Final report

CongestionSim - A study of traffic congestion

Elias Haaralahti
Aleksi Hämäläinen

May 10, 2023

Aalto University

CS-E4875 - Research Project in Machine Learning, Data Science and Artificial Intelligence, 5 ECTS

Advisor: Vesa Hirvisalo

Abstract

The increase of autonomous features in vehicles has increased the need to understand the space around vehicles and to measure traffic flow. Commonly these can be done by reading the raw sensor output of vehicles and processing the data with object detection methods. Such systems can also include fixed-position cameras, which can work alone or augment other sensors. The processed information is communicated over a wireless network, either directly to the other vehicles or to an external network node that can process the data from all vehicles.

We focus on using machine learning methods to estimate traffic congestion and create an understanding of the environment around vehicles. We present CongestionSim, a hybrid simulation system that simulates traffic in a city using CARLA and processes the collected data in a discrete-event simulation (DES). We compare this system to a deep learning model, which classifies traffic congestion directly from fixed-position camera images. In addition, we consider the inference requirements for running such a system in real-time.

Our results show that traffic congestion analysis can be difficult due to the various ways congestion can be defined. However, both our DES simulation and our deep learning model produced correct predictions, as in general intersection with more vehicles and a slower average speed was classified to be more congested than an intersection with fewer vehicles and a faster average speed. Our deep learning model was able to achieve an accuracy of 81.6% on our simulated data, but it should be noted that the labels were created based on another estimation of the ground truth.

We estimated inference requirements for the DES simulation and found that with our hardware, running it in real-time with CARLA would only be possible at around one fps with our current implementation, which might be enough for traffic congestion analysis. However, running the part of the system that processes vehicle sensor data could be run at over 30 fps, which would be enough for other use cases, such as detecting potential collisions.

We discovered that our methods are a viable way of estimating traffic congestion and extracting information from vehicle sensor data. However, both the performance and accuracy of our methods could be greatly increased, and we propose several improvements to overcome their limitations.

Contents

1 Introduction

This paper focuses on utilizing sensor data processing to understand the environment around vehicles and to detect traffic congestion. These modern systems are increasingly important as vehicles become more autonomous [12]. With the ability to automatically communicate information over wireless networks, systems like these could be used to improve traffic flow and prevent unnecessary accidents.

We study the problem of converting raw sensor data from multiple entities to information, such as how many vehicles and pedestrians there are in an intersection. With this information, congestion can be analyzed, and other vehicles could be warned of potential danger, such as a bicycle coming fast from behind a corner. The vehicles process the data from their sensors and can then send it to other vehicles directly (vehicle-to-vehicle communication) or to an external processing node (vehicle-to-infrastructure communication) [2], which processes the data from all nearby vehicles and sends information back. The sensors may be vehicles that are capable of processing data and sending it over a network. They can also be roadside units (RSUs), which are fixed location cameras capable of processing and sending data.

We also analyze the inference requirements of these systems, as we are interested in estimating the hardware requirements of running the inference in real-time. However, these systems are not limited to traffic and could be used in any other system where entities collect sensor data and can cooperate with each other to increase efficiency or safety.

We create our own experiments based on literature and existing methods and present a hybrid simulation system called CongestionSim. The system simulates traffic in a city with multiple intersections, using the CARLA simulator [4]. From the CARLA simulation, we are able to collect sensor data from vehicles and RSUs. The data is then stored and processed separately in a discrete-event simulation (DES) [17]. In the DES simulation, vehicles communicate with an external wireless processing node that is capable of combining and processing the data to form an understanding of the world. We apply computer vision and machine learning methods, such as the YOLO object recognition model [16], to infer relevant information from the raw sensor data. We also created a separate deep learning model that detects congestion from RSU sensor data.

The results of our experimentation show that congestion detection using our methods is possible. However, estimating a ground truth for congestion is difficult, as there are multiple ways of defining it [14]. Our estimations of the ground truth are based on the average velocities of vehicles in an intersection and do not account for any attributes of the area, such as how many lanes there are. Due to this, comparing the accuracy of our methods is difficult. However, based on our congestion analysis for two intersections, where the first one is more congested, our methods were able to predict that the congestion of the first intersection was higher than in the second one. Our CNN-based deep learning model, on the other hand, was able to achieve an accuracy of 81.6% on our simulated data.

We estimated inference requirements for the DES simulation and found that with our hardware, running it in real-time with CARLA would only be possible at around one fps with our current implementation. However, running the part of the system that processes vehicle sensor data could be run at over 30 fps. which is enough for many use cases.

The rest of this paper is organized as follows. In Section 2, we go over relevant literature and background where we present key problems and methods used in similar research. Section 3 presents the design and implementation of CongestionSim. We show our experimental methods and results in Section 4. Section 5 discusses our experimental results and the performance of our system. Finally, we draw conclusions on our results, summarize what we learned, and how the design choices worked for us in Section 6.

2 Background

In this section, we will give background about relevant topics related to our CongestionSim system and present related literature. In Section 2.1, we discuss traffic simulation and CARLA, which we use to simulate traffic. In Section 2.2, we attempt to define traffic congestion and how to measure it, while Section 2.3 goes over the basics of discrete-event simulation. Section 2.4 gives an overview of convolutional neural networks and inference. In Section 2.5, we present relevant simulation design choices. Section 2.6 goes through different methods for detecting traffic congestion and understanding the vehicle's surrounding environment. Finally, in Section 2.7, we review previous studies on detecting traffic congestion from images taken by RSUs.

2.1 Traffic simulation

Traffic simulation provides a safe, easy, and cost-effective way to collect data in complex urban environments. CARLA [4] is an open-source simulator designed for autonomous driving research. It can also be used to customize and simulate various traffic scenarios. The simulated environment is composed of 3D models of static objects, such as buildings, roads, and traffic signs, as well as dynamic objects, such as vehicles and pedestrians. CARLA comes with multiple premade urban cities and supports various sensors, such as cameras, lidars, and radars. The architecture of CARLA is based on a client-server model, where the server runs the simulation and renders the scene. Python clients communicate with the server and receive sensor readings in return.

2.2 Traffic congestion

Estimating traffic congestion is difficult, as there are various ways to define it. Some approaches to defining congestion include calculating the ratio between the average speed of vehicles and the speed limit, measuring travel times, as well as analyzing queue lengths [14]. Other approaches to this matter are generally based on the speed or amount of vehicles in an area. We could consider that an intersection has a maximum capacity for vehicles, so if there are a lot of them, the intersection is probably congested. However, the average speeds give information about the traffic flow, which may be more important. Even if there are a lot of vehicles, as long as the average speed remains near the typical speed for the area, there probably is no traffic congestion.

Traffic congestion can typically be classified into recurring and non-recurring congestion [14]. Recurring congestion refers to congestion that occurs on a regular basis, for example, due to poor urban planning. Non-recurring congestion, on the other hand, does not follow a recurring pattern and might appear, for example, due to a car crash or a special event. These two types of congestion could be analyzed in congestion prediction and alleviation tasks. The congestion detection task, on the other hand, does not make such a division, as it simultaneously detects both types of congestion.

2.3 Discrete-event simulation

A simulation is a model of a real system, and in general, simulations are used to study the system, including how they work, how changes affect the state of the system, and for experimentation and validation [17]. There are various ways to simulate a system, such as real-time simulation and discrete-event simulation (DES). The main difference between the two is that, in real-time simulations, the simulation time progresses at the same time as the modeled system does. For example, to model a system in the real world, the simulation would have to move at the same pace as our clocks, which can be difficult if there is a lot of processing to perform. A DES simulation allows us to simulate a system in discrete steps, which allows us to do as much processing as we require without any time constraints.

A discrete-event simulation is a type of simulation where events occur sequentially at specific simulation times, and each event changes the state of the system at the next step [17]. A DES simulation allows the processing of stored data sequentially, even though events might happen simultaneously. This allows for performing computationally demanding processing without time constraints. When all entities in the simulation are processed, the simulation time advances either to the next event or by a specified unit, such as one second.

2.4 Convolutional neural networks and inference

We use machine learning methods for visual perception and analysis of traffic congestion. Our methods are based on Artificial Neural Networks (ANNs), which is a deep learning method inspired by the human brain that uses interconnected neurons in multiple layers to learn a function [15]. As explained in the paper, ANNs use backpropagation to modify the network based on the difference between the output of the network and what the output should be, also known as a loss function. For example, an ANN model can accept the pixels of an image as input features and output a probability of there being a car in the image.

Specifically, we use Convolutional Neural Networks (CNNs). These networks are based on ANNs but are generally used for images, as they can perform operations, such as convolution and downsampling, to learn features from the images [15]. Essentially unlike ANNs, for a convolutional neural network, pixels that are next to each other are, are considered more important to each other.

In the context of ANNs, inference refers to a forward pass of the network, which involves giving the data to the model and receiving the output. Inference requirement analysis is critical if the model is used for purposes where inference speed is important. It consists of analyzing model performance on various hardware to find the minimum hardware requirements to run the model with the required throughput, which is a measure of how many times the model completes a forward pass in a second [5]. Along with throughput, memory consumption of the model is also critical. Some of the main reasons for high inference requirements are the number of parameters in the model and the size of the data.

We used pre-trained models, such as YOLO (You only look once) [16] and AlexNet [10], for object detection and image classification, respectively. Both models are based on CNNs but differ in purpose. YOLO is an object detection model that detects objects in an image and creates a bounding box for the detections. A bounding box is a rectangle shape that estimates the position and size of the object in the image as close to reality as possible. AlexNet, on the other hand, is primarily for image classification, and we use it to detect traffic congestion from an image.

2.5 Simulation design choices

A relevant design choice in systems such as these is how data processing and communication are handled. Two key methods of cooperative communication between vehicles are direct vehicle-to-vehicle communication and indirect vehicle-to-infrastructure communication, in which vehicles communicate to fixed nodes [2]. In vehicle-to-vehicle communication, vehicles directly inform other vehicles in the area about themselves and any relevant information gained from the processing of sensor data. An advantage of vehicle-to-vehicle communication is that there is no need for additional infrastructure to be built and maintained.

However, this infrastructure, or external network nodes, are not required to be everywhere. Communication happens via wireless networks, and the main concern is the delay of data transmission. A single static processing node is capable of processing data for multiple locations. However, vehicles may still require enough computing resources to process the data from their own sensors, as transferring images at 30 fps to an external node over a wireless network can be demanding, but in some cases, some of the processing could be offloaded to the external node.

There are various ways to understand the world from sensor data. A basic way of detecting traffic congestion could be to measure the speeds of vehicles in an area and compare it to the speed limit or to the average speed usual for that time. Congestion could be classified into labels such as low, medium, and high or to a percentage. This classification result could then be sent to other vehicles that intend to pass through the area. More advanced methods for detecting traffic congestion include image and video processing. On the image processing level, computer vision techniques can be used to detect objects from images, such as other vehicles and pedestrians. If the processing is applied to a video, object tracking can be done, and information, such as the vehicle velocity and trajectory, can be inferred. Estimated velocities could be used in more advanced congestion estimation, as the speeds of vehicles that do not cooperate would also be known. Trajectory planning could help estimate traffic flow and increase the analysis potential for a collision warning system.

2.6 Visual perception

Visual perception is one of our key problems to address. We want to combine information computed from raw sensor data from each vehicle to perceive and analyze the environment. There are various sensors that can be used in vehicles to perceive the environment, but commonly these sensors are cameras or a LiDAR (Light Detection and Ranging) [1]. The data from these sensors can be processed to extract relevant information, which can be communicated to other entities to achieve cooperation between vehicles. The information is used to form an understanding of the world around the vehicles, which can be analyzed further. Analysis can include estimating traffic congestion in an intersection or warning vehicles of potential collisions.

One method of converting video to a 3D world is presented in the paper Joint Monocular 3D Vehicle Detection and Tracking [6]. In this paper, various techniques are discussed to convert video to a 3D world using a monocular camera instead of stereo vision, as it is stated that in many cases, stereo vision is not available. The first key step in the paper is object detection using deep learning with CNN-based architectures or pre-trained models, such as YOLO. The model is used to find 3D bounding boxes, which also indicate the depth and orientation of objects in the images. This allows for estimating the dimensions and orientation of the vehicle. Finally, in the paper, vehicle tracking is applied over video, which increases the capabilities of the system.

Another method that is closer to our use case is presented in the paper "DisNet: A novel method for distance estimation from monocular camera" [13]. In the paper they train a model with YOLO bounding box information as features and distances measured by a laser as labels. In the paper, they use bounding box information, real dimensions of the detected object and camera parameters to estimate distance. The relationship between the bounding box size and the actual object size is expected to be related, as further away objects should have smaller bounding boxes. This allows using the size of the bounding box and knowledge of the object's average dimensions to estimate the distance with some error. The error varied between results, but was often up to several meters.

2.7 Congestion detection using RSUs

Several studies have been conducted to study traffic congestion by applying CNN-based approaches to images captured by roadside units (RSUs). Chakraborty et al. [3] have performed binary classification using YOLO and a CNN model to detect traffic congestion from camera images. Their data consisted of CCTV images collected from interstates and highways in Iowa, USA. Since manually labeling data is time-consuming, they used occupancy data from nearby radars to label the images into congested and non-congested classes. Using these approaches, they were able to get 91.5% and 90.2% accuracies for the YOLO and CNN models, respectively.

Wang et al. [19] utilized manually labeled images to explore two variations of AlexNet and VGGNet for binary classification of traffic congestion. In the first approach, they applied transfer learning to pre-trained AlexNet and VGGNet models by modifying the final layers of the models and fine-tuning them on traffic data. The second approach used support vector machines (SVMs) instead of fully connected layers in the pre-trained AlexNet and VGGNet models. Their dataset comprised images from hundreds of CCTVs in Shaanxi province, China, depicting varying weather and lighting conditions. In the first approach, they were able to obtain 90% accuracy using AlexNet and 89% accuracy using VGGNet. In the second approach, the accuracies were slightly lower, 78% for AlexNet and 81% for VGGNet.

Another study using CNN-based approaches for detecting traffic congestion was conducted by Kurviawan et al. [11]. Their study involved binary classification using a CNN model, using manually labeled CCTV images collected from 14 locations in Jakarta, Indonesia. Their approach resulted in an impressive 93% accuracy in detecting traffic congestion.

3 CongestionSim

We created a hybrid simulation system called CongestionSim [7], which consists of three main parts. First, we simulate traffic and collect sensor data using CARLA [4]. Then, we process the data using a discrete-event simulation. An HDF5 file acts as a bridge between these two systems and allows the DES simulation to process the collected data without interacting with CARLA. Additionally, we also trained a CNN-based binary classification model to predict congestion from the CARLA sensor data. A high-level overview of our setup is visualized in Figure 1, with the exception of the binary classifier. More detailed explanations are given in Sections 3.1, 3.2, and 3.3.

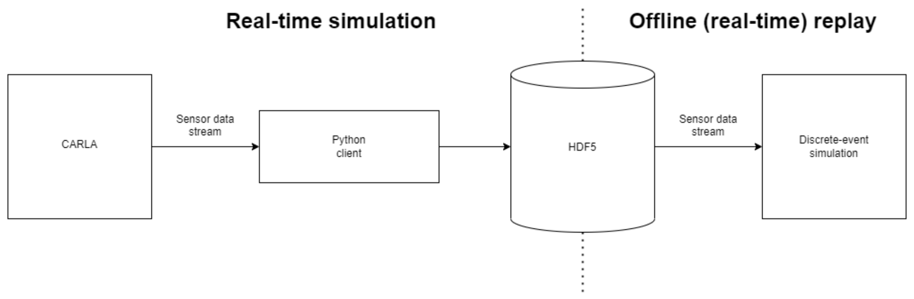
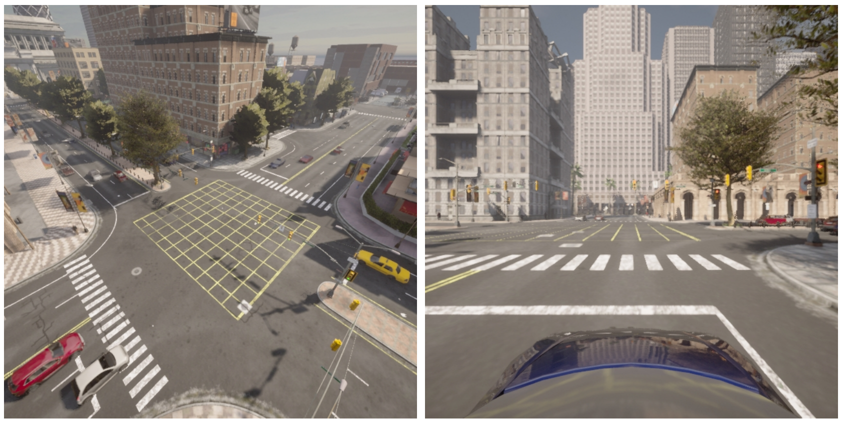


Figure 1: Overview of our setup. CARLA is used to simulate traffic and produce sensor data in real-time. The data is captured with a Python client and stored in a hierarchical data format (HDF5). The discrete-event simulation can replay the data in real-time without interacting with the CARLA simulator.

3.1 Simulating traffic and collecting sensor data

We used CARLA [4] to simulate traffic and collect sensor data, as shown in the left side of Figure 1. We created a 3D scene of a city with vehicles, RSUs, and pedestrians. Data was then collected from the simulation and stored in a hierarchical data format (HDF5). The data includes images captured by vehicles and RSUs, the locations and velocities of the vehicles, as well as metadata about the simulated scenario, such as the total number of simulated frames, camera resolution, and the locations of the RSUs and intersections. The collected datasets represent situations where autonomous vehicles move in a city and perceive their environment. RSUs, on the other hand, observe the traffic in an intersection. An example of the data is visualized in Figure 2.



(a) Image captured by RSU. (b) Image captured by vehicle.

Figure 2: An example of image data produced by CARLA.

3.2 Training a binary classifier for congestion detection

To detect traffic congestion using the RSUs, we fine-tuned a pre-trained AlexNet model [10] using PyTorch. To do this, we modified the final layer of the model to have 2 outputs instead of the original 1000. In addition, we added a softmax layer to get the probabilities of the classes. We used negative log-likelihood loss and Adam optimizer [9] with the PyTorch default parameters to train the model.

We collected training data for the model using CARLA. To generate the data, we randomly spawned 100 vehicles around the city and collected image data using the RSUs. We ran the simulation for 500 frames, and as a result, we got a total of 2000 images, 500 per RSU. The labels for the images were determined according to the average velocities of the vehicles in an intersection. We calculated the average velocities of the vehicles within 50 meters of the intersection, which were not affected by red or yellow traffic lights. Images were labeled congested if the average velocity was below half the speed limit. Otherwise, as non-congested. We also explored other ways of defining congestion but found that this approach worked the best.

Finally, we divided the dataset into training, validation, and test sets. We assigned 80% of the data to the training set, while the remaining 20% was split equally between the validation and test sets. Undersampling was used to balance the class distribution. This approach resulted in a training set of 532 images for both classes, comparable to previous studies. For example, Chakraborty et al. [3] used a training set with 700 images for both classes, while Kurniawan et al. [11] utilized a training set of 1000 images in their study.

We explored using different batch sizes and the number of epochs in training. To prevent overfitting, we used early stopping to select the model with the highest validation accuracy. In all our experiments, the accuracies varied between 70% and 80%, and the best accuracy of 81.6% was obtained with 10 epochs and a batch size of 32.

3.3 Discrete-event simulation

We used a DES simulation [17] to process the stored CARLA simulation data. This is a form of offline simulation where the data is collected beforehand. This was done to separate the process from CARLA and give us more flexibility with design choices. In our case, our DES simulation models another simulation by analyzing the data of sensors in CARLA to try to reconstruct the scene partially to extract information.

The DES simulation considers the data from CARLA as a network, where each node represents a vehicle, an RSU, or an external processing unit. When the node has finished processing, the simulation time for that node progresses by one unit. In our case, we run the simulation at 30 fps, so the simulation advances by approximately 0.03 seconds. The nodes are capable of communicating with each other using a network, which would allow the simulation of physical network traffic, including bitrate limits and communication delays. However, we did not implement these limitations in our experiments. Figure 3 shows the flow of data in our simulation.

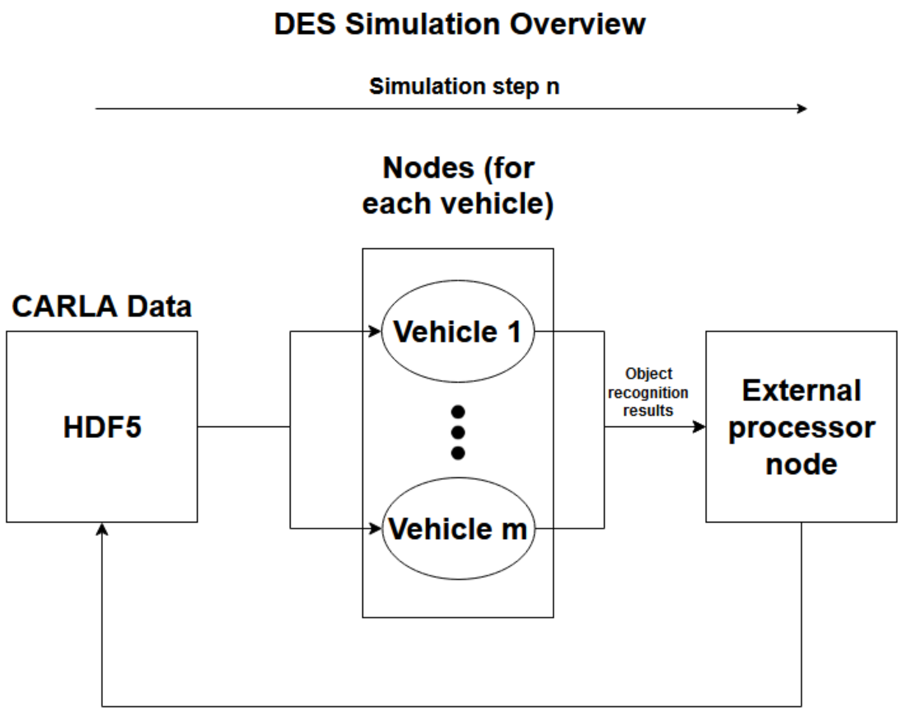


Figure 3: Structure of the discrete-event simulation data flow. At simulation step n, data is passed to corresponding nodes, which process and forward the data to an external processor. The simulation then advances to the next step.

We designed our DES simulation based on Figure 3. The figure shows the simulation at simulation step n, which means we also read data from the CARLA dataset at time step n. The data is then passed to each corresponding node, meaning that if CARLA has a car x, then node x will receive the sensor data from car x. Each node then processes the sensor data to extract information, and this information is then passed to an external processor node, which uses the data to extract information and reconstruct the CARLA scene partially. The output is then visualized to show the current state of the simulation at the current timestep. After this, the DES simulation advances for one time step, and the cycle repeats.

The implementation of the DES simulation was done with the Python library Simpy, which is a process-based discrete-event simulation framework [18]. The DES simulator reads the data from CARLA, one timestep at a time, so the simulation nodes are always synchronized relative to data from CARLA. The simulator processes all nodes during a single timestep before advancing the simulation time. The vehicles and RSUs use the YOLO real-time object detection model [16], specifically the YOLOv5 PyTorch implementation [8]. YOLOv5 has multiple versions of it, with the main difference being model parameter count. There are a total of 5 models, with the smallest model, Nano, having only 12.7 million parameters, while the largest model, YOLOv5x, has 141.8 million parameters. We test all these models to compare both inference speed and quality of results. While these models are pre-trained to predict 80 different classes of objects, we only use predictions for vehicles and pedestrians to understand the environment. From the detections, a 2D world is created that can be analyzed and visualized. Multiple techniques are employed to make this as accurate as possible, and we present them in Sections 3.3.1 and 3.3.3.

3.3.1 From object detection to world position

We estimate the distance of the object with some similarity to the methods presented in the paper "DisNet: A novel method for distance estimation from monocular camera" [13], in which bounding box size, knowledge of the average dimensions of the detected object and camera parameters are used to estimate dimensions. The paper is discussed more in Section 2.6.

We use similar methods to estimate the object position from the ratio of the bounding box and the expected size of the object, with the assumption that further away objects have a smaller bounding box. In our case, width is not used, as we do not know the orientation of the detected object, and height is less likely to change as much. However, in our case, the RSUs are placed high above the ground, and they look slightly down at the vehicles and pedestrians. This causes the distance estimation to produce bad results, as the height of the object may be completely different than it should be due to the angle. The distance based on the measured height is calculated using Equation 1:

$$D = \frac{(H_r \times F)}{H_f}, \tag{1}$$

where D is the distance from the camera, F is the focal length of the camera, H_r is the average height of the object, and H_f is the measured height of the object. All parameters are in centimeters, except H_f, which is in pixels. As there was a bit of ambiguity about what the focal length should be in the CARLA simulator, a good value was found by experimentation. To get the distance between an RSU and a detected object, the Pythagorean theorem has to be applied since RSUs are placed higher, and the DES simulation does not account for height. Now, the height of the RSU is known, along with the estimated distance to the object. With this information, the distance in 2D can be calculated.

Finally, to place the object in the 2D world, the angle between the camera and the detected object is calculated. This is done by calculating the width of the object to find the center of the detection in the horizontal direction. Then the image width is converted to 90 segments, corresponding to 90 degrees, which is the field of view (FOV) in cameras in the CARLA simulator. Then the center of the object can directly be converted to degrees with Equation 2:

$$A = (\frac{D_c}{W_i} \times F) - (\frac{F}{2}), \tag{2}$$

where D_c is the center of the detection in the horizontal direction, W_i is the image width in pixels, and F is the field of view. The resulting angle A would be in the range [0, 90], but as we want 0 degrees to represent forward, the equation, in the end, is subtracted by FOV divided by two, which results in the angle being in the range [-45, 45], where angle 0 is forward. Finally, the coordinates of the detected object can be calculated by using the unit circle to find the position based on distance and angle. Equations 3 and 4 are used to find the X and Y coordinates for the detection, respectively.

$$D_x = C_x + (D_d \times cos(A)) \tag{3}$$

$$D_y = C_y + (D_d \times sin(A)) \tag{4}$$

In these equations, C_x and C_y correspond to the X and Y coordinates of the camera. D_d is the distance to the detected object, and A is the angle in radians. The outputs D_x and D_y correspond to the global coordinates of the detection.

3.3.2 Duplicate detections

The same object can be detected by multiple sensors. When combining the sensor data from multiple vehicles, this could cause the system to believe there are more vehicles or pedestrians than there actually are. This is especially a problem if the accuracy of converting the object detection to world coordinates is bad. This places the same object in multiple places.

To avoid creating duplicates of detected objects, a match analysis is required. This analysis attempts to define if a detected object has already been detected by another sensor. This is done by measuring the distance between the detection, which may have variations in the predicted position and the position of other objects. If the distance between another vehicle and the detection is smaller than a defined threshold, the two entities can be considered to be the same. A threshold is required, as the distance estimation using this method is bound to have some variance and is not exact. However, the threshold cannot be too large, as it is possible for two objects to be close to each other.

3.3.3 Analysis of the environment

When the environment has been processed, the locations of all vehicles (including non-communicating ones), RSUs, and pedestrians are known. The world consists of multiple intersections, and analysis is done for each intersection separately. A car is part of an intersection if the distance to the intersection is within a defined threshold, such as 50 meters. Due to current limitations, information such as detected object velocity is not available and cannot be inferred, as analysis is done for a single simulation timestep at a time. We apply the following two methods to analyze traffic situations.

The first method analyzes potential collisions. This is done very simply by estimating a braking distance of an average car in good conditions. Then the distance to an object, as calculated from the YOLO model detection bounding box, is compared to the estimated braking distance. Currently, the location of the detected object relative to the sensor location is not accounted for, meaning the object does not have to be directly in front of the vehicle. This serves as a simplified collision warning system.

The second method is congestion analysis, which is based on how many vehicles and pedestrians there are in the area and what the average known speeds of those vehicles are. In our case, we analyze congestion at two intersections. The congestion status of an intersection is binary, so it either is or is not congested. However, assigning multiple states or even a probability of congestion would be possible and, with enough accuracy, provides more information for route planning purposes.

In the simulation, an intersection is considered congested if the average speed of vehicles is below a set threshold and there are enough vehicles and pedestrians. Counting vehicles and pedestrians is useful, as an intersection cannot be congested if there are only two vehicles that just happen to be parked or moving slowly, even if they could go fast.

Pedestrians may also cause congestion if they are crossing the street. To avoid a situation where there are too many vehicles considering the intersection capacity but no pedestrians, the algorithm does not strictly require only one of the conditions to be true. Pedestrian and vehicle counts are added together, and if either is high enough and the average speed is below a set threshold, the area will be considered congested.

3.3.4 Inference analysis

Simple inference analysis is also performed to estimate the inference requirements of the DES simulation and the feasibility of trying to run it in real-time with CARLA or running the processing of a single node on real vehicle hardware. The inference analysis was performed on a single computer by estimating hardware utilization during the execution of the DES simulation. The DES simulation also collects information about execution times during the simulation, which can be used to estimate the speed of the simulation as a whole and for each vehicle. For running CARLA in real-time, the system should be able to process a simulation timestep at the rate data is collected, which is 30 fps. If the DES simulation were to process the CARLA data in real-time, it would have to be able to run 30 times a second. This includes all vehicles, RSUs, and external processing nodes. This task will depend on scale, as the processing cost growth is nearly linear to the number of nodes in the simulation. In the case of running the simulation on real vehicle hardware, only one node would have to be able to finish 30 times per second, which is a lot less computationally intensive. The results of our inference requirement analysis can be found in Section 4.

4 Experiments

For our experiments, we use four different scenarios created in CARLA, which were simulated in the DES simulation, to analyze traffic congestion in two intersections separately. The scenarios have 5, 10, 15, and 20 vehicles, and each scenario also has 4 RSUs in total. Each scenario is in the same city that has two intersections with 2 RSUs in each. The scene also includes pedestrians that walk on the sidewalks and cross the roads. The intersections have varying levels of congestion, which change during the simulation. Each vehicle is equipped with a single front-facing camera, and the simulation was run for 1000 steps at 30 frames per second. The camera resolutions were set to 640x640 pixels.

The DES simulation has multiple parameters that can be fined tuned, such as what is the threshold for entities to be inside an intersection, what YOLO model is used, and what is the average speed threshold to consider an intersection congested. These parameter values were found by experimentation by comparing results. For instance, the different YOLOv5 models were compared to find the optimal results between speed and accuracy. The system has to be fast, preferably real-time, but still must be accurate enough to create consistent detections. Figure 4 shows a comparison of the model run times to understand how the simulation and model sizes affect it.

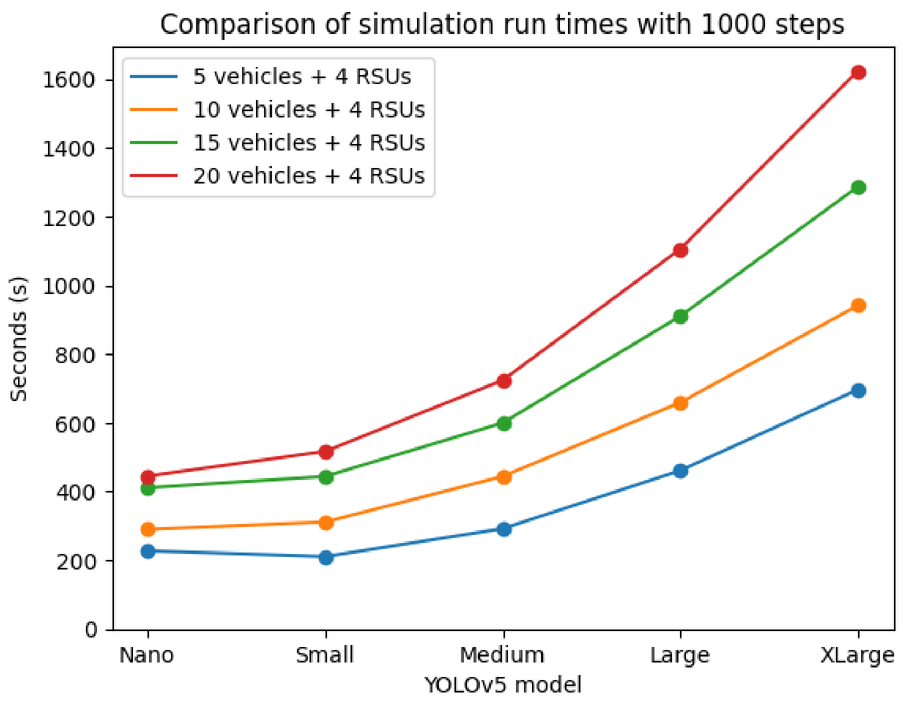


Figure 4: Performance comparison of different size YOLOv5 models.

The data indicate different runs of the simulations, with varying models and simulation sizes. Vehicle and RSU counts are indicated separately, though, in practice, they are processed mostly similarly. This is why we did not perform a comparison of the simulation without RSUs. Model accuracy also varied slightly, especially in harder conditions, such as RSUs looking down at cars. RSUs were a special case, as they looked at the traffic from a high angle, which caused some issues in detecting vehicles with smaller models. Figure 5 shows an example of such a case, with all YOLOv5 models compared.

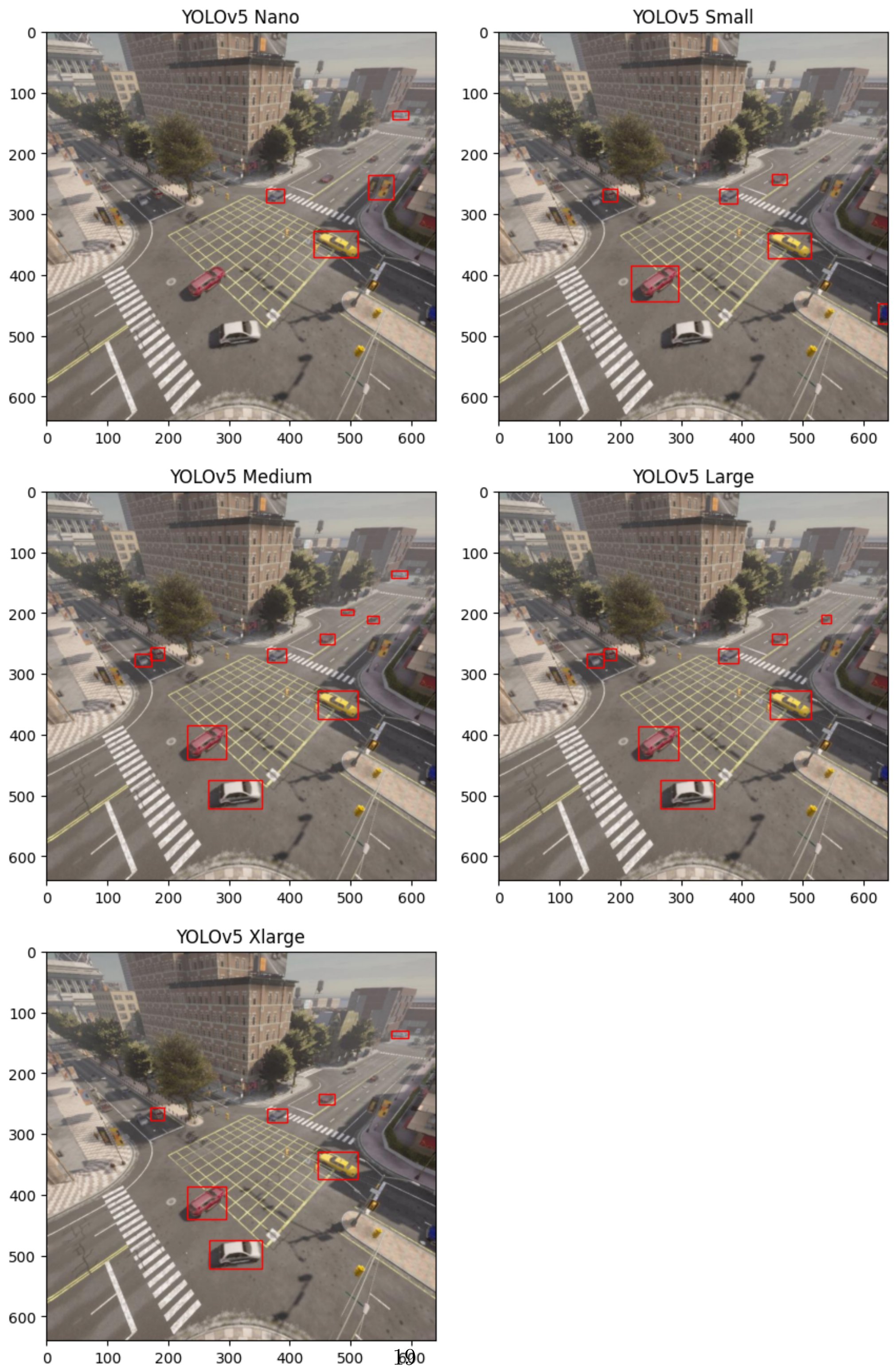
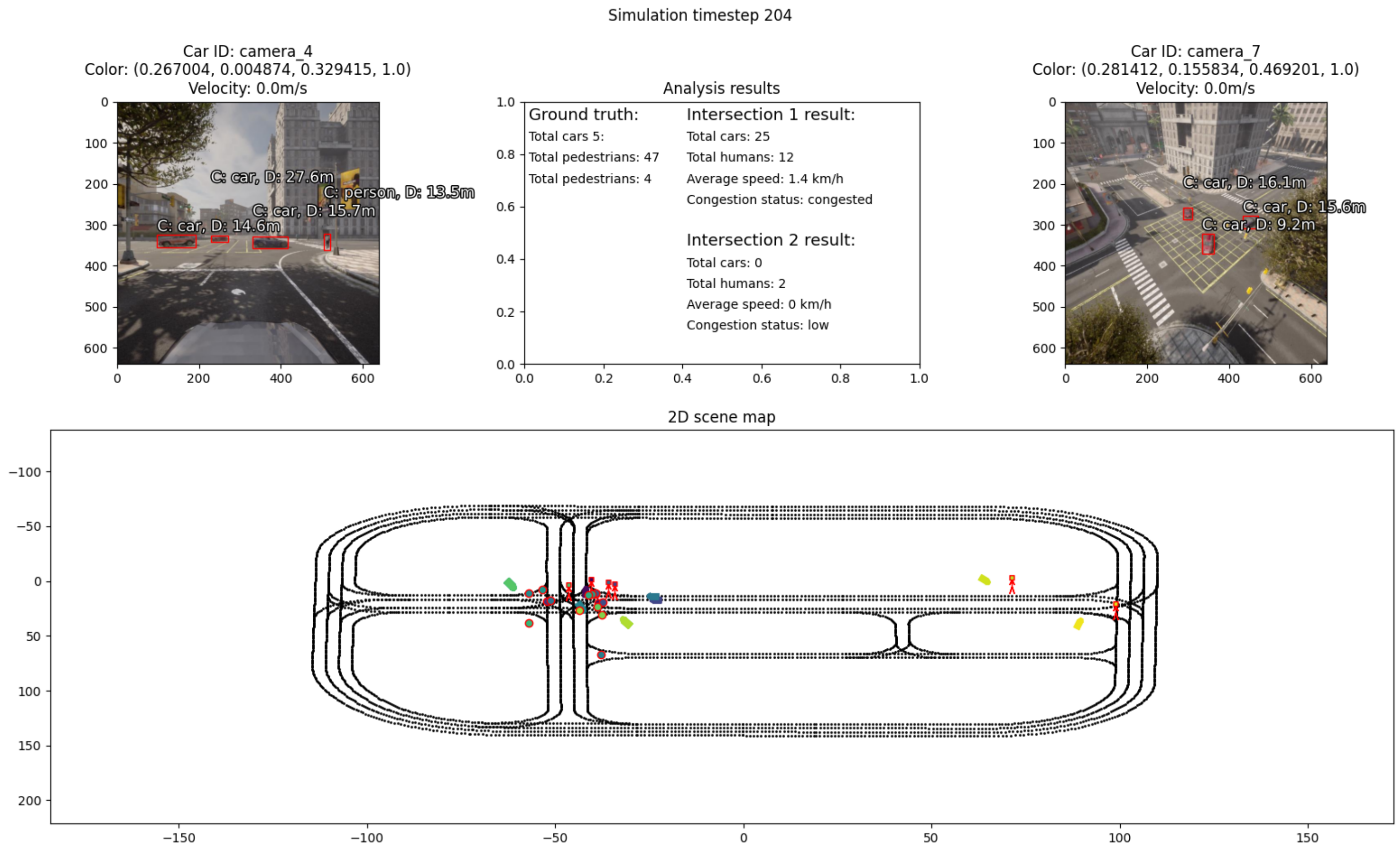


Figure 5: Accuracy comparison of different size of YOLOv5 models.

The Nano model misses two cars that are in the middle of the intersection, likely due to the angle of the vehicles. The Small model is able to detect the other car, but not the white one. The Medium model is the first model to be able to detect all vehicles in the scene. The two largest models perform similarly, though in general, in our tests, bigger models have produced more accurate results. However, in most cases, the Medium model brings sufficient accuracy without increasing run time too much. For this reason, the Medium model is what will be used in further tests unless mentioned otherwise.

The results produced by the simulation are sufficient for analysis but lack the accuracy required to ensure that the same vehicle or pedestrian is not counted multiple times, as multiple different sensors can detect it. This can be tuned by changing how far the detection and a previous detection or a known vehicle have to be before we consider them one. However, the results are slightly inaccurate, which can be seen in the below Figure 6, which is a visualization of the DES simulation results, which show that there are many more vehicles than there actually are (ground truth).



```
Ground truth:
Total cars 5:
Total pedestrians: 47
Total pedestrians: 4

Intersection 1 result:
Total cars: 25
Total humans: 12
Average speed: 1.4 km/h
Congestion status: congested

Intersection 2 result:
Total cars: 0
Total humans: 2
Average speed: 0 km/h
Congestion status: low
```

Figure 6: Visualization of the DES simulation results.

The visualization has four images. The top-left and top-right windows show images taken by two nodes, a vehicle, and an RSU, at simulation timestep 204. The image in the top-center shows data ground truths and DES simulation analysis results. The bottom image shows a 2D top-view map of the city, where black dots indicate roads. Circles indicate detections, human-like markers indicate pedestrians, car-shaped markers show vehicles, and rectangles show RSUs. Car and RSU markers are also rotated to show which direction they are facing. While the YOLO bounding boxes are mostly accurate and entities are placed on the map approximately correctly, the inaccuracy of the simulation causes the system to think that there are many more cars and pedestrians than there actually are, as multiple cameras observe the same entities. One large issue is the RSUs, which are placed far higher than vehicle sensors. The bounding box calculation does not work well, as the height of the bounding box does not describe the height of the detected object due to the angle.

However, the system has now produced a 2D world, which can be used for analysis, such as detecting if a pedestrian is in front of the car within braking distance, which adds a collision warning text to the detection annotation. This information could be used to control the car in an emergency.

Table 1 compares the congestion detection of the DES simulation and the CNN-based congestion detection model. The real congestion status (ground truth) is defined by the average speed at the intersection. Vehicles within 50 meters of the intersection are considered to be part of the intersection. The speed limit in the CARLA simulation is 30km/h. So if the average speed in the intersection is less than 15km/h, the intersection is considered congested. This is a very simplified way of determining the ground truth and is not the definitive truth. However, the CNN model data were labeled based on the ground truth, which allows comparing the accuracy of the model in both intersections. The data uses the CARLA simulation with 20 vehicles and 4 RSUs.

	Intersection 1	**Intersection 2**
"Ground Truth"	59.3 %	35.4 %
CNN Model	79.1 %	36.9 %
Simulation	99.9 %	9.5 %

Table 1: Comparison of percentage of simulation time the intersection was congested with different estimation methods.

The results presented in Table 1 give a good overview of the congestion status in the intersections. The CNN-based model and ground truth results are similar, which is expected, as the ground truth is how the CNN model training data was labeled. However, the CNN model is not perfectly accurate, so variation in results is expected. The simulation results differ greatly, but it is expected, as the simulation has more information and defines congestion differently. The simulation also accounts for the number of vehicles and pedestrians in the intersection, which may explain the difference in results. During most of the CARLA simulation, intersection 1 had a lot more vehicles and pedestrians than intersection 2, which was mostly empty or only had a few vehicles and pedestrians. This is also what the results indicate for all congestion estimation methods.

The inference requirement estimation was performed on the same hardware for all experimentations. The experiments were run on a Windows 10 PC with an Intel Core i5 12400f processor, 32 gigabytes of DDR4 RAM, and an NVIDIA GeForce RTX 3070 GPU. The simulation run time comparison of Figure 4 shows the results of comparing inference speed with CARLA environments of various sizes and different YOLOv5 models. For estimating hardware usage, the Windows performance tools and the CPUID HWMonitor software were used to collect data and verify the results. During experimentation, the actual hardware utilization on our hardware, with the Medium YOLOv5 model and a scenario with 20 vehicles and 4 RSUs, were as follows. RAM usage was consistently around 3GB. GPU utilization varied a lot but maxed out around 75% usage, and GPU memory utilization was around 3GB. CPU utilization was consistently less than 30%. However, these are only estimations, and no clear bottleneck was identified.

To measure processing speed of the simulation, we can measure time it takes to run a single node and the whole simulation. Figure 4 shows how the number of processable entities increases the processing requirements. The worst case scenario, with the largest YOLOv5 model and CARLA simulation with 20 vehicles and 4 RSUs, took around 1600 seconds to process all 1000 simulation steps. This means that each simulation step took around 1.6 seconds to complete, which would not be fast enough to process the CARLA simulation in real-time. It would result in less than 1 fps, and currently the data is collected at 30 fps. However, when using the Medium YOLOv5 model, the processing took around 700 seconds. That equals 0.7 seconds per simulation step, which would allow us to process the simulation at over one fps.

However, the inference requirements are far less demanding for processing the sensor data of a single vehicle. There were 24 processable nodes, which means that for one vehicle, the approximate processing time is 0.7 seconds divided by the number of nodes. A single vehicle would have taken around 0.03 seconds to finish processing, which would allow for around 33 frames per second, which is sufficient, as data was collected at 30 fps.

5 Discussion

In this section, we go through the weaknesses of our CongestionSim system and propose different improvement ideas. Section 5.1 discusses the issues with our data labeling approach and suggests different alternatives. In Section 5.2, we discuss the flaws and potential improvements for the DES simulation.

5.1 CNN-based classifier

When collecting images for the CNN-based binary classifier, as discussed in Section 3.2, we found that using the average velocities of vehicles in an intersection is a too simplified way of determining whether an image of an intersection is congested or not. Our labeling approach works quite well for congested scenarios but often mislabels non-congested images as congested. For example, consider a situation where a vehicle is in an intersection and has to slow down because a pedestrian is crossing the street. In our approach, the resulting image gets a congested label since now the average velocity of the vehicles in the intersection is small. An example of this kind of data is shown in Figure 7.

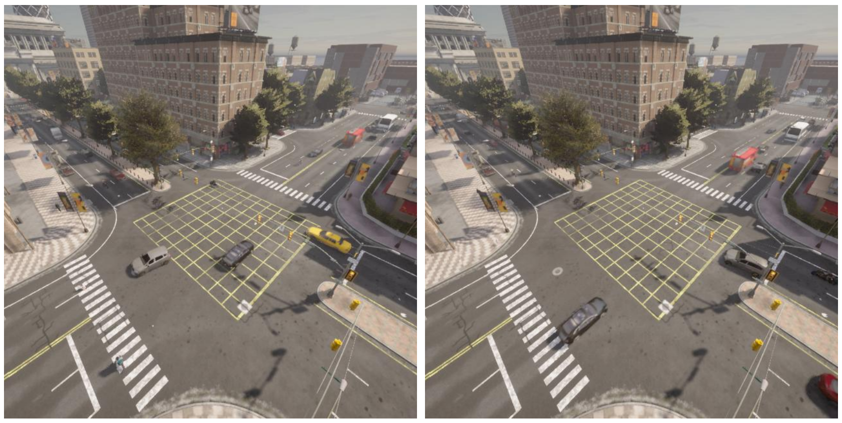

(a) Image that gets congested label using our approach. (b) Image that gets non-congested label using our approach.

Figure 7: Illustration of a case where our labeling approach fails. On the left-hand side, a vehicle has to slow down because a pedestrian is crossing the street, resulting in a lower average velocity. This makes the classification task hard since the images are so similar.

An alternative approach could be to use the traffic density to label the images. However, in this approach, an image of an intersection with many vehicles would get a congested label, even if the traffic flow is smooth. Another approach could be to label the images manually. This approach has the disadvantage that different people may have different standards for traffic congestion, as discussed by Wang et al. [19]. On the other hand, labeling hundreds of images manually is a tedious process, and it can be difficult to label the images consistently due to fatigue. However, despite the issues with the labels, we were able to get an accuracy relatively close to previous studies [3, 19, 11]. In future studies, we could try labeling the data using additional radar sensors, as was done in [3]. Using a more advanced image classifier than AlexNet would also likely increase the classification accuracy.

5.2 DES simulator

The DES simulator is a viable choice for simulating the data from CARLA, as its sequential nature helps in processing and visualizing the data without parallel processing capabilities. The results were slightly inaccurate but were still good enough for analyzing the world around the vehicles. The YOLOv5 models performed well in both accuracy and inference speed, and we were able to calculate the 2D position of detections with good accuracy. The Medium model was selected for experimentation due to the good balance of inference speed and accuracy. However, the DES simulation accuracy was not enough to ensure that the same object is not detected multiple times by different cameras, thus creating objects in the intersection that do not actually exist. The system tries to match newly detected entities to existing entities if they are within a specific distance, but the threshold would have to be even bigger to remove duplicate detections, which would cause the system to completely miss entities, such as a human walking very close to a car.

The DES simulator is currently restricted to just analyzing the situation at intersections, but in theory, the analysis could be performed for any area within the city or any part of a vehicle's route from point A to B. However, this could increase processing requirements, and the distance between external processing nodes could grow. In that case, the solution should account for the possible increase in time to communicate between a vehicle and an external processing node, as in our situation, we did not simulate a delay in data transfer, as the assumption is that in a city, the sending and receiving of data would be fast enough for it not to matter.

5.2.1 Inference requirements

In our experiments, we were able to run one step of the simulation in 0.7 seconds. It would allow running the simulation at around 1 fps. That would not be enough for processing that requires a fast response, such as real-time crash detection. However, for processing congestion, one frame per second should be enough. Of course, it should be noted that the scale of the CARLA simulation affects the run time greatly, as demonstrated by Figure 5. This would need to be accounted for in cases of extreme amounts of nodes. Perhaps the object recognition model sizes could be adapted based on simulation speed. In any case, running the full simulation in real-time with our hardware is slow, and either more processing power is required or the efficiency of the system would need to be improved.

Our experiments allowed us to process a single node at over 30 fps. This is sufficient for running a single node in real-time with the CARLA simulation. The Medium YOLOv5 model provides a good balance between speed and accuracy, which is why it was used for experimentation. Larger models may not be fast enough to process the node in real-time. Luckily the scale of the simulation does not greatly matter, as the node only has to process itself. Only external processing nodes are affected by the scale of the simulation, as data has to be combined from more nodes. However, heavy lifting can be done by vehicles, and the processing power of an external processing node can be increased as needed. Alternatively, more of them can also be installed in a busy area. These external processing nodes could also communicate results to each other to understand traffic in a larger area, such as the entire city.

5.2.2 Improvement ideas

While creating this system, we found many problems with our method that do not allow for further accuracy. The YOLO detections are lacking, especially if the cars are at an angle to the camera. This is especially noticeable with the RSUs, which are located higher than the vehicles. The objects may not be detected, and if they are, the bounding box heights do not work well with the way distance is calculated from the object's detected height. A potential slight improvement would be to account for this by estimating what the angle between the RSU and the vehicle is, but without knowing the orientation of the detection, this would be hard. A great solution to this would be to detect a 3D bounding box, which would include the orientation of the object. This would not only improve distance calculation greatly but would also allow the system to estimate which way an object is moving and visualize it properly. One example of such a method is presented in the paper Joint Monocular 3D Vehicle Detection and Tracking [6], which works in a similar setting and shows great detection results.

An alternative method would be to use a model that detects the bounding boxes of objects but also estimates the distance between the camera and the object. Dist-YOLO is an improvement on the YOLO model that predicts the distance of the detected object from a photo taken with a monocular camera [20]. This could possibly replace the manual distance calculation and increase the accuracy of results.

The methods above would likely produce great results but cause inference times to increase. An approach that would make the YOLOv5 model more accurate and possibly faster would be to fine-tune it. The fine-tuned model would only need to predict the objects we are interested in, which should increase inference time and accuracy. However, this would not solve the issue of calculating distance when RSUs are a lot higher than vehicles. Also, the improvement would likely be mostly for inference speed. Another method could be inference optimization methods, such as 8-bit quantization or other methods, that increase inference speed but decrease precision [5].

Another flaw is that the system analyzes only one timestep at a time. This means there is no consistency between frames, and we are unable to determine temporal information, such as the trajectory of a vehicle. The consistency between frames shows up in the visualization, which causes detections to disappear and re-appear across frames. By including previous frames in the data, we could attempt to track objects, which would greatly increase the consistency of the results. Also, with object tracking, we could attempt to average their positions across frames, causing smoother changes in position.

By applying the above improvements to the DES simulation, the resulting system would be more demanding during inference but would produce more accurate results and analyzing opportunities. If we were able to infer the orientation and speed of the vehicle, we could estimate trajectories and collision warnings would be far more accurate. Even more importantly, congestion detection could account for the average speed of all vehicles, not just the ones that communicate with each other or to an external entity.

6 Conclusion

Our primary goal was to analyze traffic congestion using vehicles with sensors and RSUs, which communicate to external processing node that attempts to understand the world around the vehicles. From this understanding, information, such as how congested an intersection in the city is, could be extracted. We compared methods such as vehicle-to-vehicle and vehicle-to-infrastructure communication and design choices related to these systems.

We then implemented a system for simulating city traffic and extracted sensor data, which we ran in a DES simulation. The DES simulation re-created a 2D map of the city by using image detection to place detected vehicles and pedestrians on the map. The map was useful for analyzing potential collisions and traffic congestion, but it had some inaccuracies and duplicate detections. This occurred when an object was detected by multiple sensors. However, the results were still good enough for analysis, which gave positive results for congestion detection accuracy.

We estimated inference requirements for our system. We concluded that the simulation, with our current hardware, would not be able to run in real-time, at least not at 30 fps. However, running a single node at 30 fps would be possible. We also experimented with different YOLO models to find a good mix of speed and accuracy, which resulted in us using the Medium YOLOv5 model. For running the whole simulation, either the system would need to be more efficient or more computational power is needed.

The CNN-based classifier turned out to be a viable solution for detecting congestion using image data. The biggest issue we faced was the difficulty of defining congestion. Our average velocity-based labeling approach turned out to be a too simple way of defining congestion, as discussed in Section 5.1. Despite the issues with the labels in the training data, we were able to get an accuracy of 81.6%, which is close to previous studies.

We discussed the advantages and disadvantages of our methods and suggested various improvements to them. The main considerations were the difficulty of defining traffic congestion, the inaccuracy of the information provided by our machine learning methods, and the problem of more complex systems having increased processing complexity and run times, especially as the amount of data increases.

References

[1] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B. Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M. Paixão, Filipe Mutz, Lucas de Paula Veronese, Thiago Oliveira-Santos, and Alberto F. De Souza. Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816, 2021. URL: `https://www.`

sciencedirect.com/science/article/pii/S095741742030628X, doi:https://doi.org/10.1016/j.eswa.2020.113816.

[2] Ramon Bauza, Javier Gozalvez, and Joaquin Sanchez-Soriano. Road traffic congestion detection through cooperative Vehicle-to-Vehicle communications. In *IEEE Local Computer Network Conference*, pages 606–612, 2010. doi:10.1109/LCN.2010.5735780.

[3] Pranamesh Chakraborty, Yaw Okyere Adu-Gyamfi, Subhadipto Poddar, Vesal Ahsani, Anuj Sharma, and Soumik Sarkar. Traffic Congestion Detection from Camera Images using Deep Convolution Neural Networks. *Transportation Research Record*, 2672(45):222–231, 2018. doi:10.1177/0361198118777631.

[4] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938, 2017. URL: http://arxiv.org/abs/1711.03938, arXiv:1711.03938.

[5] Yury Gorbachev, Mikhail Fedorov, Iliya Slavutin, Artyom Tugarev, Marat Fatekhov, and Yaroslav Tarkan. Openvino deep learning workbench: Comprehensive analysis and tuning of neural networks inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Oct 2019. URL: https://openaccess.thecvf.com/content_ICCVW_2019/papers/SDL-CV/Gorbachev_OpenVINO_Deep_Learning_Workbench_Comprehensive_Analysis_and_Tuning_of_Neural_ICCVW_2019_paper.pdf.

[6] Hou-Ning Hu, Qi-Zhi Cai, Dequan Wang, Ji Lin, Min Sun, Philipp Krahenbuhl, Trevor Darrell, and Fisher Yu. Joint Monocular 3D Vehicle Detection and Tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. URL: https://openaccess.thecvf.com/content_ICCV_2019/papers/Hu_Joint_Monocular_3D_Vehicle_Detection_and_Tracking_ICCV_2019_paper.pdf.

[7] Aleksi Hämäläinen and Elias Haaralahti. CongestionSim. Accessed 14/4/2023: https://github.com/EliasHaaralahti/congestion-sim, 2023.

[8] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, (Zeng Yifu), Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation. Accessed 14/04/2023: https://pytorch.org/hub/ultralytics_yolov5/, November 2022. doi:10.5281/zenodo.7347926.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 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. URL: `http://arxiv.org/abs/1412.6980`.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL: `https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[11] Jason Kurniawan, Sensa G.S. Syahra, Chandra K. Dewa, and Afiahayati. Traffic Congestion Detection: Learning from CCTV Monitoring Images using Convolutional Neural Network. *Procedia Computer Science*, 144:291–297, 2018. INNS Conference on Big Data and Deep Learning. URL: `https://www.sciencedirect.com/science/article/pii/S1877050918322397`, `doi:https://doi.org/10.1016/j.procs.2018.10.530`.

[12] M. Rezwanul Mahmood, Mohammad Abdul Matin, Panagiotis Sarigiannidis, and Sotirios K. Goudos. A Comprehensive Review on Artificial Intelligence/Machine Learning Algorithms for Empowering the Future IoT Toward 6G Era. *IEEE Access*, 10:87535–87562, 2022. `doi:10.1109/ACCESS.2022.3199689`.

[13] Danijela Ristić-Durrant Axel Gräser Muhammad Abdul Haseeb, Jianyu Guan. Disnet: A novel method for distance estimation from monocular camera. 2018. URL: `https://project.inria.fr/ppniv18/files/2018/10/paper22.pdf`.

[14] Martin Raubal Nishant Kumar. APPLICATIONS OF DEEP LEARNING IN CONGESTION DETECTION, PREDICTION AND ALLEVIATION: A SURVEY. 2021. URL: `https://arxiv.org/pdf/2102.09759.pdf`.

[15] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks, 2015. URL: `https://arxiv.org/pdf/1511.08458.pdf`, `arXiv:1511.08458`.

[16] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*, abs/1506.02640, 2015. URL: `http://arxiv.org/abs/1506.02640`, `arXiv:1506.02640`.

[17] Prateek Sharma. Discrete-Event Simulation. *International Journal of Scientific & Technology Research*, 4:136–140, 2015. URL: `https://www.ijstr.org/final-print/apr2015/Discrete-event-Simulation.pdf`.

[18] Klaus Müller Stefan Scherfke, Ontje Lünsdorf and Tony Vignaux. Simpy. Discrete event simulation for Python. URL: `https://simpy.readthedocs.io/en/latest/`.

[19] Ping Wang, Li Li, Yinli Jin, and Guiping Wang. Detection of unwanted traffic congestion based on existing surveillance system using in freeway via a CNN-architecture trafficnet. In *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 1134–1139, 2018. `doi:10.1109/ICIEA.2018.8397881`.

[20] Yu Zhang, Zhongyin Guo, Jianqing Wu, Yuan Tian, Haotian Tang, and Xinming Guo. Real-Time Vehicle Detection Based on Improved YOLO v5. *Sustainability*, 14(19), 2022. URL: `https://www.mdpi.com/2071-1050/14/19/12274`, `doi:10.3390/su141912274`.