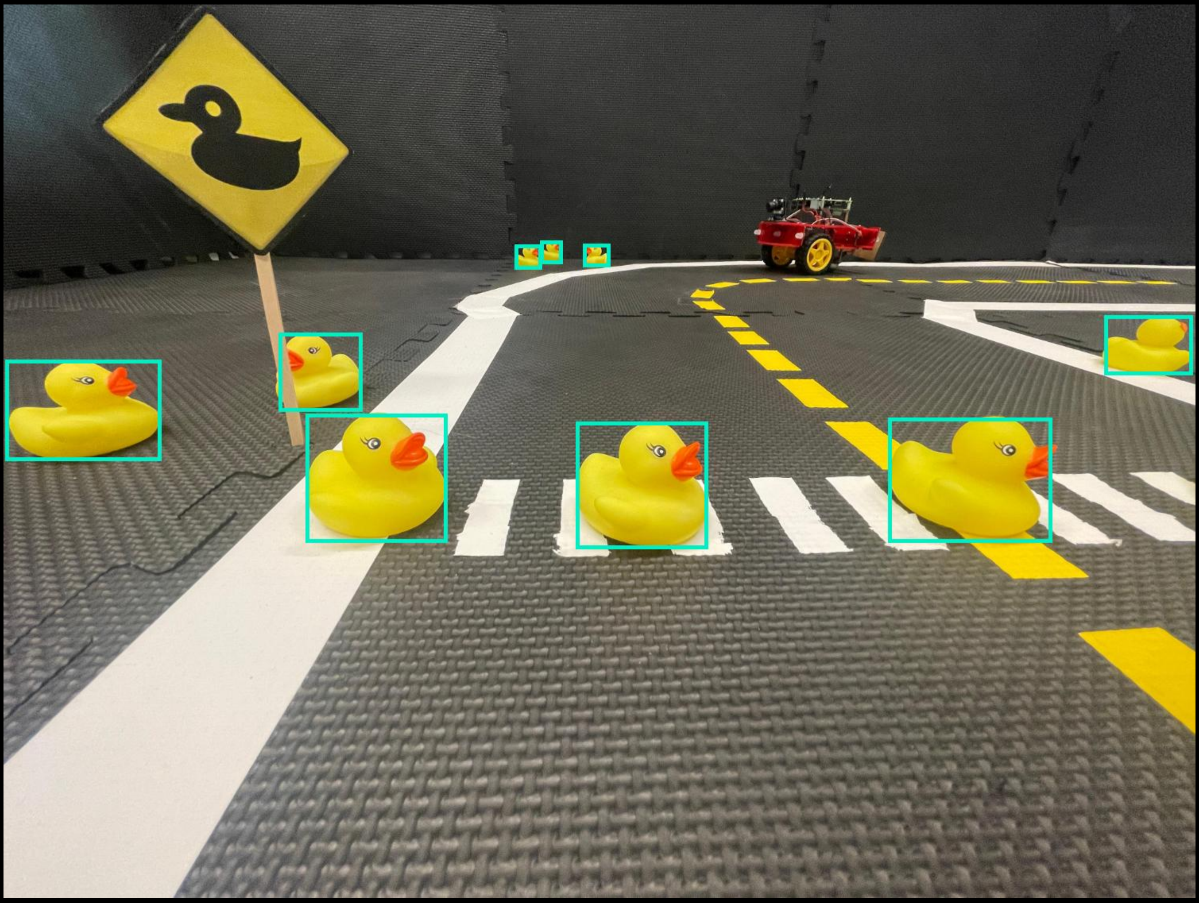
Quantifying the Reality Gap in Abstracted Pedestrian Detection in Simulated Environments



Fyor Klein Gunnewiek

Bachelor Informatica

University of Amsterdam

Quantifying the Reality Gap in Abstracted Pedestrian Detection in Simulated Environments

Fyor Klein Gunnewiek

June 23, 2023

Supervisor(s): Dr. A. (Arnoud) Visser

Abstract

As the usage of Autonomous Vehicles continues to grow rapidly, concerns about their effectiveness and safety have rightfully emerged. Simulation is currently not the preferred method for gathering data on driverless vehicles even though there are less safety and cost considerations. This is because of a significant gap between the simulation used for training and the reality that it is intended for. This study aims to analyze the reality gap in object detection specifically related to pedestrian recognition across various simulations. Given the complexity involved in understanding the causes of this difference, multiple simulation methods are explored to gain insight into the current state of this gap in learning platforms for autonomous driving within an abstract environment. To enable this research, three comprehensive and meticulously annotated datasets were created, enabling us to get an insight into the present state of pedestrian detection. By comparing the implementation of simulation methods and their accuracy in capturing real-world scenarios, conclusions about their effectiveness and suitability for simulation-to-reality training can be drawn.

Acknowledgements

I would like to thank my supervisor, Arnoud Visser[1], for their guidance during my thesis. Although I encountered significant difficulties with the DuckieTown platform, including both the simulation and physical robot, as well as my troubles with Turtlebot in the Gazebo simulation, my supervisor provided valuable assistance in steering me towards a more successful and achievable trajectory for this project.

I would like to express my sincere appreciation to Joey van der Kaaij[2], the Lab Manager at the Intelligent Robotics Lab of the University of Amsterdam, for his invaluable contribution in helping set up the Unity Environment and his efforts in debugging the DuckieBot. Despite encountering challenges, Joey's dedication and assistance were crucial to the progress of my bachelor thesis. I am very thankful for his support and assistance throughout the project.

Do not waste to much white space. The Acknowledgements are great for the 4th page, with the abstract on the 2nd

[1] https://staff.fnwi.uva.nl/a.visser/
[2] http://joeyvanderkaaij.com

Contents

CHAPTER 1

Introduction

Autonomous Driving Vehicles have made enormous progress in the last few years, to the point of mass availability for consumers [1]. Because human drivers cause an unbelievably large amount of accidents whilst driving, the push for non-human drivers has been increasing rapidly. The expectation is that these machines will be better, because they can always pay attention, don't get tired or drive recklessly. However, proving this statistically has so far been a challenge due to the difference in data between human and non-human driven miles, making it difficult to quantifiably demonstrate this risk reduction, relative to humans [2].

To drive without human interaction many problems are needed to be solved, like lane-following, localisation, environmental modelling and high-level safe decision making. However, this thesis will focus on **pedestrian detection**, which is a problem that has had quite some development over the years, but still hasn't reached its full potential. Aside from assisting in autonomous driving this problem is also relevant to numerous other real-world applications, like video surveillance [3], action recognition and tracking [4].

Claim, so give citation

Pedestrian detection has historically been a popular problem to tackle as it stems from the classic problem from computer vision: Object Detection. The basic detection of a singular person in most environments was never the true problem to solve. The problem has never been getting good at pedestrian detection, but truly mastering it [5]. In recent years the advancement has been tackled by looking at specific scenario's where detections would fail. When pedestrians were near, detections would fall apart, often due to a large overlap causing predictions to be fused together [6]–[8]. Pedestrians who were partially occluded were also causing problems, also especially when in large crowds of people [3], [9]. In these weak areas small incremental improvements have been achieved and have had an impact on pedestrian detection as a whole. However, when looking at the development and expectation from years ago [10] in comparison to today the mastery still has not been achieved [11]. Even the state of the art still has trouble when the domain shifts even slightly, because they have become tailored for target datasets and their design reflects this by being less generalizable [4].

Only on page 20-21 examples of your experiments are given. Give a preview in your Introduction.

1.1 Reality Gap

To train models for detecting pedestrians in dynamic complex environments, large real-world datasets are almost always used. The reason for not using simulated data is the difference in domain realism, which is often referred to as the reality gap. This difference can be visual, but also physical. Getting this gap as small as possible could enable training in simulation which can be a large time and monetary improvement. With simulation also comes the advantage of using automatic labelling of data, which is also a very large optimisation to the entire process of gathering training data.

It is possible to decrease this gap by changing the training process or the simulation. One of the most common ways to decrease the gap is using domain randomisation, which is a simple technique for training models on simulated images by partly randomising the visual [12] and or physical [13] domain. The thought behind domain randomisation is that with enough variability in the simulator, the real world, with all its very specific but hard to define peculiarities, might be seen as just another variation. The result is a model that is a lot more environment invariant.

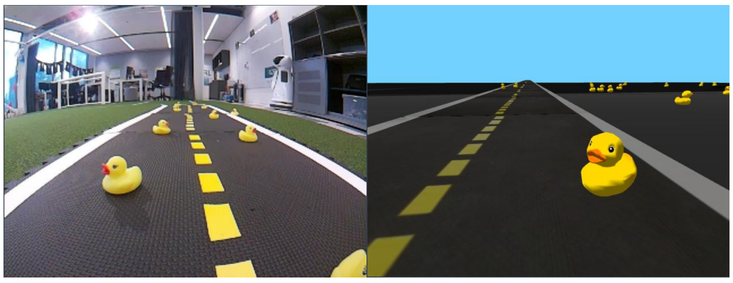
Another method that is seen as an alternative to domain randomisation is domain adaptation, which means trying to get the domain to be as close to reality as possible. This is often done by making simulations more realistic and detailed. Another approach is by using image-to-image translations to approximate the difference between simulation and reality. This translation can then be applied to new images to make them more realistic at a way lower cost than upgrading or improving an entire simulation [14], [15]. This same method has been shown to convert synthetic images from a video game to a very realistic looking image [16]. However, it did suffer from fitting too closely to the dataset it was trained on, changing large parts of the image outright instead of only improving the visual fidelity.

Even with no adaptation to reality some have been able to get state-of-the-art results with completely synthetic dataset [17]. With a large enough dataset the domain randomisation was obtained through the large variety in different environments used.

1.2 DuckieTown

To analyse autonomous driving concepts and problems in a more abstract and simplified environment, many educational platforms have been created to allow for a lower barrier of entry when entering this area of research [18]. One of these platforms is DuckieTown [19], which is a platform with this specific purpose in mind. It has a supported simulation and also an equivalent platform for real-world testing, which makes it an ideal platform for testing the transfer from simulation to reality. It provides a world for a small robot (DuckieBot) to drive round an abstract world (DuckieTown) with rubber duckies to simulate pedestrians. The simulation this platform provides is very basic, visually and physically, which creates problems when porting a simulation trained vehicle to the real world. Any trained model in this simulation will be trained to perform actions only within the domain on which it has trained. The real-world, even though it is somewhat similar, learnings often won't transfer directly.

(a) The second edition of AI-DO took place at the 2019 International Conference on Robotics and Automation (ICRA), with finals held in Montréal, Canada, in May 2019.



(b) Two images side by side indicating the lack of visual clarity in the DuckieTown platform between reality and simulation. The difference noticed by models trained in simulation between these is the simulation-to-reality gap.

There was an attempt made of porting trained Machine Learning algorithms from the DuckieTown sim to reality whilst trying to close the reality gap [20]. The attempt was unsuccessful referencing the gap between simulation and reality to be too large as one of the problems. However, others were successful crossing this gap using a Proximal Policy Optimization (PPO) method [21] for Reinforcement Learning. The latter was able to get an in DuckieTown simulation trained car to drive around a real-world track. Even here the author cited that a faster driven model still had troubles crossing the gap from simulation to reality.

Though it is known that there is gap to bridge when using Simulators like DuckieTown, however to what extent there exists a gap is not known. As this is hard to quantify absolutely I will look at it from a relative perspective. Comparing multiple simulations with one common technique. Object Detection is used as a standardised test to perform. It is a common computer vision problem to solve with artificial intelligence, which also suffers from the reality gap.

In this thesis I will look at the gap of several simulations often used for learning algorithms and general research purposes. By comparing the performance of different simulations, insight into how they compare to reality and to each other can be gained. This analysis will help understand how accurately simulations can replicate real-world scenarios, and which simulations are the most effective for training object detection models.

This study will look specifically at the current state of pedestrian detection using simulated data and create an abstraction to allow future research to experiment with a lower barrier of entry for trying to tackle this problem. The approach is three-pronged: a standardised abstracted environment is created, which is applicable to any simulation; multiple datasets are collected and annotated with the intent of detecting objects; finally, a comparative analysis of performance metrics is conducted using the collected data, aiming to gain valuable insights into the current state of pedestrian detection and its challenges.

CHAPTER 2

Method

2.1 Duckietown

Duckietown[1] [19], [22] is a modular robotics and AI ecosystem with many tightly integrated components designed for learning experiences. The aim of DuckieTown is to provide an easy to use standardised platform for testing autonomous driving implementations. The abstraction of the self driving problem is as follows: a small robot, called a DuckieBot, drives a round a road layout created from a standard tilemap which is known as the DuckieTown. Around the track Duckies can be placed and used for object-detection tasks. The DuckieBot, Duckietown and Duckies are constant between simulation and real-world implementations, which allows for comparisons to be drawn between multiple environments whilst reducing the complexity of the problem dramatically.

2.1.1 Hardware

Duckiebot is a minimal autonomy platform that allows to investigate complex behaviours, learn about real world challenges, and do research. It's designed to drive on a road layer, which is a tilemap of road segments that can be connected. Aside from a road layer it also has the ability to interact with a signal layer to receive information regarding traffic signs and road infrastructure. The only relevant piece of hardware for this thesis is the camera [2], which is a 5 Mega Pixel 1080p camera with a field of view of 160°. As this is quite wide, the camera can capture a large area to the left and right of itself, but does give a large distortion to objects close to the camera. Due to a problem with the DuckieBot, which was made available to me, the robot isn't able to move. However, this isn't a problem for my application as the only thing that is needed is its camera and the DuckieTown layout.

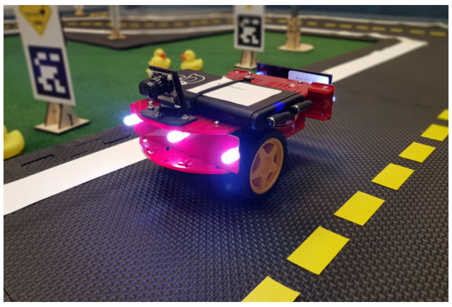
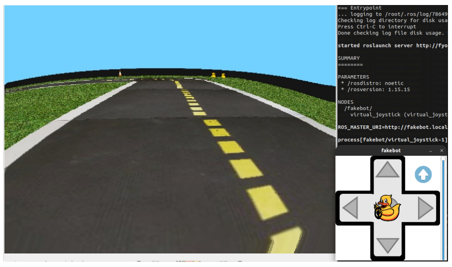


Figure 2.1: *A DuckieBot driving around a DuckieTown in the real world*

[1]`https://www.Duckietown.org/platform`

[2]`https://docs.duckietown.com/daffy/opmanual-duckiebot/preliminaries_hardware/get_hardware/index.html#camera`

2.1.2 Software



```
<== Entrypoint
... logging to /root/.ros/log/78649
Checking log directory for disk usa
Press Ctrl-C to interrupt
Done checking log file disk usage.

started roslaunch server http://fyo

SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.15

NODES
  /fakebot/
    virtual_joystick (virtual_joyst

ROS_MASTER_URI=http://fakebot.local

process[fakebot/virtual_joystick-1]
```

Figure 2.2: *The camera view of driving around a DuckieTown world in the Gym simulation*

Instead of running Duckiebot on a real track, it's also possible to do the same in simulation. Using the Duckietown Simulator [23] many of the same problems can be tested without needing to build a road. The simulation used by Duckietown is based on the Open AI gym [24], which is a Python library that standardises the communication between learning algorithms and environments. Its main use is reinforcement learning, but it has become more widely applicable. To maintain a wide application both the Gym's and the Duckietown's implementation is fairly basic, with its uniform lighting and lack of physics engine.

2.1.3 Tilemap

DuckieTown defines multiple layers, which can be seen as a level of extra detail. This can entail traffic lights, signs, street names, but most importantly the tilemap layer which defines the road network. These tiles are made up of foam with coloured tape laying out different parts of the road. There are many types of tiles that can be created, but for this research the only necessary ones are the straight and curved pieces. Each tile type has specific instructions for its creation to keep tracks reproducible, which is illustrated in figure 2.3

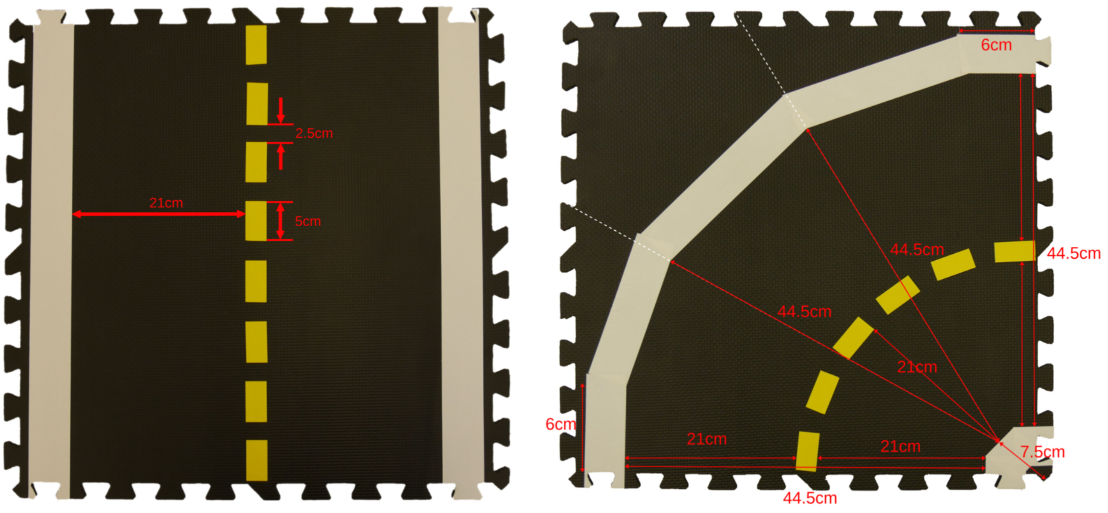


Figure 2.3: Building instructions for creating a straight and curved images, showing the exact measurements for laying out the tiles for the DuckieTown tilemap layer

[3]

2.2 Adapting DuckieTown to other Simulations

The environment used by DuckieTown is a great abstraction to use for testing across multiple Simulations. By porting the environment pieces, mainly the road tiles from the tilemap and the Duckies we can simulate the same environment wherever. These are the only things that can be somewhat simulation agnostic. The rest, which constitutes of things like random camera and

[3] https://docs.duckietown.com/daffy/opmanual-duckietown/assembly/tilemap/index.html

ducky placement and lighting configuration, is specific to each simulation and not possible to standardise.

However, with the provided scanned items and a reasonable understanding of any simulation it should be transferable to anywhere. With this portability any simulation or environment could be used as a way of recreating and adding to this research. If it allows importing models and has some kind of randomisation system, it can be used. To demonstrate further how this process should occur, these steps will be run through in detail for the Unity simulation.

2.2.1 Unity

Unity [25] is a state of the art Game Engine. It provides many features like a physics simulation, but most importantly it allows for a large amount of customisation called packages. With a package like the Unity Perception Package [26] we can extend the engine to work for us. This specific package enables us to automatically label images that we capture with automatic annotations calculated from the position of the camera and the position and dimensions of objects within the simulation.

2.3 Object Detection

Object detection [27] is collection of tasks that solves the popular problem from computer vision that aims to detect all instances of an object in an image. The result that a working object detection algorithm will produce is a 2D bounding box drawn around where it thinks an object is located. These networks are often trained on labelled data, which means that a human is required to draw each bounding box as a reference for evaluating the training and testing.

2.3.1 YOLO: You Only Look Once

YOLO [28] is a family of object recognition models that use a single neural network to predict both bounding boxes and class labels. It is known for its speed and ability to run in real time. With YOLOv8 being the latest in the family, it provides great improvement to the classic YOLOv5 object detector and allows for localising and tracking persons and objects[29]. The name comes from its differing approach to how it handles the input of images. In comparison to other implementations it doesn't go over the image multiple times, it only looks once.

The basics of the inner workings are that YOLO divides its input image into a finite grid. For each grid cell of the image it produces multiple bounding boxes with confidence scores and class predictions using a Convolutional Neural Network (CNN) backbone. It then uses Non-Maximum Suppression (NMS) to fuse highly overlapping or duplicate bounding boxes. Based on the class predictions a class is selected for each bounding box. The result is a list of bounding boxes with a corresponding confidence score. During training it uses classification and regression loss functions to optimise both the bounding boxes and the class predictions.

2.3.2 Metrics

YOLO has multiple metrics for determining the performance of the models train with it. Each metric covers a different, but important part factor in how well a model is functioning.

Average Precision

Recall

Recall calculates the proportion of correctly identified objects, true positives, to the sum of true positives and false negatives, which indicates the algorithm's ability to find all the objects of interest in the image. Higher recall values indicate that the algorithm has a lower rate of false negatives, meaning it doesn't miss any important objects in the scene. For pedestrian detection this is very important as not detecting a pedestrian could result in causing them serious harm when used in real world scenario's.

Precision

Precision measures the accuracy of the positive predictions made by the model. It indicates the proportion of correctly identified objects among all the predicted objects. A higher precision means fewer false positives, which shows the reliability of the model. This is an important metric for pedestrian detection as false positives can result in incorrect behaviour in autonomous vehicles.

2.4 Prepare simulations

object and To have our simulations be as consistent as possible, we use the same environment and object for our training and testing purposes. The environment is taken from the Duckietown world and simulated in each simulation, that will be tested with. As the DuckieTown Gym based simulation already has everything built in, this process will only be necessary for the Unity Engine. However, as this same process could later be applied to different simulations a general description will also be given. When replicating this part of the research, it is important to note that the official rules for DuckieTown building was used. This consists of exact measures for lines and rules about track layout requirements. For the exact specifications of how I got these items from physical to digital can be found in section 3.2.

The main part of the setup consists of four parts: setting up the track; setting up the randomised Duckies; setting up the camera randomisation; and lastly, configuring the camera settings. For the road, only two pieces are needed. With a straight and a curved road, all non-complex layouts can be recreated. The object that will be used for detecting will be a rubber ducky. The usage of Duckies for Pedestrian Detection is also taken from the DuckieTown world as an abstraction of pedestrians and/or other road user. These components will be scanned-in as an .obj file and placed in each respective simulation.

2.4.1 Camera specifications

As described in section 2.1.1, the DuckieBot has a camera with very unique specifications. To match the real world as closely as possible, these settings should be used when configuring the robot within simulation. The exact settings that should be used are as follows. The resolution of the images produced by the camera are 640x480 pixels. The camera is a wide angled lens, with the exact angle being 160°.

2.5 Gathering the dataset

The question of this thesis surrounds the problem of porting a simulation trained artificial intelligence to the real world. The training data will be the data from the simulation and the testing data will be from the real world. In the real world the process of data gathering will be very manual as the platform itself doesn't provide a way to capture a dataset or images for a purpose like this. Each simulation will also have a tailor made method for capturing images, which will be much easier to automate.

2.5.1 Dataset specifications

For each simulation data gathering will be performed twice. One with the duckies included and randomised and once without any duckies. The same amount of data will be gathered for with and without duckies to ensure about a 50/50 split of ducky images and null images. This ensures that each model will also be able to correctly handle the lack of duckies, which is beneficial to not detect other yellowish objects that are present in the scene like the centre-line and background objects.

2.5.2 Simulation data

In each simulation the world layout will be created using the tilemap with a semi-random placement of Duckies near the track. The camera will than either move past the Duckies to capture multiple angles and distances to the camera or random ducky placements and road layouts will be created to capture multiple locations. So its either random driving or placement for the camera and always ducky placement as random as possible. The placement of duckies does not have to random for each image if a moving robot is used as the change in position of the camera creates enough random perspectives if the capturing interval isn't too low.

2.5.3 Real-world data

Very similar to the data gathering in the simulation, the real-world data is captured on a couple tilemap layouts with Duckies placed randomly on the surface of the tiles. From the camera, images will be taken at an interval whilst the DuckieBot is pushed at a constant speed to capture multiple angles and distances of Duckies near the track and camera. To create variability in the data multiple lighting environments will be used whilst recording, which can be controlled through blinds and/or putting the track in the shadow or direct sunlight.

Many things like, the rotation and position relative to windows determines a lot about how the camera and its internal sensor will react. The dataset that will be gathered for the real world data should be as broad as possible to capture many different scenario's as to emulate how real world pedestrian detection should also be widely applicable. If not enough variety is present in the dataset, the performance of other models on the data won't be determined by how well they adapt, but how well they happen to match the same exact circumstances.

2.6 Label all the data

After all images have been gathered the data needs to be labelled. In the Unity Game Engine this can be automated, but in the DuckieTown Gym this isn't possible as with real world data. Using Roboflow, an online platform providing a wide range of tools for machine learning related problems, each image can be annotated with a class. Because the dataset only contains Duckies, there is only one class necessary.

2.7 Train the object-detection

YOLO is used to train a model on each datatset. Each dataset is split into three sections, with each their own purpose. The training part will be used as data to improve the model. In batches the model is shown a set of images to improve slightly with. The results is then checked after each epoch against the validation dataset. Here some hyper-parameters are tweaked to allow for the next training steps to be as successful as possible. After all the data has been processed the data is then checked against the testing part of the dataset, which it has never seen before. The result on the test set is therefore as unbiased as it could be and illustrates the performance the model will likely have on other data it has never seen before.

The parameters of each training session are slightly varied for each dataset to see if any change will make a very significant change in its performance. After multiple models have been trained and analysed a set of arguments is chosen and will stay consistent throughout each test. These tests are not essential for the main research question and will therefore be placed as an addition to the thesis in section 7.2

2.8 Evaluate the object-detection on the real-world data

When each dataset has a trained model with the same parameters for training, the models can be used for testing. Firstly, all models will be evaluated on their own testing data, which will illustrate how closely fitted it is to its own data. Each model will then also be run on the same part of real world data, the testing split. From this we will get the performance of each model on their own and the real worlds test data. From this we will be able to quantify the reality gap through multiple performance metrics.

2.8.1 Compare data through Metrics

We take the following criteria that are popular with pedestrian detection as measures for their performance: Averaged Precision (AP), Precision and Recall. More information about their inner workings is explained in section 2.3.2.

CHAPTER 3

Experiments

For my experiments to be possible, multiple datasets were needed to be created. To allow each simulation to be used for similar data gathering some preparations were needed to be made. Each simulation has its own special features and constraints, which need to be taken into account when adapting environments to the same purpose. With that purpose being placing randomised duckies within a realistic DuckieTown track, whilst taking images from a predefined camera angle and configuration.

When this setup is completed, the actual images can be captured or generated depending on the operating domain. For both the real world data and the DuckieTown Gym data, the images are then annotated to create a ground truth for models to be trained and tested against. The Unity simulation has the ability to automate this process by generating them through its exact knowledge about objects location and orientation.

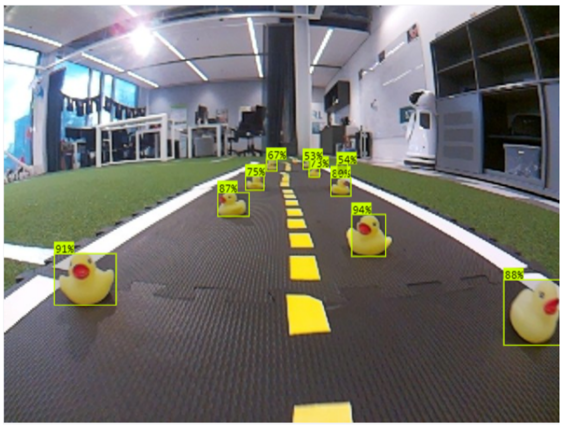


Figure 3.1: *A real-world image with a predicted location and confidence level for each ducky.*

Each simulation now has its own or multiple datatsets captured with it. The datasets are then split into three sections: training, validation and testing. The separation of data is needed to see how the training of the model performs during and after training. As the term suggests, the training data is used for the improvement of the YOLO model through seeing new data. During each iteration the model is tweaked slightly with the validation data. Only after the entire learning procedure is complete, the model is tested on the testing data, which it has never seen before. This performance of the model will be the metric for how it would react to similar data it has never seen before.

To more easily distribute and manage the datasets the Roboflow[30] platform is used to upload the images, including the corresponding annotations if applicable. The platform then allows the images without annotations to be annotated. At first this was done by hand using the rectangle tool. After about 100 images were hand annotated, a model was trained on the platform with this data to use as an intermediate model. The model could then be used as label assist to speed up the process.

The question of how models that have been trained in simulation transfer to the real world is answered by using each simulation model on the test part of the real world data and see how they compare.

3.1 Real World Setup

Using a corner and straight road from the DuckieTown set tiles two tracks are created, a small circular track and a larger, more complex track. On the track 25 duckies are placed at random, which was achieved by throwing the duckies on the track and letting them bounce away. The DuckieBot is placed on the track and moved along the track whilst capturing an image from its camera every 2 seconds. This process is then repeated in other lighting conditions. The three types of lighting conditions that were used are as follows:

- Soft Natural Light: With no harsh shadows or direct sunlight
- Soft Artificial Light: With no harsh shadows, but with direct artificial light
- Harsh Natural Light: With harsh shadows and direct natural light

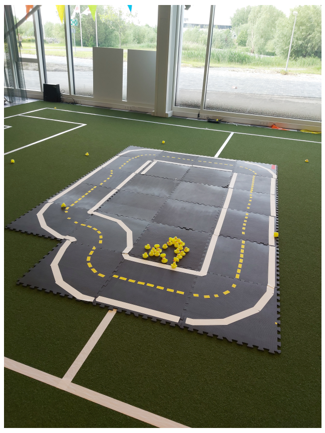
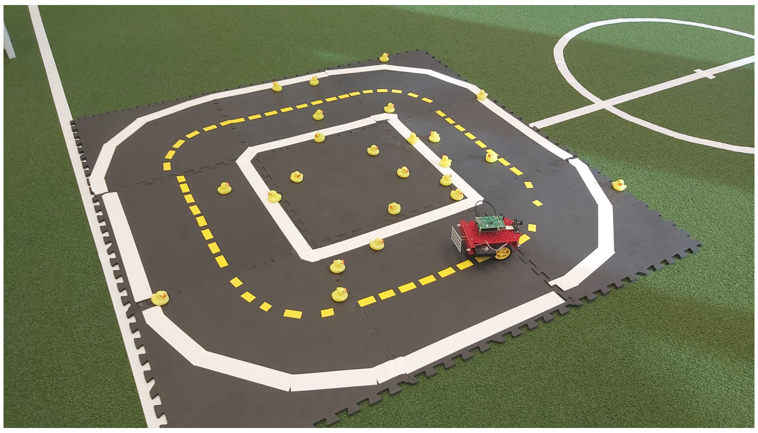



Figure 3.2: Multiple angles of the setup in the real world for capturing its dataset. The images show a DuckieTown floor laying on a green underground

3.1.1 Annotation

Each image was annotated using the Roboflow platform [30]. After 100 images were labelled an intermediate model was trained that was able to assist with annotation through this platform.

3.2 Unity Simulation Setup

In Unity and environment is set up with a model closely representing the actual location where the real world dataset is recorded. From the DuckieTown platform multiple elements are scanned into the simulation. Using a picture of two types of tiles a model is closely traced. With only a straight and corner piece many different layouts can be created. The layout in use throughout the dataset, pictured in 3.3, is kept the same as when capturing the real world dataset.

As the object to detect the rubber ducky is used. Due to some troubles with scanning in the ducky with the 3D scanner available, an exact replica of the type of ducky used in DuckieTown was sourced online[1]. Due to slight imperfections in the scan, like multiple shadows on the texture and its material simplicity, some changes were made to the texture to improve its similarity to its real world counterpart. This included, but was not limited to increased brightness, exposure and a slight emissive texture to emanate the almost glowing appearance of the real duckies. The version of the ducky that was eventually used is shown in 3.4 and can be found with the other created files.

[1]https://sketchfab.com/3d-models/rubber-duck-3d-scan-ead0a5b7b0ba4a6299a60fd77d85f99a

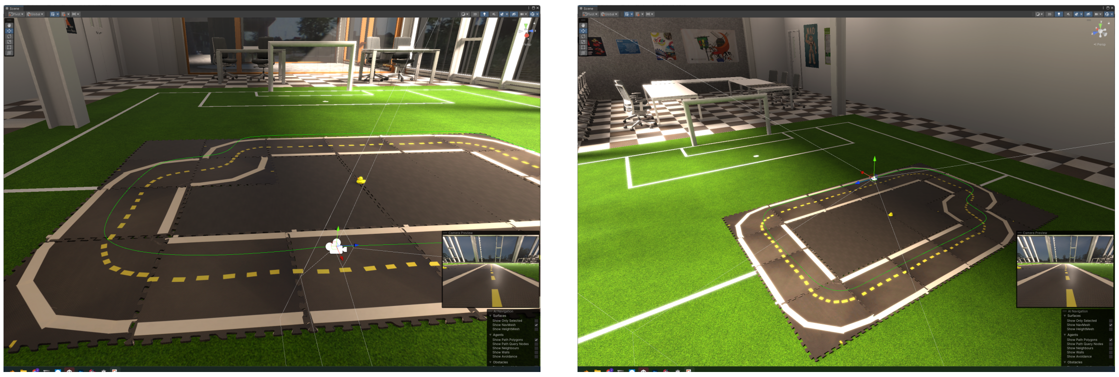


Figure 3.3: Multiple angles of the setup within the Unity Game Engine for capturing its dataset. The images show a DuckieTown floor laying in a room, which is a very close approximation of the location where the real world data was gathered.

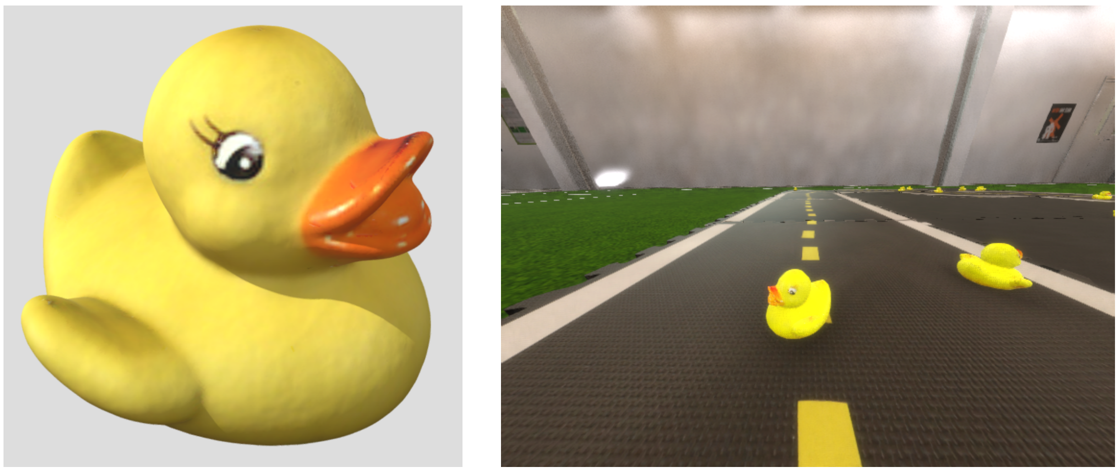


Figure 3.4: On the left an image of the model used for the ducky is shown. On the right this same model is shown in context after slight texture changes were applied.

The models and textures created for this thesis can be found on Github[2]

On the area of the tiles 25 duckies are placed randomly for every iteration. Their placement consists of a random position within the bounds of the tiles. This does not take into account the collision between duckies, meaning there is a small chance that two of them can collide with each other causing some visual glitching. But due to the large area of the track the change of this happening is negligible and won't have any real effect on any model trained on it.

Now that the track is set up, the camera is placed semi-randomly around the drivable area of the track. This is accomplished by drawing a spline following the center of the road. The camera's position is based on the addition of a random point on this spline and some noise allowing the car to also be located to either side of the track.

Within the Unity Simulation it is possible to also randomise the lighting conditions, however for the main dataset this is kept fairly constant to keep the difference between simulations mainly focused on the level of fidelity, not their ability to perform domain randomisation.

The results of this section including the models, the Unity Environment and the scripts used for capturing the data and labels can be found on Github[3]

[2] `https://github.com/IntelligentRoboticsLab/Robolab-Unity-Env/tree/main/Assets/Models/Custom/DuckieTown`
[3] `https://github.com/IntelligentRoboticsLab/Robolab-Unity-Env`

3.2.1 Extension: Unity Domain Randomisation

As an extension to my experiments two extra datasets were created with more varying settings for the lighting conditions within the simulation. The main Unity dataset is very static in its usage of these features, but because of the ease of using the simulation these two extra datasets were created to investigate the effect that these features can have. One of these uses these settings to create a range of conditions that are within reason of conditions that could happen in the real world. The other is with the settings cranked up the extreme, which creates far more unrealistic conditions. Though this might make it harder for the model to fit to its own data, their is a possibility for it to become more generalisable.

3.2.2 Annotation

In contrast to the real world, the camera in the simulation knows the exact position and outline of each object in the scene. Using the Unity Perception Package we are able to generate images with annotations already created. This alleviates the need for hand or assisted annotating, which is one of the largest tasks for generating a dataset. The direct output from the simulation is annotations in the SOLO file format. This is not the format that will eventually be used to train the models to perform object detection. So, a script that converts SOLO to YOLO annotations is used[4]. Aside from some small problems with this script, the dataset is now complete, annotated and ready for training.

A portion of the capturing of the dataset in Unity was recorded, which gives a great deal of context to the explanation above.[5]

[4]https://github.com/Unity-Technologies/pysolotools
[5]https://youtu.be/vVYbwfosOi8

3.3 Gym DuckieTown Simulation Setup

In contrast to Unity, the Gym DuckieTown simulation doesn't need a lot of setup. Using a Reinforcement learning trained driving agent trained [21], the DuckieBot is able to follow any track with a good speed. The simulation loops over a list of maps picked that are somewhat similar to the environment of the other setups. The list of maps is as follows: "zigzag_dists", "straight_road", "udem1", "small_loop", "loop_empty". After a certain amount of iterations (if the robot doesn't go out of bounds) the next map is loaded in. Before loading each map 25 duckies are placed randomly over the entire track. This resets after each new map load. At an interval of 2 seconds an image is taken. This makes sure that two images taken after each other are not too similar.

The code used for collecting the dataset within the DuckieTown Gym Environment can be viewed online. [6]*. It's forked from previous work that created the RL driving agent in DuckieTown [21].*

3.3.1 Annotation

Because of the limitations of the Gym based simulation it wasn't possible to create a program to automatically annotate the images within the time frame of this thesis. This meant that each image had to be hand annotated, with the later help of an intermediate model.

3.4 Datasets

My Datasets	# images	# duckies	# duckies / img
Real world [31]	2001	6811	3.4
Gym DuckieTown [32]	2082	7629	3.7
Unity DuckieTown [33]	2000	8421	4.2

Other Datasets	# images	# duckies	# duckies / img
Duckietown Object Detection [34]	200	552	2.8
Duckietown Dataset [35]	1432	2,546	1.8
Duckiebotvision [36]	1446	2,590	1.8

Table 3.1: *Ducky density* of each dataset. Excluding the part of the dataset specifically meant to contain no objects.

3.4.1 Other Unity Datasets

Unity Datasets	# images	# duckies	# duckies / img
Static [33]	2000	8421	4.2
Dynamic [37]	2000	8137	4.1
Extreme [38]	2000	8211	4.1

Table 3.2: *Ducky density* of the alternative Unity datasets

[6]`https://github.com/Fyor/gym-duckietown-data`

3.5 Training

Each dataset is used to train a singular model. The training parameters will stay consistent for each dataset and have been chosen by playing around with the settings throughout the project and finding a good balance. The most significant parameters that have been analysed are the batch size, the number of training epochs, the optimiser used and the learning rate. As YOLOv8 is a very stable and advanced system, many things worked fine or the best with the default settings or different parameters didn't provide any significant improvement. For some alternative training results with other parameters, take a look at section 7.2.

For training the YOLOv8 models a Jupyter Notebook[7] is used with a template provided by the creators of Roboflow[30], which imports datasets directly from their website. A link to the resulting models can be found in the following section 4.

[7]`https://github.com/roboflow/notebooks/blob/main/notebooks/train-yolov8-object-detection-on-custom-dataset.ipynb`

CHAPTER 4

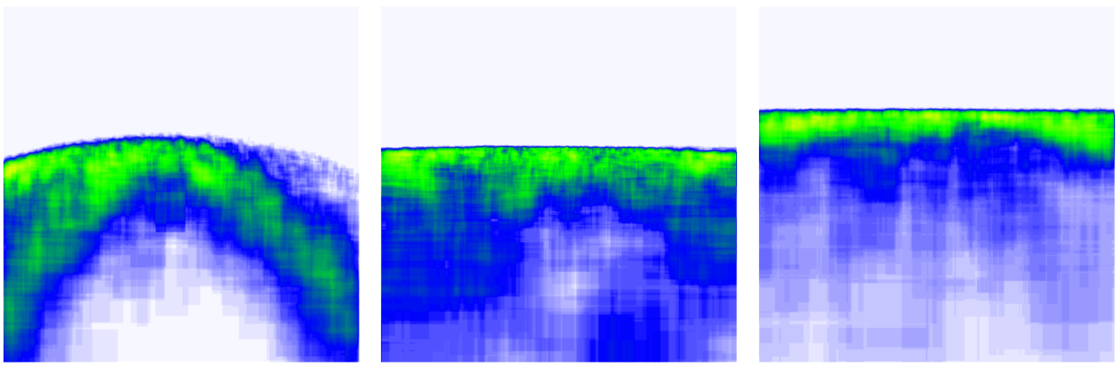
Results

As described in section 3.5 multiple experiments have been carried out with their results being described in the following chapter. The three different environments, real world, Unity and the Gym have been tested on both the validation(valid) and testing(test) part of the real world dataset. With the aim of this experiment being to analyse the difference between the domains without any augmentations. The second test compares multiple datasets captured in the same environment, but with each having varying settings. With the aim of this last experiment to see how well augmentations can help with getting the performance closer to the benchmark of the real world trained model.

The main trained models can be found online on Github[1]

4.1 Datasets

Through the Roboflow platform some analysis can be performed to show and illustrate characteristics of the data, which could lead to biases in the model results.



(a) Real-world (b) Unity (c) DuckieTown Gym

Figure 4.1: A heatmap showing the distribution of duckies throughout the camera view for each environment. Here blue indicates a low number of duckies and lime indicates a high numer of duckies.

[1]`https://github.com/Fyor/DuckieTown-Ducky-Models`

4.2 Training

In this section the results of the training procedure will be shown. Though this is not relevant to the actual conclusions that can be drawn from each of the results, it's still important to show how each model performed during its training. With this can be shown that each model did train correctly and was able to fit to its own data well enough.

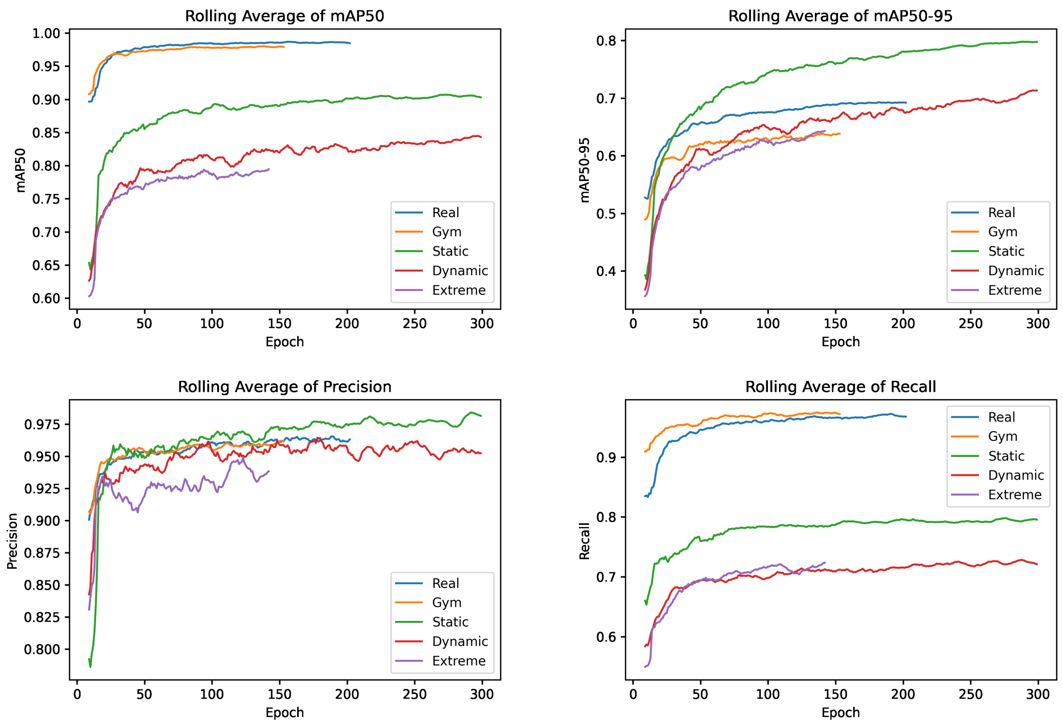


Figure 4.2: Four plots showing metrics of the model throughout the training procedure. Each model fits slightly different to its data, due to differing distributions of characteristics between datasets. As the training furthers each metric increases eventually resulting in a better model.

4.3 Testing

For each model the following metrics will be tracked. These metrics will be used to evaluate how each model trained on its own training data. Each model will then also be evaluated on a separate part of the real-world dataset which no model has seen during training to get a fair comparison between their actual real world performance.

- mAP
- Precision
- Recall

4.3.1 Training result

Dataset	m	mAP	Precision	Recall
Real world	66.0	96.8	92.7	93.8
Gym DuckieTown	69.3	99.2	96.6	97.7
Unity	78.6	90.3	98.9	77.4
Dynamic Unity	75.6	88.9	97.1	77.1
Extreme Unity	68.2	85.3	95.6	73.1

Table 4.1: A comparison of object detection metrics between datasets on tested on the test data of each dataset. For the metrics mAP, Precision and Recall a higher value is better.

4.3.2 On test data

Model training data	m	mAP	Precision	Recall
Real world	66.0	96.8	92.7	93.8
Unity DuckieTown	58.8	92.6	90.6	85.9
Gym DuckieTown	37.1	62.8	73.0	53.8

Table 4.2: A comparison of object detection metrics between datasets on tested on the unseen test data from real world dataset. For the metrics mAP, Precision and Recall a higher value is better.

4.3.3 Unity Environment Comparison

The extension adds two more datasets which were generated with Unity, but with increasingly more complex environment settings. The Unity DuckieTown results from previous tests are now used as the "Static Unity" variant, which the two other models are compared against.

Model training data	m	mAP	Precision	Recall
Static Unity	58.8	92.6	90.6	85.9
Dynamic Unity	59.3	93.5	92.0	87.0
Extreme Unity	58.2	93.7	90.6	89.7

Table 4.3: A comparison of relevant pedestrian metrics between Unity trained models, tested on real world data. For the metrics mAP, Precision and Recall a higher value is better.

CHAPTER 5

Conclusion

As a result from this thesis there are now numerous datasets created, which each on their own exceed any other similar dataset before it. The datasets are all meticulously annotated and contain a wide variety of conditions to encompass the domain it was created from. Using these datasets a model is created to detect abstracted pedestrians, in the DuckieTown world. Aside from some analysis on its own corresponding test data, each model is assessed on the unseen test data from the real world as a way of analysing their simulation-to-reality transfer ability. By using the real-world model as a baseline, it becomes evident that the visual fidelity significantly impacts the portability of object detection models. The Unity simulation closely approaches the baseline performance, while the DuckieTown Gym noticeably lags behind. From the second experiment we can see the effect that domain randomisation on data can have to models trained with it. A slight but significant improvement is noticed when a reasonable amount of randomisation is applied, which is plausible for the domain. However, pushing this too far beyond the scope of the domain offers no clear improvement to its baseline of a static environment.

CHAPTER 6

Discussion

One of the metrics that I felt was very relevant to pedestrian detection and this thesis was the Miss Rate, written as MR^{-2}, which stands for *log-average Miss Rate on False Positive Per Image (FPPI).* This metric is often used in pedestrian detection as the MR^{-2} punishes false positives more [5]. A smaller MR^{-2} indicates better performance. Due to time constraints of the project I was sadly not able to include this metric in my evaluation. The implementation is not a standard for object detection models in general, which is why wide support is unavailable out of the box. A custom implementation would have been necessary. However, its insights are invaluable for pedestrian and critical object detection purposes.

DuckieTown provides two mediums for testing autonomous driving or related tasks: A simulation and a physical robot. At the start of the project the physical robot was not in a working state and the simulation was described as difficult to work with. In the end, it was not possible to get the physical robot working even after reaching out to the helpful people from the DuckieTown slack community. The simulation did work, but required a large time investment to understand and get working. Each method of working with the simulation provided its own problems, which includes python pip dependencies being deprecated, disappearing and throwing errors before even starting modifying the code. Eventually, I was able to get it working on one particular workstation after many modifications, but performing the same steps on a different workstation did not turn out as fruitful.

The utilisation of domain randomisation appears very much underused in this thesis, which was not intended at the start of the project. Where domain randomisation really shines is in areas where visual fidelity fails. Applying various techniques to the DuckieTown Gym environment would have been an interesting area to research, because of its large simulation-to-reality gap to be crossed. However, the large task of annotating for this simulation took quite some time as there exists no way of automating this procedure. Theoretically this could be possible as the simulation does keep track of the camera location and all objects in the scene. Using these with some information about the dimensions of the object you're trying to visualise, a very close approximation of the location of objects in the scene could be automatically generated. If a tool like this would exist, the amount of experimentation on this platform could increase dramatically. Due to uncovering this problem late in the project together with the difficulty of getting the DuckieTown Gym simulation to a working state, creating this solution was not within the scope of this project. However, future research could easily build further on this thesis if such a tool were to be created.

When analysing the data from each dataset in detail for the Results section it appeared that something went wrong with configuring some of the camera settings. When looking at the heatmaps of each dataset a clear lack of distortion seemed to be missing. The real world dataset clearly shows the horizon as being curved, which is logical with the camera being so wide angled. However, when looking at both simulations a clear curve in the data was missing. For the Unity simulation this could be explained by the camera settings not being saved in the online repository when porting the Unity environment to a different workstation. In the DuckieTown

Gym it appears that this settings, which is available was also missed. Due to them both being equally wrong in this aspect the comparison should still be unbiased by this. However, a slight increase in accuracy could be expected if these settings were both set correctly. When looking at the height of the horizon the Unity simulation does seem to have the correct orientation as it reaches the same height as the real world data. The provided DuckieTown simulation, however, shows a clear error with the horizon line being noticeably higher than the two other heatmaps. This could have an impact on the results as the distribution of data differs, possibly making the predictions more or less accurate at different positions relative to the camera. The degree of this error is not known and hard to predict, but future research might need to take this into account when transferring simulation trained models to reality.

Bibliography

[1] M. Dikmen and C. Burns, “Autonomous driving in the real world: Experiences with tesla autopilot and summon,” Oct. 2016, pp. 225–228. DOI: 10.1145/3003715.3005465.

[2] C. Ryan, F. Murphy, and M. Mullins, “End-to-end autonomous driving risk analysis: A behavioural anomaly detection approach,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1650–1662, 2021. DOI: 10.1109/TITS.2020.2975043.

[3] C. Chi, S. Zhang, J. Xing, Z. Lei, S. Li, and X. Zou, “Pedhunter: Occlusion robust pedestrian detector in crowded scenes,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 10 639–10 646, Apr. 2020. DOI: 10.1609/aaai.v34i07.6690.

[4] I. Hasan, S. Liao, J. Li, S. Ullah Akram, and L. Shao, “Generalizable pedestrian detection: The elephant in the room,” Jun. 2021, pp. 11 323–11 332. DOI: 10.1109/CVPR46437.2021.01117.

[5] P. Dollar, C. Wojek, B. Schiele, and P. Perona, “Pedestrian detection: An evaluation of the state of the art,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 743–761, 2012. DOI: 10.1109/TPAMI.2011.155.

[6] X. Chu, A. Zheng, X. Zhang, and J. Sun, “Detection in crowded scenes: One proposal, multiple predictions,” in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 12 211–12 220. DOI: 10.1109/CVPR42600.2020.01223.

[7] S. Shao, Z. Zhao, B. Li, *et al.*, “Crowdhuman: A benchmark for detecting human in a crowd,” *ArXiv*, vol. abs/1805.00123, 2018. DOI: 10.48550/arXiv.1805.00123.

[8] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, “Soft-nms — improving object detection with one line of code,” *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5562–5570, 2017. DOI: 10.48550/arXiv.1704.04503.

[9] S. Rujikietgumjorn and R. T. Collins, “Optimized pedestrian detection for multiple and occluded people,” in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 3690–3697. DOI: 10.1109/CVPR.2013.473.

[10] R. Benenson, M. Omran, J. Hosang, and B. Schiele, “Ten years of pedestrian detection, what have we learned?,” vol. 8926, Nov. 2014, ISBN: 978-3-319-16180-8. DOI: 10.1007/978-3-319-16181-5_47.

[11] S. Zhang, R. Benenson, M. Omran, J. Hosang, and B. Schiele, “How far are we from solving pedestrian detection?,” Jun. 2016, pp. 1259–1267. DOI: 10.1109/CVPR.2016.141.

[12] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 23–30. DOI: 10.1109/IROS.2017.8202133.

[13] Y. Chebotar, A. Handa, V. Makoviychuk, *et al.*, “Closing the sim-to-real loop: Adapting simulation randomization with real world experience,” May 2019, pp. 8973–8979. DOI: 10.1109/ICRA.2019.8793789.

[14] F. Sadeghi and S. Levine, “Cad2rl: Real single-image flight without a single real image,” Jul. 2017. DOI: 10.15607/RSS.2017.XIII.034.

[15] H. G. Lekanne gezegd Deprez, BSc thesis, University of Amsterdam, Jul. 2020. [Online]. Available: https://staff.fnwi.uva.nl/a.visser/education/bachelorAI/thesis_hidde_lekanne_deprez.pdf.

[16] S. R. Richter, H. A. AlHaija, and V. Koltun, "Enhancing photorealism enhancement," *arXiv:2105.04619*, 2021.

[17] M. Fabbri, G. Brasó, G. Maugeri, *et al.*, "Motsynth: How can synthetic data help pedestrian detection and tracking?" In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 10 849–10 859. DOI: https://doi.org/10.48550/arXiv.2108.09518.

[18] B. Balaji, S. Mallya, S. Genc, *et al.*, "Deepracer: Autonomous racing platform for experimentation with sim2real reinforcement learning," May 2020, pp. 2746–2754. DOI: 10.1109/ICRA40945.2020.9197465.

[19] L. Paull, J. Tani, H. Ahn, *et al.*, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1497–1504. DOI: 10.1109/ICRA.2017.7989179.

[20] V. C. Pablo Miralles, *Study and demonstration of domain adaptation techniques applied to reinforcement and supervised learning algorithms.* https://upcommons.upc.edu/handle/2117/333264, Nov. 2020.

[21] T. Wiggers and A. Visser, "Learning to drive fast on a duckietown highway," in *Annual Meeting of the IEEE Industry Applications Society*, 2021. DOI: 10.1007/978-3-030-95892-3_14.

[22] J. Tani, L. Paull, M. T. Zuber, *et al.*, "Duckietown: An innovative way to teach autonomy," in *Educational Robotics*, 2016. DOI: 10.1007/978-3-319-55553-9_8.

[23] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta, and L. Paull, *Duckietown environments for openai gym*, https://github.com/duckietown/gym-duckietown, 2018. DOI: 10.48550/arXiv.1606.01540.

[24] G. Brockman, V. Cheung, L. Pettersson, *et al.*, *Openai gym*, 2016. eprint: arXiv:1606.01540.

[25] Unity Technologies, *Unity*, https://unity3d.com, 2017.

[26] Unity Technologies, *Unity Perception package*, https://github.com/Unity-Technologies/com.unity.perception, 2020.

[27] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023. DOI: 10.1109/JPROC.2023.3238524.

[28] G. Jocher, A. Chaurasia, and J. Qiu, *YOLO by Ultralytics*, version 8.0.0, Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.

[29] J. Terven and D. Cordova-Esparza, "A comprehensive review of yolo: From yolov1 to yolov8 and beyond," *arXiv preprint arXiv:2304.00501*, 2023. DOI: 10.48550/arXiv.2304.00501.

[30] B. Dwyer, J. Nelson, and J. Solawetz, *Roboflow*, version 1.0, 2022. [Online]. Available: https://roboflow.com.

[31] Duckies, *Real world duckietown duckies dataset*, https://universe.roboflow.com/duckies-1cfdk/real-world-duckietown-duckies, Open Source Dataset, visited on 2023-06-22, Jun. 2023. [Online]. Available: https://universe.roboflow.com/duckies-1cfdk/real-world-duckietown-duckies.

[32] Duckies, *Duckietown gym simulation duckies dataset*, https://universe.roboflow.com/duckies-1cfdk/duckietown-gym-simulation-duckies, Open Source Dataset, visited on 2023-06-22, Jun. 2023. [Online]. Available: https://universe.roboflow.com/duckies-1cfdk/duckietown-gym-simulation-duckies.

[33] D. in Simulations, *Unity duckies static lighting dataset*, https://universe.roboflow.com/duckies-in-simulations/unity-duckies-static-lighting, Open Source Dataset, visited on 2023-06-21, Jun. 2023. [Online]. Available: https://universe.roboflow.com/duckies-in-simulations/unity-duckies-static-lighting.

[34] Duckietown, *Duckietown object detection dataset*, `https://universe.roboflow.com/duckietown-phrqi/duckietown-object-detection`, Open Source Dataset, visited on 2023-06-06, Jun. 2022. [Online]. Available: `%5Curl%7Bhttps://universe.roboflow.com/duckietown-phrqi/duckietown-object-detection%7D`.

[35] CVduckies, *Duckietown dataset*, `https://universe.roboflow.com/cvduckies-ahtmc/duckietown_dataset`, Open Source Dataset, visited on 2023-06-06, Jun. 2022. [Online]. Available: `%5Curl%7Bhttps://universe.roboflow.com/cvduckies-ahtmc/duckietown_dataset%20%7D`.

[36] duckiebotvision, *Duckiebotvision dataset*, `https://universe.roboflow.com/duckiebotvision/duckiebotvision`, Open Source Dataset, visited on 2023-06-06, Dec. 2022. [Online]. Available: `%5Curl%7Bhttps://universe.roboflow.com/duckiebotvision/duckiebotvision%20%7D`.

[37] D. in Simulations, *Unity duckies dynamic lighting dataset*, `https://universe.roboflow.com/duckies-in-simulations/unity-duckies-dynamic-lighting`, Open Source Dataset, visited on 2023-06-22, Jun. 2023. [Online]. Available: `https://universe.roboflow.com/duckies-in-simulations/unity-duckies-dynamic-lighting`.

[38] D. in Simulations, *Unity duckies extreme dynamic lighting dataset*, `https://universe.roboflow.com/duckies-in-simulations/unity-duckies-extreme-dynamic-lighting`, Open Source Dataset, visited on 2023-06-22, Jun. 2023. [Online]. Available: `https://universe.roboflow.com/duckies-in-simulations/unity-duckies-extreme-dynamic-lighting`.

CHAPTER 7

Appendices

7.1 Appendix A: Model Training Parameters

```
task: detect               save_json: false           workspace: 4
mode: train                save_hybrid: false         nms: false
model: yolov8l.yaml        conf: 0.001                lr0: 0.01
data: ‘data directory‘     iou: 0.7                   lrf: 0.01
epochs: 300                max_det: 300               momentum: 0.937
patience: 50               half: false                weight_decay: 0.001
batch: 32                  dnn: false                 warmup_epochs: 3.0
imgsz: 640                 plots: true                warmup_momentum: 0.8
save: true                 source: ultralytics/assets/warmup_bias_lr: 0.1
cache: false               show: false                box: 7.5
device: ’’                 save_txt: false            cls: 0.5
workers: 8                 save_conf: false           dfl: 1.5
project: null              save_crop: false           fl_gamma: 0.0
name: null                 hide_labels: false         label_smoothing: 0.0
exist_ok: false            hide_conf: false           nbs: 64
pretrained: false          vid_stride: 1              hsv_h: 0.015
optimizer: SGD             line_thickness: 3          hsv_s: 0.7
verbose: true              visualize: false           hsv_v: 0.4
seed: 0                    augment: false             degrees: 0.0
deterministic: true        agnostic_nms: false        translate: 0.1
single_cls: false          classes: null              scale: 0.9
image_weights: false       retina_masks: false        shear: 0.0
rect: false                boxes: true                perspective: 0.0
cos_lr: false              format: torchscript        flipud: 0.0
close_mosaic: 10           keras: false               fliplr: 0.5
resume: false              optimize: false            mosaic: 1.0
overlap_mask: true         int8: false                mixup: 0.15
mask_ratio: 4              dynamic: false             copy_paste: 0.3
dropout: false             simplify: false            cfg: null
val: true                  opset: 17                  v5loader: false
```

7.2 Appendix B: Model Training Alternatives

TODO: *Show graphs of how the changed parameters like batch size and optimizer didn't have a real impact on the resulting metrics.*

7.3 Appendix C: DuckieTown Troubles

TODO: *Describe what went wrong with Duckietown and what the original plan was for the thesis.*