combined with RNN (VGG and GRU in our experiments) and (2) a 3D CNN as proposed in the PCPA model. Second, we tried the models with and without the global feature (semantic segmentation). Finally, we tried different fusion strategies that include later fusion, early fusion, and hierarchical fusion so that they can be compared with the proposed hybrid fusion strategy. Later fusion (Figure 7.3) is the same as that proposed in PCPA [179]. Early fusion (Figure 7.4) concatenates non-visual features and visual features directly and then sends them into one RNN module followed by an attention module. Hierarchical fusion (Figure 7.5)

gradually fuses both visual features and non-visual features by RNN modules in the same manner as in Figure 7.2(a), followed by an attention module.

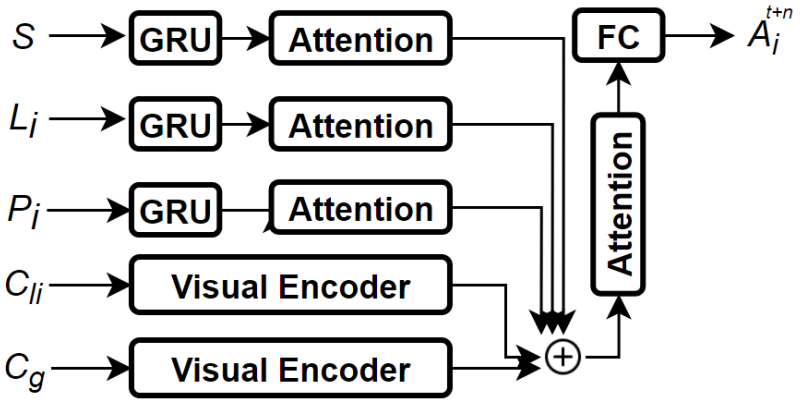


Figure 7.3: Illustration of Later Fusion

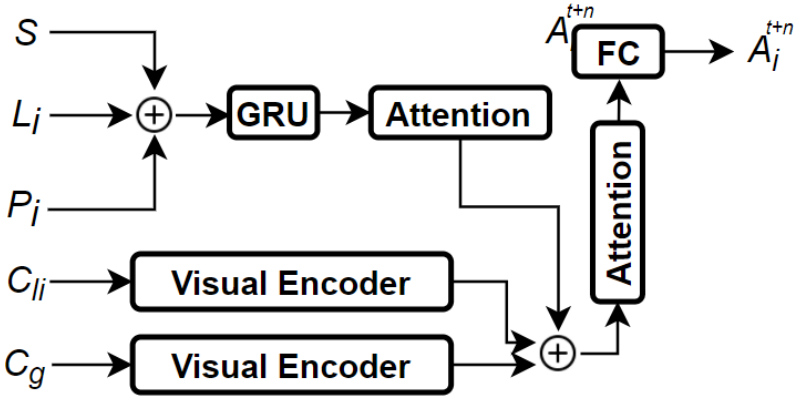


Figure 7.4: Illustration of Early Fusion

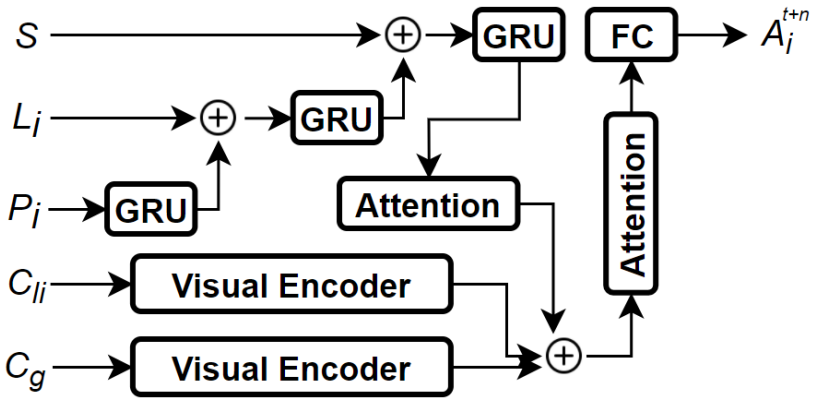


Figure 7.5: Illustration of Hierarchical Fusion

7.5 Results

7.5.1 Quantitative Results

Table 7.1 shows the qualitative results on the JAAD_{beh} dataset. It compares the proposed model with baseline models of SingleRNN [171], SF-GRU [176] and PCPA [179]. The proposed model achieved the best scores in accuracy, F1, and recall. F1 score is an balanced metric considering both recall and precision. For binary classification, it is the most important indicator of quality of the model. Our model achieved about 4% improvement in F1. In addition to F1, accuracy is another important metric, and our model also achieved the best score.

Table 7.2 shows the qualitative results on the JAAD_{all} dataset. JAAD_{all} has additional samples of non-crossing behaviors. It is larger than JAAD_{beh}. The data distribution is more similar to real world scenarios. As illustrated by Table 7.2, the proposed method achieved the best in accuracy, AUC, F1, and precision. Similar to the results in JAAD_{beh}, our model achieved the best score in terms of the two important metrics, F1 and accuracy.

Table 7.3 shows the qualitative results on the PIE dataset. On such a comprehensive dataset, our proposed method outperforms other methods with a considerable gap, which shows the importance and advantages of designing a hybrid fusion strategy and utilizing global context.

Table 7.1: Quantitative Results on the JAAD Behavior Subset

Models	**Model Variants**			**JAAD_{beh}**				
	Visual Encoder	Global Context	Fusion Approach	Accuracy	AUC	F1 Score	Precision	Recall
SingleRNN [171]	VGG + GRU	✗	✗	0.60	0.54	0.70	0.65	0.76
SF-GRU [176]	VGG + GRU	✗	hierarchical-fusion	0.58	**0.56**	0.65	**0.68**	0.62
PCPA [179]	3D CNN	✗	later-fusion	0.56	0.54	0.63	0.66	0.60
Ours	VGG + GRU	✓	hybrid-fusion	**0.62**	0.54	**0.74**	0.65	**0.85**

Table 7.2: Quantitative Results on the JAAD All Dataset

Models	**Model Variants**			**JAAD_{all}**				
	Visual Encoder	Global Context	Fusion Approach	Accuracy	AUC	F1 Score	Precision	Recall
SingleRNN [171]	VGG + GRU	✗	✗	0.78	0.77	0.54	0.42	0.75
SF-GRU [176]	VGG + GRU	✗	hierarchical-fusion	0.76	0.77	0.53	0.40	0.79
PCPA [179]	3D CNN	✗	later-fusion	0.77	0.79	0.56	0.42	**0.83**
Ours	VGG + GRU	✓	hybrid-fusion	**0.83**	**0.82**	**0.63**	**0.51**	0.81

Table 7.3: Quantitative Results on the PIE Dataset

Models	**Model Variants**			**PIE**				
	Visual Encoder	Global Context	Fusion Approach	Accuracy	AUC	F1 Score	Precision	Recall
SingleRNN [171]	VGG + GRU	✗	✗	0.83	0.78	0.69	0.72	0.67
SF-GRU [176]	VGG + GRU	✗	hierarchical-fusion	0.84	0.80	0.71	0.72	0.71
PCPA [179]	3D CNN	✗	later-fusion	0.87	0.85	0.78	0.76	0.81
Ours	VGG + GRU	✓	hybrid-fusion	**0.89**	**0.86**	**0.80**	**0.79**	**0.81**

Note that the results of PCPA were generated based on the official implementation released by the PCPA author. We cannot reproduce the same results as reported in the PCPA paper. After communicating with PCPA's authors, they confirm that our reproduced result is normal.

We also analyzed the computational cost of the above models. The total number of model parameters is used as an indicator of computational cost. Table 7.4 shows the comparison. PCPA [179] has the highest number of model parameters as it utilizes 3D convolution. Our model has only one-tenth of the parameters in PCPA, but still achieved better performance. This provides advantages in real-time deployment.

Table 7.4: Comparison of Computational Cost

Model	Number of Params.
SingleRNN [171]	1,016,321
SF-GRU [176]	2,595,329
PCPA [179]	31,165,953
Ours	2,988,545

7.5.2 Qualitative Results

Figure 7.6 provides qualitative results for the proposed model of pedestrian crossing intention prediction. We mainly compared the proposed method with the PCPA model. In the provided examples, our method correctly predicted the crossing intention but the PCPA failed. Taking a closer look at the examples, the following argument is raised. Without utilizing the global context, the task of crossing intention prediction may face the problems of (1) unknown direction of the pedestrian (Case a in Figure 7.6), (2) occlusion (Case b in Figure 7.6), and (3) poor vision (Case c in Figure 7.6). Global context can provide additional information to account for the interaction between the whole scene and the target pedestrian.

Figure 7.7 provides more qualitative results to analyze the advantages of the proposed model over the PCPA model as well as a few failure cases. Figure 7.7-(a) and Figure 7.7-(b) show cases when the proposed model generated correct predictions but the PCPA failed. The main reason is that our model considers the global visual context that contains the semantic segmentation of the drivable area. The model can learn from this whether the pedestrian is moving toward or on the drivable area, which is an important indicator of pedestrian crossing intention.

Figure 7.7-(c) and Figure 7.7-(d) show cases when both the proposed model and the PCPA failed. Figure 7.7-(c) shows an intersection scenario. The pedestrian (yellow bounding box) has already crossed the ego road but is near the edge of the road on the other side. This may mislead the model to generate a prediction of crossing. The failure in Figure 7.7-(d) was mainly due to the poor illumination such that the model cannot obtain enough detailed features.

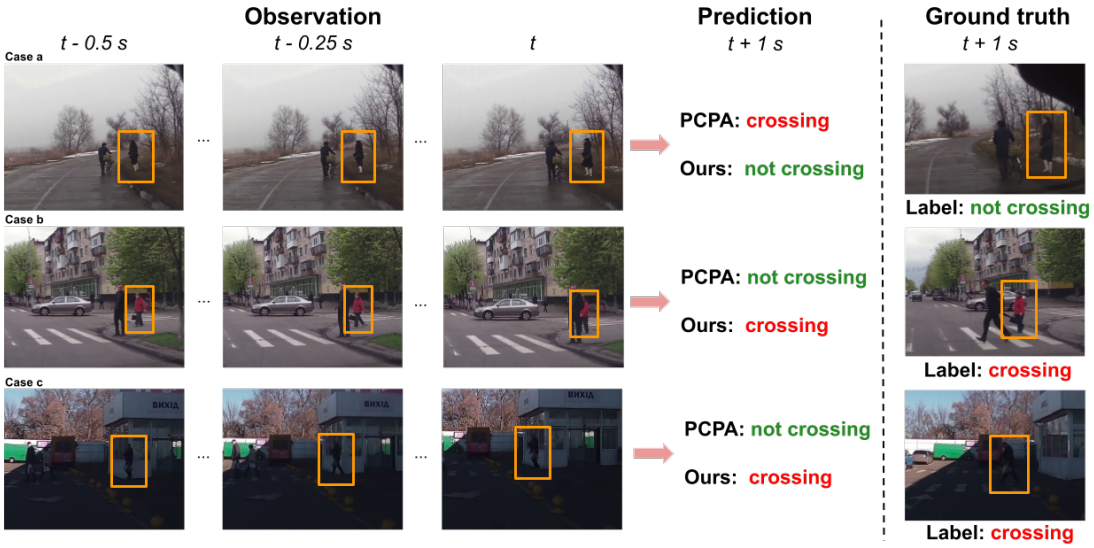


Figure 7.6: Qualitative results on the JAAD dataset produced by and our proposed model (Ours). The target pedestrians in images are enclosed by orange **bounding boxes**. The prediction results as well as ground truth labels are represented as red **crossing** or green **not crossing**.

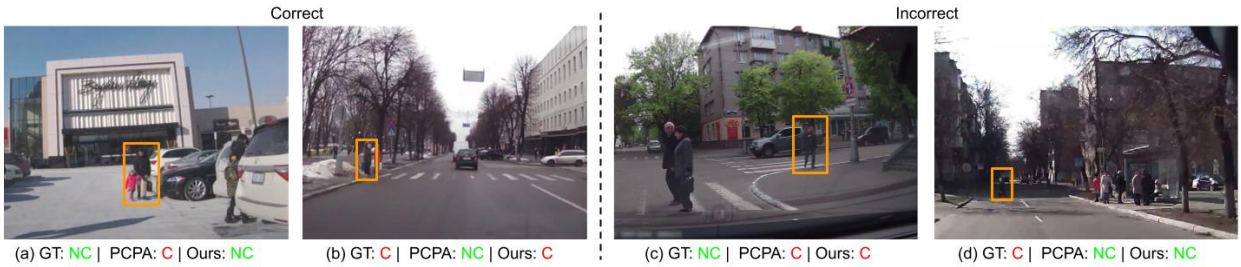


Figure 7.7: More qualitative results. (a) and (b) show cases of correct prediction by the proposed model for which the PCPA failed. (c) and (d) show results when both the proposed and the PCPA model failed.

7.5.3 Results of Ablation Study

Table 7.5 and Table 7.6 show the ablation study on the JAAD_{beh} and JAAD_{all} datasets, respectively. Table 7.7 shows the ablation study on the PIE dataset. Different model variants are denoted by Ours1, Ours2, ..., Ours7. By comparing Ours5 with Ours4 and Ours1 with the PCPA model, it shows that introducing global context can improve the model performance. In terms of fusion strategies, the proposed hybrid fusion strategy achieved the best performance, as seen by comparing Ours with Ours5, Ours6, and Ours7. If we further compare Ours4 with the PCPA model, it shows that using a 2D CNN plus RNN instead of a 3D CNN has a minimal impact on performance. This evidence also demonstrates that the improvement of our proposed method is mainly due to the new hybrid fusion strategy and global context.

7.5.4 Effect of Longer Prediction Horizon

It is claimed in some traffic studies [204] that the most suitable prediction horizon, i.e., time-to-event (TTE), is 1-2 seconds, because longer prediction horizon is impractical due to unpredictable

Table 7.5: Ablation Study on the JAAD Behavior Subset

Models	Model Variants			JAAD_{beh}				
	Visual Encoder	Global Context	Fusion Approach	Accuracy	AUC	F1 Score	Precision	Recall
Ours	VGG + GRU	✓	hybrid-fusion	0.62	0.54	**0.74**	0.65	**0.85**
Ablations								
Ours1	3D CNN	✓	later-fusion	0.59	0.53	0.69	0.65	0.75
Ours2	3D CNN	✓	early-fusion	0.59	0.54	0.69	0.65	0.74
Ours3	3D CNN	✓	hierarchical-fusion	0.57	0.48	0.70	0.62	0.81
Ours4	VGG + GRU	✗	later-fusion	0.59	0.51	0.72	0.63	0.83
Ours5	VGG + GRU	✓	later-fusion	**0.64**	**0.59**	0.73	**0.68**	0.78
Ours6	VGG + GRU	✓	early-fusion	0.60	0.56	0.70	0.67	0.73
Ours7	VGG + GRU	✓	hierarchical-fusion	0.54	0.50	0.64	0.63	0.65

Table 7.6: Ablation Study on the JAAD All Dataset

Models	Model Variants			JAAD_{all}				
	Visual Encoder	Global Context	Fusion Approach	Accuracy	AUC	F1 Score	Precision	Recall
Ours	VGG + GRU	✓	hybrid-fusion	**0.83**	**0.82**	**0.63**	**0.51**	0.81
Ablations								
Ours1	3D CNN	✓	later-fusion	0.77	0.77	0.54	0.42	0.76
Ours2	3D CNN	✓	early-fusion	0.77	0.74	0.51	0.41	0.69
Ours3	3D CNN	✓	hierarchical-fusion	0.78	0.77	0.55	0.43	0.75
Ours4	VGG + GRU	✗	later-fusion	0.75	0.79	0.54	0.40	**0.85**
Ours5	VGG + GRU	✓	later-fusion	0.77	0.80	0.56	0.43	0.84
Ours6	VGG + GRU	✓	early-fusion	0.79	0.74	0.52	0.43	0.66
Ours7	VGG + GRU	✓	hierarchical-fusion	0.80	0.81	0.59	0.46	0.84

nature of most urban scenarios and human dynamics [204]. However, to show the generalization ability, we still evaluated the proposed model with a longer TTE prediction horizon of 2-3 seconds. This was done by recreating the samples with a larger number of future frames. Table 7.8 shows the effect of different prediction horizons for the proposed model. It can be seen from the table that the model performance drops on both the JAAD and PIE datasets. This supports the claims that a TTE of 1-2 seconds is more suitable than a TTE of 2-3 seconds.

7.5.5 Comparison of Different Prediction Task Configurations

There are some works that formulate the pedestrian intention prediction task in a different setting. Although using the same datasets, JAAD and PIE, they prepare the training, evaluation, and testing samples in a different way. The quantitative results cannot be directly compared with the proposed method. We analytically compared their results with ours with the conditions

Table 7.7: Ablation Study on the PIE Dataset

Models	Model Variants			PIE				
	Visual Encoder	Global Context	Fusion Approach	Accuracy	AUC	F1 Score	Precision	Recall
Ours	VGG + GRU	✓	hybrid-fusion	**0.89**	**0.86**	**0.80**	**0.79**	0.81
Ablations								
Ours1	3D CNN	✓	later-fusion	0.84	0.85	0.76	0.67	0.86
Ours2	3D CNN	✓	early-fusion	0.85	0.85	0.76	0.68	**0.87**
Ours3	3D CNN	✓	hierarchical-fusion	0.86	0.84	0.76	0.75	0.77
Ours4	VGG + GRU	✗	later-fusion	0.85	0.85	0.76	0.69	0.84
Ours5	VGG + GRU	✓	later-fusion	0.86	0.84	0.76	0.74	0.78
Ours6	VGG + GRU	✓	early-fusion	0.74	0.64	0.47	0.55	0.41
Ours7	VGG + GRU	✓	hierarchical-fusion	0.86	0.84	0.77	0.74	0.80

Table 7.8: Effect of Longer Prediction Horizon

Dataset	TTE	Acc.	AUC	F1	Precision	Recall
$JAAD_{beh}$	1-2s	**0.62**	**0.54**	**0.74**	**0.65**	**0.85**
	2-3s	0.53	0.47	0.65	0.62	0.68
$JAAD_{all}$	1-2s	**0.83**	**0.82**	**0.63**	**0.51**	**0.81**
	2-3s	0.79	0.78	0.57	0.46	0.76
PIE	1-2s	**0.89**	**0.86**	**0.80**	**0.79**	**0.81**
	2-3s	0.78	0.77	0.65	0.59	0.73

• The **bold** result indicates he best result among the models.

described. The comparison can be found in Table 7.9. The following works were compared:

Table 7.9: Comparison of Different Pedestrian Intention Prediction Task Configurations

Work	Prediction Task Configuration	Results	Comparison to Ours
Liu et al. [183]	Both pedestrian-centric and location-centric configuration	Achieved 0.77 accuracy on JAAD dataset for action prediction at 1s	Achieved 0.83 accuracy on JAAD dataset for 1-2s future action prediction
Zhang et al. [205]	Pedestrian crossing intention at red-light scenario	Achieved 0.91 accuracy on self-created red-light scenario dataset	Unable to directly compare as the self-collected dataset is not accessible
Chen et al. [206]	Utlized a balanced sampling strategy, observing 15 frames (0.5s), predicting the action for 45 frames (1.5s)	Achieved 0.79 accuracy and 0.78 F1 score on randomly sampled testing set (balanced) using PIE dataset	Achieved 0.89 accuracy and 0.80 F1 score on PIE dataset with the commonly adopted configuration in [179]
Rasouli et al. [178]	Jointly predicting pedestrian action (intention), trajectory, and grid position	Achieved 0.91 accuracy with the auxiliary labels on PIE dataset	Achieved 0.89 accuracy with only action (intention) labels on PIE dataset

- Liu's work [183] formulated the prediction task in two different perspectives of pedestrian-centric and location-centric settings. In addition to the JAAD dataset, it also introduces a specifically designed new dataset. The spatio-temporal information is encoded by GCN and RNN. They reported an accuracy of 0.77 on the JAAD dataset for predicting exactly 1 second into the future. We use the more commonly recognized benchmark proposed in [179] in our work. We achieved an accuracy of 0.83 on the JAAD dataset for future actions of 1-2 seconds. With a longer prediction horizon and higher accuracy scores, the effectiveness of our proposed method is validated.

- Zhang's work [205] focuses on pedestrian's crossing intention at red-light scenario. It analyzed 4 different machine learning models, SVM, RF, GBM, and XGBoost, that are fed with pedestrian pose features. It achieved an accuracy of 0.91. However, the results were obtained on a self-collected and self-labeled red-light scenario dataset. Our proposed model cannot be directly compared with this method due to the inability of accessing the dataset and the different task configurations. Nevertheless, our method achieved an accuracy of 0.89 on the PIE dataset, which is very similar to Zhang's results.

- Chen's work [206] utilized a balanced sampling strategy to extract the samples for pedestrian crossing prediction. They use 15 frames (0.5s) as observation to predict the pedestrian crossing action for 45 future frames (1.5s). A graph convolutional autoencoder is used to embed spatio-temporal information. It achieved 0.79 accuracy and 0.78 F1 scores on a randomly

sampled testing set (balanced) using the PIE dataset. Our model uses the commonly adopted configuration in [179]. We achieved 0.89 accuracy and 0.80 F1 score on the PIE dataset.

- Rasouli's work [178] formulates the pedestrian crossing intention as a sub-task of a multitask prediction framework, i.e., jointly predicting action (intention), trajectory, and grid position. They use a combined independent and joint encoding strategy with a categorical interaction module to fuse all the input channels. With the auxiliary labels, their work achieved 0.91 accuracy on the PIE dataset. As a comparison, our model achieved 0.89 accuracy on PIE but with only the action (intention) labels. Without auxiliary labels, our model still achieves comparable results. This validates the effectiveness of our model.

7.6 Conclusion

In this work, we proposed a novel method for vision-based pedestrian crossing intention prediction. Our method explicitly considers the global context as a channel representing the interaction between the target pedestrian and the whole scene. We also proposed a hybrid fusion strategy for different features using 2D CNNs, RNNs, and attention mechanisms. Experiments on the JAAD and PIE datasets show that the proposed method achieves the state-of-the-art against baseline methods in the pedestrian action prediction benchmark.

Future work can focus on improving our model's robustness in unexpected situations, e.g., poor vision and occlusion. Additionally, feature fusion with more information sources can be explored. Finally, fine-tuning the model for particular pedestrian subsets, such as children and disabled people, can increase overall safety and performance.

Chapter 8

Photorealism in Driving Simulations: Blending Generative Adversarial Image Synthesis With Rendering

8.1 Introduction

Driving simulations are important for developing and evaluating intelligent transportation systems [207]. A good simulation environment should have accurate vehicle dynamics, realistic traffic behavior, and high visual fidelity. Visual fidelity is especially crucial for validating vision-based algorithms and conducting human-in-the-loop experiments. There are numerous studies [208, 209, 210, 211, 212, 213] that utilize a driving simulation whose integrity greatly depends on the visual quality of the simulation environment.

The aforementioned studies all use rendered images that are generated by a simulation environment. However, limited work has been done on evaluating and improving the visual fidelity of state-of-the-art driving simulators. Here we investigate a new approach: introducing generative photorealism to virtual driving environments using deep learning. Data-centric applications trained or fine-tuned in a photorealistic driving simulation can be more confidently deployed to the real world. Furthermore, automated driving systems can be tested with photorealistic-looking dangerous scenes that are difficult to obtain outside a simulation environment. In addition, if non-realistic repetitive patterns can be replaced by photorealistic scenery, the degree of immersion for human-in-the-loop simulation experiments can be increased.

The fidelity of a conventional driving simulator depends on the quality of its computer graphics pipeline, which consists of 3D models, textures, and a rendering engine. High-quality 3D models and textures require artisanship, whereas the rendering engine must run complicated physics calculations for the realistic representation of lighting and shading [214]. These processes are labor-intensive, and images obtained this way are not photorealistic. Here we investigate alternatives for alleviating the aforementioned costs. An overview of our approach is shown in Figure 8.1.

The alternative to rendering is neural network based generative adversarial image synthesis. The advent of Generative Adversarial Networks (GAN) [215] enabled the realization of photo-realistic image synthesis [216, 217, 218, 219, 220, 221, 222]. A particular sub-problem, conditional image synthesis [223, 224, 225, 226, 227], delves into the more specific task of mapping a pixel-wise semantic layout to a complying photo-realistic image. The conditional semantic layout is the key link between the 3D scene and the generative synthesizer in our framework. More recently, video-to-video synthesis [228] was proposed as an alternative to image synthesis. The temporal dimension

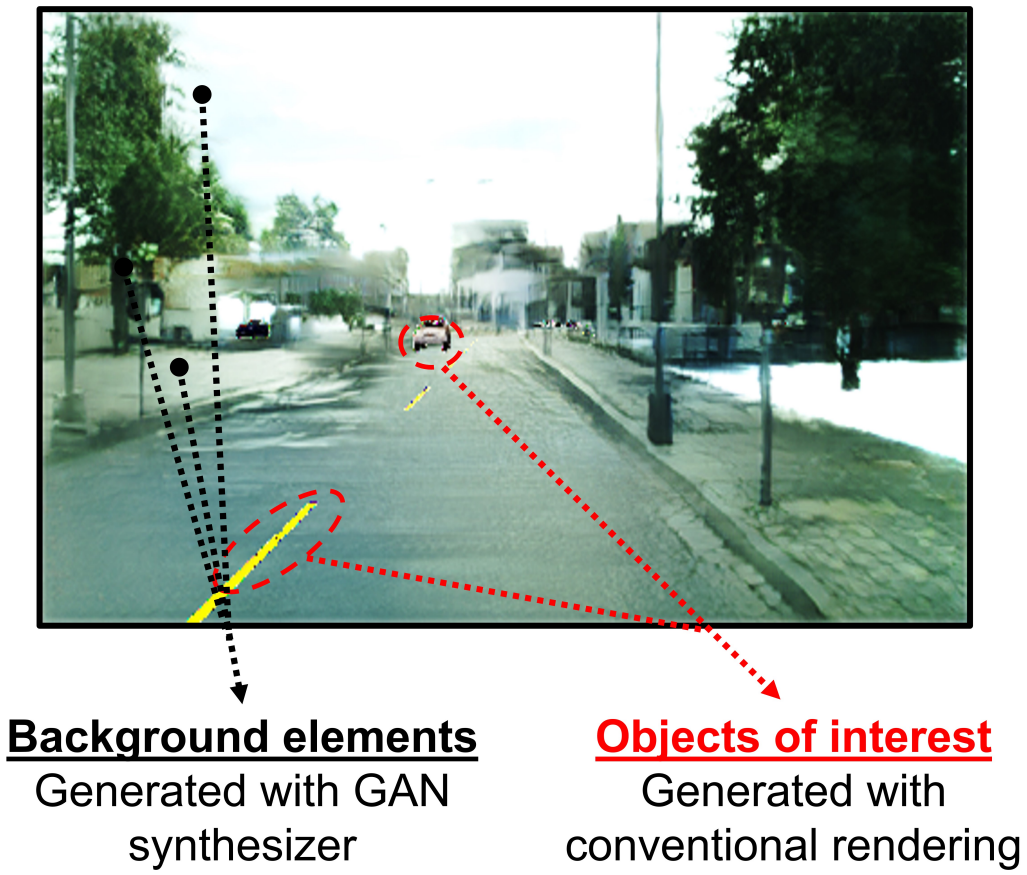


Figure 8.1: The proposed framework generates photorealistic imagery for driving simulators. First, we obtain the semantic layout of the scene through a conventional simulation pipeline with textureless simple 3D models. Then, this semantic layout is converted into a photorealistic RGB image using GANs with the proposed image formation and blending strategy.

was added to the generative process to reduce inconsistencies between synthesized frames.

The main motivation of this work is twofold: to increase the visual fidelity of driving simulations and reduce the manual labor requirements for 3D mesh and texture creation. With the use of GAN-based photorealistic image synthesizers, background objects such as trees, mountains, and the sky can be generated without detailed meshes or texture information. However, conventional rendering is still needed to have full control over important objects of interest, such as vehicles and road markers.

In this work, we propose to integrate generative adversarial image synthesis into a driving simulation. For each time step, CARLA, an open-source driving simulator [39], determines the scene's semantic layout with simple, textureless 3D models that are radiant with a unique class color. It should be noted that there is no illumination source other than the radiant 3D objects and no reflections or ambient occlusion are considered at this step. Then, a virtual pinhole camera is used to form a 2D semantic image from this scene. This image is the equivalent of a pixel-wise semantic segmentation mask. Next, the GAN-based image synthesizer converts the 2D semantic image to a photorealistic image. Conditional GAN (cGAN) [229] and CYcle GAN (Cy-GAN) [223] are the main techniques for this step. Simultaneously, a few objects of interest are partially rendered using a conventional rendering engine [230]. This is necessary as full control over some critical

objects, such as lane markings and vehicles in a driving scene, is only achieved with a conventional graphics pipe. Finally, a blending GAN mixes the cGAN/Cy-GAN synthesized image with the individually rendered objects. The proposed method was evaluated with semantic segmentation [231], an important driving-related perception task.

The main contributions of this work are:

- We introduce a novel driving simulation graphics pipeline for expediting scene creation using automated synthesis of background elements such as buildings, vegetation, and sky. To the best of our knowledge, this is the first GAN-render hybrid graphics pipeline for driving simulations.
- Blending GAN-based image synthesis with physics-based partial rendering.
- Replacing recurring patterns, such as repeating tree and building models, that are common in driving simulations with generative photorealistic surfaces as shown in Figure 8.2. Repetitive patterns can break immersion for human-in-the-loop simulation experiments. In addition, machine learning algorithms trained or fine-tuned in a repetitive environment can fail in the real world due to overfitting. As such, the proposed approach aims at increasing the integrity of simulation-based intelligent transportation research.

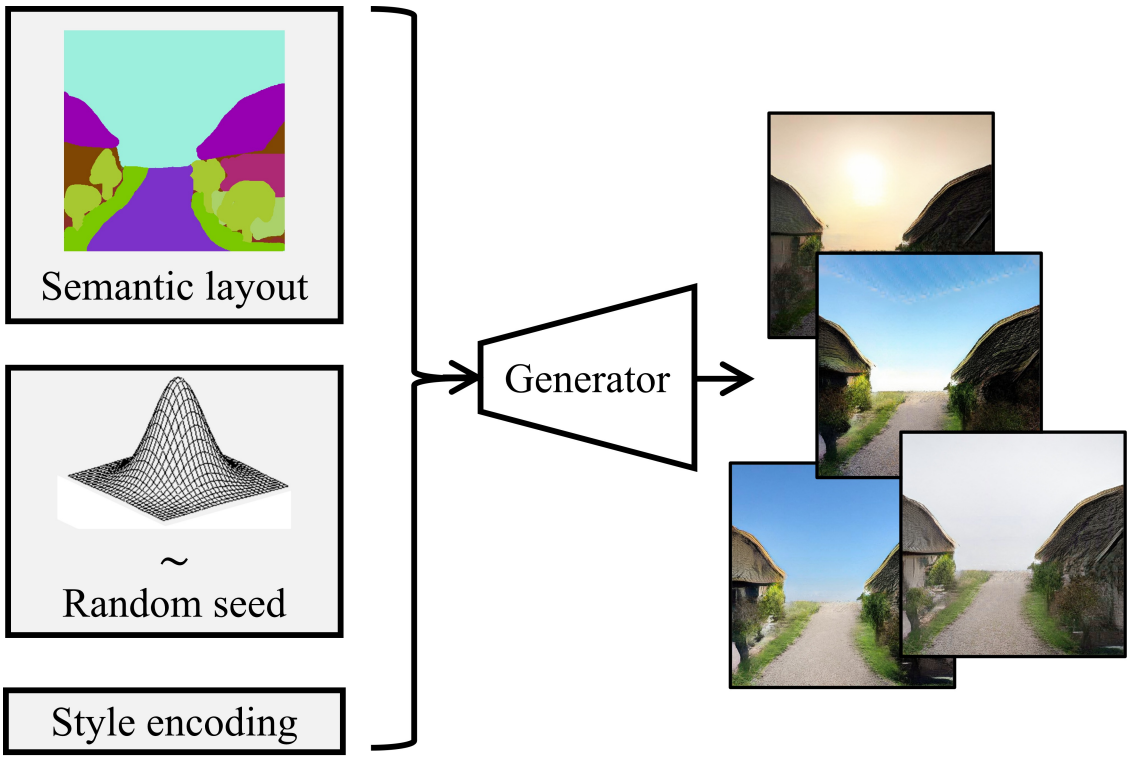


Figure 8.2: We first create the semantic layout, and then use a GAN-based image synthesizer with different style encodings to generate random but photorealistic RGB background imagery. Repetitive patterns that are common in driving simulations are memorazible by learning algorithms and break immersion for human driver subjects. The proposed approach alleviates these shortcomings.

8.2 Related work

Simulation based driving studies. Human driver reaction to various driving-related stimuli has been observed via simulation environments in numerous studies. The simulation's visual fidelity is critical for such experiments, as humans are accustomed to a real-world driving setting. Driving simulators have been used to study the driver's reaction during an automated driving take-over [210], to monitor human responses to stressful driving stimuli [208], to find the effect of inter-vehicular distances on human car following behavior [211], and to measure the effect of acoustic cues on situational awareness of human drivers [213]. An automated highway driving system with human-like decision-making capabilities has been developed via a driving simulator [212]. Another study [209] focused on human pose estimation using simulated images and showed that data-centric algorithms fine-tuned in these simulations could be used in real-world scenarios.

A recent study showed that human subjects gaze with higher variance and exhibit more diverse steering activity in driving simulations that have better visual fidelity [232]. Higher visual fidelity is always desired in human-in-the-loop experiments because human driving behavior deviates from real-world behavior in unrealistic simulation environments [233]. Furthermore, data-centric methods that are trained on synthetic data generated by conventional rendering engines fail to perform with real-world images [234].

The number and significance of these studies underline the importance for improving the visual fidelity in driving simulations. New technologies can be developed more effectively with a better simulator. For example, if photorealism can be achieved, a learning-based lane-boundary detection algorithm [235] can be trained in a simulation and deployed in the real world.

Rendering. Physics-based rendering [214] has been used at the end of the line of conventional computer graphics pipelines to form 2D imagery from virtual 3D scenes for a long time. The most common approaches, rasterization and ray-tracing, require a full pipeline of detailed 3D models, their surface textures and materials, and a physics engine such as Unreal Engine 4 [230] to run complicated calculations for representing light and shading. Here, we propose to partially replace this pipe with much simpler 3D models and remove light, texture, and material information for most of the objects in the scene. We also show that the visual fidelity can be increased with the proposed method.

Neural rendering. Recent work [236] demonstrated that 2D image formation could be achieved given a camera pose and light position in a 3D scene using differentiable convolutional networks. The key enabler here is the formulation of the discrete rasterization problem [237]. With a differentiable rendering framework, a neural network can be trained with backpropagation. There is additional work [238, 239, 240] focusing on the different aspects of differentiable rendering formulations and approximations. Neural rendering is a promising technique. However, this approach still requires detailed 3D models and is incapable of generating texture information, which reduces the visual fidelity of the output. In comparison, we propose to use generative models for reducing 3D model and texture complexity.

Generative adversarial image synthesis. Generative adversarial image synthesis omits rasterization and rendering. Physical phenomena such as lighting and reflectivity are completely ignored by GAN based neural image synthesizers [216, 217, 218, 219, 220, 221, 222]. Instead, the photorealism is achieved by training the GAN with real-world data. In other words, the network learns to generate photorealistic images by capturing a latent probability distribution underlying real-world datasets. This approach has one major drawback: there is no constraint on the semantic layout of the generated 2D image. Hence, no association with 3D scenery can be constructed. As such, this methodology cannot be applied for our image formation purposes.

Conditional generative adversarial image synthesis. On the other hand, conditional

GANs [223, 224, 225, 226, 228, 227], [241] have been effectively used for image synthesis while retaining a semantic constraint. Typically, this constraint is a pixel-wise semantic segmentation mask, but other modalities such as text [242] have also been used. One limiting factor for Conditional GAN (cGAN) is the paired data requirement. The dataset must contain semantic segmentation masks and the corresponding real-world images. Building such paired datasets is labor-intensive because every real-world image needs a corresponding semantic segmentation label assigned by a human annotator.

Cycle-consistency and domain adaptation. Cycle consistent GANs and unsupervised domain adaptation techniques make the paired dataset requirement uneccessary [243, 234, 244, 245, 246, 247, 244]. These works have illustrated that high fidelity image synthesis can also be achieved with unpaired data. Cycle-consistency is very promising and has a huge application range. For example, CyCADA [234] can translate an existing game-engine generated image into a photorealistic image.

The aforementioned GAN-based image synthesis techniques have not been integrated into driving simulation pipelines until now. This contribution makes our proposed method novel. We propose to use simple 3D models radiant with unique class color-codes without textures to form a 2D semantic image. This image is analogous to a 2D semantic segmentation mask. Then, a state-of-the-art GAN-based image synthesizer trained on real-world datasets is used to generate RGB imagery. We tried both cGAN and Cy-GAN variants. Additionally, we render certain important objects of interest, such as cars in an urban scene, with Unreal Engine 4. Images obtained by blending the partial-render foreground and GAN background are more realistic. Blended images also retain the semantic layout of the scene better.

8.3 Preliminaries

Generative Adversarial Networks (GAN) [215] use a generator G and a discriminator D in a simultaneous adversarial training strategy. The goal of G is to generate data $\hat{x}$ that is indistinguishable from the real data $x \in X$. During training, G captures the probability distribution p_{data} which should closely match the distribution underlying the real data. This is achieved by training a generative mapping function $G(z)$ that maps an a priori noise distribution $p_z(z)$ to the data domain X. While G tries to generate the most realistic $\hat{x}$, the discriminator D tries to discriminate fake data $\hat{x}$ from real data x. The output of $D(x)$ is the probability that x is real. $G(z)$ and $D(x)$, both of which are neural networks, are trained simultaneously with the following min max function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\left[\log D(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}\left[\log(1 - D(G(\boldsymbol{z})))\right]. \tag{8.1}$$

8.4 Method

8.4.1 Problem formulation

We define a virtual 3D driving scene S with a 6-tuple $(O_1, O_2, P_1, P_2, T_2, \mathbf{x})$. Where $O_1 = (\mathbf{o}_{1,1}, \mathbf{o}_{1,2}, \cdots, \mathbf{o}_{1,n})$ is a list of object pose vectors, $\mathbf{o} \in \mathbb{R}^6$, and $P_1 = (M_1, M_2, \cdots, M_n)$ is the list of corresponding simple object meshes. We assume P_1 is radiant with unique class color-codes. O_2 is a sublist of O_1 for certain objects of interest, and it has a corresponding list of more complicated object meshes P_2. P_2 is not radiant. T_2 is a list of texture maps that corresponds to P_2. $\mathbf{x} \in \mathbb{R}^6$

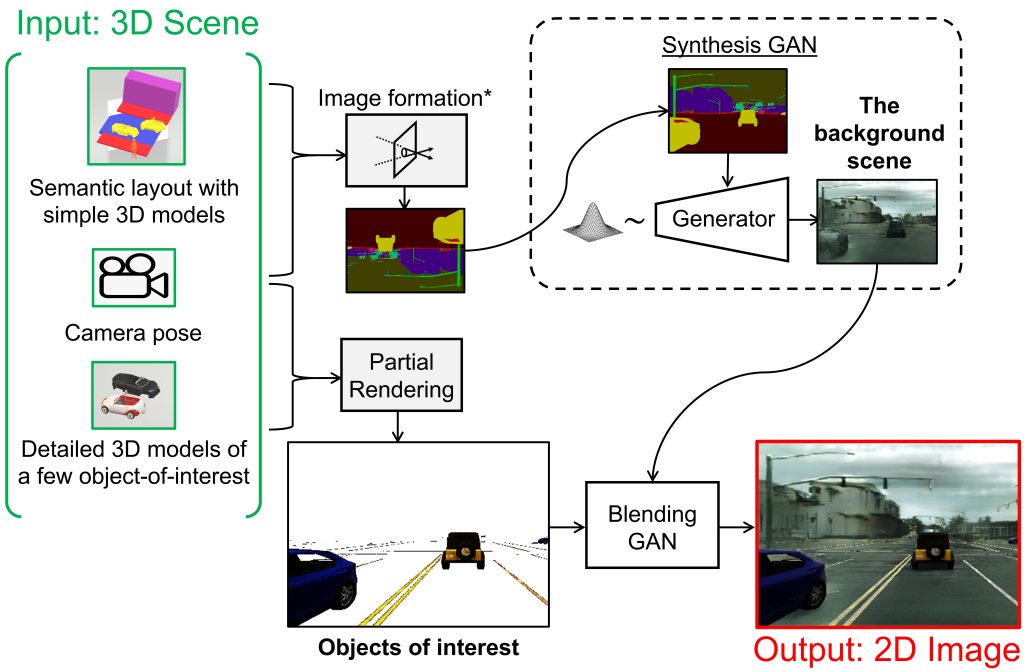


Figure 8.3: Overview of the proposed method. We introduce a novel neural graphics pipeline to form 2D image representations from virtual 3D scenes. Most of the scene is generated with very simple 3D models without texture except for a few partially rendered objects of interest. We then blend the cGAN synthesized image with a physics-based partial render for increasing visual fidelity *and* to maintain full control over the appearance of objects of interest.

is the pose vector of a virtual camera. It should be noted that a corresponding T_1 to O_1 does not exist.

We follow the formal definition of a triangular mesh given in [248]. $M := (V, Q)$ is a triangular mesh defined with faces $Q \subseteq \{1, \cdots, |V|\}^3$ and vertices $V \subseteq \mathbb{R}^3$, where $q = (q_1, q_2, q_3) \in Q$ is a triangular face with corresponding vertices v_{q_1}, v_{q_2}, and v_{q_3}. $E(Q)$, the edges between the vertices, are defined by the faces implicitly.

Problem 1. Given S, we are interested in finding a mapping function $U : \mathbf{x} \rightarrow \mathbb{R}^{H \times W \times 3}$ that will convert the camera pose vector $\mathbf{x}$ to a photo-realistic RGB image with height H and width W.

The overview of our solution, Hybrid Generative Neural Graphics (HGNG) is shown in Figure 8.3 and Algorithm 3, and the formal description follows.

8.4.2 Semantic Image formation

A semantic image formation function h can be obtained with O_1, P_1 and a pinhole camera model. Let $\mathbf{m} \in \mathbb{M}^{H \times W}$ be a pixel-wise semantic image whose entries correspond to the semantic classes of the scene. Then $h : \mathbf{x} \rightarrow \mathbb{M}^{H \times W}$ maps $\mathbf{x}$ to an integer subspace ($\mathbb{M} \subset \mathbb{Z}$) using the pinhole camera model [249] given by:

Algorithm 3: HGNG$(O_1, O_2, P_1, P_2, T_2, \mathbf{x})$

Input:
O_1, the list of object pose vectors
P_1, the list of simple object meshes w/o texture.
O_2, a sublist of O_1, corresponds to objects of interest
P_2, the list of complex object meshes.
T_2, the list of texture maps that corresponds to P_2, O_2.
$\mathbf{x}$, the pose vector of the pinhole camera
Output:
$\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$, 2D RGB image.
Main algorithm:
$\mathbf{m} = h_{\text{pinhole}}(O_1, P_1, \mathbf{x})$;
$\mathbf{I}_{\text{background}} = f_{\text{generator}}(\mathbf{m}, z \sim N)$;
foreach $i(1, 2...n)$ **do**
  $\mathbf{I}_i^{\text{object-of-interest}} = L_{\text{rendering}}(O_2(i), P_2(i), T_2(i))$;
end
$\mathbf{I}_{\text{foreground}} = \sum_i^n \mathbf{I}_i^{\text{object-of-interest}}$;
$\mathbf{I} = b_{\text{generator-blending}}(\mathbf{I}_{\text{background}}, \mathbf{I}_{\text{foreground}})$;

$$\begin{pmatrix} m_1 \\ m_2 \end{pmatrix} = -\frac{d}{p_3} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} \tag{8.2}$$

where (p_1, p_2, p_3) are the 3D coordinates of point $\mathbf{p}$ in $\mathbb{R}^3$, (m_1, m_2) are the corresponding pixel coordinates in $\mathbf{m}$, and d is the distance between the focal point and image formation plane. $\mathbf{m}$ is an upside-down image as shown in Figure 8.3. $\mathbf{m}$ is rotated 180° for the next step. For simplicity, we use the same notation $\mathbf{m}$ for the rotated image in the remainder of the chapter.

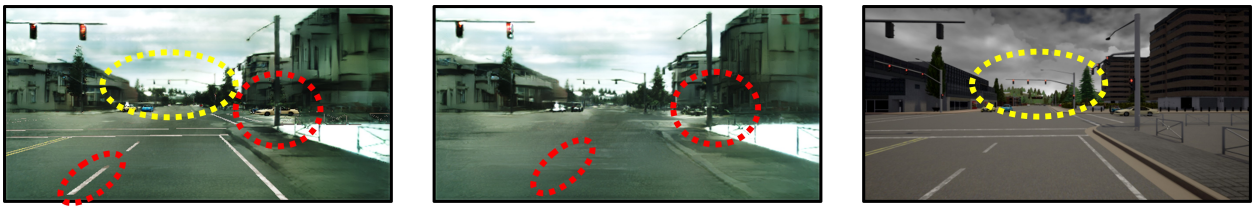
Then, the problem narrows down to finding $f : \mathbf{m} \rightarrow \mathbb{R}^{H \times W \times 3}$. This is the exact same goal as the well-studied [223, 224, 225, 226] conditional image synthesis problem.

8.4.3 Generative Adversarial Image Synthesis with cGANs and Cy-GANs

We propose to use the generator networks of cGANs or Cy-GANs to map $G : \mathbf{m} \rightarrow \mathbb{R}^{H \times W \times 3}$. Training is to be done on a real-world paired dataset of RGB images and pixel-wise semantic masks for cGAN. On the other hand, Cy-GANs can be trained with an unpaired dataset.

cGAN [229] extends the original GAN and can generate realistic fake data while retaining a conditional constraint. This is achieved by pairing the conditional constraint y with the data x and creating a new paired dataset (x, y). This pair can be an RGB image and pixel-wise semantic layout pair, or an image and text pair. x and y do not have to share the same modality. Details of cGAN can be found in [229]. cGAN can successfully generate photo-realistic fake data with a conditional constraint. However, the paired dataset requirement increases the cost of this approach.

In comparison, building an unpaired X and Y is relatively easy. Cycle GAN (Cy-GAN) [246] enables photo-realistic image synthesis with unpaired data. In summary, Cy-GAN contains two generators, $G(x)$ and $F(y)$, which map $X \rightarrow Y$ and $Y \rightarrow X$ respectively. Also, two discriminators, D_X and D_Y, try to distinguish fake data from real data. The adversarial losses are similar to the original GAN; the addition is the novel cycle consistency loss. This loss prevents the mappings of



(a) Proposed cGAN-partial render blend

(b) Only cGAN [19]

(c) Conventional full render [23]

Figure 8.4: The proposed framework (a) converts the semantic layout of the scene into a photorealistic image by blending partially rendered foreground objects with a GAN generated background. The conventional rendering engine (CARLA) (c) requires detailed models and texture information while outputting unrealistic background trees and vegetation (shown with a yellow circle). On the other hand, using only the SPADE cGAN (b) approach leads to poor car shapes and omitting road markings (shown with a red circle), while removing the need for texturing and rendering calculations. The proposed method (a) has the best of both worlds.

G and F from diverging from each other. The key idea of cycle GAN is the use of two generators to create a cycle. First, $G(x)$ generates fake $\hat{y}$, then $F(G(x))$ translates the fake $\hat{y}$ back to $\hat{x}$. If the cycle is consistent, then $x \approx \hat{x}$.

The baseline cGAN employed in this study is a SPatially-Adaptive-(DE)-normalization (SPADE) [225] network, which is a state-of-the-art cGAN based image synthesizer. SPADE outperforms other image-to-image synthesizers by retaining semantic information against conventional normalization operations [225]. This is achieved through the following de-normalization operation where the activation value at layer i is given by:

$$\gamma^i_{c,y,x}(\mathbf{m})\frac{h^i_{n,c,y,x} - \mu^i_c}{\sigma^i_c} + \beta^i_{c,y,x}(\mathbf{m}) \tag{8.3}$$

where $h^i_{n,c,y,x}$ is the activation before normalization and μ^i_c and σ^i_c are the mean and standard deviation in channel c. $\gamma^i_{c,y,x}(\mathbf{m})$ and $\beta^i_{c,y,x}(\mathbf{m})$ are learned variables that modulates the normalization process. We refer the readers of the original SPADE paper [225] for more details.

We use a SPADE network pre-trained on the Cityscapes dataset [19] as the mapping function f_s and obtain the synthesized image with it as $\mathbf{I} = f_s(\mathbf{m})$.

8.4.4 Partial rendering

To increase visual fidelity and have full control over certain objects of interest, we propose using physics-based rendering to obtain partially-rendered images $\mathbf{I}_r$. Besides O_2, P_2, T_2 and $\mathbf{x}$, a light source is also needed for rendering. Here we assume that the properties and location of the light source are fixed and known relative to $\mathbf{x}$. Then the rendering equation [214] can be used to render objects of interest.

$$\begin{aligned} L_0(\mathbf{p}, \omega, \lambda, t) = L_e(\mathbf{p}, \omega_0, \lambda, t) + \\ \iint_{\Omega} f_r(\mathbf{p}, \omega_i, \omega_0\lambda, t) L_i(\mathbf{p}, \omega_i, \lambda, t)(\omega_i.\mathbf{n}) d\omega_i \end{aligned} \tag{8.4}$$

where $L_0(\mathbf{p}, \omega, \lambda, t)$ is the total spectral radiance, λ is wavelength, ω_0 is the outgoing light direction, ω_i is the incoming light direction, t is time and $\mathbf{p}$ is a point in 3D space. $L_e(\mathbf{p}, \omega_0, \lambda, t)$ is the emitted spectral radiance, Ω is a unit hemisphere with the surface normal center $\mathbf{n}$ of $\mathbf{p}$ and it contains all values for ω_i, $f_r(\mathbf{p}, \omega_i, \omega_0\lambda, t)$ is the bidirectional reflectance function and finally $L_i(\mathbf{p}, \omega_i, \lambda, t)$ is the spectral radiance of the incoming wavelength.

With Equation 8.4, the spectral radiance of each 3D point on a few objects of interest is obtained. Then, the partially rendered image $\mathbf{I}_r$ is formed with the same pinhole camera model introduced in Equation 8.2.

8.4.5 Blending

Here we propose to blend the synthesized image $\mathbf{I}$ with the partially rendered image $\mathbf{I}_r$ to obtain a hybrid image $\mathbf{I}_h$ as shown in Figure 8.3. The hybrid image is defined as:

$$\mathbf{I}_h := b(\mathbf{I}, \mathbf{I}_r) \tag{8.5}$$

where the blending function $b : (\mathbf{I}, \mathbf{I}_r) \rightarrow \mathbb{R}^{HxW\times 3}$ maps the synthesized and partially rendered images to a new hybrid RGB image. We compared three different blending functions b in this study.

Alpha blending. Taking $\mathbf{I}$ as the background image and $\mathbf{I}_r$ as the foreground image, the alpha blended image $\mathbf{I}_h$ can be obtained with:

$$\mathbf{I}_h = \alpha\mathbf{I} + (1 - \alpha)\mathbf{I}_r. \tag{8.6}$$

Pyramid blending. With the gaussian pyramid mask G_R [250], L_a the laplacian pyramid of the foreground $\mathbf{I}_r$, and L_b the laplacian pyramid of background $\mathbf{I}$, the laplacian blended pixel $b(i, j)$ can be obtained with:

$$b(i, j) = G_R(i, j)L_a(i, j) + (1 - G_R(i, j))L_b(i, j). \tag{8.7}$$

GAN blending. As a third blending option, we employed GP-GAN [251]. The generator of GP-GAN converts a naive copy-paste blended image to a realistic well-blended image. Besides conditional GAN loss, GP-GAN employs an auxiliary l_2 loss to sharpen the image. The overall combined loss function is given by:

$$\mathcal{L}(x, x_g) = \lambda\mathcal{L}_{l_2}(x, x_g) + (1 - \lambda)\mathcal{L}_{\text{adv}}(x, x_g) \tag{8.8}$$

where $\mathcal{L}(x, x_g)$ is the final loss, $\mathcal{L}_{l_2}$ is the l_2 loss and $\mathcal{L}_{\text{adv}}$ is the adverserial loss. λ is a hyperparameter and set to 0.999.

8.5 Experiments

8.5.1 Implementation details

We used SPADE [225] as our cGAN image synthesizer to convert the semantic layout of the scene to a photorealistic background image. The network was trained on Cityscapes [19], an urban driving dataset with paired semantic mask and image data. CARLA [39], an open-source driving simulator

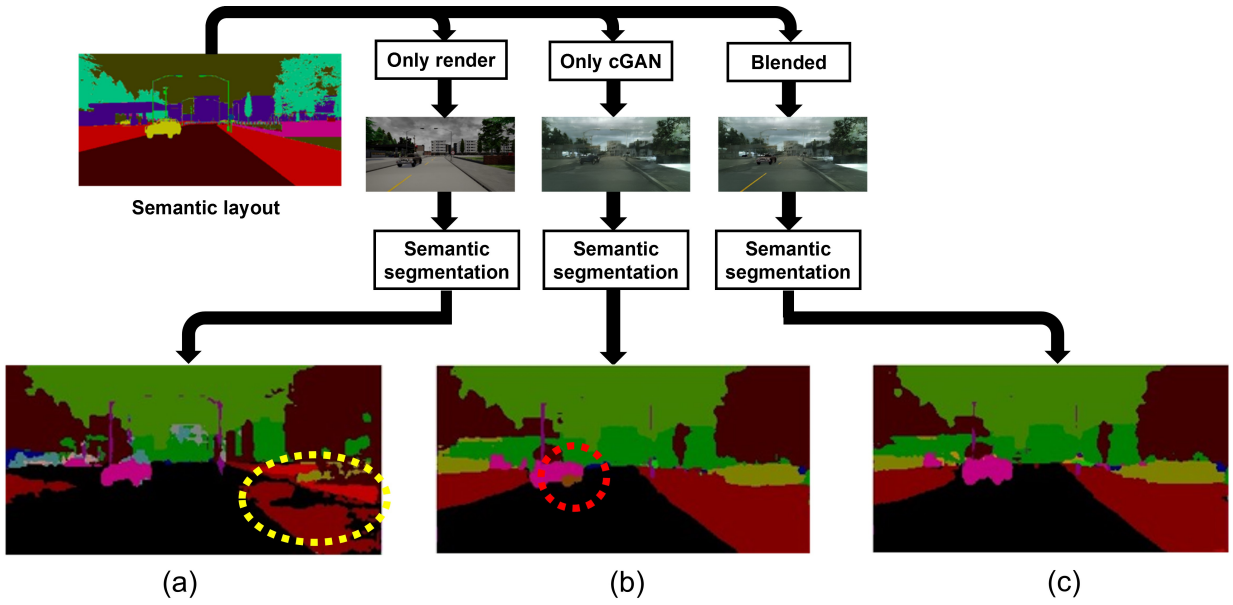


Figure 8.5: An illustration of semantic retention analysis. The semantic segmentation result should stay true to the initial semantic layout. (a) Full-render yields unrealistic shadows. On the bottom right-hand side of the left-most image (shown with a yellow circle), shadows of trees cast on the sidewalk were misclassified as a road by DeepLabV3. (b) cGAN generated vehicles do not retain their shapes perfectly (middle image, shown with a red circle). (c) Blending retains the semantic relationship with the source layout (right-most image). This figure employs different color codes to distinguish the semantic layout formation and semantic segmentation processes for illustration purposes.

built upon Unreal Engine 4, was utilized to obtain the semantic layout and partially rendered images. We used the shading and lighting engine [230] of Unreal Engine 4 in our experiments. Only vehicles and lane markings were considered as objects of interest. For blending, we used a GP-GAN [251] trained on the Transient Attributes Database [252]. All computational experiments were conducted with an Nvidia RTX 2080.

8.5.2 Evaluation

Semantic retention

Figure 8.5 illustrates semantic retention analysis, a common [224, 227, 225] evaluation method for fake image synthesis. Semantic retention measures the semantic correspondence between the conditional semantic mask and the synthesized image. In summary, an external semantic segmentation network is used to segment the synthesized image. Then, the discrepancy between the conditional semantic layout (input of the synthesizer) and the semantic mask obtained from the generated image (output of the pre-trained external segmentation network) is calculated with top-1 accuracy. A good synthesizer should produce photorealistic images while retaining the initial conditional semantic layout. In other words, the initial semantic layout is accepted as the ground truth, and the image synthesizer's mask accuracy is calculated to obtain the retention score. A higher retention score is favorable.

In this study, we employed DeepLabV3 [231], a state-of-the-art semantic segmentation network,

to measure semantic retention. The network was trained on Cityscapes, an urban driving dataset [19].

FID

Frechet Inception Distance (FID) [253] is a commonly used [225, 228] performance metric for measuring visual fidelity. In summary, a deep neural network is employed to extract features of all images in a dataset. Then, the covariance and mean of features obtained from synthesized and real datasets are compared to generate a score. We do not have any real-data corresponding to our virtual 3D scene, but FID can still be used with unpaired data. As such, three different real-world datasets [19, 254, 255] were utilized as the ground truth.

An InceptionV3 [256] model that was trained on ImageNet [257] was employed as the feature extractor. After features were extracted from the synthesized images and from real-world images from Cityscapes [19], KITTI [254], and ADE20K [255], the FID is calculated as follows:

$$d^2 = ||\mu_1 - \mu_2||^2 + Tr(C_1 + C_2 - 2\sqrt{(C_1 C_2)}) \tag{8.9}$$

where μ_1, μ_2 are the means of features, and C_1, C_2 are the covariances obtained from datasets 1 and 2 respectively, where the first dataset consists of real images and the second synthesized images. The smaller the distance d^2, the more similar are the two datasets. In other words, a small FID indicates that fake data is similar to real-world data.

The synthesized images were then compared against each other using FID scores as shown in Table 8.2. μ_1 and C_1 were obtained from the real datasets and do not change in a column, whereas μ_2 and C_2 were obtained from synthesized images and vary with each row. A lower FID indicates high visual fidelity.

Inception score

Inception Score (IS) was initially proposed to evaluate the generator performance of GANs [258]. In summary, a pre-trained image classifier is run over a GAN generated fake dataset. The distribution of predicted classes, along with the confidence intervals, were then compared against a real dataset. A higher Inception Score indicates higher image quality and diversity. IS differs from FID by its use of actual classification results, whereas FID utilizes latent features. Details of IS can be found in [258].

Comparisions and ablations

Ablation studies were conducted to demonstrate the effect of each component of the proposed method. The ablation list is:

1. No partial-render (only vanilla cGAN or Cy-GAN)
2. No cGAN or Cy-GAN (only full render)
3. Alpha blend (cGAN or Cy-GAN + partial render)
4. Pyramid blend (cGAN or Cy-GAN + partial render)
5. GAN blend (cGAN or Cy-GAN + partial render)

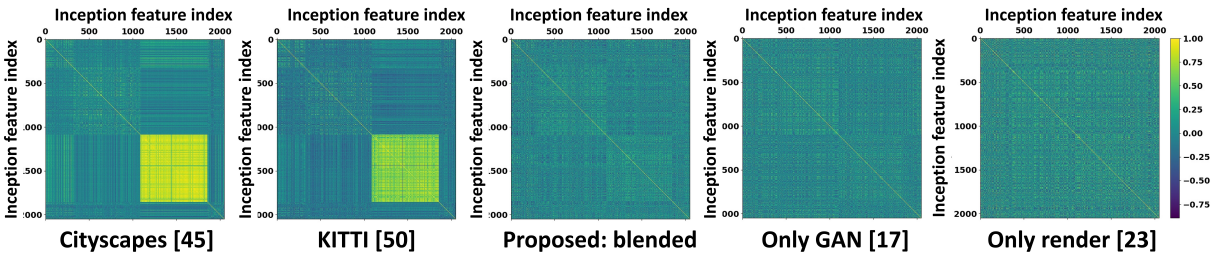


Figure 8.6: InceptionV3 feature vector correlation matrices of real and synthetic data. The synthetic dataset that was generated with the proposed blending approach shows a similar correlation pattern with real data. This pattern does not emerge with the only render or only GAN methods.

Where SPADE [225] was used as the vanilla cGAN and the original Cycle-GAN [223] was employed as the vanilla Cy-GAN variant.

We used CARLA [39] to obtain fully rendered images of urban scenery. The semantic layout of the scene was also imported from CARLA and used as the conditional input for the generative adversarial image synthesizers. Only vehicles and lane markings were considered by the partial-renders. In this work, the image synthesis was done frame-by-frame with a fixed random seed.

8.5.3 Results

Qualitative results

The qualitative results are shown in Figures 8.4, 8.5, and 8.6. These figures illustrate fully rendered, blended, and only cGAN images. As can be seen in Figure 8.5, rendered shadows are unrealistic, while only cGAN generated vehicles cannot retain their shapes. These results underline the importance of partial rendering of objects of interest such as cars, vans, and lane markings. The hybrid approach combines the accuracy of a full-render with the realism of a generative model.

Treating foreground objects differently from background scenery with the proposed blending technique improves the photorealism of the final image as can be seen in Figures 8.4, 8.5, 8.6. The appearance of foreground objects, such as vehicles, needs to be controllable and rendered in detail. This necessitates employing conventional rendering techniques with high-detail models and textures. However, background scenery's cannot be controlled at the same level of detail because they contain a higher number of elements such as mountains, trees, and buildings. In practice, conventional driving simulators pay less attention to these background elements by lowering 3D model quality and using less rendering focus. This causes lower overall visual fidelity. In contrast, GAN-based image synthesizers can automate background scene generation while achieving higher visual fidelity by learning the background compositions of real-world data. Our use of a GAN-based image synthesizer completely removes the texture and detailed model requirements of background scenery generation while increasing visual fidelity. In Figure 8.4, the conventional rendering method produced unrealistic trees and vegetation (shown with a yellow circle) around the vanishing point of the image. At the same time, the proposed method and the only-cGAN approach generated a more blended background scene with vegetation at the same spot.

On the other hand, only using a GAN-based image synthesizer reduces control over the appearance of objects of interest, such as cars. This causes lower visual fidelity. In Figure 8.4, the only GAN-based approach failed to generate road-markings. In addition, the surface quality of cars (shown with a red circle) was much lower than the full-render approach and the proposed method.

Figure 8.5 demonstrates the unrealistic shadows of full-render and incomplete vehicle shapes of the only GAN approach with a yellow and red circle. The proposed method alleviates these issues. These results indicate a qualitative validation of our hypothesis: the proposed approach, blending GAN-based synthesizers with conventional rendering, has the best of both worlds.

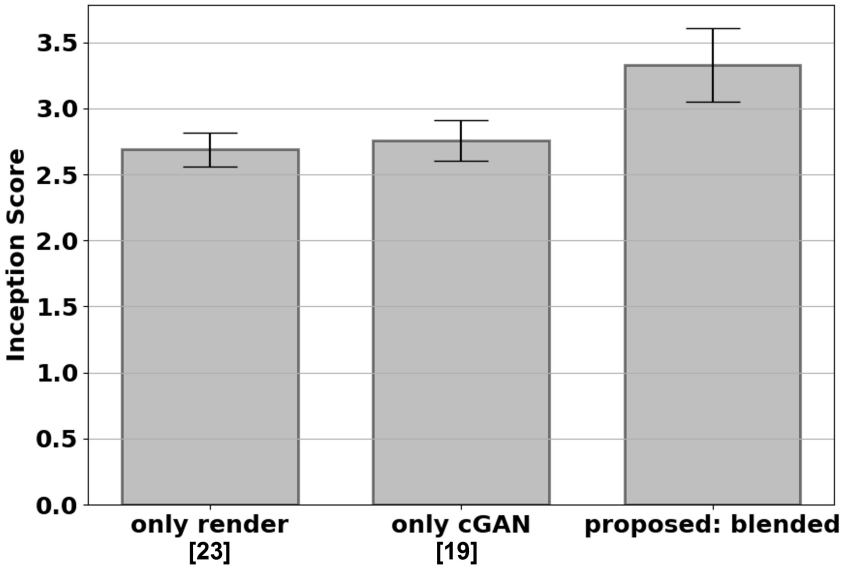


Figure 8.7: Inception score results. A high Inception score indicates better image quality and higher diversity.

Quantitative results

Figure 8.6 shows Inception V3 feature vector correlations. Even though real-world images of the Cityscapes and KITTI datasets are entirely different, they have similar latent feature correlations. However, this pattern does not emerge with a synthetic dataset of low visual fidelity. The proposed blended synthetic dataset has, albeit being weak, a similar correlation pattern. In comparison, the same pattern does not emerge with the conventional render or pure GAN approaches. This shows that the proposed blending approach is a good strategy for the realistic representation of driving scenes.

IS (Inception Score [258]), FID, and semantic retention scores are given in Table 8.1, Table 8.2 and Figure 8.7. These results indicate that the proposed hybrid blending approach consistently outperforms conventional rendering and pure generative adversarial image synthesis. Using detailed models with a conventional rendering engine for objects-of-interest produces high-quality visuals. However, building the rest of the driving scene with this level of detail is extremely challenging. Our hybrid method mitigates this problem by replacing the background elements with a GAN synthesizer and blending high-quality objects of interest renders into the scene. This blending strategy is the main reason for achieving higher visual fidelity.

The Cityscapes dataset contains only urban driving scenes, while ADE20K also has miscellaneous scenes. All of our virtual 3D scenes were in an urban environment. As such, most of the methods received better FID scores for the Cityscapes dataset, as can be seen in Table 8.2.

GAN blend and Alpha blend showed similar performances as shown in Table 8.1 and Table 8.2. However, it should be noted that the blending GAN was not trained on an urban driving dataset.

Table 8.1: Semantic retention performance- higher scores are better. Our methods outperform the physics-based rendering approach.

Method	Semantic retention ↑
Baseline: only render [39]	0.819
only Cycle GAN [223]	0.343
Proposed	
cy-GAN alpha blend	0.362
cy-GAN pyramid blend	0.353
cy-GAN GAN blend	0.318
cGAN alpha blend	0.879
cGAN pyramid blend	0.868
cGAN GAN blend	0.846

Table 8.2: FID performance- lower scores are better. Our methods outperform the physics-based computer graphics pipeline. Cy-R stands for CyGAN-Render blend, and c-R stands for cGAN-Render blend.

Method	FID ↓		
	Cityscapes[19]	KITTI[254]	ADE20K[255]
Only render [39]	231.768	285.222	361.496
Proposed			
Cy-R alpha blend	175.832	220.223	272.069
Cy-R pyramid blend	196.911	228.277	279.704
Cy-R GAN blend	**194.191**	234.087	266.615
c-R alpha blend	188.809	220.161	272.877
c-R pyramid blend	202.120	**214.488**	265.603
c-R GAN blend	194.898	217.663	**260.404**

The blending performance can possibly be increased with a better blending dataset for training the blending GAN.

The cGAN variants performed better on average as expected, as shown in Table 8.1 and 8.2. The synthesized images were both realistic and loyal to the initial semantic layout. However, cGAN requires a paired dataset for training. The full render is better at semantic retention than Cy-GAN variants, but Cy-GAN variants have a higher FID score than rendering. This means that Cy-GAN can generate realistic images but fails to retain the semantic constraints.

8.6 Conclusions

This work introduced and investigated the feasibility of Hybrid Generative Neural Graphics (HGNG). The proposed approach utilizes a GAN-based image synthesizer to remove the need for rendering calculations and labor-intensive texture-making steps for background elements while increasing photorealism. In addition, our method achieves full control over the appearance of objects of interest using partial-rendering. Our novel image formation strategy blends the GAN-generated background image with these partial renders and outperforms conventional approaches.

Experimental results indicate that conventional generation of driving simulation graphics now has a strong alternative.

In order to train the cGAN-based synthesizers, real-world urban images and their semantic labels, i.e., a paired dataset, are needed. Therefore, with the publication of more paired real-world datasets, the performance of the proposed method can be further increased. On the other hand, CyGANs remove this paired dataset requirement with the use of cycle consistency, but cyGANs do not perform as well as cGANs. As such, without a paired dataset, the proposed system cannot outperform the conventional pipelines yet. However, potential future developments in domain adaptation and cycle consistency can greatly benefit HGNG and may remove the paired dataset requirement in the future.

This work focused on frame-by-frame image formation with GANs. However, computer graphics applications such as driving simulations may require more temporally consistent approaches. Each subsequent frame of a driving simulation needs to be consistent with the overall sequence. The proposed method already achieves temporal consistency for objects of interest using partial rendering. The temporal consistency of the GAN-generated background scene can potentially be increased with larger urban video datasets. To this end, future work can focus on creating better urban video datasets and developing GAN-based video-to-video synthesis methods.

Chapter 9

Conclusion

In this report, we have detailed a number of potential applications of image processing, including both traditional and neural network and deep learning image processing technologies, to automated vehicle sensing and control, traffic scene analysis, pedestrian detection and intention estimation, and improved image synthesis for automated vehicle simulation and testing. Seven specific problems were defined, studied through a survey of relevant literature and current state of the art practices, and addressed. For each the technological and methodological approaches explored, the implementation process, and the analysis and validation results were presented.

For the first application, integrated deep reinforcement learning with model-based path planners for automated driving, a novel hybrid approach for integrating path planning into model-free DRL frame-works was proposed. A proof-of-concept implementation and experiments in a virtual environment showed that the proposed method is capable of learning to drive. This work is part of a larger effort to investigate and improve the safety, robustness, and explainability of end-to-end automated driving systems.

For the second application, optical flow based potential field for autonomous driving, we demonstrated that it is possible to obtain a visual potential field from the optical flow information from a monocular camera. The novelty of this work consists on the formulation of the potential field for both the obstacles and the road boundaries and applying it to control a vehicle. This visual based navigation method is less computationally expensive than learning based techniques, but at the same time, it allows to capture the features of dynamically changing environment. For this reason, it can serve as a baseline for comparison with both classical and learning based approaches.

For the third application, automated traffic surveillance using existing cameras on transit buses, we developed and evaluated a fully automatic vision-based method for traffic count and flow estimation by counting and tracking vehicles captured in video imagery from cameras mounted on buses for the purpose of estimating traffic flows on roadway segments using a previously developed moving observer methodology. The proposed method was implemented and tested using imagery from in-service transit buses, and its feasibility and accuracy was shown through experimental validation.

For the fourth application, a vision-based social distancing and critical density detection system for COVID-19, we proposed and implemented an AI and monocular-camera-based real-time system to detect and monitor social distancing, the spacing between individuals, and crowd densities. In addition, our system utilized the proposed critical social density value to avoid overcrowding by modulating inflow to the area of interest. The proposed approach was demonstrated using three different pedestrian crowd datasets. Quantitative validation was conducted over the Oxford Town Center Dataset that provides ground truth pedestrian detections.

For the fifth application, faraway-frustum: dealing with lidar sparsity for 3D object detection using fusion, we proposed an alternative 3D bird's-eye-view pedestrian detector, named Faraway-Frustum, to deal with lidar sparsity of faraway objects. Our method takes advantage of relatively dense image data to find faraway objects and circumvents the disadvantages of pointcloud-driven neural networks working on very sparse points. Moreover, our alternative detector can be flexibly combined with a state-of-the-art method to form an overall 3D BEV object detection system via setting faraway thresholds. The experiments demonstrated the feasibility of our approach, but they also exposed a significant shortcoming of state-of-the-art object detection methods: relying on learned representations of very sparse lidar points to detect faraway objects is not a good strategy.

For the sixth application, predicting pedestrian crossing intention with feature fusion and spatio-temporal attention, we proposed a novel method for improving vision-based pedestrian crossing intention prediction that explicitly considers the global context as a channel representing the interaction between the target pedestrian and the whole scene. We also proposed a hybrid fusion strategy for different features using 2D CNNs, RNNs, and attention mechanisms. Experiments on the JAAD and PIE datasets show that the proposed method achieves the state-of-the-art results against baseline methods in the pedestrian action prediction benchmark.

And finally, for the seventh application, photo-realism in driving simulations: blending generative adversarial image synthesis with rendering, we introduced and investigated the feasibility of producing more realistic imagery in driving simulators using Hybrid Generative Neural Graphics. The proposed approach utilizes a GAN-based image synthesizer to remove the need for rendering calculations and labor-intensive texture-making steps for background elements while increasing photo-realism. In addition, our method achieves full control over the appearance of objects of interest using partial-rendering. Our novel image formation strategy blends the GAN-generated background image with these partial renders and outperforms conventional approaches. Experimental results indicate that conventional generation of driving simulation graphics now has a strong alternative.

9.1 Research Products from this Project

9.1.1 Available Source Code

The source code from the work described in Chapter 2, Integrating Deep Reinforcement Learning with Model-based Path Planners for Automated Driving, is open-source from theand available online at: `https://github.com/Ekim-Yurtsever/Hybrid-DeepRL-Automated-Driving`.

The implementation and experimental data from the work described in Chapter 5, A Vision based Social Distancing and Critical Density Detection System for COVID-19, is open-sourced and available at: `https://github.com/dongfang-steven-yang/social-distancing-monitoring`.

The source code from the work described in Chapter 6, Faraway-Frustum: Dealing with Lidar Sparsity for 3D Object Detection using Fusion with Vision, is open-source and publicly available at: `https://github.com/dongfang-steven-yang/faraway-frustum`.

The source code from the work described in Chapter 7, Predicting Pedestrian Crossing Intention With Feature Fusion and Spatio-Temporal Attention, is open-source and publicly available: `https://github.com/OSU-Haolin/Pedestrian_Crossing_Intention_Prediction`.

9.1.2 Available Datasets

The original video data used in the work described in Chapter 4, Automated Traffic Surveillance using Existing Cameras on Transit Buses, as well as the manually extracted ground truth records are available in the Zenodo repository at DOI 10.5281/zenodo.7955464.

Bibliography

[1] Ekim Yurtsever, Linda Capito, Keith Redmill, and Ümit Özgüner. Integrating deep reinforcement learning with model-based path planners for automated driving. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1311–1316, 2020.

[2] Linda Capito, Ümit Özgüner, and Keith Redmill. Optical flow based visual potential field for autonomous driving. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 885–891, 2020.

[3] Keith A. Redmill, Ekim Yurtsever, Rabi G. Mishalani, Benjamin Coifman, and Mark R. McCord. Automated traffic surveillance using existing cameras on transit buses. *Sensors*, 23(11), 2023.

[4] Dongfang Yang, Ekim Yurtsever, Vishnu Renganathan, Keith A. Redmill, and Ümit Özgüner. A vision-based social distancing and critical density detection system for covid-19. *Sensors*, 21(13), 2021.

[5] Haolin Zhang, Dongfang Yang, Ekim Yurtsever, Keith A. Redmill, and Ümit Özgüner. Faraway-frustum: Dealing with lidar sparsity for 3d object detection using fusion. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 2646–2652, 2021.

[6] Dongfang Yang, Haolin Zhang, Ekim Yurtsever, Keith A. Redmill, and Ümit Özgüner. Predicting pedestrian crossing intention with feature fusion and spatio-temporal attention. *IEEE Transactions on Intelligent Vehicles*, 7(2):221–230, 2022.

[7] Ekim Yurtsever, Dongfang Yang, Ibrahim Mert Koc, and Keith A. Redmill. Photorealism in driving simulations: Blending generative adversarial image synthesis with rendering. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):23114–23123, 2022.

[8] Puneet Kohli and Anjali Chadha. Enabling pedestrian safety using computer vision techniques: A case study of the 2018 uber inc. self-driving car crash. In *Future of Information and Communication Conference*, pages 261–279. Springer, 2019.

[9] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020.

[10] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[11] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 163–168. IEEE, 2011.

[12] Junqing Wei, Jarrod M Snider, Junsung Kim, John M Dolan, Raj Rajkumar, and Bakhtiar Litkouhi. Towards a viable autonomous driving research platform. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 763–770. IEEE, 2013.

[13] Alberto Broggi, Michele Buzzoni, Stefano Debattisti, Paolo Grisleri, Maria Chiara Laghi, Paolo Medici, and Pietro Versari. Extensive tests of autonomous driving technologies. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1403–1415, 2013.

[14] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

[15] Naoki Akai, Luis Yoichi Morales, Takuma Yamaguchi, Eijiro Takeuchi, Yuki Yoshihara, Hiroyuki Okuda, Tatsuya Suzuki, and Yoshiki Ninomiya. Autonomous driving based on accurate localization using multilayer lidar and dead reckoning. In *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2017.

[16] Erico Guizzo. How google's self-driving car works. *IEEE Spectrum Online*, 18(7):1132–1141, 2011.

[17] Julius Ziegler, Philipp Bender, Markus Schreiber, Henning Lategahn, Tobias Strauss, Christoph Stiller, Thao Dang, Uwe Franke, Nils Appenrodt, Christoph G Keller, et al. Making Bertha drive – an autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*, 6(2):8–20, 2014.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[19] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[20] Ekim Yurtsever, Yongkang Liu, Jacob Lambert, Chiyomi Miyajima, Eijiro Takeuchi, Kazuya Takeda, and John HL Hansen. Risky action recognition in lane change video clips using deep spatiotemporal networks with segmentation mask transfer. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3100–3107. IEEE, 2019.

[21] Matthew McNaughton, Chris Urmson, John M Dolan, and Jin-Woo Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *2011 IEEE International Conference on Robotics and Automation*, pages 4889–4895. IEEE, 2011.

[22] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[23] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[24] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006.

[25] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[26] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. *arXiv preprint*, 2017.

[27] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[28] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

[29] Shumeet Baluja. Evolution of an artificial neural network based autonomous land vehicle controller. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 26(3):450–463, 1996.

[30] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068. ACM, 2013.

[31] Konstantinos Makantasis, Maria Kontorinaki, and Ioannis Nikolos. A deep reinforcement learning driving policy for autonomous road vehicles. *arXiv preprint arXiv:1905.09046*, 2019.

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[33] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[34] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[35] Carl-Johan Hoel, Katherine Driggs-Campbell, Krister Wolff, Leo Laine, and Mykel Kochenderfer. Combining planning and deep reinforcement learning in tactical decision making for autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2019.

[36] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[37] Blazej Osinski, Adam Jakubowski, Piotr Milos, Pawel Ziecina, Christopher Galias, and Henryk Michalewski. Simulation-based reinforcement learning for real-world autonomous driving. *arXiv preprint arXiv:1911.12905*, 2019.

[38] Sentdex. Carla-rl. `https://github.com/Sentdex/Carla-RL`, 2020.

[39] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[40] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots: A survey. *Journal of intelligent and robotic systems*, 53(3):263, 2008.

[41] Naoya Ohnishi and Atsushi Imiya. Visual navigation of mobile robot using optical flow and visual potential field. In *International Workshop on Robot Vision*, pages 412–426. Springer, 2008.

[42] Huiqi Miao and Yan Wang. Optical flow based obstacle avoidance and path planning for quadrotor flight. In *Chinese intelligent automation conference*, pages 631–638. Springer, 2017.

[43] Wesley H Huang, Brett R Fajen, Jonathan R Fink, and William H Warren. Visual navigation and obstacle avoidance using a steering potential function. *Robotics and Autonomous Systems*, 54(4):288–299, 2006.

[44] Ted Camus and Ted Camus. *Calculating time-to-contact using real-time quantized optical flow*. US Department of Commerce, National Institute of Standards and Technology, 1995.

[45] Axel Gern, Rainer Moebus, and Uwe Franke. Vision-based lane recognition under adverse weather conditions using optical flow. In *Intelligent Vehicle Symposium, 2002. IEEE*, volume 2, pages 652–657. IEEE, 2002.

[46] David Lieb, Andrew Lookingbill, and Sebastian Thrun. Adaptive road following using self-supervised learning and reverse optical flow. In *Robotics: science and systems*, pages 273–280, 2005.

[47] Ken-ichi Yoshimoto, Masatoshi Sakatoh, Makoto Takeuchi, and Hideki Ogawa. An automatic steering control algorithm using optical flow. *JSAE review*, 16(2):165–169, 1995.

[48] Yuki Okafuji, Takanori Fukao, and Hiroshi Inou. Development of automatic steering system by modeling human behavior based on optical flow. *Journal of Robotics and Mechatronics*, 27(2):136–145, 2015.

[49] Kahlouche Souhila and Achour Karim. Optical flow based robot obstacle avoidance. *International Journal of Advanced Robotic Systems*, 4(1):2, 2007.

[50] Ruijuan Chang, Rong Ding, Mengxiang Lin, Dechao Meng, Zeye Wu, and Meng Hang. An experimental evaluation of balance strategy based obstacle avoidance. In *Control, Automation, Robotics and Vision (ICARCV), 2016 14th International Conference on*, pages 1–6. IEEE, 2016.

[51] J Guldner, VI Utkin, H Hashimoto, and F Harashima. Tracking gradients of artificial potential fields with non-holonomic mobile robots. In *American Control Conference, Proceedings of the 1995*, volume 4, pages 2803–2804. IEEE, 1995.

[52] Antonella Ferrara and Matteo Rubagotti. Gradient tracking based second order sliding mode control of a wheeled vehicle. In *Control Conference (ECC), 2007 European*, pages 3810–3817. IEEE, 2007.

[53] Eloy Snapper. Model-based path planning and control for autonomous vehicles using artificial potential fields. 2018.

[54] UZA Hamid, Y Saito, H Zamzuri, and P Raksincharoensak. Collision avoidance performance analysis of a varied loads autonomous vehicle using integrated nonlinear controller. *PERINTIS eJournal*, 8(1):17—43, 2018.

[55] Pongsathorn Raksincharoensak, Takahiro Hasegawa, Akito Yamasaki, Hiroshi Mouri, and Masao Nagai. Vehicle motion planning and control for autonomous driving intelligence system based on risk potential optimization framework. In *The Dynamics of Vehicles on Roads and Tracks: Proceedings of the 24th Symposium of the International Association for Vehicle System Dynamics (IAVSD 2015), Graz, Austria, 17-21 August 2015*, page 189. CRC Press, 2016.

[56] Linda J Capito Ruiz. Optical flow-based artificial potential field generation for gradient tracking sliding mode control for autonomous vehicle navigation. Master's thesis, The Ohio State University, 2019.

[57] Jianbo Shi and Carlo Tomasi. Good features to track. Technical report, Cornell University, 1993.

[58] Jean-Yves Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10):4, 2001.

[59] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[60] Ümit Özgüner, Tankut Acarman, and Keith Redmill. *Autonomous ground vehicles*. Artech House, 2011.

[61] Vadim Utkin, Jürgen Guldner, and Jingxin Shi. *Sliding mode control in electro-mechanical systems*. CRC press, 2009.

[62] Benjamin Coifman, David Beymer, Philip McLauchlan, and Jitendra Malik. A real-time computer vision system for vehicle tracking and traffic surveillance. *Transportation Research Part C: Emerging Technologies*, 6(4):271–288, 1998.

[63] Boris A Alpatov, Pavel V Babayan, and Maksim D Ershov. Vehicle detection and counting system for real-time traffic surveillance. In *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4. IEEE, 2018.

[64] Da Li, Bodong Liang, and Weigang Zhang. Real-time moving vehicle detection, tracking, and counting system implemented with opencv. In *2014 4th IEEE International Conference on Information Science and Technology*, pages 631–634. IEEE, 2014.

[65] Kunfeng Wang, Zhenjiang Li, Qingming Yao, Wuling Huang, and Fei-Yue Wang. An automated vehicle counting system for traffic surveillance. In *2007 IEEE International Conference on Vehicular Electronics and Safety*, pages 1–6. IEEE, 2007.

[66] Santiago Felici-Castell, Miguel García-Pineda, Jaume Segura-Garcia, Rafael Fayos-Jordan, and Jesus Lopez-Ballester. Adaptive live video streaming on low-cost wireless multihop networks for road traffic surveillance in smart cities. *Future Generation Computer Systems*, 115:741–755, 2021.

[67] Jia-Ping Lin and Min-Te Sun. A yolo-based traffic counting system. In *2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 82–85. IEEE, 2018.

[68] Jianxiao Zhu, Xu Li, Peng Jin, Qimin Xu, Zhengliang Sun, and Xiang Song. Mme-yolo: multi-sensor multi-level enhanced yolo for robust vehicle detection in traffic surveillance. *Sensors*, 21(1):27, 2021.

[69] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[70] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[71] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[72] Bin Tian, Brendan Tran Morris, Ming Tang, Yuqiang Liu, Yanjie Yao, Chao Gou, Dayong Shen, and Shaohu Tang. Hierarchical and networked vehicle surveillance in its: a survey. *IEEE transactions on intelligent transportation systems*, 16(2):557–580, 2014.

[73] Benjamin Coifman, Mark McCord, Rabi G Mishalani, Michael Iswalt, and Yuxiong Ji. Roadway traffic monitoring from an unmanned aerial vehicle. In *IEE Proceedings-Intelligent Transport Systems*, volume 153, pages 11–20. IET, 2006.

[74] Ying Zhang, Daiyin Zhu, Peng Wang, Gong Zhang, and Henry Leung. Vision-based vehicle detection for videosar surveillance using low-rank plus sparse three-term decomposition. *IEEE Transactions on Vehicular Technology*, 69(5):4711–4726, 2020.

[75] Ilker Bozcan and Erdal Kayacan. Au-air: A multi-modal unmanned aerial vehicle dataset for low altitude traffic surveillance. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8504–8510. IEEE, 2020.

[76] Srishti Srivastava, Sarthak Narayan, and Sparsh Mittal. A survey of deep learning techniques for vehicle detection from uav images. *Journal of Systems Architecture*, page 102152, 2021.

[77] Keith A Redmill, Benjamin Coifman, Mark McCord, and Rabi G Mishalani. Using transit or municipal vehicles as moving observer platforms for large scale collection of traffic and transportation system information. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1089–1095. IEEE, 2011.

[78] Benjamin Coifman, Keith Redmill, Rong Yang, Rabi Mishalani, and Mark McCord. Municipal vehicles as sensor platforms to monitor roadway traffic. *Transportation Research Record*, 2644(1):48–54, 2017.

[79] Mark R McCord, Rabi G Mishalani, and Benjamin Coifman. Using municipal vehicles as sensor platforms to monitor the health and performance of the traffic control system. *Final Research Report, Mobility 21 A USDOT National University Transportation Center*, page 32, 2020.

[80] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

[81] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*, pages 3464–3468. IEEE, 2016.

[82] Indrabayu, Rizki Yusliana Bakti, Intan Sari Areni, and A Ais Prayogi. Vehicle detection and tracking using gaussian mixture model and kalman filter. In *2016 International Conference on Computational Intelligence and Cybernetics*, pages 115–119. IEEE, 2016.

[83] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[84] Xinyu Hou, Yi Wang, and Lap-Pui Chau. Vehicle tracking using deep sort with low confidence track filtering. In *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2019.

[85] Adel Ammar, Anis Koubaa, Mohanned Ahmed, Abdulrahman Saad, and Bilel Benjdira. Vehicle detection from aerial images using deep learning: A comparative study. *Electronics*, 10(7):820, 2021.

[86] Seung Woo Ham, Ho-Chul Park, Eui-Jin Kim, Seung-Young Kho, and Dong-Kyu Kim. Investigating the influential factors for practical application of multi-class vehicle detection for images from unmanned aerial vehicle using deep learning models. *Transportation Research Record*, 2674(12):553–567, 2020.

[87] Qingtian Wu and Yimin Zhou. Real-time object detection based on unmanned aerial vehicle. In *2019 IEEE 8th Data Driven Control and Learning Systems Conference (DDCLS)*, pages 574–579. IEEE, 2019.

[88] Xiaofei Liu, Tao Yang, and Jing Li. Real-time ground vehicle detection in aerial infrared imagery based on convolutional neural network. *Electronics*, 7(6):78, 2018.

[89] Yongzheng Xu, Guizhen Yu, Yunpeng Wang, Xinkai Wu, and Yalong Ma. Car detection from low-altitude uav imagery with the faster r-cnn. *Journal of Advanced Transportation*, 2017, 2017.

[90] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Dota: A large-scale dataset for object detection in aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3974–3983, 2018.

[91] Shawn M Turner, William L Eisele, Robert J Benz, and Douglas J Holdener. Travel time data collection handbook. Technical report, United States. Federal Highway Administration, 1998.

[92] Zhongren Wang. Using floating cars to measure travel time delay: how accurate is the method? *Transportation research record*, 1870(1):84–93, 2004.

[93] Frederick W Cathey and Daniel J Dailey. Estimating corridor travel time by using transit vehicles as probes. *Transportation Research Record*, 1855(1):60–65, 2003.

[94] Robert L Bertini and Sutti Tantiyanugulchai. Transit buses as traffic probes: Use of geolocation data for empirical evaluation. *Transportation Research Record*, 1870(1):35–45, 2004.

[95] Liang Zou, Jian-Min Xu, and Ling-Xiang Zhu. Arterial speed studies with taxi equipped with global positioning receivers as probe vehicle. In *Proceedings. 2005 International Conference on Wireless Communications, Networking and Mobile Computing, 2005.*, volume 2, pages 1343–1347. IEEE, 2005.

[96] Benjamin Coifman and SeoungBum Kim. Measuring freeway traffic conditions with transit vehicles. *Transportation research record*, 2121(1):90–101, 2009.

[97] Douglas Thornton and Benjamin Coifman. Signal progression impacts on transit buses as travel time probes. *Journal of Transportation Engineering*, 141(8):04015009, 2015.

[98] Ross J Haseman, Jason S Wasson, and Darcy M Bullock. Real-time measurement of travel time delay in work zones and evaluation metrics using bluetooth probe tracking. *Transportation research record*, 2169(1):40–53, 2010.

[99] Maria Martchouk, Fred Mannering, and Darcy Bullock. Analysis of freeway travel time variability using bluetooth detection. *Journal of transportation engineering*, 137(10):697–704, 2011.

[100] Ernest OA Tufuor and Laurence R Rilett. Validation of the highway capacity manual urban street travel time reliability methodology using empirical data. *Transportation research record*, 2673(4):415–426, 2019.

[101] John Glen Wardrop and George Charlesworth. A method of estimating speed and flow of traffic from a moving vehicle. *Proceedings of the Institution of Civil Engineers*, 3(1):158–171, 1954.

[102] Keemin Sohn and Daehyun Kim. Dynamic origin–destination flow estimation using cellular communication system. *IEEE Transactions on Vehicular Technology*, 57(5):2703–2713, 2008.

[103] Noelia Caceres, Luis M Romero, Francisco G Benitez, and Jose M del Castillo. Traffic flow estimation models using cellular phone data. *IEEE Transactions on Intelligent Transportation Systems*, 13(3):1430–1441, 2012.

[104] Wei Ma and Sean Qian. High-resolution traffic sensing with probe autonomous vehicles: A data-driven approach. *Sensors*, 21(2):464, 2021.

[105] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[106] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[107] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010.

[108] Charles Courtemanche, Joseph Garuccio, Anh Le, Joshua Pinkston, and Aaron Yelowitz. Strong social distancing measures in the united states reduced the covid-19 growth rate: Study evaluates the impact of social distancing measures on the growth rate of confirmed covid-19 cases across the united states. *Health Affairs*, 39:10–1377, 2020.

[109] Cong T Nguyen, Yuris Mulya Saputra, Nguyen Van Huynh, Ngoc-Tan Nguyen, Tran Viet Khoa, Bui Minh Tuan, Diep N Nguyen, Dinh Thai Hoang, Thang X Vu, Eryk Dutkiewicz, and et al. Enabling and emerging technologies for social distancing: A comprehensive survey. *arXiv preprint arXiv:2005.02816*, 2020.

[110] Sonali Agarwal, Narinder Singh Punn, Sanjay Kumar Sonbhadra, P Nagabhushan, KK Pandian, and Praveer Saxena. Unleashing the power of disruptive and emerging technologies amid covid 2019: A detailed review. *arXiv preprint arXiv:2005.11507*, 2020.

[111] Marco Cristani, Alessio Del Bue, Vittorio Murino, Francesco Setti, and Alessandro Vinciarelli. The visual social distancing problem. *IEEE Access*, 8:126876–126886, 2020.

[112] Narinder Singh Punn, Sanjay Kumar Sonbhadra, and Sonali Agarwal. Monitoring covid-19 social distancing with person detection and tracking via fine-tuned yolo v3 and deepsort techniques. *arXiv preprint arXiv:2005.01385*, 2020.

[113] Mahdi Rezaei and Mohsen Azarmi. Deepsocial: Social distancing monitoring and infection risk assessment in covid-19 pandemic. *Applied Sciences*, 10(21):7514, 2020.

[114] Imran Ahmed, Misbah Ahmad, Joel JPC Rodrigues, Gwanggil Jeon, and Sadia Din. A deep learning-based social distance monitoring framework for covid-19. *Sustainable Cities and Society*, 65:102571, 2020.

[115] Davide Antonio Maria Cota. *Monitoring COVID-19 prevention measures on CCTV cameras using Deep Learning*. PhD thesis, Politecnico di Torino, 2020.

[116] Prateek Khandelwal, Anuj Khandelwal, and Snigdha Agarwal. Using computer vision to enhance safety of workforce in manufacturing in a post covid world. *arXiv preprint arXiv:2005.05287*, 2020.

[117] Antoni B Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–7. IEEE, 2008.

[118] Faisal Z Qureshi. Object-video streams for preserving privacy in video surveillance. In *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 442–447. IEEE, 2009.

[119] Sangho Park and Mohan M Trivedi. A track-based human movement analysis and privacy protection system adaptive to environmental contexts. In *IEEE Conference on Advanced Video and Signal Based Surveillance, 2005.*, pages 171–176. IEEE, 2005.

[120] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems*, 28:91–99, 2015.

[121] David S Hui, Esam I Azhar, Tariq A Madani, Francine Ntoumi, Richard Kock, Osman Dar, Giuseppe Ippolito, Timothy D Mchugh, Ziad A Memish, Christian Drosten, and et al. The continuing 2019-ncov epidemic threat of novel coronaviruses to global health—the latest 2019 novel coronavirus outbreak in wuhan, china. *International Journal of Infectious Diseases*, 91:264–266, 2020.

[122] Michael Greenstone and Vishan Nigam. Does social distancing matter? *University of Chicago, Becker Friedman Institute for Economics Working Paper 2020-26*, 2020.

[123] Daniel Costa and João Paulo Peixoto. Covid-19 pandemic: A review of smart cities initiatives to face new outbreaks. *IET Smart Cities*, 2:64–73, 07 2020.

[124] Musa Ndiaye, Stephen S. Oyewobi, Adnan M. Abu-Mahfouz, Gerhard P. Hancke, Anish M. Kurien, and Karim Djouani. Iot in the wake of covid-19: A survey on contributions, challenges and evolution. *IEEE Access*, 8:186821–186839, 2020.

[125] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*, 2019.

[126] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, 2019.

[127] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.

[128] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017.

[129] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision (ECCV)*, pages 21–37. Springer, 2016.

[130] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10781–10790, 2020.

[131] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *European Conference on Computer Vision (ECCV)*, pages 734–750, 2018.

[132] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *IEEE International Conference on Computer Vision*, pages 9627–9636, 2019.

[133] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[134] Shorf Mohammad, M. Shamim Hossain, and Mohammed Alhamid. Towards the sustainable development of smart cities through mass video surveillance: A response to the covid-19 pandemic. *Sustainable Cities and Society*, 64:102582, 01 2021.

[135] LandingAI. Landing ai creates an ai tool to help customers monitor social distancing in the workplace. `https://landing.ai/landing-ai-creates-an-ai-tool-to-help-customers-monitor-social-distancing-in-the-workplace/`, 2020 (accessed May 3, 2020.

[136] Jon Hall. Social distance monitoring. `https://levelfivesupplies.com/social-distance-monitoring/`, 2020 (accessed June 1, 2020).

[137] StereoLabs. Using 3d cameras to monitor social distancing. `https://www.stereolabs.com/blog/using-3d-cameras-to-monitor-social-distancing/`, 2020 (accessed June 20, 2020).

[138] Barret Zoph, Ekin D Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V Le. Learning data augmentation strategies for object detection. *arXiv preprint arXiv:1906.11172*, 2019.

[139] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[140] Ben Benfold and Ian Reid. Stable multi-target tracking in real-time surveillance video. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3457–3464. IEEE, 2011.

[141] Ke Chen, Chen Change Loy, Shaogang Gong, and Tony Xiang. Feature mining for localised crowd counting. In *British Machine Vision Conference (BMVC)*, volume 1, page 3, 2012.

[142] Bolei Zhou, Xiaogang Wang, and Xiaoou Tang. Understanding collective crowd behaviors: Learning a mixture model of dynamic pedestrian-agents. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2871–2878. IEEE, 2012.

[143] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[144] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10529–10538, 2020.

[145] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018.

[146] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019.

[147] Zhe Liu, Xin Zhao, Tengteng Huang, Ruolan Hu, Yu Zhou, and Xiang Bai. Tanet: Robust 3d object detection from point clouds with triple attention. In *AAAI*, pages 11677–11684, 2020.

[148] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia. 3dssd: Point-based 3d single stage object detector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11040–11048, 2020.

[149] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

[150] Zhuo Deng and Longin Jan Latecki. Amodal detection of 3d objects: Inferring 3d bounding boxes from 2d ones in rgb-depth images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5762–5770, 2017.

[151] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

[152] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L Waslander. Joint 3d proposal generation and object detection from view aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, 2018.

[153] Shuran Song and Jianxiong Xiao. Deep sliding shapes for amodal 3d object detection in rgb-d images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 808–816, 2016.

[154] Zhixin Wang and Kui Jia. Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1742–1749, 2019.

[155] Maosheng Ye, Shuangjie Xu, and Tongyi Cao. Hvnet: Hybrid voxel network for lidar based 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1631–1640, 2020.

[156] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–779, 2019.

[157] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Std: Sparse-to-dense 3d object detector for point cloud. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1951–1960, 2017.

[158] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

[159] Vishwanath A Sindagi, Yin Zhou, and Oncel Tuzel. Mvx-net: Multimodal voxelnet for 3d object detection. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7276–7282. IEEE, 2019.

[160] Sourabh Vora, Alex H Lang, Bassam Helou, and Oscar Beijbom. Pointpainting: Sequential fusion for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4604–4612, 2020.

[161] Xinxin Du, Marcelo H Ang, Sertac Karaman, and Daniela Rus. A general pipeline for 3d detection of vehicles. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3194–3200. IEEE, 2018.

[162] Kiwoo Shin, Youngwook Paul Kwon, and Masayoshi Tomizuka. Roarnet: A robust 3d object detection based on region approximation refinement. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2510–2515. IEEE, 2019.

[163] Xiaoke Shen and Ioannis Stamos. Frustum voxnet for 3d object detection from rgb-d or depth images. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 1698–1706, 2020.

[164] Xin Zhao, Zhe Liu, Ruolan Hu, and Kaiqi Huang. 3d object detection using scale invariant and feature reweighting networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9267–9274, 2019.

[165] Pei Cao, Hao Chen, Ye Zhang, and Gang Wang. Multi-view frustum pointnet for object detection in autonomous driving. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 3896–3899. IEEE, 2019.

[166] Michael Fürst, Oliver Wasenmüller, and Didier Stricker. Lrpd: Long range 3d pedestrian detection leveraging specific strengths of lidar and rgb. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7. IEEE, 2020.

[167] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

[168] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[169] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[170] Amir Rasouli, Iuliia Kotseruba, and John K Tsotsos. Are they going to cross? a benchmark dataset and baseline for pedestrian crosswalk behavior. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 206–213, 2017.

[171] Iuliia Kotseruba, Amir Rasouli, and John K Tsotsos. Do they want to cross? understanding pedestrian intention for behavior prediction. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1688–1693. IEEE, 2020.

[172] Javier Lorenzo, Ignacio Parra, Florian Wirth, Christoph Stiller, David Fernandez Llorca, and Miguel Angel Sotelo. Rnn-based pedestrian crossing prediction using activity and pose-related features. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1801–1806. IEEE, 2020.

[173] Amir Rasouli, Iuliia Kotseruba, Toni Kunic, and John K Tsotsos. Pie: A large-scale dataset and models for pedestrian intention estimation and trajectory prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6262–6271, 2019.

[174] Zhijie Fang and Antonio M López. Is the pedestrian going to cross? answering by 2d pose estimation. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1271–1276. IEEE, 2018.

[175] Francesco Piccoli, Rajarathnam Balakrishnan, Maria Jesus Perez, Moraldeepsingh Sachdeo, Carlos Nunez, Matthew Tang, Kajsa Andreasson, Kalle Bjurek, Ria Dass Raj, Ebba Davidsson, et al. Fussi-net: Fusion of spatio-temporal skeletons for intention prediction network. *arXiv preprint arXiv:2005.07796*, 2020.

[176] Amir Rasouli, Iuliia Kotseruba, and John K Tsotsos. Pedestrian action anticipation using contextual feature fusion in stacked rnns. In *BMVC*, 2019.

[177] Amir Rasouli, Tiffany Yau, Mohsen Rohani, and Jun Luo. Multi-modal hybrid architecture for pedestrian action prediction. *arXiv preprint arXiv:2012.00514*, 2020.

[178] Amir Rasouli, Mohsen Rohani, and Jun Luo. Bifold and semantic reasoning for pedestrian behavior prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15600–15610, 2021.

[179] Iuliia Kotseruba, Amir Rasouli, and John K Tsotsos. Benchmark for evaluating pedestrian action prediction. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1258–1268, 2021.

[180] Joko Hariyono and Kang-Hyun Jo. Detection of pedestrian crossing road. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 4585–4588. IEEE, 2015.

[181] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 304–311. IEEE, 2009.

[182] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[183] Bingbin Liu, Ehsan Adeli, Zhangjie Cao, Kuan-Hui Lee, Abhijeet Shenoi, Adrien Gaidon, and Juan Carlos Niebles. Spatiotemporal relationship reasoning for pedestrian intent prediction. *IEEE Robotics and Automation Letters*, 5(2):3485–3492, 2020.

[184] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[185] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103, 2014.

[186] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[187] Khaled Saleh, Mohammed Hossny, and Saeid Nahavandi. Real-time intent prediction of pedestrians for autonomous ground vehicles via spatio-temporal densenet. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9704–9710. IEEE, 2019.

[188] Khaled Saleh, Mohammed Hossny, and Saeid Nahavandi. Spatio-temporal densenet for real-time intent prediction of pedestrians in urban traffic environments. *Neurocomputing*, 386:317–324, 2020.

[189] Mohamed Chaabane, Ameni Trabelsi, Nathaniel Blanchard, and Ross Beveridge. Looking ahead: Anticipating pedestrians crossing with future frames prediction. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2297–2306, 2020.

[190] Pratik Gujjar and Richard Vaughan. Classifying pedestrian actions in advance using predicted video of urban driving scenes. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2097–2103. IEEE, 2019.

[191] Zhijie Fang and Antonio M López. Intention recognition of pedestrians and cyclists by 2d pose estimation. *IEEE Transactions on Intelligent Transportation Systems*, 21(11):4773–4783, 2019.

[192] Zixing Wang and Nikolaos Papanikolopoulos. Estimating pedestrian crossing states based on single 2d body pose. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, volume 2, 2020.

[193] Pablo Rodrigo Gantier Cadena, Ming Yang, Yeqiang Qian, and Chunxiang Wang. Pedestrian graph: Pedestrian crossing prediction based on 2d pose estimation and graph convolutional networks. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2000–2005. IEEE, 2019.

[194] Amir Rasouli, Mohsen Rohani, and Jun Luo. Pedestrian behavior prediction via multitask learning and categorical interaction modeling. *arXiv preprint arXiv:2012.03298*, 2020.

[195] Amir Rasouli, Tiffany Yau, Peter Lakner, Saber Malekmohammadi, Mohsen Rohani, and Jun Luo. Pepscenes: A novel dataset and baseline for pedestrian action prediction in 3d. *arXiv preprint arXiv:2012.07773*, 2020.

[196] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[197] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[198] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *IEEE transactions on pattern analysis and machine intelligence*, 43(1):172–186, 2019.

[199] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[200] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[201] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[202] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

[203] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[204] Amir Rasouli and John K Tsotsos. Autonomous vehicles that interact with pedestrians: A survey of theory and practice. *IEEE transactions on intelligent transportation systems*, 21(3):900–918, 2019.

[205] Shile Zhang, Mohamed Abdel-Aty, Yina Wu, and Ou Zheng. Pedestrian crossing intention prediction at red-light using pose estimation. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[206] Tina Chen, Renran Tian, and Zhengming Ding. Visual reasoning using graph convolutional networks for predicting pedestrian crossing intention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3103–3109, 2021.

[207] Vincenzo Punzo and Biagio Ciuffo. Integration of driving and traffic simulation: Issues and first solutions. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):354–363, 2010.

[208] Antonio Lanatà, Gaetano Valenza, Alberto Greco, Claudio Gentili, Riccardo Bartolozzi, Francesco Bucchi, Francesco Frendo, and Enzo Pasquale Scilingo. How the autonomic nervous system and driving style change with incremental stressing conditions during simulated driving. *IEEE Transactions on Intelligent Transportation Systems*, 16(3):1505–1517, 2014.

[209] Dennis Ludl, Thomas Gulde, and Cristóbal Curio. Enhancing data-driven algorithms for human pose estimation and action recognition through simulation. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3990–3999, 2020.

[210] Saad Minhas, Aura Hernández-Sabaté, Shoaib Ehsan, and Klaus D McDonald-Maier. Effects of non-driving related tasks during self-driving mode. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[211] M. Aramrattana, T. Larsson, C. Englund, J. Jansson, and A. Nåbo. A simulation study on effects of platooning gaps on drivers of conventional vehicles in highway merging situations. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–7, 2020.

[212] Wei Yang, Ling Zheng, Yinong Li, Yue Ren, and Zhoubing Xiong. Automated highway driving decision considering driver characteristics. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[213] Uijong Ju, Lewis L Chuang, and Christian Wallraven. Acoustic cues increase situational awareness in accident situations: A vr car-driving study. *IEEE Transactions on Intelligent Transportation Systems*, 21:2350–2359, 2020.

[214] James T Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 143–150, 1986.

[215] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[216] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

[217] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *Stat*, 1050:21, 2018.

[218] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5907–5915, 2017.

[219] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan++: Realistic image synthesis with stacked generative adversarial networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1947–1962, 2018.

[220] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B. Tenenbaum, William T. Freeman, and Antonio Torralba. Visualizing and understanding generative adversarial networks. In *International Conference on Learning Representations*, 2019.

[221] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In *Advances in Neural Information Processing Systems*, pages 700–708, 2017.

[222] Takeru Miyato and Masanori Koyama. cGANs with projection discriminator. In *International Conference on Learning Representations*, 2018.

[223] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1125–1134, 2017.

[224] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1511–1520, 2017.

[225] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019.

[226] Xiaojuan Qi, Qifeng Chen, Jiaya Jia, and Vladlen Koltun. Semi-parametric image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8808–8816, 2018.

[227] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8798–8807, 2018.

[228] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. In *Advances in Neural Information Processing Systems*, pages 1144–1156, 2018.

[229] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[230] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4, 2013.

[231] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2017.

[232] Peter M van Leeuwen, Carla Gómez i Subils, Arnau Ramon Jimenez, Riender Happee, and Joost CF de Winter. Effects of visual fidelity on curve negotiation, gaze behaviour and simulator discomfort. *Ergonomics*, 58(8):1347–1364, 2015.

[233] Xi Zhao and Wayne A Sarasua. How to use driving simulators properly: impacts of human sensory and perceptual capabilities on visual fidelity. *Transportation Research Part C: Emerging Technologies*, 93:381–395, 2018.

[234] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[235] Ruochen Fan, Xuanrun Wang, Qibin Hou, Hanchao Liu, and Tai-Jiang Mu. Spinnet: Spinning convolutional network for lane boundary detection. *Computational Visual Media*, 5(4):417–428, 2019.

[236] Thu H Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yongliang Yang. Rendernet: A deep convolutional network for differentiable rendering from 3d shapes. In *Advances in Neural Information Processing Systems*, pages 7891–7901, 2018.

[237] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7708–7717, 2019.

[238] Ioannis Gkioulekas, Anat Levin, and Todd Zickler. An evaluation of computational imaging techniques for heterogeneous inverse scattering. In *European Conference on Computer Vision*, pages 685–701. Springer, 2016.

[239] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3907–3916, 2018.

[240] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision*, pages 154–169. Springer, 2014.

[241] Miao Wang, Guo-Ye Yang, Ruilong Li, Run-Ze Liang, Song-Hai Zhang, Peter M Hall, and Shi-Min Hu. Example-guided style-consistent image synthesis from semantic labeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1495–1504, 2019.

[242] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *33rd International Conference on Machine Learning*, pages 1060–1069, 2016.

[243] Weijian Deng, Liang Zheng, Qixiang Ye, Guoliang Kang, Yi Yang, and Jianbin Jiao. Image-image domain adaptation with preserved self-similarity and domain-dissimilarity for person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 994–1003, 2018.

[244] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3722–3731, 2017.

[245] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 469–477, 2016.

[246] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2223–2232, 2017.

[247] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7167–7176, 2017.

[248] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon Mesh Processing*. CRC Press, 2010.

[249] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[250] Tom Mertens, Jan Kautz, and Frank Van Reeth. Exposure fusion: A simple and practical alternative to high dynamic range photography. In *Computer Graphics Forum*, volume 28, pages 161–171. Wiley Online Library, 2009.

[251] Huikai Wu, Shuai Zheng, Junge Zhang, and Kaiqi Huang. Gp-gan: Towards realistic high-resolution image blending. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2487–2495, 2019.

[252] Pierre-Yves Laffont, Zhile Ren, Xiaofeng Tao, Chao Qian, and James Hays. Transient attributes for high-level understanding and editing of outdoor scenes. *ACM Transactions on Graphics (TOG)*, 33(4):1–11, 2014.

[253] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[254] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[255] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 633–641, 2017.

[256] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[257] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[258] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.