MASTER'S THESIS 2023

Generating Synthetic Scenarios to Test an AI-Enabled Traffic Measurement System

Elias Sjöberg

ISSN 1650-2884
LU-CS-EX: 2023-44

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-44

Generating Synthetic Scenarios to Test an AI-Enabled Traffic Measurement System

Simulering av trafiksituationer för testning av ett AI-system

Elias Sjöberg

Generating Synthetic Scenarios to Test an AI-Enabled Traffic Measurement System

Elias Sjöberg
fys13esj@student.lu.se

November 2, 2023

Master's thesis work carried out at RISE Research Institutes of Sweden.

Supervisor: Markus Borg, markus.borg@cs.lth.se
Examiner: Per Runeson, per.runeson@cs.lth.se

Abstract

To make the future of the transport system efficient, safe and sustainable, understanding of the complexities of traffic is vital. Viscando provides detailed traffic data by utilizing Machine Learning and computer vision algorithms in their infrastructure sensor OTUS3D. The purpose of this study was to test OTUS3D using simulated traffic scenarios. Using the MathWorks package RoadRunner, we created a digital model based on a real-life junction. This digital model was imported into the open-source traffic simulator CARLA. We developed two models to generate parameter values that were used to create traffic scenarios: one of the models used random sampling while the other was based on the Genetic Algorithm NSGA-II. Due to time limitations, it was not possible to create a sufficient number of simulated scenarios to properly evaluate the two models. However, some conclusions could be drawn regarding which traffic scenarios were particularly challenging for OTUS3D to estimate.

Keywords: software testing, machine learning, genetic algorithms, simulators, automotive

Acknowledgements

I would especially like to thank Markus Borg for all the advice, feedback and support given, as well as his enthusiasm throughout this process.
I would also like to thank Viscando for making this work possible, particularly Yury Tarakanov for his feedback and for the rewarding discussions, and Ulf Erlandsson for his help in processing the scenarios.

Contents

Chapter 1
Introduction

In recent years, to say that there has been a lot of hype for Machine Learning (ML) solutions to engineering problems would perhaps be an understatement. There is no doubt however, that the use of Deep Neural Networks, coupled with more powerful hardware and a seemingly ever increasing ability to gather and store data, has had a significant impact on numerous research fields. One example of such a field is computer vision, where recent developments have radically improved the ability for machines to automatically detect, classify and track moving objects. While such developments certainly enable the development of exciting and useful innovations, especially in the field of automated driving, there are also novel challenges that arise in their wake. Of primary concern to this report is the problem of performing quality assurance testing on systems that are based on computer vision and ML. For traditional software, dividing the system under test into smaller components that can be tested individually is often an efficient and effective strategy to find possible improvements. The black box nature of ML systems, where performance is the result of a complex set of connections between the different components, makes this approach infeasible in most cases. Instead, other methods must be considered. The search for such methods of testing different aspects of ML systems has developed into a field of research in itself with a wide range of different approaches [28], [24]. Since this thesis is concerned about quality assurance testing of an ML system, it was decided that a suitable approach would one based on Search-Based Software Testing (SBST).

In the description of the 14th Intl. Workshop on Search-Based Software Testing [1], SBST is simply defined as "the application of optimizing search techniques (...) to solve problems in software testing". The search technique can be performed in a multitude of ways; what is important is that there is one or more *objective functions* (sometimes called *fitness functions*) that can be used to evaluate candidate solutions. When testing a piece of software, it is common that the objective function takes the form of an error measurement that results from some test input, and the goal is to use SBST to generate *critical input* that maximizes this error [28]. When it comes to search techniques, some of the more popular ones - that are also used in

this report - are based on Genetic Algorithms (GAs). In brief terms, GAs can be used to find critical input parameters in software by initializing a diverse set of parameter values and then letting the ones that have the greatest potential of provoking system faults reproduce while the others are removed. GAs will be presented and explained in depth in Chapter 2.

There is one more challenge regarding quality assurance testing of ML systems that should be mentioned since it is highly relevant to this report: the difficulty of finding a *test oracle*, i.e. a procedure that distinguishes between correct and incorrect behavior of the system under test by comparing the output to some ground truth. The simple reason behind this difficulty is that ML is frequently used to solve problems where no other method produces satisfactory results. This challenge, which also exists in non-ML systems, is referred to as *The Oracle Problem* in the literature [5]. In the case of this report, The Oracle Problem was solved by performing tests in a simulator. This will be further explained below, but first we will have a closer look at the problems that form the basis of this report.

In order to make future transportation systems efficient, safe and sustainable, having access to detailed and accurate traffic data is vital. With the help of the already mentioned ML-enabled solutions that have been developed in the field of computer vision, the company Viscando is able to provide such data. Through use of DNNs, Viscando's solution OTUS3D is able to segment, classify and track distinct objects in traffic; including cars, vans, pedestrians and bicyclists. This data collection makes it possible to capture the complex dynamics that can occur in traffic situations. The data can be used to for example discover traffic safety issues, to increase perception for autonomous and automated vehicles and to analyze how to prevent occlusions and improve efficiency on roads. Based on the discussion on ML quality assurance testing presented above, one might ask how a system like OTUS3D can be tested. Are there specific traffic scenarios, perhaps coupled with certain weather conditions or a certain time of day, that are more prone to provoke system faults? Note that the traffic scenarios mentioned here correspond to the critical input mentioned in the section introducing SBST, while provoking system faults corresponds to maximizing objective functions, in this work defined as differences between ground truth and estimations made by OTUS3D.

The testing environment was implemented in the open-source traffic simulator CARLA. Created with the intention of being used for development of autonomous driving systems, CARLA contains a large amount of functionalities, some of which include pre-made 3D scenes, assets for different types of vehicles and human walkers and many different types of sensors that can be used to collect data. Furthermore, CARLA is built upon Unreal Engine 4 which enables state-of-the-art rendering quality and realistic physics. It is designed as a server-client system where the server runs the simulation and renders the scene while the client sends commands to the server and receives sensor readings [15]. As was briefly mentioned above, one advantage with performing tests in a simulator is that ground truth is easily collected. Another advantage is that multiple traffic scenarios can be tested efficiently by a simple tuning of the input parameters, an approach that works well with the use of GAs. This tuning can easily be done in the CARLA simulator, as will be further explained in Chapter 3.

To make the simulated scenarios as relevant to the testing as possible, we created a low-

fidelity digital model of a real-life junction and imported it into CARLA. The junction that the digital model was based on is located in Lindholmen, Gothenburg, where OTUS3D has gathered data. All of the simulated scenarios were created in the environment of this digital model.

1.1 Research questions

The thesis aims to answer the following questions:

RQ1: Can a low-fidelity digital model of a traffic junction be used to find faults in Viscando's system?
RQ2: Is SBST an efficient and effective approach to provoke system faults in the digital model?
RQ3: Which parameter configurations are the most effective at provoking system faults?

1.2 Contributions and related work

Previously, there has been extensive research where simulators and SBST have been used to test different aspects of Advanced Driver-Assistance Systems (ADAS). The studies have mainly been centered around testing pedestrian detection systems and automatic emergency braking [6], [7], [16], but other aspects have also been tested; one example includes a lane-keeping system [22]. It should be noted that these studies have in common that they all found success in using some variant of the GA algorithm NSGA-II to create critical traffic scenarios, which is also the algorithm that is used in this report. The main novelty of this work is that it does not aim to test an ADAS but instead a system that collects traffic data. One consequence of this was that novel objective functions had to be formulated; we decided to base them on the errors in estimated positions, speeds and classifications.

Rosique et al. performed a systematic review on perception systems and simulators for autonomous vehicles research [25]. For perception systems, they found that there was a clear need to combine the advantages of different types of sensors and systems such as millimeter wave RADARs, 3D LiDARs, cameras and Real-Time Kinematic systems in order to produce adequate results. In order to combine the different types of sensor data, specific algorithms that perform data fusion were frequently used. More important to this report was the second part of the literature review concerning simulators. Rosique et al. found that the use of simulations to test the perception system of autonomous vehicles is in most cases essential. The reason for this is economic and legislative obstacles that in many cases make comprehensive real-world testing impossible. Moreover, they found that while robotics simulators were frequently used in the field of research, they are not always able to provide sufficient realism. Thus, there is a recent tendency to develop more advanced simulators that use realistic 3D rendering and advanced physics engines.

CARLA is a popular traffic simulator, introduced by Dosovitskiy et al. [15]. In their paper, they describe how CARLA has been developed specifically to support development, training

and validation of autonomous driving systems. To showcase its capabilities, they used the simulator to train and test systems using three different approaches to autonomous driving: a classic modular pipeline, a deep network trained by imitation learning and a deep network trained by reinforcement learning. They found that CARLA was capable of delivering results that could be further analyzed.

State-of-the-art perception systems rely on ML, which introduces novel challenges from a testing perspective, as was mentioned previously. Zhang et al. presented a review of the emerging research field [28]. In the review, they divide the field of ML testing research into four categories: testing of workflow (referring to for example input generation, test adequacy evaluation and other properties that have to do with *how* to conduct ML testing), testing of properties (i.e. correctness, robustness and other properties that have to do with *what* to test), testing of components (such as the data or the learning program) and testing of application scenarios. They present the current research status for each of these categories and for ML testing in general. They found several challenges and research opportunities in the field. Most notable for this report is their observation that test input generation and The Oracle Problem both present challenges that merit further research. In the case of test input generation, Zhang et al. highlight SBST as a promising technique.

Afzal et al. conducted a systematic literature review on SBST for non-functional system properties [4]. They found that most of the studies used SBST to investigate execution time, security and usability of software. Moreover, GAs were the most widely used search techniques, while other techniques such as simulated annealing and particle swarm optimization were used less frequently. The authors conclude that there is plenty of future potential for testing non-functional system properties using search-based approaches, and that more studies are needed to draw more generalized conclusions about the benefits of different approaches.

Borg et al. compared SBST for an automotive pedestrian detection system in two simulated environments, PreScan and ESI Pro-SiVIC [8]. They found that while SBST could be used to effectively detect failure revealing test scenarios in both simulators, the specific type of scenarios found between the two simulators differed. This lead them to two conclusions: that using multiple simulators when testing an automotive pedestrian detection system is beneficial, as is using fitness functions that are minimally affected by implementation details in a particular simulator.

Chapter 2
Background

We introduce Evolutionary Algorithms and their close relatives Genetic Algorithms, and describe the parts that make up the latter. We also introduce multiobjective optimization and the Genetic Algorithm NSGA-II.

2.1 Evolutionary Algorithms

As one might suspect, the concept of Evolutionary Algorithms is inspired by nature, or more specifically by the principle of "survival of the fittest" found in classical Darwinian evolution. Given a set of candidate solutions (sometimes the word 'individual' is used instead of 'solution') to some problem, the idea is to select the most 'fit' solutions to 'reproduce' in order to create new candidates. The least fit solutions do not reproduce and are eventually removed from the population. The expectation is of course that 'child' solutions created through reproduction are similar to their 'parents', so that the selection process leads to a population of fitter individuals as the number of generations grow. 'Fitness' is here defined by a numerical value that is attained by applying a fitness function (also known as an objective function) to some solution. Reproduction is typically divided into two parts: crossover and mutation. In a crossover operation, there is an exchange between two parent solutions to create one or more children. In a mutation, random perturbations are made to an individual solution.

Perhaps the first important aspect that one has to decide on when creating an Evolutionary Algorithm is the *representation* of the candidate solutions. In this report, all solutions are represented as vectors containing a number of parameter values. The parameter values are either of floating point type or of integer type. Thus, when the word 'solution' or 'individual' is used in this report, it should always be assumed that it is a one- or multidimensional vector that is being referred to. Furthermore, a solution vector of length n can be viewed as a point in what will be referred to as the n-dimensional *solution space*. It is worth noting that aside from vectors, there are many other ways of representing the solutions; two examples covered

by Luke in [21] are tree representations and graph representations.

Genetic algorithms (GAs) are one form of Evolutionary algorithms. They were first conceived in the 1970s, when the algorithm that is today often referred to as the 'simple GA' was developed [17]. This first GA differs in a few ways from the GAs used in this report; one important difference is that in early GAs, the candidate solutions were often binary coded (all parameter values were either 0 or 1). Exactly what differentiates GAs from other Evolutionary Algorithms is not always clear and will not be covered in depth here. Traditionally, one special property of GAs is that they replace the entire parent population by the child population in a single generation [17], [21] however as we will see below, this is often not the case in more recent GAs.

For the purposes of this report, we will assume that a GA can be decomposed into the following six steps:

1. Mutation
2. Crossover
3. Parent selection
4. Existence of elitism
5. Number of objective functions
6. Evaluation

Unsurprisingly, there is a large amount of existing theory about each of these components. This is not least true for mutations and crossovers, where a vast amount of different methods have been developed over the years [20]. A few of these methods are described in the following sections, where we briefly discuss all of the above mentioned components of a GA.

2.2 Mutation

The point of the mutation operator in a GA is to increase the variation in the population by modifying one or more parameter values in a solution. For applications of the original GA that uses a binary representation of the individuals, the most common method of mutation has been to simply 'flip' each bit with a probability of p_m [17]. Thus, if one bit in an individual has a value of 1, there is a probability of p_m that it will be changed to a zero after the mutation, and analogously if the bit has a value of 0. For applications that use real-valued representations, it is common to use methods that in principle are similar to this 'bit-flip' operator. In what is called a *Gaussian mutation*, the mutation probability p_m works like before, but instead of flipping a bit we make a zero-mean Gaussian distributed perturbation to the parameter value. The idea is that we want to encourage making small changes to parameter values, while larger changes are made rare but not impossible. Exactly how rare larger perturbations are is governed by the standard deviation σ of the Gaussian distribution, which is added as a user-defined parameter.

An option to sampling the perturbations from a Gaussian distribution is to do it from a polynomial distribution. A proposal by Deb and Goyal [12] is to sample from the probability distribution

$$P(\delta) = 0.5(\eta_m + 1)(1 - |\delta|)^{\eta_m},$$

where $\delta \in (-1, 1)$ and η_m is a positive user-defined parameter called the *distribution index*. We can easily sample variables $\bar{\delta}$ from this distribution by sampling uniformly distributed random variables $u \in (0, 1)$ and then setting

$$\bar{\delta} = \begin{cases} (2u)^{\frac{1}{\eta_m+1}} - 1, & \text{if } u \leq 0.5 \\ 1 - (2(1-u))^{\frac{1}{\eta_m+1}}, & \text{if } u > 0.5. \end{cases}$$

Given a parent solution $x_p = (x_p^1, ..., x_p^n)$ and a maximal allowed perturbance $\Delta_{\max}$ we can then create the i:th parameter value of the child solution x_c as $x_c^i = x_p^i + \bar{\delta}\Delta_{\max}$. Note that typically, one $\bar{\delta}$ is sampled for each parameter.

The mutation scheme described above assumes that all of the parameter values are unbounded. If we instead assume that the i:th parameter value should be bounded between upper and lower values x_l^i and x_u^i we can sample $\bar{\delta}$ as follows [14]:

$$\delta_1 = \frac{x_p^i - x_l^i}{x_u^i - x_l^i},$$

$$\delta_2 = \frac{x_u^i - x_p^i}{x_u^i - x_l^i},$$

$$\bar{\delta} = \begin{cases} (2u + (1-2u)(1-\delta_1)^{\eta_m+1})^{\frac{1}{\eta_m+1}} - 1, & \text{if } u \leq 0.5 \\ 1 - (2(1-u) + 2(u-0.5)(1-\delta_2)^{\eta_m+1})^{\frac{1}{\eta_m+1}}, & \text{if } u > 0.5. \end{cases}$$

Similarly to above, we then compute the child as

$$x_c^i = x_p^i + \bar{\delta}(x_u^i - x_l^i).$$

We note that the bounded scheme reduces to the unbounded one if we set $x_u^i = \infty$ and $x_l^i = -\infty$. When it comes to setting the distribution index η_m, it can be shown that the expected normalized perturbance $\frac{x_c^i - x_p^i}{x_u^i - x_l^i}$ on both the positive and negative side is of order $O(1/\eta_m)$ [10]. Thus, if we were to set $\eta_m = 10$, we would get a mutation effect of around 10%. Moreover, an increase in η_m implies a decrease in the mutation effect. The fact that the setting of a single hyperparameter create perturbances that are relative to the upper and lower bounds of each parameter value of the individual makes bounded polynomial mutation very useful, not least in the experiments carried out in this report.

2.3 Crossover

Crossover is an operation where parameter values are exchanged between two individuals (parents) to create two or more new individuals (children). Again we will start by examining operators that are used together with a binary representation of individuals. One traditionally very common method of doing binary crossover is the *one-point crossover*. From two

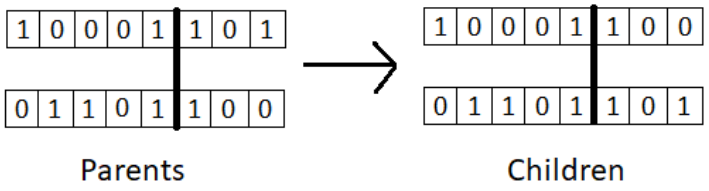


Figure 2.1: The one-point crossover. The left and right parts of the two parents are combined to create two children.

parents it creates two children as follows. Given two parents $\boldsymbol{A}$ and $\boldsymbol{B}$, the idea is to first randomly select an index. One of the children is created by combining the bit values that are to the left of the chosen index of parent $\boldsymbol{A}$ with the ones that are to the right of the same index of parent $\boldsymbol{B}$ — and vice versa for the other child. A visualization of the procedure is shown in figure 2.1. The idea behind this type of crossover is that building blocks consisting of several bit values are crucial in deciding an individual's fitness and should therefore to some extent be maintained across multiple generations [21]. This is one property that we want to be carried over when using real-valued representation.

When it comes to real-valued crossover, it is common to not simply combine parameter values of the parents, but to actually change the values themselves [20]. One simple way to do this is to create a child y by taking a convex combination of the two parents x_1 and x_2, i.e. $y = \alpha x_1 + (1 - \alpha)x_2$ where $\alpha \in [0, 1]$. This type of operator is called a *line crossover*, since from a geometric point of view the child will be located on the line between the two parents in the solution space.

A more complicated real-valued crossover scheme was introduced by Deb and Agrawal [11]; namely the Simulated Binary Crossover (SBX). The authors' stated goal was to create a real-valued version of the one-point crossover (explained above), hence the name. While it can be shown that the SBX has roughly the same *search power* as the one-point crossover, i.e a similar ability to create an arbitrary child solution from a given pair of parent solutions, there are also significant differences between the two schemes; as we will see below.

The SBX can be implemented as follows, according to Deb [10]. First we sample β from the probability distribution

$$P(\beta) = \begin{cases} 0.5(\eta_c + 1)\beta^{\eta_c}, & \text{if } \beta \leq 1, \\ 0.5(\eta_c + 1)\frac{1}{\beta^{\eta_c+2}}, & \text{otherwise,} \end{cases} \tag{2.1}$$

where η_c, similarly to the distribution used in the polynomial mutation scheme, is a user defined, positive distribution index. A small η_c increases the probability of child solutions being farther away from the parents while a large η_c decreases that same probability. Note however that child solutions closer to the parents are always monotonically more likely to appear; the value of η_c only determines how quickly this probability decreases for points

farther away from the parents. The reason behind sampling from this particular distribution is to make the search power similar to the one attained in one-point binary crossover. Given a uniformly distributed random variable $u \in (0, 1)$ we can sample from the distribution (2.1) by setting

$$\beta = \begin{cases} (2u)^{\frac{1}{\eta_c+1}}, & if\ u \leq 0.5, \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise.} \end{cases}$$

Again like in the polynomial mutation scheme, one β is typically sampled for each parameter in the solution vectors. Now, given two parent solutions $x_{(p,1)} = (x^1_{(p,1)}, ..., x^n_{(p,1)})$ and $x_{(p,2)} = (x^1_{(p,2)}, ..., x^n_{(p,2)})$ we can create the i:th parameter value of two child solutions $x_{(c,1)}$ and $x_{(c,2)}$ as

$$x^i_{(c,1)} = 0.5\Big((1+\beta)x^i_{(p,1)} + (1-\beta)x^i_{(p,2)}\Big), \tag{2.2}$$

$$x^i_{(c,2)} = 0.5\Big((1-\beta)x^i_{(p,1)} + (1+\beta)x^i_{(p,2)}\Big). \tag{2.3}$$

An important implication of the above equations can be seen by subtracting equation (2.2) from equation (2.3), yielding

$$x^i_{(c,2)} - x^i_{(c,1)} = \beta\Big(x^i_{(p,2)} - x^i_{(p,1)}\Big).$$

What this rewriting shows is that the distance between the two children is proportional to the distance between the parents. This is desirable, since in cases where parent solutions are farther apart (typically in the early iterations of a GA run) we generally want to emphasize the exploration of the solution space while in cases where the parents are closer to each other (typically in the later iterations) we generally want to further narrow down the search. This balance between *exploring* the solution space and *exploiting* the best solutions is a recurring theme when it comes to SBST in general and GAs in particular. This will be more apparent as we move on.

As was the case with the polynomial mutation scheme, the procedure described above assumes that solution parameter values are unbounded. Assuming instead that the i:th parameter has the lower and upper values x^i_l and x^i_u respectively, we generate the children as follows [2]. First, we define $\bar{x}_1 = \min(x^i_{(p,1)}, x^i_{(p,2)})$ and $\bar{x}_2 = \max(x^i_{(p,1)}, x^i_{(p,2)})$. For the first parent we then define

$$\gamma = 1 + \frac{2\left(\bar{x}_1 - x^i_l\right)}{\bar{x}_2 - \bar{x}_1},$$

and for the second

$$\gamma = 1 + \frac{2\left(x^i_u - \bar{x}_2\right)}{\bar{x}_2 - \bar{x}_1}.$$

Next, letting $\alpha = 2 - \gamma^{\eta_c+1}$ and again letting $u \in (0, 1)$ denote uniformly distributed random variables, we can sample β as follows:

$$\beta = \begin{cases} (\alpha u)^{\frac{1}{\eta_c+1}}, & if\ u \leq \frac{1}{\alpha}, \\ \left(\frac{1}{2-\alpha u}\right)^{\frac{1}{\eta_c+1}}, & \text{otherwise.} \end{cases}$$

Finally, we create the children as

$$x^i_{(c,1)} = 0.5\Big((1+\beta)\bar{x}_1 + (1-\beta)\bar{x}_2\Big),$$
$$x^i_{(c,2)} = 0.5\Big((1-\beta)\bar{x}_1 + (1+\beta)\bar{x}_2\Big).$$

2.4 Parent selection

In this section we discuss different ways of selecting the parents in a population, i.e the individuals that we apply mutation and crossover to. The first kind of methods that we cover are based on the principle of *fitness proportionate selection*. These methods all have in common that they select parents with a probability proportional to fitness. Individuals with a high fitness value have a high probability of being selected and vice versa for individuals with a low fitness value. A common downside of these types of methods is that they essentially get reduced to random search when the fitness values in the population are close to each other. For example, if the least fit individual has a value of 0.48 and the most fit has a value of 0.50, we are almost as likely to select the worst individual as we are the best one. In cases like this, the internal *rankings* of the fitness values are often more important than the values themselves. This is one main reason why the selection operators of many GAs are instead often based on what is called *tournament selection*.

In tournament selection, a number of individuals are randomly selected to participate in a tournament. The winner of the tournament, i.e the one with the largest fitness value, is selected as a parent. This process is then repeated until the entire parent population has been allocated. Typically, each individual can be selected at most two times [10]. The number of individuals participating in each tournament, the *tournament size*, can be viewed as a tunable parameter. The probability of selecting the most fit individual is directly related to the tournament size; it approaches 1 as the tournament size grows. See algorithm 1 for the details. The simplicity of the algorithm and the fact that it works well for most kinds of fitness functions has made tournament selection into a very popular choice of selection operator for GAs [21].

Algorithm 1 Tournament selection algorithm

```
P ← population
t ← tournament size, t ≥ 1
Best ← individual picked at random from P with replacement
for i from 2 to t do
    Next ← individual picked at random from P with replacement
    if Fitness(Next) > Fitness(Best) then
        Best ← Next
if Best has been selected 2 times then
    P ← P \ Best
return Best
```

2.5 Elitism

As mentioned above, GAs traditionally replace all of the individuals in a parent generation with the newly created children. The concept of elitism, where the fittest individuals (the 'elite') are retained in the population across multiple generations, breaks with this tradition. As it turns out, it does so for good reason; some form of elitism is commonly used in all types of EAs and GAs [21]. When it comes to multiobjective GAs, which this report is mostly concerned about, there are multiple studies showing that algorithms that use elitism outperform their counterparts that do not [19], [9]. There are two main reasons behind this success: firstly by keeping some of the elite individuals in the population the overall fitness is kept from deteriorating, and secondly the probability of creating even fitter offspring is increased [10]. However, it should be emphasized that keeping too many of the elite individuals reduces the overall diversity in the population and risks leading to premature convergence. This is another case of balancing the principles of exploration and exploitation when constructing a GA.

2.6 Multiobjective optimization

The concept of multiobjective optimization is not hard to understand: it simply means that more than one objective function are to be optimized simultaneously. That problems of such kind often appear in the real world is also not hard to realize. One example provided by Luke [21] is the problem of building a house that simultaneously consumes as little energy as possible while still keeping building costs down. Optimizing one objective may very well have a detrimental effect on the other, so it is clear that we will not be able to find some kind of objectively single best solution in this case. However, what we can say is that a house that is both more expensive and that consumes more energy is clearly outclassed by a cheaper and more energy efficient house. Using the terminology of multiobjective optimization, we say that the second house *Pareto dominates* the first one.

More formally, we say that one solution $\boldsymbol{A}$ Pareto dominates $\boldsymbol{B}$ if $\boldsymbol{A}$ is at least as 'good' as $\boldsymbol{B}$ in all objectives and is 'better' than $\boldsymbol{B}$ in at least one objective. Whether a 'good' solution means an objective function taking larger or smaller values depends on the application. In

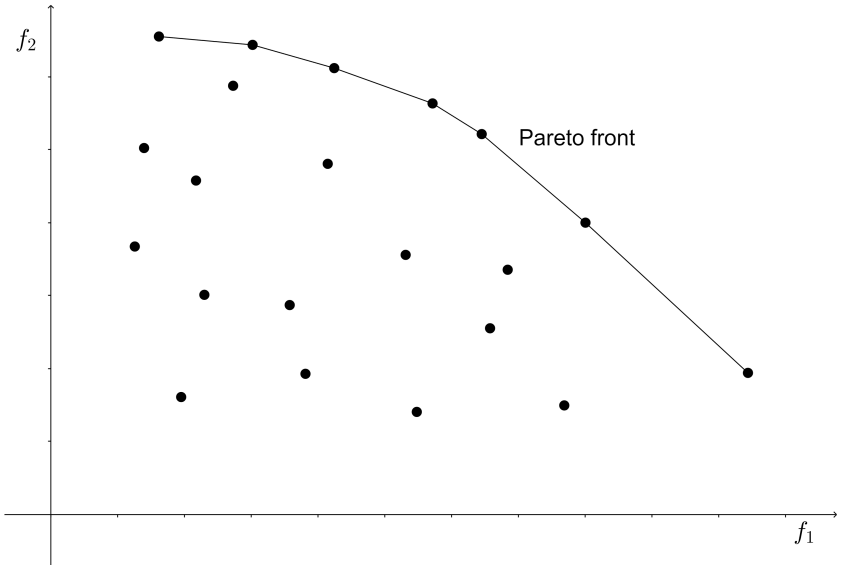


Figure 2.2: The plot shows an example of what the Pareto front might look like for a multiobjective optimization problem with two objective functions f_1 and f_2. Each point corresponds to the objective function values of one solution.

either case, the takeaway here should be that, given a set of candidate solutions to a multiobjective optimization problem, we might as well restrict ourselves to the ones that are not Pareto dominated. This subset of solutions is commonly referred to as the *Pareto front* of the space of solutions. Figure 2.2 shows a visualization of the concept behind the Pareto front in a case with two objective functions. Clearly, the Pareto front gives us a way to evaluate and compare different solutions, which is of great importance when creating a selection operator for a GA. Moving on, we would like to expand on this notion of comparing different solutions to each other.

One particularly useful way of evaluating and comparing solutions is to measure their respective *closeness to the Pareto front*. There are multiple ways to do this measurement; here, we will use a concept called the *Pareto front rank*. The idea behind it is to utilize the fact that solutions that dominate a large number of other solutions are most likely preferable to solutions that are themselves largely dominated. Thus, all individuals are assigned an integer ranking, the lower the ranking the better. Individuals belonging to the Pareto front are given a Pareto front ranking of 1. A ranking of 2 is given to all individuals belonging to the Pareto front after all individuals of rank 1 have been removed, and likewise a ranking of 3 is given to the members of the Pareto front after individuals of rank 2 have been removed, and so on. This notion of ranking allows us to select between individuals even when one does not dominate the others. Of course, there could potentially still be several individuals that have the same Pareto front rank. This serves as motivation for us to look at yet another way of evaluating individuals: how much they contribute to the spread of the Pareto front.

Beyond trying to find solutions at the Pareto front, there is another important goal in multiobjective optimization that has not yet been mentioned: to maximize the *spread* of the solutions across a Pareto front. A set where all individuals are gathered closely together across

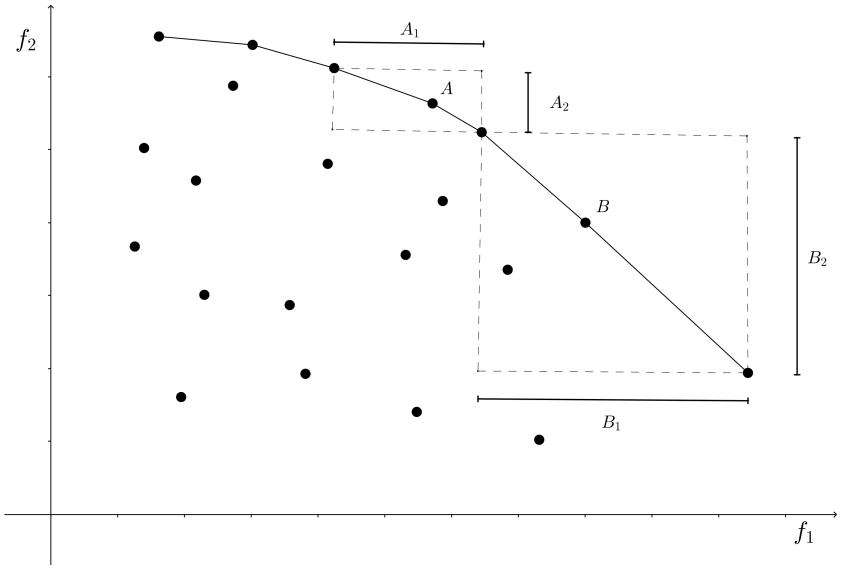


Figure 2.3: The plot shows the idea behind the assignment of sparsity to a multiobjective optimization problem, in this case with two objective functions. The sparsity of solution $\boldsymbol{B}$ is greater than that of solution $\boldsymbol{A}$ since $A_1 + A_2 < B_1 + B_2$.

the Pareto front does not tell us much about what options are available when trying to select suitable solution(s). Instead, it is much more preferable to have a set of solutions that is more spread out across the front, giving us greater diversity when choosing. As to how this notion of spread can be quantified, there are again several different ways. We will here focus on the notion of *sparsity*, defined in algorithm 2 below. The algorithm closely follows the corresponding one presented in [21]. In short, the sparsity of a solution x_i is the sum of the distance between the closest neighbors in each individual dimension among solutions with the same Pareto front rank as x_i, divided by a normalizing factor. The solutions at the far ends of the front are assigned an infinite sparsity. The idea behind the concept is shown in figure 2.3. Naturally, we would like solutions to have as large of a sparsity as possible.

Algorithm 2 Assignment of Sparsity

$F \leftarrow \{F_1, ..., F_m\}$ ▷ a Pareto front rank of m individuals
$O \leftarrow \{O_1, ..., O_n\}$ ▷ objective functions to assess with
for each individual $F_j \in F$ **do**
    Sparsity$(F_j) \leftarrow 0$
for each objective $O_i \in O$ **do**
    $F' \leftarrow$ Sort(F, O_i) ▷ sort F with respect to the objective O_i
    Sparsity$(F'_1) \leftarrow \infty$
    Sparsity$(F'_m) \leftarrow \infty$
    if $F'_1 = F'_m$ **then**
        continue
    for j from 2 to $m - 1$ **do**
        Sparsity$(F'_j) \leftarrow$ Sparsity$(F'_j) + \frac{\text{ObjectiveValue}(O_i, F'_{j+1}) - \text{ObjectiveValue}(O_i, F'_{j-1})}{m(\text{Range}(O_i))}$

2.7 A Genetic Algorithm: NSGA-II

With the above concepts defined, all the tools are in place for us to describe the Non-dominated Sorting Genetic Algorithm II, better known as NSGA-II. Ever since it was introduced by Deb et al. [9], it has become one of the more popular multiobjective GAs and has been widely used in many different applications [26]. Variants of it have even been previously used to create simulated traffic scenarios [8], [6], [7].

The algorithm itself can in short terms be described as an elitist GA that uses binary tournament selection based on Pareto front ranks and sparsity as they were defined above. The two individuals that get chosen to participate in the tournament are in the first case evaluated based on their respective front rank, where the individual with a larger rank is selected. In the case where the individuals happen to have the same front rank, the one with the larger sparsity value is chosen. The elitism is introduced through the introduction of an archive with a fixed size, where the individuals that have the largest front ranks are stored. If there are not enough slots in the archive to fill it with all individuals of a certain front rank, the ones with the largest sparsities are again prioritized. All individuals compete for a place in the archive every generation, while only members of the archive can be selected to crossover and mutate in the tournament selection. The details are presented in algorithm 3, again closely following the corresponding algorithm presented by Luke [21]. Note that the expression Breed(A) means the execution of parent selection, crossover and mutation in that order.

Algorithm 3 The Genetic Algorithm NSGA-II

$m \leftarrow$ desired population size
$a \leftarrow$ desired archive size ▷ typically a = m
$P \leftarrow \{P_1, ..., P_m\}$ ▷ build initial population
$A \leftarrow \{\}$ ▷ archive
while termination criteria are not satisfied **do**
  AssessFitness(P) ▷ assess the fitness so that the Pareto front ranks can be computed
  $P \leftarrow P \cup A$
  $BestFront \leftarrow$ Pareto front of P
  $R \leftarrow$ compute front ranks of P
  $A \leftarrow \{\}$
  for each front rank $R_i \in R$ **do**
    AssignSparsities(R_i) ▷ assign sparsities to all individuals in R_i
    if $||A|| + ||R_i|| \geq a$ **then** ▷ add the sparsest individuals of a front rank into the archive
      $A \leftarrow A \cup$ the sparsest $a - ||A||$ individuals in R_i
      break from the **for** loop
    else ▷ add an entire front rank into the archive
      $A \leftarrow A \cup R_i$
  $P \leftarrow$ Breed(A), using Algorithm 1 for selection (with a tournament size of 2)
return $BestFront$

One consequence of the use of elitism in NSGA-II is that the archive will be almost entirely composed of only nondominated solutions in the case where we have a large number of ob-

jectives (say, around 10 or more). This can easily lead to situations where the diversity in the population deteriorates. This is one reason why NSGA-II is best used when having problems with only a few objectives [13].

2.8 Assessing the solutions

Many methods are proposed as to how the solutions obtained from search-based algorithms can be evaluated. In their review [27], Wang et al. differentiate between methods that measure how close the obtained solutions are to the optimal Pareto front and methods that measure the diversity among the solutions. Most of the methods they discuss require either that the optimal Pareto front is known or that it is estimated in some other way, making them rather inconvenient to use. We will therefore only cover one of the methods they present: the hypervolume (HV) indicator.

The HV indicator measures the volume that is covered by the solutions in the objective space. We first define $\boldsymbol{S}$ to be a set containing solutions $\boldsymbol{s}_i$, where each $\boldsymbol{s}_i$ belongs to the Pareto front. Now, the idea is to for each $\boldsymbol{s}_i$ create a corresponding hypercube v_i that has one corner at $\boldsymbol{s}_i$ and its other diagonal corner at a reference point. The reference point is often selected by combining the worst objective values among all solutions. The HV value is then defined as the union of all the hypercubes, or in mathematical terms

$$\text{HV} = \text{volume}\left(\cup_{i=1}^{|S|} v_i \right).$$

Figure 2.4 illustrates the idea behind the HV indicator. Looking at the figure, it is clear that a larger HV value indicates both a greater diversity among the solutions and that the computed Pareto front is closer to the optimal front. Thus, it is desirable for an algorithm to yield solutions with as large of an HV value as possible.

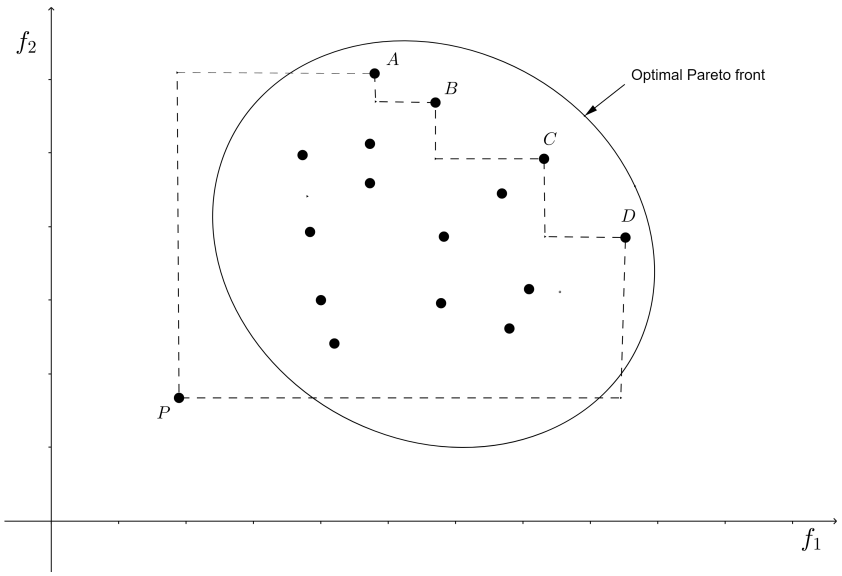


Figure 2.4: The plot illustrates how the HV indicator for a set of solutions is computed for a problem with two objective functions. The inside of the ellipse denotes the region where solutions are defined. Thus, the upper right part of the ellipse constitutes the optimal Pareto front and the solutions ***A*** to ***D*** are the members of the computed Pareto front. The solution ***P*** is a reference point. In this example, the HV value is equal to the area enclosed by the dashed lines. Note that a larger HV value indicates both a greater diversity among the solutions and that the computed Pareto front is closer to the optimal front. Thus, larger HV values are desirable.

Chapter 3
Approach

We introduce the tools that were used to carry out the experiments (most importantly the CARLA simulator) and describe how they were used. We also describe how the simulated traffic scenarios were generated and how they were used to test OTUS3D. An overview of the workflow of the entire project is presented in figure 3.1.

3.1 The system under test

The system under test was the solution OTUS3D, developed by Viscando. OTUS3D is based on the principle of stereoscopic vision where, similarly to the human eyes, images from two separate cameras placed close to each other enable the creation of a 3D view of the environment. By then utilizing computer vision and ML algorithms, the system is able to segment the images and then classify and track pedestrians, cyclists and motor vehicles. The tracked objects all have properties such as their positions, directions and speeds estimated by OTUS3D at different time points.

3.2 The CARLA simulator

The traffic scenarios were simulated using the open-source traffic simulator CARLA. Being built upon Unreal Engine 4, CARLA provides high-end rendering quality and realistic physics. The CARLA server runs the simulation while receiving commands from the client. These commands could for example be instructions on how to change the steering and acceleration of one or more vehicles, or they could affect properties of the simulation environment such as weather, illumination or walker behavior. The client API is implemented in Python. What this means is that the server, and the entire simulation, can be controlled through the execution of Python scripts. In the documentation of the API that is presented at the CARLA webpage [3], there is a large number of pre-defined classes and methods that sim-

plify the implementation of such scripts.

One of the most important classes of the CARLA API is the Actor class. Three different subclasses of the Actor class were used in this project: vehicles, walkers and sensors. Note that the vehicle subclass includes both bicycles and cars. There exists a number of blueprint 3D models in CARLA so that vehicles and walkers of different kinds can be created. Moreover, both of these types of actors can be controlled either manually or by autopilot. Sensors are actors that retrieve data from simulations; some examples include RGB cameras, collision detectors and LIDAR sensors. Different sensors naturally produce different types of output. For example, RGB cameras retrieve images from the simulation, collision detectors retrieve frames at which collisions occured and LIDAR sensors retrieve point clouds.

Weather conditions are easily customizable in CARLA. Cloudiness, precipitation, fog thickness and wind intensity are some examples of weather effects that can be customized simply by setting an intensity parameter. The placement of the sun can also be fully customized.

By default, the CARLA server runs as fast as possible without waiting for the client. What this means is that when executing a script, it is not possible to know how long the simulation has progressed between two lines of code. This is not ideal in situations where we want to collect simulation data at a set time interval. In these types of situations, it is better to use what the CARLA documentation calls *synchronous mode*. When set to synchronous mode, the server will only advance a frame when given permission from the client. The amount of time in the simulation that each frame will correspond to can be set by the user.

When it comes to the installation of CARLA, there are two existing options: installing the package version or the source version. The source version is more arduous to install, as it requires additional steps and more kinds of different software. The upside of installing it is that 3D scenes can be directly customized in the Unreal Engine editor. This is true both for the default scenes created by the CARLA developers and for imported ones. Moreover, the process of importing scenes is a lot simpler in the source version of CARLA. For these reasons, we used the source version of CARLA in this project.

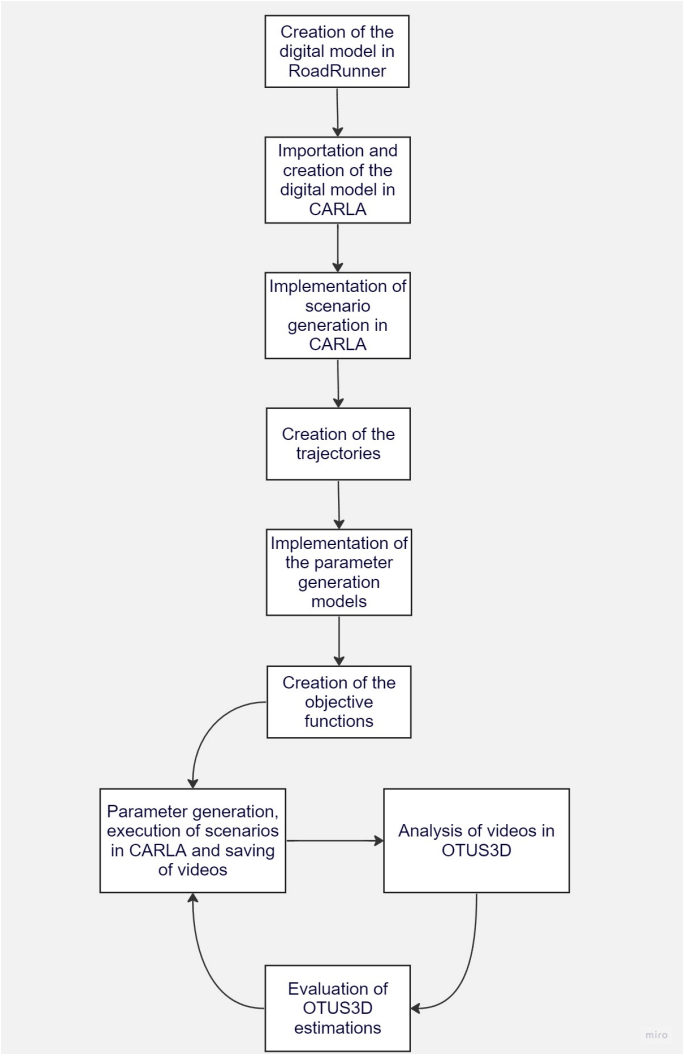


Figure 3.1: An overview of the workflow of the project.

3.3 DEAP

The implementation of NSGA-II that was used in this project was created using the Python evolutionary computation framework DEAP [18]. All parts of the algorithm, including the computation of the hypervolume used for the evaluation, was implemented using DEAP. The main part that we had to implement ourselves was the calculation of fitness of the individuals; how this was done is explained in section 3.7.

3.4 Creation of the digital model

As was mentioned in the introduction, the environment that was created in the simulation was based on a real life junction, one where OTUS3D had gathered data. This junction is located in Lindholmen, Gothenburg (lat 57.7086727, long 11.9395434). The junction is shown in figure 3.2. Another image taken from Eniro Maps is shown in figure 3.3. The digital model of this junction was created using the MathWorks package RoadRunner. RoadRunner is an interactive editor built for the purpose of creating 3D models of traffic environments. With its help, roads can easily be created and customized and there exists a large number of blueprints of 3D models such as signs, guardrails, trees and buildings that can be inserted into the environment.

When exporting the scene that was created in Roadrunner, the data was stored in one xodr file and one fbx file. The xodr extension indicates the OpenDRIVE file format. OpenDRIVE files describe road network data: the geometry of roads, lanes and markings as well as features along the road like stop signals are saved in such files. The fbx file on the other hand contains 3D geometry and animation data: in the case of our scene it contained information about 3D objects like trees, bushes and street lights. With the data contained in the xodr and fbx files, importing a 3D scene into a version of CARLA that had been built from source was easily done by using methods that were provided with the installation. Images of the digital model that was built in RoadRunner are shown in figure 3.4 and figure 3.5.

After the scene was imported into CARLA, some of the 3D assets had either disappeared or did not look particularly visually pleasing. This is shown in figure 3.6. This problem was fixed by using the Unreal Engine editor to add blueprints developed by the CARLA team. The results can be seen in figure 3.7.

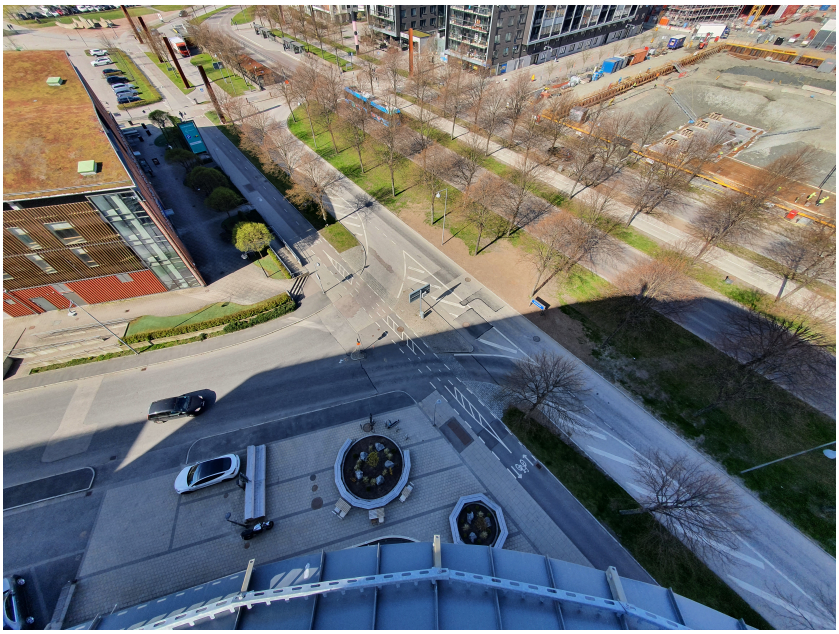


Figure 3.2: A photograph taken above the junction in Lindholmen, Gothenburg where Viscando has collected data. Note in particular the distinct shadow from the building in the morning sun (April 29th, 9:45 am).

Figure 3.3: An image of the Lindholmen junction taken from Eniro Maps.

Figure 3.4: The digital model of the Lindholmen junction was created in RoadRunner. This is an image of the model. Compare with figure 3.2.

Figure 3.5: The digital model of the Lindholmen junction was created in RoadRunner. This is an image of the model. Compare with figure 3.3.

Figure 3.6: The digital model of the Lindholmen junction after it was imported into CARLA. Note that some of the signs present in figure 3.5 are missing. Some of the textures, particularly the leaves and the pavement material, also do not look particularly realistic. 3.3.

Figure 3.7: The improved digital model of the Lindholmen junction after it was imported into CARLA. Compared to figure 3.6, the tree and bush textures look more realistic, some ground material has been changed and some signs have been added. The building to the left in the image has also been re-scaled to better resemble the real one shown in figure 3.3.

3.5 Creation of trajectories

When defining a traffic scenario, the paths that vehicles and walkers follow, i.e their *trajectories*, are of great importance. In this project, all trajectories were kept constant over the course of the experimental testing that was made. In other words, a number of trajectories were manually created before any experiments were made, and these trajectories were not changed during or in between the experiments. A trajectory was defined as a vector of *waypoints*, i.e points in 2D space that specified the path that the actor should strive to follow. Suitable waypoints were found with the help of data that had already been collected from the real-life Lindholmen junction by Viscando. This data contained two pieces of information that were of interest to us: the estimated positions of passing vehicles and walkers in the form of 2D points and the corresponding estimated speeds of these objects, with a time interval of 0.05 seconds between each data point. The 2D points were, after some modifications to better suit the digital model, suitable to use as waypoints. The final trajectories consisted of approximately between 150 and 350 waypoints.

In practice, the actors (cars, bicycles and walkers) followed the set trajectories and kept their assigned velocities by setting the CARLA simulator to synchronous mode and advancing it in a frame by frame fashion. After the creation of an actor, that actor was assigned a target velocity and was set to steer towards the next waypoint. Once that waypoint was reached, the actor was assigned a new target velocity and was set to steer to yet another waypoint. This was repeated until the entire trajectory was completed. It should be noted that the exact velocities and positions of the actor were ultimately decided by the CARLA server; the waypoints and target velocities were treated as just that - targets. Therefore, the exact position and velocity data had to be fetched from the simulator and saved to later provide ground truth for the estimations made by OTUS3D.

While simulating a scenario, images that could be used as input to the OTUS3D system

Figure 3.8: A captured image from the simulation that was used as input to OTUS3D.

needed to be captured. This was done by placing RGB cameras at the two spots in the simulated environment that corresponded to the placements of the two cameras used by the OTUS3D sensor in the real Lindholmen junction. These cameras were set to capture 20 images per second, since this was an appropriate rate for OTUS3D to receive image input. Finally, the images were converted into video format - with one video for each scenario - and these videos were used as input for OTUS3D. In figure 3.8 we see an example of a captured simulated image that was used as input to OTUS3D.

3.6 Defining a traffic scenario

In this section, we slightly formalize what is meant by a traffic scenario in the context of this report. We assume that a traffic scenario is defined through the following properties:

- The number of actors following each of the trajectories. There could for example be two actors following trajectory 0 and zero actors following trajectory 1.
- The starting time for each actor. This is defined by an integer denoting the number of frames passed since the start of the simulation.
- Target velocities of the actors at specific points in their respective trajectory.
- The color of each car. This is defined by a triplet of integers from 0 to 255 denoting the RGB-value.
- The setting of weather parameters.

Moving forward, we will refer to the numbers describing the above properties as *scenario parameters*. It should be noted that the blueprint model of each actor potentially constituted

another property of a traffic scenario. However, to simplify the implementation of the GA, it was decided that the same blueprint model should be used for each car, bicycle and walker across all experiments.

The more formalized view of traffic scenarios presented above allows us to generate synthetic scenarios in a systematic manner. In particular, it makes it possible to keep some of the properties of a traffic scenario fixed while treating others as variable parameters. Moreover, the allowed ranges of the variable parameters is also something that needs to be considered.

3.7 Setting the position of the sun

The setting of the weather parameters describing the position of the sun was not as straightforward as for the other weather parameters and therefore needs a more detailed description. In CARLA, the position of the sun in the sky is decided by two parameters: the altitude angle and the azimuth angle. The altitude angle is the number of degrees above the horizon that the sun is situated: a value of 90 implies that the sun is directly overhead. The azimuth angle describes the position of the sun's projection onto the horizontal plane: it takes a value between 0 and 360 degrees. In the case of CARLA, the direction of a zero azimuth angle is decided arbitrarily by Unreal Engine.

In order to make the simulated scenarios as realistic as possible we wanted the placement of shadows from different objects, and by extension the position of the sun, to be roughly the same as in the real Lindholmen junction. This was achieved by using data from the Swedish Meteorological and Hydrological Institute, SMHI. Figure 3.9 shows sun-path diagrams in Gothenburg at different times of the year. The diagram that was selected as the one to follow in the simulated scenarios is specified in the figure; it shows the sun-path in Gothenburg on August 1st. We wanted to select one of the summer months because the blueprint models of the trees had leaves on them. The diagram in question was approximated as a second degree polynomial function. During simulation, when using the sun position as a variable parameter, the altitude angle was found by plugging the azimuth angle parameter into the polynomial function.

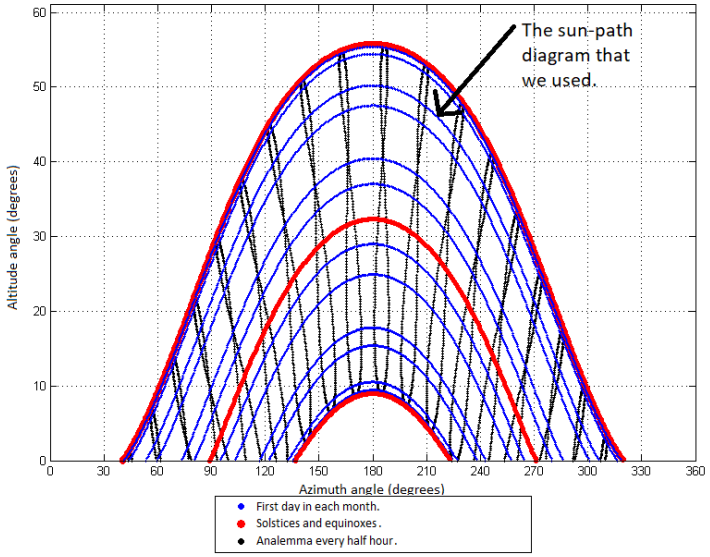


Figure 3.9: Sun-path diagrams for Gothenburg, for different times of the year. The x-axis denotes the azimuth angle in degrees, where 180 means a straight southward direction. The y-axis denotes the altitude angle in degrees. The diagram that we used in the project is marked in the figure; it shows the sun-path in Gothenburg on August 1st. The image is an adapted version of the original found at `https://www.smhi.se/polopoly_fs/1.178701!/goteborg%5B1%5D.png`.

3.8 Objective functions

The formulation of the objective functions (fitness functions) was crucial when using NSGA-II to generate the scenario parameters. Since the purpose of the experiments was to find traffic scenarios that tested the limits of OTUS3D, we wanted to develop fitness functions that in some sense maximized the errors between the ground truth and the estimations made by OTUS3D. We decided to focus on three different error measurements that each corresponded to one objective function. In order to formulate these functions we define the i:th ground truth trajectory as $x^i = x_1^i, ..., x_{n_i}^i$, where each x_j^i corresponds to one waypoint. The corresponding i:th estimated trajectory we define analogously as $\tilde{x}^i = \tilde{x}_1^i, ..., \tilde{x}_{n_i}^i$. Similarly, we define the corresponding ground truth speeds as a sequence of points $v_1^i, ..., v_{n_i}^i$ and the estimated speeds as a sequence $\tilde{v}_1^i, ..., \tilde{v}_{n_i}^i$. Letting m denote the number of trajectories, we can then define the three objective functions as follows:

- Distance errors:

$$O_1 = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{n_i}\sum_{j=1}^{n_i}\sqrt{\left(x_j^i - \tilde{x}_j^i\right)^2}.$$

  This is essentially the mean euclidean distance between the ground truth trajectories and the trajectories estimated by OTUS3D.

- Speed errors:

$$O_2 = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{n_i}\sum_{j=1}^{n_i}\left|v_j^i - \tilde{v}_j^i\right|.$$

  This is essentially the mean absolute errors between the ground truth speeds and the estimated speeds.

- The third objective concerns the percentage of misclassifications across the trajectories. Assuming that the i:th trajectory had c_i misclassifications we define

$$O_3 = \frac{1}{m}\sum_{i=1}^{m}\frac{c_i}{n_i}.$$

Note that the ground truth trajectories were matched with the estimated trajectories that minimized the O_1-value presented above. Using these three objective functions, each simulated scenario could be assigned three fitness values that could then be used in the NSGA-II algorithm 3.

3.9 Overview of the experimental setup

An overview of the experimental setup is presented in figure 3.10. In short, the hyperparameters set the general behavior and boundaries of the whole experiment - they were also kept constant throughout. They include properties such as GA parameters, values of fixed

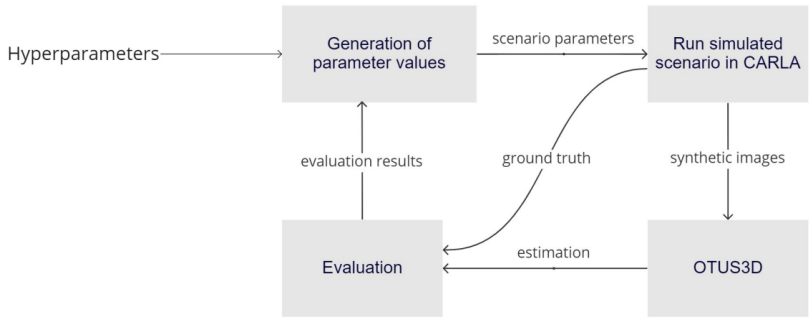


Figure 3.10: An overview of the experimental setup. In the baseline model, the scenario parameters were randomly generated while in the GA model, the NSGA-II algorithm was used to generate the parameter values (see algorithm 3 for details).

scenario parameters and minimum and maximum values for variable scenario parameters.

The scenario parameters were generated based on the hyperparameters. More specifically, as will be explained in section 4.1, the scenario parameters were generated either by sampling from uniform probability distributions (the baseline model) or by use of the GA NSGA-II. One set of scenario parameters defined one traffic scenario that was then simulated in CARLA. The captured images were converted into a video format that were used as input to OTUS3D. OTUS3D in turn produced classifications and estimations of the positions and speeds of the actors participating in the simulation, which could be evaluated by comparing to the ground truth that was produced by CARLA. Note that in the baseline model, there was no feedback of evaluation results to the parameter generation since the parameter values were randomly generated. When using NSGA-II, the procedure described in algorithm 3 was followed, using the objective functions defined in section 3.8 and with one set of scenario parameters constituting a single individual in the population.

It should be pointed out that not all parts of the setup as presented in figure 3.10 were implemented in a single automatic loop. Specifically, since we had no direct access to the OTUS3D system, the synthetic images (converted into videos) that were created by the running of traffic scenarios had to be manually sent to Viscando so that they could be processed by OTUS3D to produce estimations that could be evaluated. Because of this manual step, there was no time to test the two models for more than a few number of generations. There was also no time to test different values for the hyperparameters.

A complete reproduction package is available on GitHub: https://github.com/EliasSjoberg/rise-viscando-thesis

Chapter 4
Evaluation

We describe the experimental setup in greater detail and present and analyze the results. We also attempt to answer the Research Questions and point out the limitations and the lessons learned from this work.

4.1 Experimental setup

In order to perform the experiments, trajectories that decided the paths that cars, bicycles and walkers should follow in the simulated scenarios were created using past data from Viscando. Note that both the positions and the velocities of the actors were determined using this data. Figure 4.1(a), (b) and (c) show the trajectories that were created for cars, bicycles and walkers respectively. Two different sets of trajectories were used in the experiments: one set that contained the car trajectories and another that contained the bicycle trajectories. Moreover, both sets contained the walker trajectories. In all scenarios, only one instance of each trajectory was used, and this number was kept fixed across all generations. The main thought process behind selecting these trajectories was that they seemed be a realistic representation of the trajectories that could appear in the real Lindholmen junction. They were also selected to minimize the number of collisions between actors, as these could lead to unrealistic scenarios.

In addition to the two sets of trajectories, experiments were performed using two different models to generate scenario parameters: one baseline model where parameter values were sampled from a uniform random distribution and one GA model where parameter values were generated using the NSGA-II algorithm with tournament selection, bounded polynomial mutation and bounded Simulated Binary Crossover. The following scenario parameters were set to be variable and thus constituted the individuals in the two models:

- The starting time for each actor participating in the simulation. It ranged from a minimum of 0 simulated seconds to a maximum of 10 simulated seconds.

population size	20
η_m	20
η_c	20
crossover probability	0.9
car model	vehicle.tesla.model3
bicycle model	vehicle.bh.crossbike
walker model	walker.pedestrian.0001

Table 4.1: GA specific parameter values and blueprint models used in the experiments. All of these values were kept fixed.

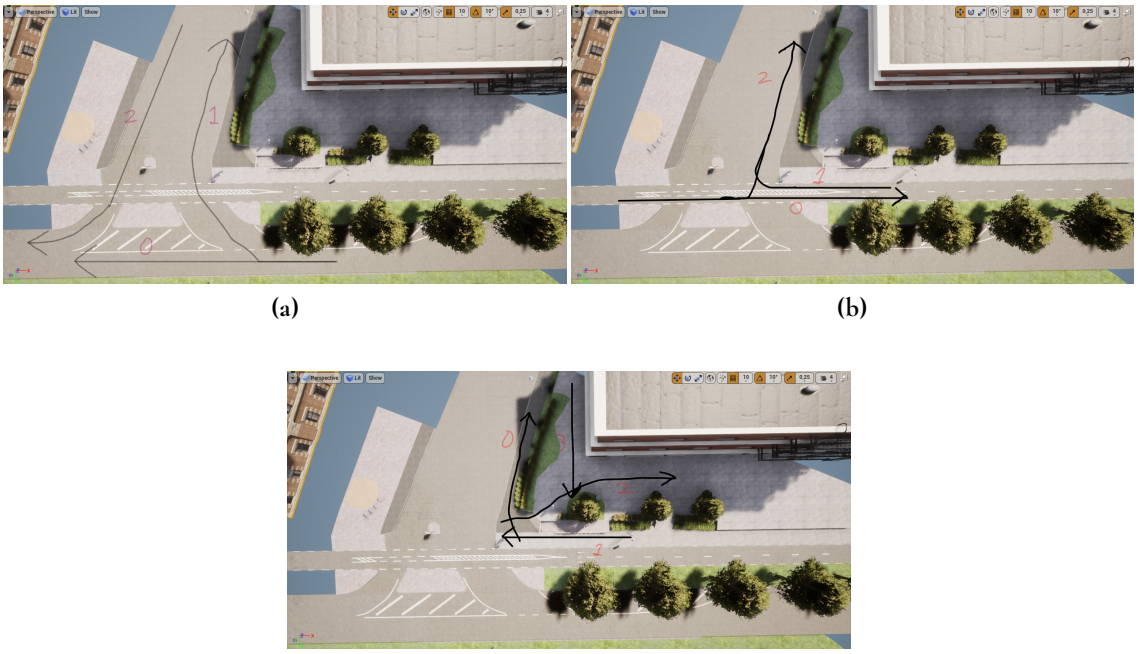
- The color of each car. It consisted of triplets of integers denoting the RGB-values between 0 and 255.
- The level of cloudiness in the simulation, described by a float value between 0 and 100. Images of the Lindholmen junction captured in CARLA at different levels of cloudiness are presented in figure 4.3(a)-(c).
- The sun azimuth angle. As we described in section 3.6, the altitude angle was calculated based on the azimuth angle and was set so that it mimicked the position of the sun in the real Lindholmen junction during early August. Moreover, the azimuth angle was set to a minimum of 25^o and a maximum of 255^o to keep the captured images from being too dark. Images of the Lindholmen junction captured in CARLA at morning, midday and evening are presented in figure 4.2(a)-(c).

These parameters were chosen mainly because they were believed to represent realistic variations that occur in the real Lindholmen junction. Moreover, since we expected varying levels of visibility and occlusion in the simulation to be highly relevant to the performance of OTUS3D, these parameters seemed appropriate. Three generations of both models were created, i.e. a population of scenario parameters was generated three times. A population size of 20 was used. For the first generation of the NSGA-II model, the parameter values were generated by sampling from a uniform random distribution. The values of some other parameters that were set to be constant are presented in table 4.1.

The total number of variable parameters was different depending on the set of trajectories that was used. With the car and walker trajectories, there were a total of seven actors that each had a starting time. Moreover, there were three cars that each had one triplet of integers describing the color, and two weather parameters. This created a total of $7 + 3 \cdot 3 + 2 = 18$ parameters. With the bicycle and walker trajectories there were no color parameters and thus only a total of $7 + 2 = 9$ parameters.

Since we had two sets of trajectories and two models, a total of four experiments were performed. Since there were 20 individuals (i.e. scenarios) in the population and three generations, a total of $20 \cdot 4 \cdot 3 = 240$ scenarios were simulated. Half of these scenarios contained the three car trajectories in figure 4.1(a), while the other half contained the three bicycle trajectories in (b). All scenarios contained the walker trajectories in (c).

As the experiments were carried out, the models were evaluated by computing the distance error, speed error and number of misclassifications as was described in section 3.8. Furthermore, the hypervolume was computed for each model and generation. We also did a number of investigations with the aim of being able to draw conclusions on the effects that different parameter configurations had on the ability of OTUS3D to produce correct estimations. To this end, we computed the mean of the three error measurements for each of the 10 trajectories and closely examined how the location of the actors affected the performance of OTUS3D. Moreover, we investigated the effects that occlusion, the position of the sun, the cloudiness value and the relative luminance values of the cars had on the error measurements.

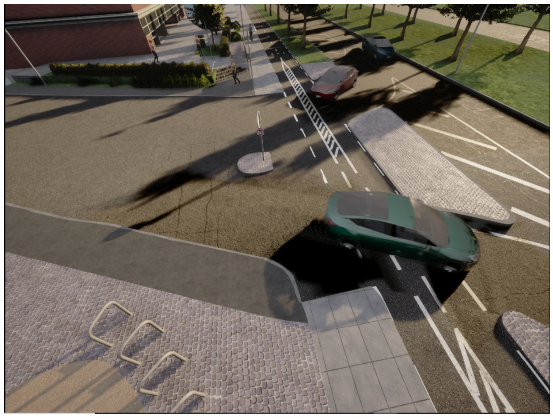


(a) **(b)**

(c)

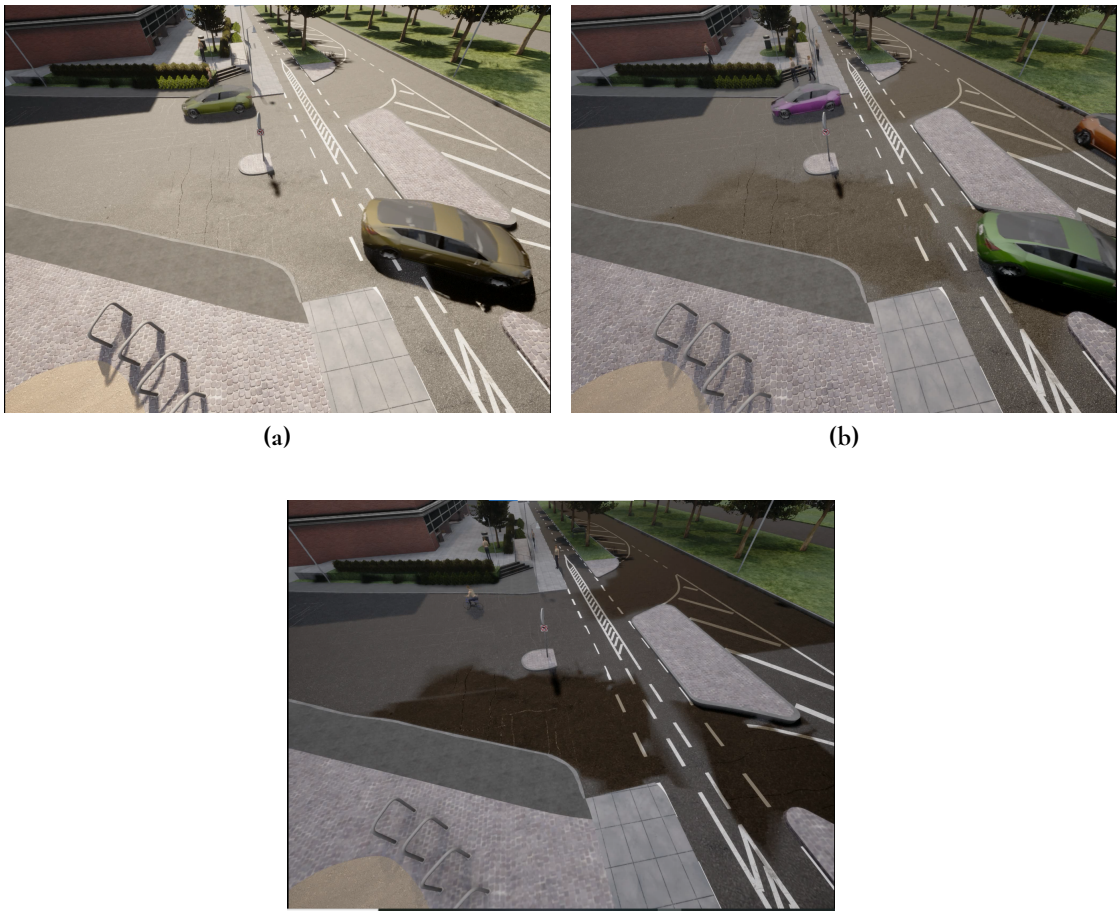
Figure 4.1: A visualization of the (a) car trajectories, (b) bicycle trajectories and (c) walker trajectories that were used to create traffic scenarios.

(a) (b)



(c)

Figure 4.2: (a)-(c) show the Lindholmen junction in CARLA in the early morning, at midday and in the evening respectively. The azimuth angles are 26.2°, 134° and 240° while the corresponding altitude angles are 8.54°, 49.9° and 18.2°.



(a) **(b)**

(c)

Figure 4.3: (a)-(c) show the Lindholmen junction in CARLA with a cloudiness value of 14.0, 82.3 and 95.1 respectively. The azimuth angle was kept roughly at a constant 170º in all three cases. We see that visibility is still quite good in (b), but in (c) the images start to become quite dark.

4.2 Results

The results can be divided into three parts. In the first part, we present the performances of the NSGA-II model and the baseline model. These results are presented in table 4.2. In the second part, we examine the errors resulting from each of the trajectories used. These results are presented in tables 4.3–4.5 and in figures 4.4–4.7. In the third part, we investigate the effects that the different scenario parameters (car colors, sun positions, cloudiness values and actor starting times) had on the error measurements. These results are presented in figures 4.8–4.13 and in tables 4.6–4.8.

Table 4.2 contains the three error measurements and the hypervolume computations for each generation using the car/walker and the bicycle/walker trajectories on the baseline model and the NSGA-II model. Ideally, the error measurements as well as the hypervolume computations would grow as the number of generations increases in the NSGA-II model. However, with just three generations, it is not surprising that no such trend can be discerned.

Tables 4.3, 4.4 and 4.5 contain the mean errors for each of the car, bicycle and walker trajectories respectively. The errors for the i:th trajectory have been computed by adding together all of the individual estimations made at each waypoint for every simulation of that trajectory (120 simulations in the case of the car and bicycle trajectories and 240 simulations for the walker trajectories). We see that the errors vary a fair bit depending on the trajectory. Generally speaking, cars seem to be more difficult to estimate with regards to their positions and speeds while the walkers are more difficult to correctly classify.

	gen 0	gen 1	gen 2
distance error (m) c+w/baseline	2.27	2.26	2.27
distance error (m) c+w/NSGA-II	2.26	2.21	2.22
distance error (m) b+w/baseline	1.89	1.92	1.92
distance error (m) b+w/NSGA-II	1.90	1.89	1.99
speed error (m/s) c+w/baseline	0.61	0.57	0.65
speed error (m/s) c+w/NSGA-II	0.62	0.55	0.59
speed error (m/s) b+w/baseline	0.39	0.40	0.40
speed error (m/s) b+w/NSGA-II	0.39	0.40	0.36
misclassifications (%) c+w/baseline	25.8	29.5	26.4
misclassifications (%) c+w/NSGA-II	24.8	24.1	22.5
misclassifications (%) b+w/baseline	30.1	27.5	29.1
misclassifications (%) b+w/NSGA-II	27.8	29.7	29.2
hypervolume c+w/baseline	4.41	3.93	4.10
hypervolume c+w/NSGA-II	4.81	3.16	3.55
hypervolume b+w/baseline	2.69	3.54	2.59
hypervolume b+w/NSGA-II	2.91	2.93	2.60

Table 4.2: Error measurements and hypervolume computations for the baseline model and the NSGA-II model with car/walker and bicycle/walker trajectories for three generations.

	traj car0	traj car1	traj car2
distance error (m)	2.81	2.97	2.76
speed error (m/s)	1.49	0.96	0.50
misclassifications (%)	12.8	12.9	2.45

Table 4.3: Mean errors for each of the three car trajectories. The numbering is in accordance with figure 4.1(a).

	traj bike0	traj bike1	traj bike2
distance error (m)	1.49	2.27	2.11
speed error (m/s)	0.41	0.66	0.31
misclassifications (%)	1.55	31.2	12.3

Table 4.4: Mean errors for each of the three bicycle trajectories. The numbering is in accordance with figure 4.1(b).

	traj walker0	traj walker1	traj walker2	traj walker3
distance error (m)	2.04	1.98	1.54	1.82
speed error (m/s)	0.24	0.50	0.39	0.18
misclassifications (%)	11.1	14.0	43.6	85.2

Table 4.5: Mean errors for each of the four walker trajectories. The numbering is in accordance with figure 4.1(c).

Figures 4.4, 4.5 and 4.6 illustrate the error measurements for the car, bicycle and walker trajectories in greater detail. In these figures, the simulated junction has been divided into a grid and the mean errors have been computed inside each square (of size 1×1 meter) of the grid. Thus, we can see how the positions of simulated actors affect the errors. The location and viewing direction of the OTUS3D system is represented by an arrow. The most obvious conclusion from these three figures is that there seems to be a general trend of errors in grid squares that are either far away from the OTUS3D system or in the periphery of its field of view being greater. There are of course exceptions to this tendency; the trend mainly seems to hold for the walker trajectories.

In figure 4.7, the ground truth x- and y-values have been subtracted from the estimated x- and y-values for all trajectories. This results in plots illustrating to what extent the estimated trajectories systematically differ from the ground truth trajectories. Since the errors in the x-direction in most cases are below zero, we can conclude that OTUS3D systematically underestimates the x-values. Put differently, OTUS3D estimates that the actors are closer to the system than they actually are. In contrast, the errors in the y-direction are generally significantly smaller and also tend to be positive, meaning that the estimations are skewed a bit too far to the right. This trend is not as strong as the one in the x-direction however.

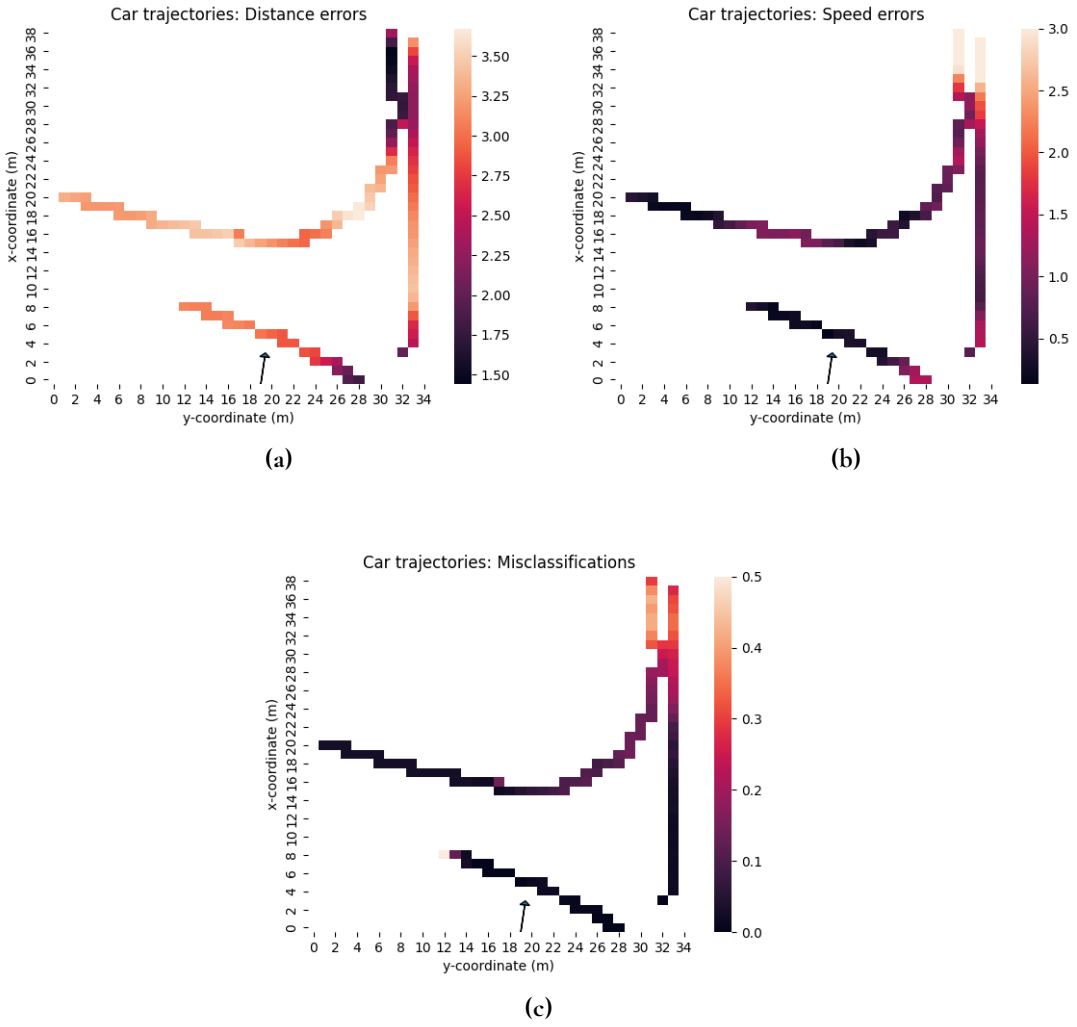


Figure 4.4: Heatmaps showing the distance errors (m), speed errors (m/s) and misclassifications (decimal percentage) of the car trajectories at set positions. Note that the x-axis is the vertical axis while the y-axis is the horizontal one. The arrow close to the bottom of the plots shows the position and direction of the OTUS3D system.

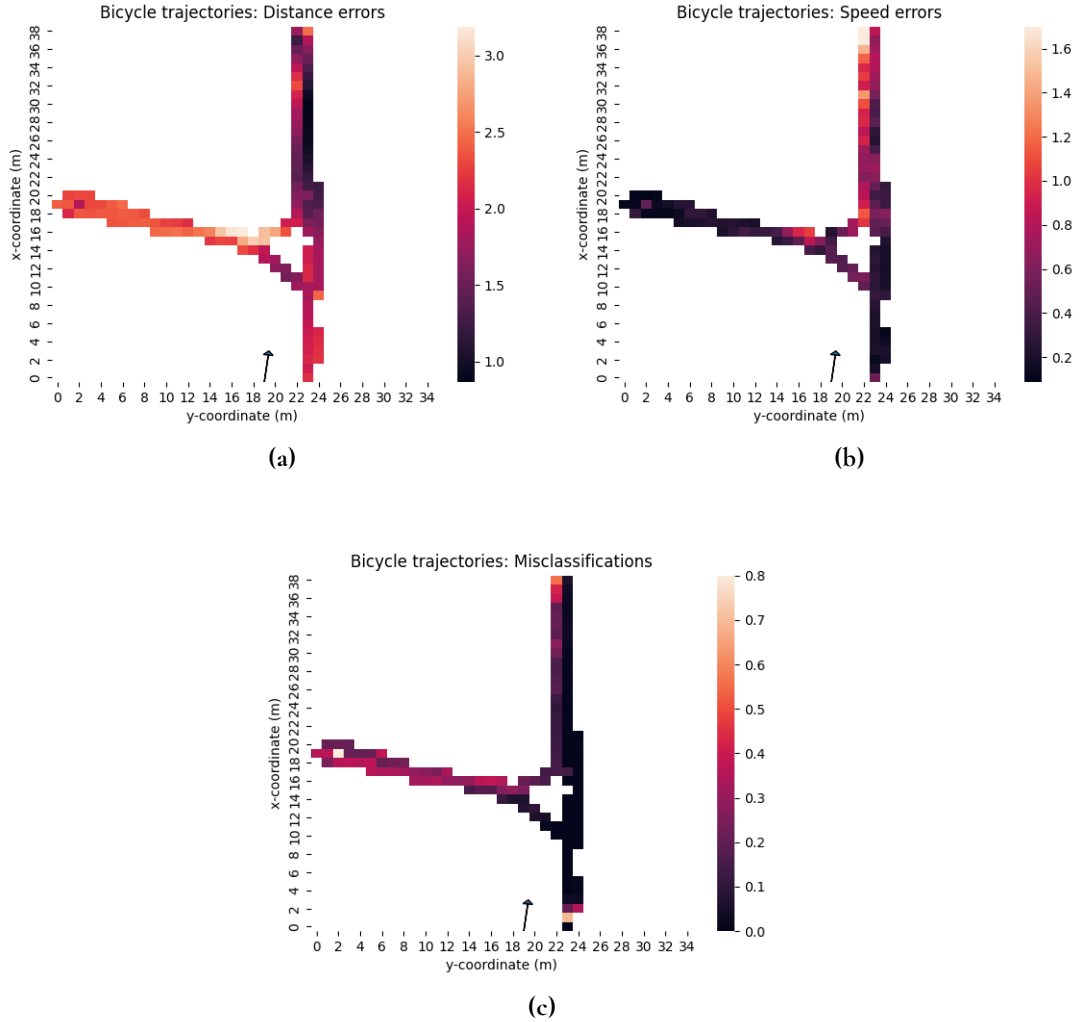


Figure 4.5: Heatmaps showing the distance errors (m), speed errors (m/s) and misclassifications (decimal percentage) of the bicycle trajectories at set positions. Note that the x-axis is the vertical axis while the y-axis is the horizontal one. The arrow close to the bottom of the plots shows the position and direction of the OTUS3D system.

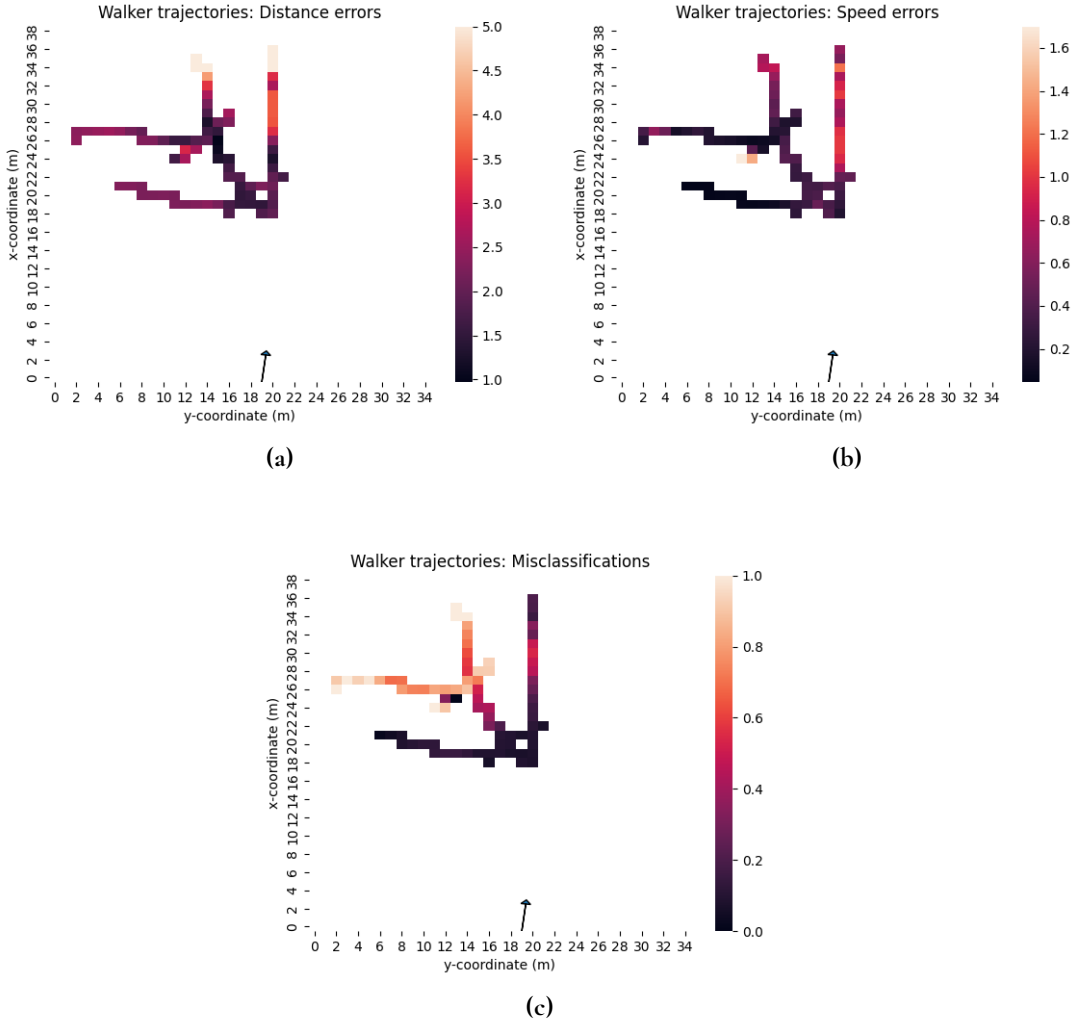
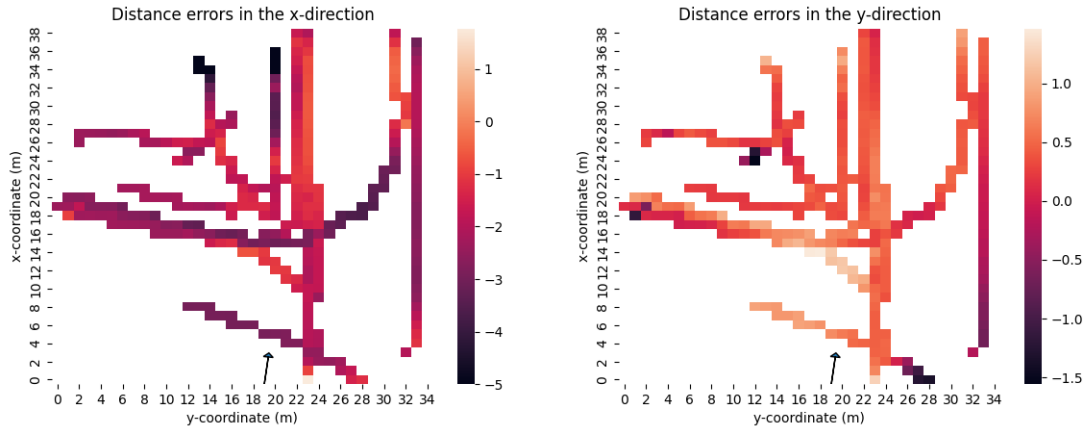


Figure 4.6: Heatmaps showing the distance errors (m), speed errors (m/s) and misclassifications (decimal percentage) of the walker trajectories at set positions. Note that the x-axis is the vertical axis while the y-axis is the horizontal one. The arrow close to the bottom of the plots shows the position and direction of the OTUS3D system.



(a) (b)

Figure 4.7: Heatmaps showing the distance errors ((m), ground truth subtracted from the OTUS3D estimation) in the x- and y-directions respectively for all the trajectories. Note that the x-direction is the vertical direction while the y-direction is the horizontal one. We see that most of the errors in the x-direction are below zero, meaning that OTUS3D systimatically estimates the actors to be closer than they actually are.

Figure 4.8 shows how different luminance values affect the distance errors of the car trajectories. Each point in the plot corresponds to one car trajectory; in total, there are $120 \cdot 3 = 360$ points. As we can see, there appears to be no correlation between the distance errors and luminance values. Figure 4.9 similarly shows the relationship between the sun altitude angles and the distance errors for cars, bicycles and walkers respectively. The expectation here would of course be that scenarios with a lower sun altitude would result in larger errors. In figure 4.9, there are no significant correlations when it comes to the car and bicycle trajectories however. For the walker trajectories, there appears to be a small negative correlation; computing the Pearson correlation coefficient between the two arrays gives a value of -0.17. Figure 4.10 shows the distance errors plotted against the cloudiness values for the car, bicycle and walker trajectories. Here, there are no significant correlations. It should be pointed out that the speed and classification errors for the luminance values, altitude angles and cloudiness values were also examined, but they are not included here since they did not contribute any significant additional information.

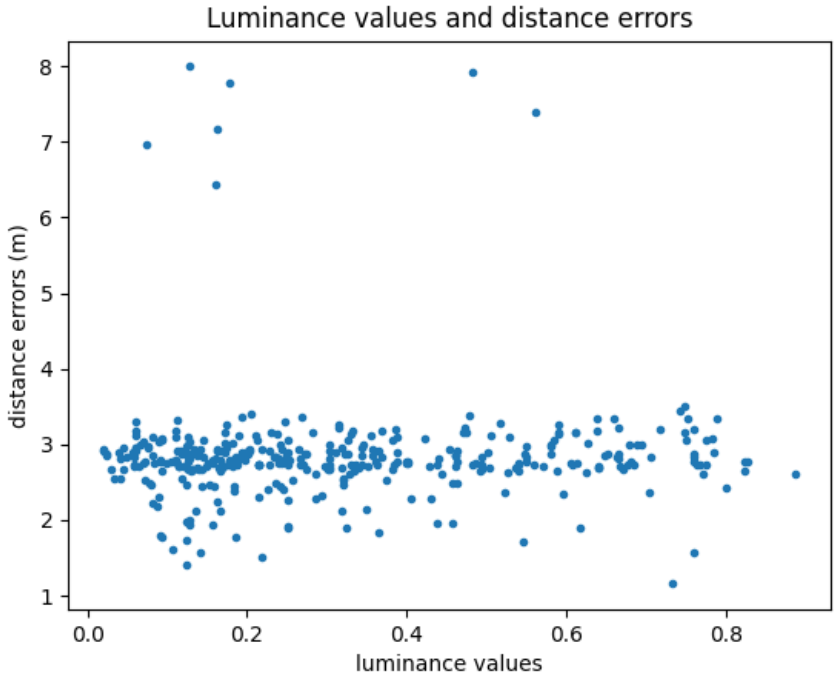
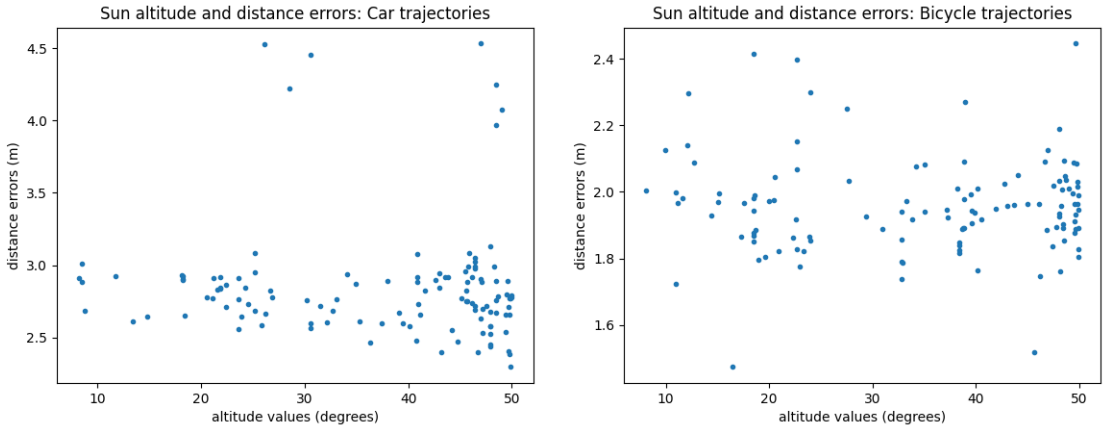
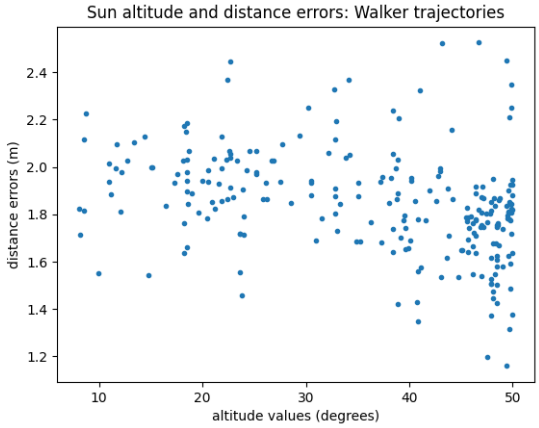


Figure 4.8: Scatterplot of the luminance values and the corresponding mean distance errors for each car trajectory across all simulated scenarios. There appears to be no clear correlation between the errors and luminance values.

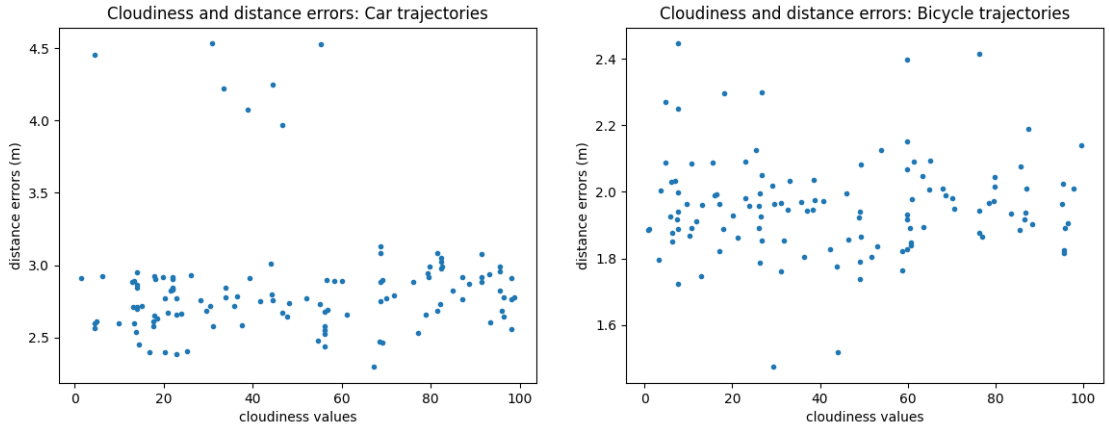


(a) (b)

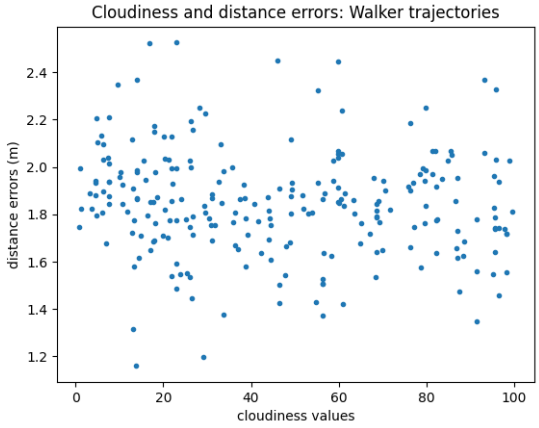


(c)

Figure 4.9: Scatterplots of the sun altitude angles and the corresponding mean distance errors for the car trajectories, bicycle trajectories and walker trajectories respectively. There are no obvious correlations, except possibly for the walker trajectories.



(a) (b)



(c)

Figure 4.10: Scatterplots of the cloudiness values and the corresponding mean distance errors for the car trajectories, bicycle trajectories and walker trajectories respectively. There are no clear correlations.

It is important to note that while the sun altitude angle is important in determining the visibility for a traffic scenario, the azimuth angle has a larger impact on the placement of shadows and is therefore also worth examining. Figure 4.11 shows the azimuth angles and the corresponding distance errors for walker trajectory 2. We see that there is a trend of the errors being lower when the azimuth angle is between 150° and 200°. This can be confirmed by computing the mean of these errors: it was found that this mean was 1.50, compared to 1.60 for errors outside of this range. Another plot showing the relationship between the azimuth angles and distance errors is presented in figure 4.12, this time for car trajectory 2. Here, we see that the errors are larger for azimuth angles between approximately 80° and 130°. The mean of these errors is 2.78, compared to 2.74 for errors outside of the range. Lastly, in figure 4.13(a) the misclassifications for bicycle trajectory 2 are plotted for different azimuth angles. For azimuth values between 200° and 255° the mean error was 15.0 compared to 7.06 for the other errors. The corresponding cloudiness values for the same errors are plotted in figure 4.13(b).

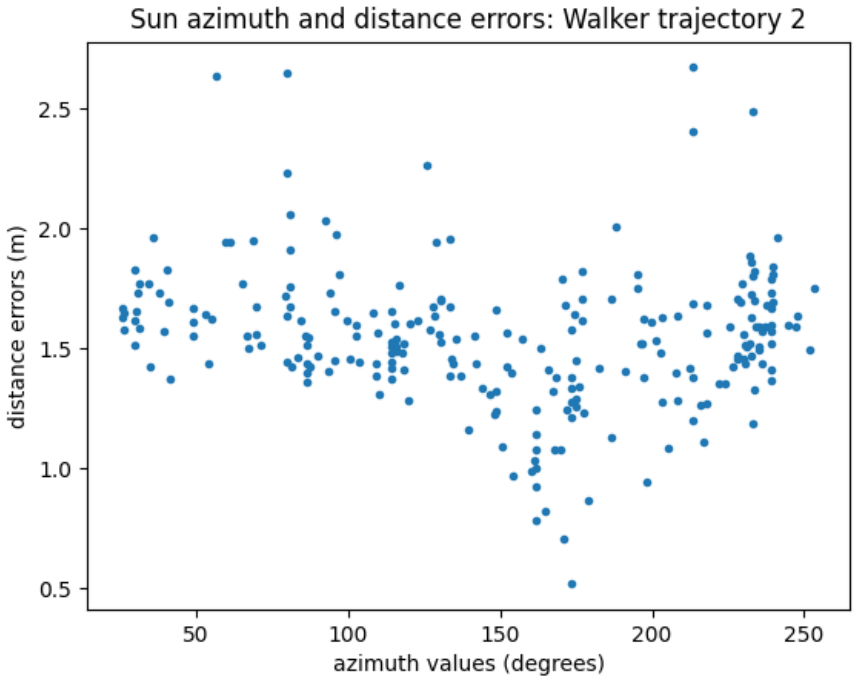


Figure 4.11: Scatterplot of the azimuth values and the corresponding mean distance errors for walker trajectory 2. The errors are quite clearly lower for azimuth angles between 150° and 200°.

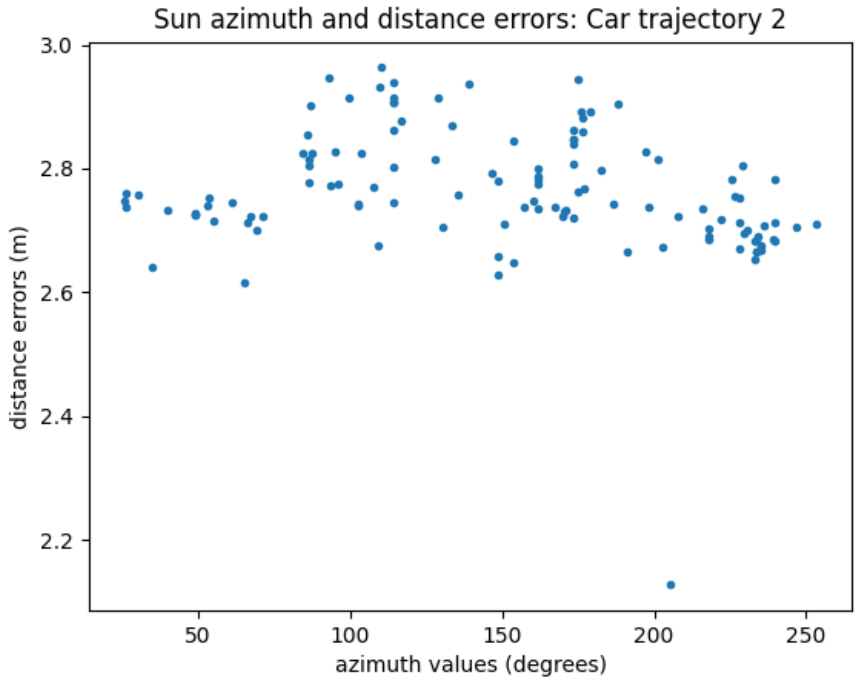
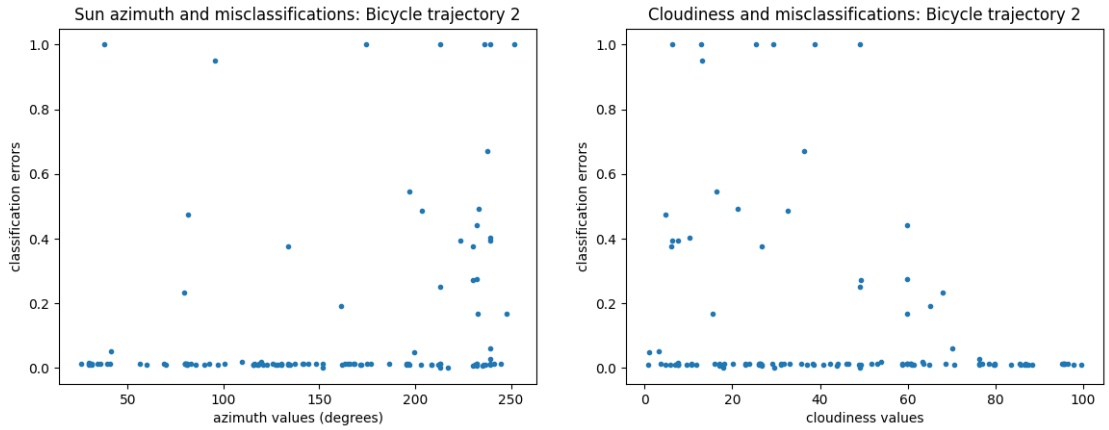


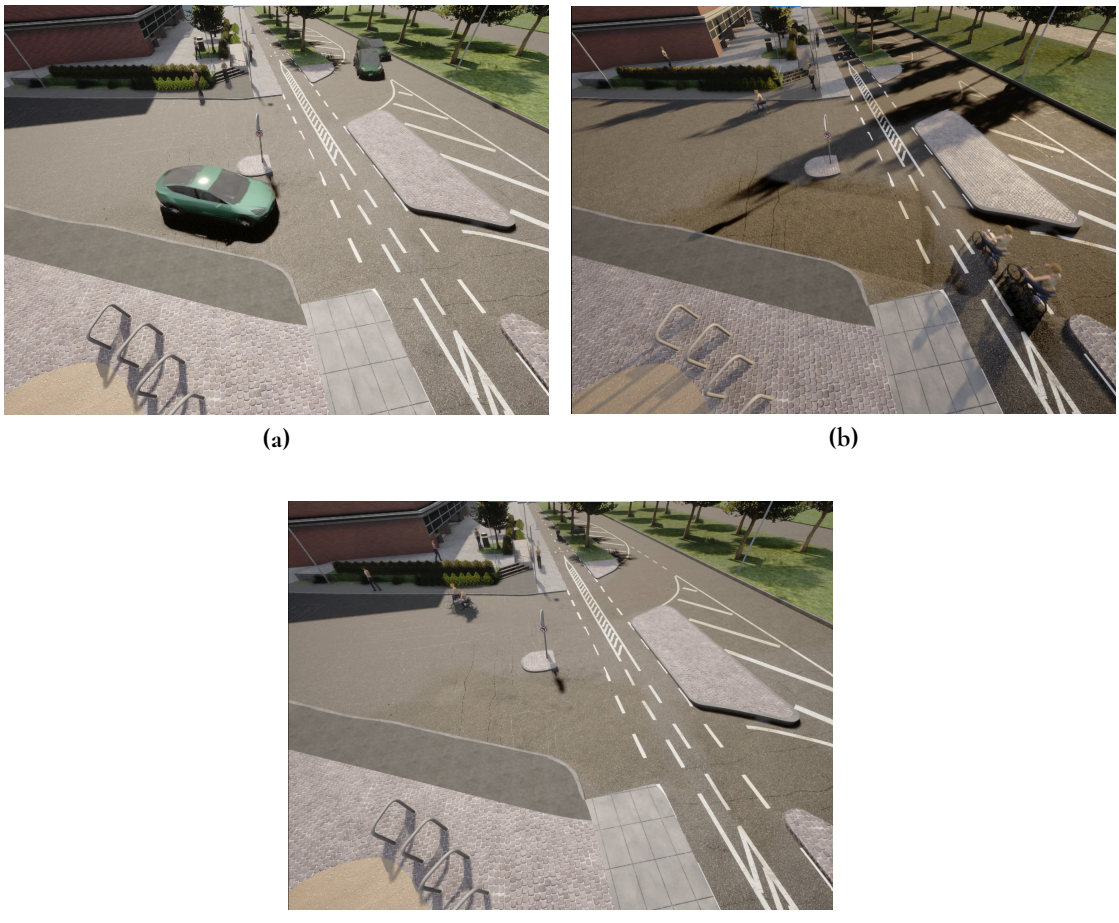
Figure 4.12: Scatterplot of the azimuth values and the corresponding mean distance errors for car trajectory 2. The errors are quite clearly larger for azimuth angles between 100° and 150°



(a) **(b)**

Figure 4.13: In (a), we see a scatterplot of the azimuth values and the corresponding mean misclassifications for bicycle trajectory 2. We see that the errors are larger for large azimuth angles. In (b), we see the same misclassifications plotted against different cloudiness values. Here, the errors are larger for small cloudiness values.

Some investigations were also done to assess the effect of occlusion on the estimations made by OTUS3D. To this end, we created three sets, each of which contained scenarios where some occlusion event occurred between two trajectories. In the first set (containing 21 scenarios), car trajectory 1 started 2.5 seconds or less before car trajectory 0. In the second (containing 10 scenarios), bicycle trajectory 2 started 1 second or less before bicycle trajectory 0. In the third (containing 13 scenarios), bicycle trajectory 1 started between 3.5 and 5 seconds before bicycle trajectory 2. In the first two sets, the occlusion event happened at the start of the two trajectories, while in the third, the event happened right in the middle of the intersection. Images illustrating the first, second and third occlusion events are presented in figure 4.14(a), (b) and (c) respectively.



(a) **(b)**

(c)

Figure 4.14: The top right of (a) shows the occlusion event between car trajectory 0 and car trajectory 1. The bottom right of (b) shows the occlusion event between bicycle trajectory 0 and bicycle trajectory 2. The left of (c) shows the occlusion event between bicycle trajectory 1 and bicycle trajectory 2.

Tables 4.6, 4.7 and 4.8 each correspond to one of the three aforementioned sets and contain the resulting mean errors of the trajectories involved in the occlusion. The mean errors across all trajectories are also included for reference. Examining table 4.6, we see that the occlusion

has no significant effect on the errors for car trajectory 1. For trajectory 0 however, there seems to be a large effect from the occlusion both on the mean distance error and mean speed error. That we see effects on trajectory 0 but not on trajectory 1 is entirely expected since in these scenarios, the former car is blocked by the latter. In table 4.7, we see that the errors are more significant for bicycle trajectory 0 than for trajectory 2 when there is occlusion, with the exception of the number of misclassifications which is large in both cases. Lastly, in table 4.8 we see that the speed error is the most significant error for bicycle trajectory 1 while both the speed error and especially the classification error are large for trajectory 2.

	traj car0 occl	traj car0 all	traj car1 occl	traj car1 all
distance error (m)	4.13	2.81	2.92	2.97
speed error (m/s)	2.39	1.49	0.72	0.96
misclassifications (%)	12.8	12.8	11.7	12.9

Table 4.6: The first and third columns contain the mean trajectory errors for the 21 scenarios where car trajectory 1 started 2.5 seconds or less before car trajectory 0. The second and fourth columns contain the mean trajectory errors across all of the scenarios, the same values as in table 4.3.

	traj bike0 occl	traj bike0 all	traj bike2 occl	traj bike2 all
distance error (m)	1.60	1.49	2.11	2.11
speed error (m/s)	0.52	0.41	0.26	0.31
misclassifications (%)	7.96	1.55	34.7	12.3

Table 4.7: The first and third columns contain the mean trajectory errors for the 10 scenarios where bicycle trajectory 2 started 1 second or less before bicycle trajectory 0. The second and fourth columns contain the mean trajectory errors across all of the scenarios, the same values as in table 4.4.

	traj bike1 occl	traj bike1 all	traj bike2 occl	traj bike2 all
distance error (m)	2.19	2.27	2.18	2.11
speed error (m/s)	0.85	0.66	0.46	0.31
misclassifications (%)	35.8	31.2	21.9	12.3

Table 4.8: The first and third columns contain the mean trajectory errors for the 13 scenarios where bicycle trajectory 1 started between 3.5 and 5 seconds before bicycle trajectory 2. The second and fourth columns contain the mean trajectory errors across all of the scenarios, the same values as in table 4.4.

4.3 Discussion

4.3.1 Answering the research questions

In this section, we will analyze the results further, answer the research questions as well as point out future research that could be done to further investigate these questions. We will start by reminding ourselves of the three research questions:

RQ1: Can a low-fidelity digital model of a traffic junction be used to find faults in Viscando's system?
RQ2: Is SBST an efficient and effective approach to provoke system faults in the digital model?
RQ3: Which parameter configurations are the most effective at provoking system faults?

When it comes to the results relating to the performance of the NSGA-II algorithm, which is related to RQ2, it is difficult to draw conclusions. More generations would have to be evaluated to determine whether NSGA-II is efficient and effective at finding faults in OTUS3D. However, there are still signs pointing towards the approach of using SBST to generate simulated traffic scenarios to be one worth further research, as will be argued below.

Before answering RQ1 and RQ3, we will need to examine the results a bit more closely.

As was previously mentioned, figure 4.7 shows that OTUS3D was found to have a clear tendency to underestimate the x-values of detected actors, i.e. how far away they were in the viewing direction. The reasons behind this tendency are largely unknown at the time of writing. It is possible that the discrepancy between the digital model and the real-life Lindholmen junction is part of the problem. We did take a great deal of care, including making several distance measurements, to have the digital model be as close to reality as possible, but some differences are unavoidable.

When it comes to the mean errors for the car trajectories, table 4.3 shows that the errors were smallest for trajectory 2, which is unsurprising considering its closeness to the OTUS3D system. In contrast, the small size of the distance errors for points far away, in the top right of figure 4.4(a) is more of a mystery; it contradicts the speed and classification errors which are both large at the same spot. And while simple explanations can be made as for why the car trajectories generally have larger speed errors and lower classification errors (they have greater speed in general and their size make them easier to classify) it is not as obvious why the distance errors are as large as they are.

Regarding the errors for the bicycle trajectories, table 4.4 shows that the errors are the largest for trajectory 1. Figure 4.5(a) shows that the sharp turn in the middle of trajectory 1 is associated with large distance errors. This is also true for the similar turn made in car trajectory 1. There also seem to be elevated errors before this turn, when the bicycle is moving directly towards the OTUS3D system. These errors are also present for the walker trajectory taking a similar path, as seen in figure 4.5(a)-(c). Lastly regarding the bicycle errors, it should be mentioned that in consultation with Viscando it was pointed out that regions close to the

edges of the field of view of OTUS3D are more prone to errors. The elevated distance and classification errors present in the left part of figure 4.4(a) and (c) are possibly a result of this.

For the walker trajectory errors, there is less to discuss. Clearly, all three errors are large for points far away. Walkers were also more difficult to correctly classify in general.

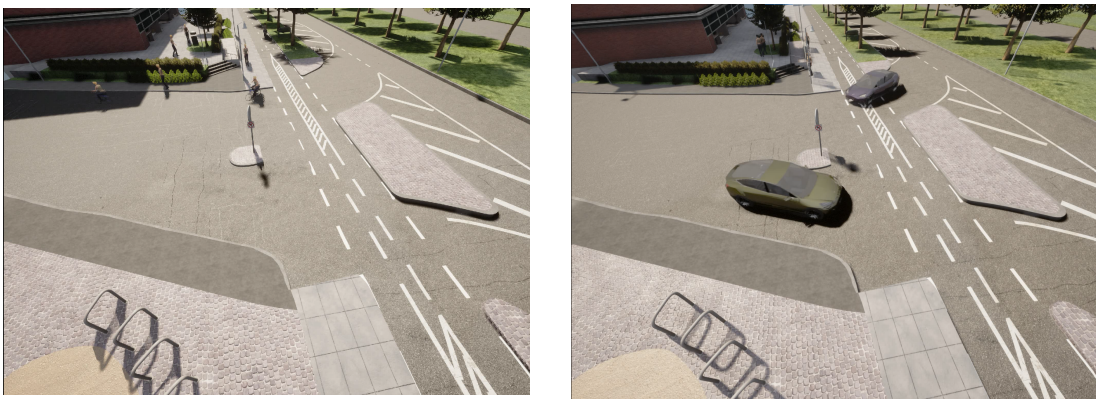
The fact that the cloudiness values and the sun altitude angle had such a small impact on the scenarios was surprising. It is likely that the minimum and maximum azimuth angles did not allow for scenarios to be sufficiently dark to create major estimation problems.

The three plots containing azimuth angles all potentially point towards the significance of shadows on the estimations. In figure 4.11, the angles that resulted in smaller errors correspond to the shadow from the large building not being present, giving a clearer view of that walker trajectory, see figure 4.15(a). For car trajectory 2, it is possible that the small shadow in front of the car (see figure 4.15(b)) that is created when the azimuth angle is in the 100º to 150º range is the cause of the elevated distance error that we see in figure 4.12. Similarly, the large classification errors for bicycle trajectory 2 seen in figure 4.13 when the azimuth angles were large could also be due to shadows creating problems, see figure 4.15(c). In this case, there is also a trend where lower cloudiness values are associated with larger errors. This shows that although larger cloudiness values do decrease visibility, they also make shadows less pronounced which could in some cases make errors smaller.

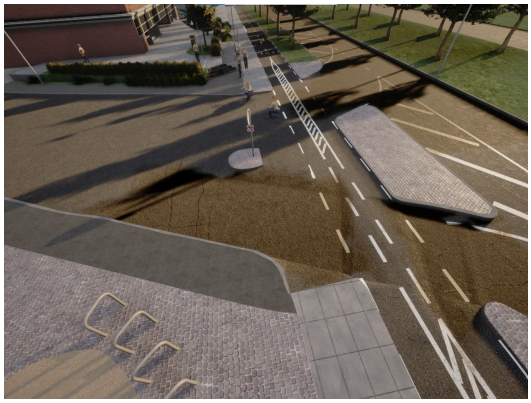
Now, turning our attention back to RQ1, we can say that yes, the digital model could indeed be used to find faults in OTUS3D. Some faults were more expected, like the difficulty of estimating actors in occlusion events or the difficulty of estimating actors far away in the viewing direction or far away into the periphery. Others were not so expected, such as the systematic distance error in the x-direction. Further investigations would have to be done to determine which of these faults are largely due to the properties of the digital model or due to the simulator, and which are transferable to reality. One obvious but time consuming way of testing this would be to build a new digital model and then test this new model and the old one in both CARLA and in a different simulator.

We can also draw some conclusions regarding RQ3. The results suggest firstly, that the estimation errors were very much dependent on the properties of the trajectories. Generally, trajectories that had paths far away from the system under test and far into the periphery of its field of view proved to be particularly challenging to estimate; this was especially the case for walker trajectories. Secondly, occlusion events seemed to be a source of significant estimation errors. The placement of shadows seemed to be of less importance, but still a possible source of errors in some cases. The luminance values of the cars, the altitude of the sun and the amount of cloudiness were all found to be relatively insignificant sources of errors. In the case of all of these parameters, further investigations are needed to draw more and stronger conclusions. For instance, it is an open question as to how low the illumination in a simulated scenario can become before the estimation errors start to increase. Much more can also be done to investigate the circumstances under which occlusion events lead to increased errors. The fact that the effects of shadows could only be found in such a limited number of cases also merits further investigation. Other parameterizations, such as varying the velocities or

the trajectories themselves or introducing more weather parameters could also be done, but that would also increase the complexity of the optimization problem.



(a) (b)



(c)

Figure 4.15: The top left of (a) shows that the shadow from the large building is small, which leaves the entirety of walker trajectory 2 in sunlight. (b) shows a small shadow in front of car trajectory 2. (c) shows the long shadows that possibly cause problems in estimating bicycle trajectory 2.

4.3.2 Contributions

There are three main contributions of this work. Firstly, it provides a digital model of the Lindholmen junction that can either be further developed or used for further testing of OTUS3D as it is. Secondly, this work shows that this digital model can be used to find faults in OTUS3D when the trajectories and weather conditions are within the range of what would commonly be encountered in the junction. Third, this work identifies some traffic scenario parameters that seem to be more effective than others at provoking system faults, which could be valuable knowledge for future research; particularly when making future scenario parametrizations. In short, this work shows that search-based simulated testing of OTUS3D is possible while providing some results and insights that could potentially be of use to future research.

4.3.3 Lessons learned and limitations

There are a few experiences taken from this work that could hopefully be helpful to future research.

First, a few words on our experience of using the CARLA simulator. Despite the complexity and the sheer amount of things that are possible to do in CARLA, it is still easy to learn and to use. There were certainly a large number of excessive features considering that this project had such a limited scope, which in turn made the installation process quite long and complicated. At the same time, some features were not properly developed at the time when the simulator was used: mainly a simple way to generate complex traffic scenarios with the use of waypoints. The scenario generation that we made, with actors manually steering towards each waypoint, was a bit complicated to implement, but the results were satisfactory. Combining the use of CARLA with the parameter generation was particularly appropriate since all code execution could be done with Python scripts. All in all, our experience of working with CARLA was positive.

Next, we discuss the fitness function that was used in this report. It consisted of three objectives: distance errors, speed errors and misclassifications. One potential problem with using these three measurements is that they to an extent are dependent, as can be seen for instance by comparing figure 4.4 (b) and (c). Thus, they partly describe the same thing without giving any new information. An arguably more important problem however, is that a large amount of information is lost when error measurements of seven largely independent actors are crammed into just three numbers. One solution to this problem could of course be to increase the number of objectives, and to for instance use one objective for each actor. Doing so would make NSGA-II an inappropriate algorithm to use however, since it underperforms when using a large number of objectives [13]. Other algorithms such as NSGA-III [13] and MOSA [23] that have been developed to handle many-objective optimization problems would be more suitable in this case. An alternative could of course be to simply lower the number of actors taking part in the simulation. This approach would be more in line with previous research [6], [7], [16] where the scenarios have as few as two actors.

There is also the possibility to include objectives that can test aspects of OTUS3D that were overlooked in this report. One example is related to an issue that appeared when matching the ground truth trajectories with the trajectories estimated by OTUS3D. It was found that while the ground truth scenarios always had exactly seven trajectories, there were often ten or more trajectories that were estimated by OTUS3D. This problem was solved by combining each ground truth trajectory with the estimated trajectory that minimized the mean distance error. More work could be done to take a wider view at all of the estimated trajectories and possibly include them into some objective function.

There are some more limitations of this work that should be pointed out. First, and perhaps most obvious, is the question of relevance that arises when using simulated scenarios to test a system that has been developed to measure real-life scenarios. This question is of course related to the discrepancies between the digital model and the real-life Lindholmen junction. To what extent the results found in this work are transferable to reality will have

to be decided by future investigations.

Another limitation of this work is the existence of counterexamples that contradict some of the drawn conclusions. While it seems reasonable to assume that the placement of shadows contributed to the errors shown in figures 4.11–4.13, there were many seemingly similar cases where the shadows did not affect the errors. This calls into question whether the shadow placements really were the main cause of the errors shown in these figures. Similarly, figures 4.4–4.6 show that some of the errors were small for points far away from the OTUS3D system and significantly larger for points close to it, contrary to our expectations and conclusions. What all of this speaks to is that the relationship between input and output in an ML system like OTUS3D is highly complicated, and one has to be careful to not draw too strong conclusions.

Lastly, it should be pointed out that while the trajectories that we used for the testing were a realistic representation of the trajectories that occur in the real Lindholmen junction, they were far from comprehensive. It is possible that using trajectories where walkers and bicyclists follow less orthodox pathways would be a suitable approach to provoke system faults within OTUS3D.

Chapter 5
Conclusions

To answer RQ1, we created a digital model of the Lindholmen junction that could be used to simulate a wide range of traffic scenarios in CARLA. Videos of these synthetic scenarios provided input to OTUS3D, which returned estimations of the actors (cars, bicycles, walkers) participating in the simulation. These estimations could then be compared with the ground truth. Noting that certain scenarios produced particularly large errors between the estimations and ground truth, we can conclude that the digital model indeed could be used to find faults in Viscando's system.

To answer RQ2, we implemented a version of the GA NSGA-II and used it to generate parameters that defined a traffic scenario. We also implemented a baseline model which generated scenario parameters randomly. We were only able to execute three generations of both models, which was not enough to give a sufficient answer to RQ2. However, the fact that some parameter configurations generated by NSGA-II were particularly good at revealing faults in the system under test suggests that a future search-based approach could be suitable to further investigate the parameter configurations used in this work or potentially other ones.

To answer RQ3, we investigated the effect that cloudiness, occlusion, the color of the cars, and the position of the sun had on the estimation errors. We also examined each individual trajectory to investigate which positions were the most prone to errors. It was found that there was a large difference in errors depending on the trajectory, with certain trajectories being more "critical" than others. It was also found that occlusion events, and in some cases specific placements of shadows, were important factors in creating large errors. The sun altitude angle, the level of cloudiness and the luminance values of the cars were less impactful.

We can recommend the use of CARLA to simulate traffic scenarios, particularly when using an SBST approach to generate these scenarios. Future research into testing OTUS3D or similar traffic measurement systems should further investigate which parameter configurations are critical at provoking system faults; testing of the circumstances under which occlusion,

illumination and shadows result in elevated errors appears particularly promising. More unorthodox bicycle and walker trajectories could potentially be used to further test the limits of OTUS3D. If future testing is done with an SBST approach, we recommend either reducing the number of actors participating in the simulations or increasing the number of objective functions and using an algorithm that is better suited for many-objective optimization than NSGA-II is. There is also the possibility of including new objective functions that evaluate other aspects of OTUS3D. One such aspect could be the tendency of OTUS3D to introduce new trajectories. An objective that punishes this behavior could for example be introduced. Lastly, more research is needed to determine to what extent the faults found in this report are transferable to reality.

References

[1] About SBST. https://sbst21.github.io/. Accessed 2022-02-14.

[2] Deap: crossover.py. https://github.com/DEAP/deap/blob/master/deap/tools/crossover.py. Accessed 2022-09-22.

[3] Python api reference. https://carla.readthedocs.io/en/latest/python_api/. Accessed 2022-06-07.

[4] Wasif Afzal, Richard Torkar, and Robert Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.

[5] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2015.

[6] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Testing advanced driver assistance systems using multi-objective search and neural networks. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 63–74, 2016.

[7] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Testing vision-based control systems using learnable evolutionary algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1016–1026. IEEE, 2018.

[8] Markus Borg, Raja Ben Abdessalem, Shiva Nejati, François-Xavier Jegeden, and Donghwan Shin. Digital twins are not monozygotic–cross-replicating ADAS testing in two industry-grade automotive simulators. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 383–393. IEEE, 2021.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[10] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, first edition, 2001.

[11] Kalyanmoy Deb and Ram Bhushan Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(2):115–148, 1995.

[12] Kalyanmoy Deb and Mayank Goyal. A combined genetic adaptive search (geneas) for engineering design. *Computer Science and Informatics*, 26:30–45, 1996.

[13] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[14] Kalyanmoy Deb and Santosh Tiwari. Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization. *European Journal of Operational Research*, 185(3):1062–1087, 2008.

[15] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[16] Hamid Ebadi, Mahshid Helali Moghadam, Markus Borg, Gregory Gay, Afonso Fontes, and Kasper Socha. Efficient and effective generation of test cases for pedestrian detection-search-based software testing of baidu apollo in SVL. In *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*, pages 103–110. IEEE, 2021.

[17] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, second edition, 2015.

[18] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[19] Abdullah Konak, David W. Coit, and Alice E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering System Safety*, 91(9):992–1007, 2006. Special Issue - Genetic Algorithms and Reliability.

[20] Siew Mooi Lim, Abu Bakar Md. Sultan, Md. Nasir Sulaiman, Aida Mustapha, and K. Y. Leong. Crossover and mutation operators of genetic algorithms. *International Journal of Machine Learning and Computing*, 7(1):9–12, 2017.

[21] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

[22] Mahshid Helali Moghadam, Markus Borg, Mehrdad Saadatmand, Seyed Jalaleddin Mousavirad, Markus Bohlin, and Björn Lisper. Machine learning testing in an ADAS case study using simulation-integrated bio-inspired search-based testing. *Journal of Software: Evolution and Process*, n/a(n/a):e2591.

[23] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. Reformulating branch coverage as a many-objective optimization problem. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–10, 2015.

[24] V. Riccio, G. Jahangirova, A. Stocco, et al. Testing machine learning based systems: a systematic mapping. *Empirical Software Engineering*, 25:5193–5254, 2020.

[25] Francisca Rosique, Pedro J Navarro, Carlos Fernández, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19(3):648, 2019.

[26] Abdel Salam Sayyad and Hany Ammar. Pareto-optimal search-based software engineering (posbse): A literature survey. In *2013 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pages 21–27, 2013.

[27] Shuai Wang, Shaukat Ali, Tao Yue, Yan Li, and Marius Liaaen. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 631–642, 2016.

[28] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 48(1):1–36, 2022.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2023-08-24

EXAMENSARBETE Generating Synthetic Scenarios to Test an AI-Enabled Traffic Measurement System
STUDENT Elias Sjöberg
HANDLEDARE Markus Borg (LTH)
EXAMINATOR Per Runeson (LTH)

Simulering av trafiksituationer för testning av ett AI-system

POPULÄRVETENSKAPLIG SAMMANFATTNING **Elias Sjöberg**

Den pågående utvecklingen inom AI har gjort det möjligt att samla in och analysera trafikdata på ett helt nytt sätt. I detta examensarbete har simulerade trafiksituationer genererats för att testa ett AI-system som samlar in trafikdata.

De senaste årens utveckling inom AI har revolutionerat flera teknikområden. Företaget Viscando använder sig av modern AI-teknik för att göra trafiken i våra städer smartare, säkrare och mer hållbar. Genom att samla in och analysera stora mängder data över rörelsemönster hos fotgängare, bilar och cyklister kan slutsatser dras för till exempel var, när och hur olyckor sker, bilköer uppstår eller trafikförseelser begås. Dessa slutsatser kan sedan användas för att utvärdera och utveckla rådande infrastruktur. Allt detta är möjligt tack vare Viscandos AI-system OTUS3D, som placeras ut där man vill samla in trafikdata.

Syftet med detta examensarbete var att bidra till utvecklingen av OTUS3D genom att försöka hitta trafiksituationer där detta AI-system underpresterar. För att bättre kunna styra över trafiksituationer att testa så simulerade vi dessa i en digital miljö. Därför byggde vi en digital modell (se bild) över en korsning i Lindholmen i Göteborg där ett exemplar av OTUS3D finns utplacerat; målet var att den simulerade miljön skulle vara så lik den riktiga som möjligt. Därefter använde vi en sökalgoritm för att på ett så effektivt sätt som möjligt anpassa de simulerade trafiksituationerna och på så sätt hitta de situationer som resulterade i störst fel hos OTUS3D.

Utifrån resultaten kunde vi identifiera några egenskaper hos trafiksituationerna som resulterade i att OTUS3D uppvisade särskilt stora fel. Dels visade det sig att bilar, cyklister och fotgängare som rörde sig på vissa platser skapade problem. Likadant var det i situationer där bilar och cyklister blockerade varandra och i situationer där skuggor försämrade sikten. Solhöjd, molnighet och färg på bilarna verkade inte påverka felen i lika hög utsträckning. Vår bedömning är att det finns stora möjligheter att i framtiden bygga vidare på dessa resultat; dels för att ytterligare kunna fastslå vilka trafiksituationer som skapar särskilt stora fel och dels för att kunna avgöra vilka fel i den simulerade miljön som är mest överförbara till verkligheten.