Universiteit Leiden
The Netherlands

Bachelor Computer Science

Differential Siamese Network for the Avoidance of Moving Obstacles

Jerry Schonenberg

Supervisors:
Dr. Erwin M. Bakker & Dr. Michael S. Lew

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

26/08/2020

Abstract

Ever since the public roads are getting increasingly crowded, a demand arises for autonomously driven vehicles. The ability to avoid obstacles is a basic necessity for it to become a viable option for the consumer. It is one of the reasons that obstacle avoidance is a very active research topic in the field of computer vision. Our research primarily focuses on the avoidance of moving obstacles. Existing approaches to this problem often make use of multiple sensors (e.g. LIDAR and RADAR) in addition to one or more cameras. Here, we propose a Differential Siamese Network (DSN) which utilises a monocular RGB camera as its sole input sensor. Since the focus lies on the avoidance of moving obstacles, work from other researchers serve as a baseline for our research. The work from Khan and Parker [KP19] is selected due to its performance and robustness, and provides a convolutional neural network architecture to manoeuvre in an environment with static obstacles.

The DSN is assessed in an environment with both static and moving obstacles to determine whether there is a significant difference in trajectories between the DSN and baseline. Virtual environments, constructed with CoppeliaSim, are utilised to ensure a fair comparison.

From the experiments came forth that the DSN's ability to avoid static obstacles is not inferior to the baseline. The distance to obstacles is measured during every run and stated that they both keep a similar distance to all obstacles. Moreover, the DSN passes moving obstacles more efficient than the baseline. The DSN is able to recognise the movement of an obstacle and readjust his trajectory accordingly. So would the DSN turn left when approaching an obstacle moving from left to right, while the baseline would turn right and hereafter encounters the obstacle again.

Contents

1 Introduction

With an ever increasing amount of vehicles on the public roads, the call for autonomously driven vehicles has never been greater than before. Numerous car manufacturers have a program in place to conduct research on the matter and subsequently putting it into practice. The vehicles are equipped with multiple sensors to be able to safely drive on the public roads. For instance, RADAR, LIDAR and vision sensors are utilised to make a vehicle drive and navigate autonomously.

A vital component of self-driving vehicles is the ability to avoid obstacles. This is one of the use-cases of the sensors; the ability to detect surrounding objects. On the basis of the observations, evasive actions can be taken to avoid potentially unsafe scenarios.

However, most sensors come with one or more drawbacks. For instance, RADAR operates within a relatively small range of circa 60 meters and encounters interference from other objects which may deteriorate the accuracy. LIDAR is sensitive to lighting conditions and is ineffective in various weather conditions (e.g. fog and heavy rain). As for vision sensors, they do not provide any information with regards to depth; it maps a three-dimensional space to a two-dimensional plane. On the other hand, they provide rich information about the environment, are cheap and power-efficient. Consequently, only utilising vision sensors to make a vehicle drive autonomously is beneficial to its production costs and in particular to its energy efficiency.

Recently, several researches have been conducted on the obstacle avoidance problem where only vision sensors are utilised. For instance, Khan and Parker [KP19], next to Xie et al. [XWMT17], made use of a monocular RGB camera to safely manoeuvre in an environment with exclusively static obstacles.

We propose a solution to the problem, but in particular to the avoidance of moving obstacles. The solution makes use of a monocular RGB camera with which a robotic vehicle is able to autonomously manoeuvre in an environment with moving obstacles. Convolutional neural networks (CNN) are utilised to determine the trajectory on the basis of the RGB images. Since the focus lies on moving obstacles, work from other researchers has been taken as a baseline on top of which our solution is build. The work from Khan and Parker is selected due to its robustness in ever changing environments and overall performance. It was able to safely traverse in an environment with never-seen-before objects. Khan and Parker's solution consists of a CNN which maps a command to an image. Further details on the method are given in Section 3.

This baseline is then extended with the ability to safely manoeuvre around obstacles with uniform movements. This extension, the Differential Siamese Network (DSN), is our main contribution to the obstacle avoidance problem.

In the end, a comparison is made between the DSN and baseline. Both find themselves in identical environments containing both static and moving obstacles. Experiments are conducted to determine whether the DSN's ability to avoid static obstacles is inferior to the baseline. The distance to the surrounding obstacles is measured during every run, with which a comparison can be made. Furthermore, an experiment is conducted to compare the behaviour of both when approaching and passing moving obstacles. A method, that assumes that all obstacles are static, would opt for the less efficient trajectory. For instance, it would turn right when approaching an obstacle that is moving from left to right, resulting in an encounter with the obstacle in the near future. Figure 1 visualises this situation. Here, turning left would be the ideal trajectory.

Figure 1: Situation where a method, that assumes all obstacles are static, will opt for the less efficient trajectory.

The remainder of the thesis is organised as follows.

Section 2 covers related research and the state-of-the-art with regards to computer vision and obstacle avoidance; Section 3 elaborates on the baseline; Section 4 discusses the Differential Siamese Network; Section 5 describes the setup of the experiments; Section 6 gives the results acquired from the experiments; Section 7 contains the conclusion of the whole project next to further research possibilities.

2 Related Research

Current approaches to the obstacle avoidance problem often rely on machine learning methods. Image classification is an essential component and usually the starting point for approaches utilising visions sensors. Different architectures can be used and dictate the performance of the final method.

Krizhevsky et al. [KSH12] laid out a foundation for image classification by means of deep learning and CNNs. Their architecture, referred to as AlexNet, was developed with as purpose to classify images into 1.000 disjoint classes. It performed significantly better than the at the time state-of-the-art when comparing the top-1 and top-5 error rates. AlexNet consists of five convolutional layers followed by three fully-connected layers. This depth contributed to the performance since removing a single layer resulted in a deterioration in performance. Consequently, adding more convolutional layers to the architecture would result in an improvement. But due to the limitation in computational power at the time, they stopped deepen the network due to the prolonged training time. Additionally, their architecture is optimised to work on two GPUs. This is achieved by dividing the last four convolutional layers into two halves, thus dividing the workload between the GPUs.

AlexNet is trained on a subset of the ImageNet dataset [DDS+09], but the training needed to be sped up due to the network's great depth. To achieve this without losing a significant amount of performance, rectified linear units (ReLU) were used instead of the more traditional activation functions. The training was sped up by a factor 6 using ReLU, however this factor can vary between different architectures.

A deep network often comes with overfitting issues, as was the case for AlexNet. In order to reduce it, overlap pooling and data augmentation were used. For overlap pooling, a window size of 3×3 and a stride of 2 were taken. They observed that the models with overlap pooling overfitted to a lesser degree than the models without it. With regards to the data augmentation, two forms are used; reflecting the images horizontally and altering the intensity of the RGB-channels. These augmentations require little computation and can be executed by the CPU while the GPUs are simultaneously processing the previous batch. So there is no time penalty for augmenting the data.

Wang et al. [WJQ+17] developed the Residual Attention Network (RAN), which makes use of multiple, stacked upon another, Attention Modules. These modules consist of a trunk and mask branch. The trunk branch is used for feature processing, using residual units and convolutional layers. It can be altered to any network structure, resulting in the ability to incorporate various deep network architectures such as ResNet and Inception. This is one of the important properties of RAN. The mask branch utilises the output of the trunk branch to softly weigh its features in the form of a mask. This mask is then combined with the feature map and results in an enhancement of the feature map; background noise is eliminated and the features are refined. This enhanced feature map is the output of one module.

Another property of RAN is the fact that adding more Attention Modules results in a consistent improvement in performance, for the reason that each module applies a different type of attention to the feature map (i.e. soft attention).

RAN is compared with architectures which achieve state-of-the-art performance for the CIFAR-10 [Kri09], CIFAR-100 [Kri09] and ImageNet datasets. From all datasets came forth that RAN outperforms the state-of-the-art architectures while using significantly less parameters than its competitors. For instance, RAN used for the CIFAR datasets $8.6 \cdot 10^6$ parameters compared to $36.5 \cdot 10^6$ and $10.3 \cdot 10^6$ parameters for respectively ResNet and Wide ResNet. All in all, RAN is able to achieve lower error rates with fewer parameters.

With AlexNet in mind, Khan and Parker [KP19] developed an architecture to make a robot drive safely through an environment without colliding into any obstacle. As mentioned in the previous section, their work serves as the baseline for this research.

Khan and Parker developed a CNN that consists of multiple convolutional, normalization and pooling layers. They ran their architecture on a TurtleBot and experimented with it in a laboratory containing exclusively static obstacles. The environment changed for every experiment since it was used by students everyday; chairs may be relocated and garbage (e.g. pencils and paper) may be present on the floor. Consequently, the environment becomes more complex and dynamic for the robot since it may encounter never-seen-before obstacles and is not able to discover a pattern when driving in the environment.

Initially, the network was trained on the CIFAR-10 dataset and obtained a top-1 error rate of 26.0%. Then they made an additional dataset, that consists of images taken while manually driving the robot in the laboratory and not colliding into any object. They mapped the user input (left, right, straight) to every image and used this dataset to finetune the network. In the end, the network obtained a top-1 error rate of 10.0% on this dataset and did not collide during any of the experiments.

Xie et al. [XWMT17] also made use of a CNN, but it is only a part of their solution. They developed the deep double-Q network (D3QN) architecture with which a robot is able to manoeuvre in an environment using a monocular RGB camera. The network makes use of reinforcement learning, even though it requires significantly more training data in comparison to supervised learning methods, such as the CNN from Khan and Parker.

This training data is acquired via a virtual environment. But since a virtual simulator is vastly different to the real world (e.g. textures, lighting, etc.), the RGB image is transformed into a depth image. The first part of the D3QN, a convolutional residual network (based on the fully convolutional residual network from [LRB+16]), handles this task. The second part consists of the Q-network where the output is determined based on the depth image. The output consists of two components; a linear and angular velocity, with a select amount of options for each. The reward function uses these two components to determine the reward. It punishes the robot when it is rotating stationary and rewards the robot when it is driving at the maximum linear velocity. This results in the robot driving as efficient as possible.

Experiments were conducted from which can be concluded that D3QN performs significantly better than the DQN and DDQN architectures. This is based on the average rewards given during the runs in the virtual environment.

Mancini et al. [MCVC16] specifically developed a network aimed at vehicles that reach high speeds. For these vehicles applies that there is less time to detect and avoid an obstacle, therefore the network should be able to detect obstacles from a long distance and be able to operate on a high frame rate. The network only takes RGB images as input, with the option to expand it with the optical flow. This optical flow is computed by applying the Brox algorithm [BBPW04] to the previous and current image. It gives an indication with regards to the dimensions of an obstacle and the image's depth.

The network consists of an encoder and decoder component, which respectively consist of convolutional and deconvolutional layers. Padding and stride are applied in such way that the output of the network has the same dimensions as the input image (and optical flow). Consequently, every pixel of the output image contains the depth of the corresponding pixel in the input image. Then the trajectory can be determined and adjusted based on the surrounding obstacles with this image.

Experiments indicate that the network is on par with the state-of-the-art with regards to long distance obstacle detection. However, the network has as weak point its ability to detect obstacles which are nearby.

Similar to our research, Kim and Do [KD12] developed an algorithm with which a robot is able to detect and avoid moving obstacles. The only constraint is that the obstacles must move towards the robot. The algorithm makes use of a monocular camera and a site map. This map contains information with regards to the environment; it describes all static obstacles, including their dimensions.

The algorithm consists of a Block-Based Motion Estimation method to make an estimation with regards to the moving objects. Every image gets divided into non-overlapping blocks. Then for every block, a best matching block from the previous image is determined. This search for the best matching block is limited by a search window wherein it must be located. Then once the best matching block is found, a motion vector is computed with the spatial distance between the

current block and the best matching block from the previous image. If the motion vector is above a threshold value, it gets interpreted as a moving obstacle.

From the experiments came forth that the algorithm is sensitive to various lighting conditions. Moreover, moving obstacles, which are far away, are not detected by the algorithm since the movements appear slower at greater distances and are thus not exceeding the threshold value.

Cherubini et al. [CGSC13] provided a different take on the moving obstacle problem. They developed an algorithm that uses a Kalman-based observer to estimate the velocities of surrounding obstacles. Their robot is equipped with a pinhole camera and LIDAR.

The objective for the robot is to follow a path represented by an ordered set of key images. These key images were taken while manually driving a taught path. During the run, multiple images are taken with a subset being the key images. There were no obstacles present on the taught path.

The robot follows the taught path as long as possible, until obstacles are detected with the use of LIDAR. Then multiple candidate trajectories are formed, referred to as tentacles. Obstacle-velocities are estimated with the use of the Kalman-based observer, which are utilised to determine the trajectory of every obstacle with the assumption that its movement is uniform. The trajectories of the obstacles behind the robot are also taken into consideration. Then, the safest tentacle is determined by comparing the earliest time of collision for each candidate solution and then selecting the one with the highest time.

A comparison is made between their solution and the framework on which it is build on. This framework does not monitor the velocity of objects in the environment. Both approaches had to follow the same path in the environment. From this experiment came forth that their solution has a smoother trajectory than the framework. The robot was able to predict the future position of an obstacle and adjust its trajectory to avoid collision and maintain speed, while the framework did not and had to come to a stop to avoid colliding with the obstacle.

3 The baseline

This section discusses the work from Khan and Parker [KP19], which is a method to let a robotic vehicle safely manoeuvre in an environment with exclusively static obstacles. It is selected as the baseline for our research due to its robustness against never-seen-before obstacles and overall performance. The work consists of a CNN that maps a command to a RGB image. This command can be either `Left`, `Right`, or `Straight`. Then, the robotic vehicle, which runs the CNN, performs that command and sends a new image to the CNN to map that to the next command. This cycle repeats itself indefinitely, resulting in the vehicle driving autonomously.

3.1 The architecture

The baseline's architecture consists of three convolutional layers, with each directly followed by a pooling and normalisation layer (excluding the fully connected layer). See Figure 2 for a visualisation of the architecture. The figure also shows the dimensions of the input image, kernels and of all convolutional and pooling layers. The network terminates with a fully connected layer that has three outputs, corresponding with the three possible commands. This layer uses softmax to compute the confidence for every command. The activation function used by the convolutional layers is ReLU,

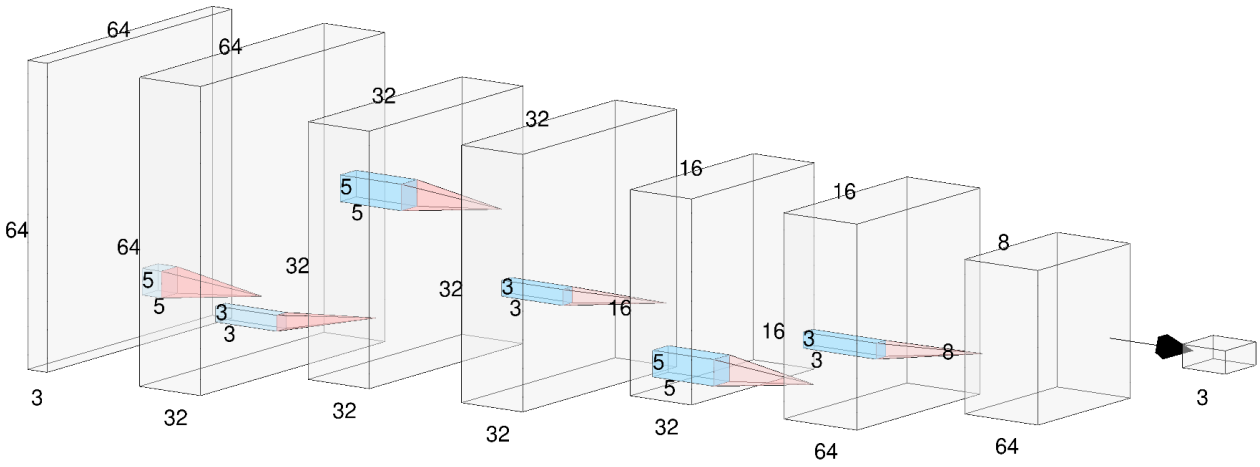


Figure 2: The architecture of the baseline consists of three convolutional layers with every layer directly followed by a pooling layer. The normalisation layers are not included in the visualisation.

due to its computional speed and it accelerates the convergence in comparison to its traditional counterparts such as sigmoid. It is described with the following equation:

$$f(x) = x^+ = \max(0, x) \tag{1}$$

However, this activation function does not prevent the saturation of neurons, since it is only lower bounded. So normalisation layers have been added to prevent this from occurring. Local Response Normalisation (LRN) [KSH12] is selected as normalisation method and it directly follows the first two pooling layers. LRN is described with the following equation:

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta \tag{2}$$

Here, $a^i_{x,y}$ is the activity of a neuron at position (x, y) of kernel i, N is the amount of kernels in the layer and $b^i_{x,y}$ is the normalised activity of the concerning neuron. The sum goes over the activation of neurons in n consecutive kernel maps which lie at the same position as the concerning neuron. The constants k, n, β and α are hyperparameters, and determined with the use of a validation set. The baseline uses the following values: $k = 2$, $n = 5$, $\alpha = 10^{-4}$ and $\beta = 0.75$.

LRN enhances a neuron with high activity and then dampens its surrounding neurons. If those surrounding neurons are low on activity, then it will further enhance the feature map. But if there is a patch in the feature map with high activity, then the whole patch will be dampened. Consequently, the small peaks, that represent useful information, are enhanced, while the patches are dampened. These patches do not give much information with regards to the feature. Those two effects combined result in better feature recognition performance and thus improving the overall performance of the network.

3.2 Baseline and AlexNet: A Comparison

Here, a comparison is made between the baseline and AlexNet. As mentioned in Section 2, AlexNet is a CNN that achieved state-of-the-art results in 2010 at top-1 and top-5 error rates when classifying images into one of 1.000 disjoint classes from the ImageNet dataset [DDS+09]. Figure 3 visualises the architecture of AlexNet.

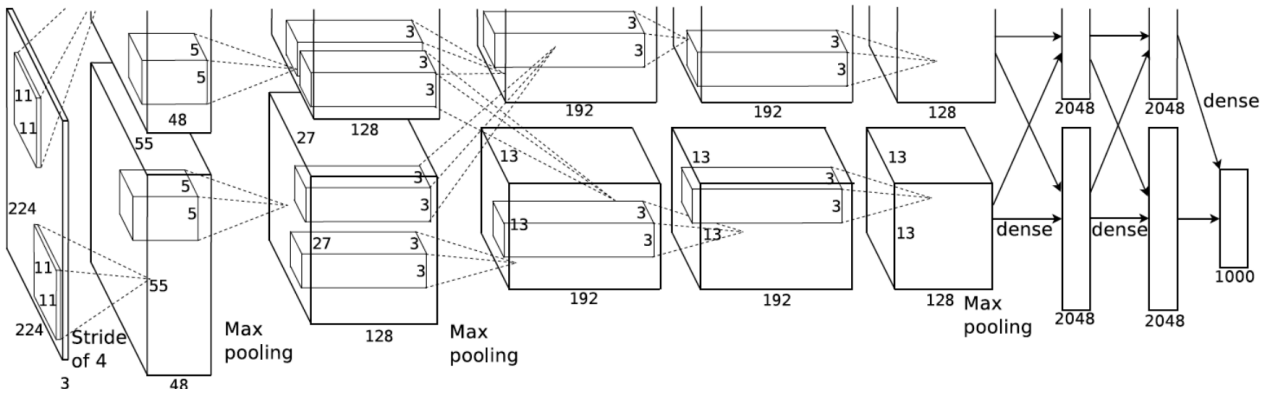


Figure 3: Visualisation of AlexNet. The network is split into two parts to optimise the training of the network with two GPUs. Acquired from [KSH12].

The baseline's depth is significantly more compact than AlexNet's; the baseline has two convolutional and two fully connected layers less than AlexNet. This is due to the difference in task both networks have to perform. Classifying images with 1.000 classes requires more network capacity than classifying CIFAR-10 and avoiding obstacles with only three possible outputs. This also applies to the amount of kernels per convolutional layer, the baseline has substantially less kernels than AlexNet.

In spite of these differences in size, the baseline is similar to AlexNet. Both have LRN-layers that follow the first and second convolutional layer and ReLU as activation function for every convolutional layer. Both make use of overlap pooling, although the baseline makes use of average and max pooling while AlexNet only uses max pooling.

As Khan and Parker denoted in their paper, they took AlexNet as a basis for their obstacle avoidance problem. Not all layers were adopted in their architecture to optimise the compactness of the network, while maintaining the performance as much as possible. It makes use of the same methods for normalisation, activation and pooling as AlexNet. So all in all, Khan and Parker minimised the amount of layers and parameters for their problem, but kept the overall structure and methods of AlexNet.

3.3 Modifications to the baseline

In contrary to the paper, we initially trained the baseline on the CIFAR-100 dataset [Kri09]. It obtained a top-1 error rate of 53.0%. This is significantly higher than the reported 26.0% with CIFAR-10 and due to the difference in datasets; CIFAR-100 contains substantially more classes than CIFAR-10 while containing an equal amount of images. So CIFAR-100 has less images per

class in comparison to CIFAR-10. Moreover, a decrease in performance indicates that the capacity of the baseline is insufficient for the additional classes.

On the basis of this, the baseline is modified to increase its capacity and thus decreasing the error-rate. These modifications are described in this subsection.

Since there was no source code available, the network had to be rebuild from scratch. Khan and Parker developed their network with the framework Caffe. We, on the other hand, have implemented it with the framework Keras (v2.3.1) and Tensorflow (v2.2.0) as backend.

Consequently, a verification must be made to assure that our implementation matches the reported performance of the paper. According to the paper, the baseline achieved a top-1 error rate of 26.0% on CIFAR-10. Our implementation obtained 26.7% ($^{+}/_{-}$ 0.9%) after training it multiple instances. Every training instance took circa 0.3 hours. From this can be concluded that our implementation matches the results of the paper and thus the fact that this implementation is a fair representation.

As for the modifications, the LRN layers are replaced with Batch Normalisation (BN) layers [IS15]. BN accelerates the training of a network significantly in comparison to LRN. It allows for higher learning rates without risk of divergence, due to the fact that BN reduces the internal covariate shift during training. Internal covariate shift is that the distribution of activations differs after each training iteration due to changing network parameters. So BN allows to achieve the same performance as LRN with less epochs. Its hyperparameter momentum is set to 0.4.

Since Krizhevsky et al. [KSH12] reported that increasing the size of AlexNet resulted in an improvement in performance, we inserted one convolutional and BN layer into the CNN before the third convolutional layer. Moreover, the amount of kernels for three of the four convolutional layers are increased. The amount of kernels are now respectively 32, 64, 128 and 128 for the four convolutional layers.

At last, one new direction is added to the set of commands, namely `Backwards`. This direction can be useful when the robot is about to collide with an object, or finds itself in a dead end.

All other aspects of the baseline remain unchanged; the convolutional layers have a kernel size of 5×5, a stride of 1, a padding of 2 and ReLU as activation function. As for the pooling layers, a kernel size of 3×3 and a stride of 2 are used. The fully connected layer utilises softmax as activation function. At last, the CNN contains L2-regularizers with a weight decay of 0.004 and stochastic gradient descent with a learning rate of 0.001 and a momentum of 0.9.

In the end, the top-1 error rate decreased with 16.4%, resulting in an error rate of 37.7% on the CIFAR-100 dataset. From here on in, the baseline of the research is the network of Khan and Parker with the above modifications applied to it. Figure 4 visualises the final architecture.

4 Differential Siamese Network

As of right now, the baseline is capable to traverse in an environment containing exclusively static obstacles. Its next command is solely based on the current state of the environment. Here, a state describes the position of all objects, including the robot, and the orientation of the robot. Subsequently, this is insufficient to detect moving obstacles since one state does not give any

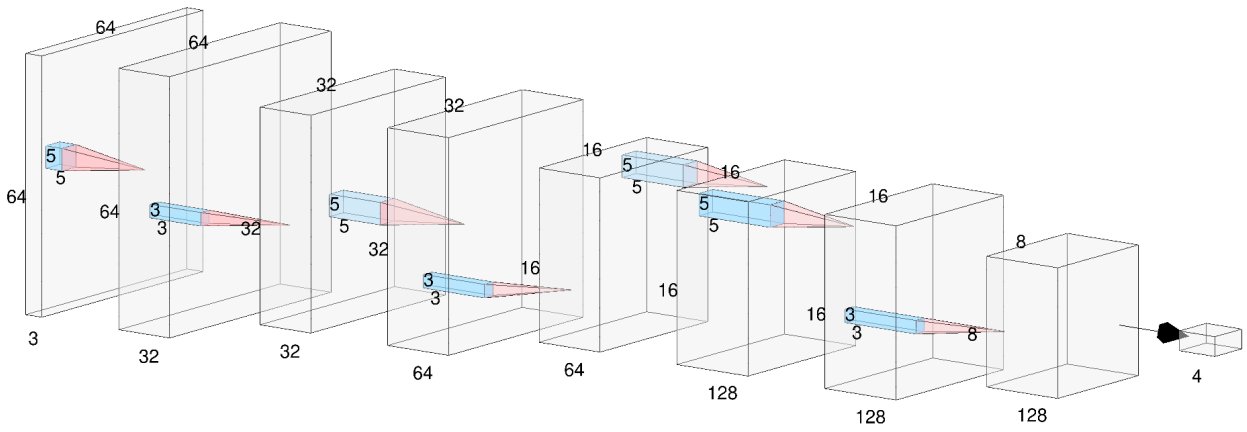


Figure 4: The architecture of the modified and final baseline, consisting of four convolutional layers and three pooling layers. The BN layers are left out in this visualisation.

information about movements. Hence the fact that the baseline is not capable to safely traverse in an environment with moving obstacles.

In order to detect the moving obstacles, multiple states have to be taken into account. With the use of these states can be determined whether an object moved from one to another state. This is a trivial task in the case of a non-moving camera, for instance background subtraction can be used to determine whether an object is moving.

In our case, detecting moving obstacles is more complex. The camera is part of the robot and consequently moving after every command. Therefore, the earlier mentioned method can not be applied here since the images will not correspond with each other due to the ever changing position and orientation of the camera.

For our proposed method, we take inspiration from the Siamese CNNs. These are used for image matching [MKR16], where the architecture consists of two or more CNNs and each producing a feature vector for its input image. These feature vectors can be utilised to determine whether the images match or not. Our method will determine whether two images are significantly different, instead of checking whether the images match.

We propose the Differential Siamese Network (DSN) which has the ability to detect moving obstacles, provided that they are always on the ground. That is, the DSN is not able to avoid an object appearing from above. Moreover, it assumes that the movement of the obstacles are uniform.

The DSN computes for two images their command vectors $\vec{C} = [c_l, c_r, c_s, c_b]$ (i.e. the output of the baseline). The first image I_c is from the current state of the environment, and the second image I_p is a prediction of the current state. This prediction is constructed with the image I_f from a preceding state and its corresponding command. The prediction assumes that all objects in the environment are static. Consequently, a significant difference between the two command vectors indicates that a moving obstacle is in front of the robot.

The network merges both command vectors into the final command vector from which the next command is determined. Figure 5 visualises the DSN, where the architecture is split up into four

components. Here, both CNNs are the baseline with identical weights W.

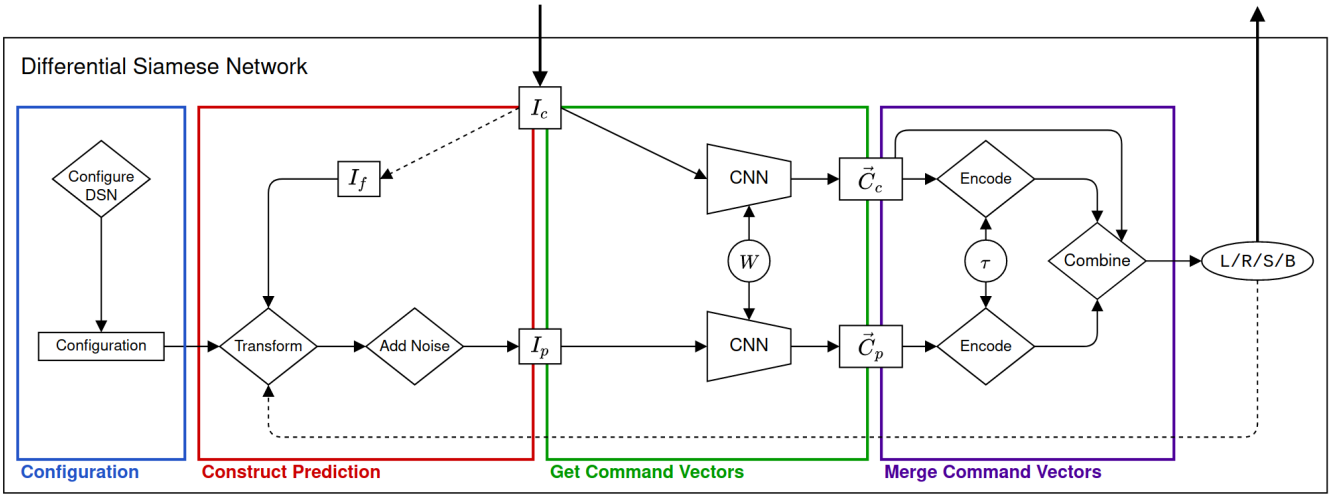


Figure 5: The architecture of the proposed DSN. It is split up into four components, denoted by the colored boxes. The figure contains dotted arrows which represent actions that occur after an iteration. For instance, I_c and the previous command will be used in the next iteration to determine I_p. Moreover, the bold arrows are the in- and output of the DSN.

There is an option with regards to which preceding state is used to construct the prediction with. For instance, a preceding state which occurred three states before the current state can be selected in order to increase the interval between the two states. Consequently, the potential difference between the current state and the predicted state enlarges as the interval between the states increases.

To denote which state is utilised to construct the prediction with, the architecture is referred to as DSN-n, where n indicates which state is used. So if we have states s_1, s_2, s_3 and s_4 which occur sequentially, then DSN-1 and DSN-3 take respectively s_3 and s_1 as the preceding states to construct the prediction with. As for Figure 5, it visualises the DSN-1.

The following subsections will discuss each of the components of Figure 5.

4.1 Constructing the prediction

The prediction I_p is constructed with the image I_f from a preceding state as its base. A set of transformations is applied to I_f resulting in the prediction of the current state. These transformations are based on the commands that are executed in between the preceding and current state.

A command can be split up into its angular and longitudinal component. The angular component is simulated by shifting the image, and the longitudinal component is simulated by applying zoom to the image. Then, I_f is right-shifted for a left turn and left-shifted for a right turn. The blank pixels that are a result of the shifting are filled in with the nearest pixel on the same row. In order to avoid horizontal lines, noise is added to these pixels. The transformations are visualised in Figure 6. The original image is also included as reference.

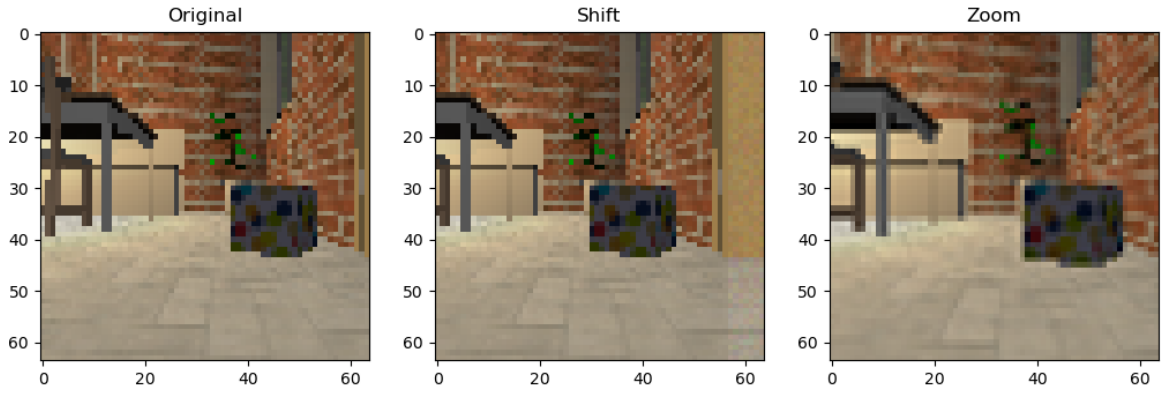


Figure 6: The transformations shift and zoom applied to an image with a resolution of 64×64. The middle image is left-shifted by 7 pixels and represents the prediction after executing the command `Right`. The right image is the result of zooming in with 4 pixels on each side and represents `Straight`.

However, the method comes with one downside. After applying shift-transformations to the image, there are less pixels that provide information about the environment, as can be seen in Figure 6. Subsequently, there are certain restrictions on what preceding state can be used. For instance, if there are only `Left` commands executed by the DSN in between a preceding and current state, then there is a point where there are very few to no useful pixels remaining in the prediction. Then the command derived from this prediction is useless. Therefore, we will only consider DSN-1, DSN-2 and DSN-3 for the experiments of this research.

Moreover, a preceding state I_f is not considered by the DSN if `Backwards` follows after I_f, due to the fact that the image from I_f will be mainly occupied by a nearby obstacle. After applying transformations to the image, the concerning obstacle will still occupy most of the image while I_c will not. So constructing a prediction of I_c under these circumstances will not result in a meaningful comparison between I_c and I_p with regards to their command vectors. So, if I_f is followed by `Backwards`, then the command vectors will not be merged and the command vector from I_c will be taken to determine the next command.

Configuring the DSN

The DSN needs to be configured before it is able to construct predictions and approximate the current state. The amount of shifting/zoom applied to the image is dependent on the definition of the commands. For instance, there are multiple ways to let a differential wheeled robot turn left; stationary (both wheels rotate in the opposite direction) or moving partially forward (both wheels rotate in the same direction). Moreover, the speed of rotation can be defined for the command. Consequently, every definition of `Left` requires a different amount of shifting/zoom to approximate the angular and longitudinal components. Otherwise, the prediction would be inaccurate. Likewise for the other commands.

Therefore, the DSN must be configured with the command specifications before it can be run. The user defines the rotation of the motors for every command. Then, a configuration run takes place in CoppeliaSim [RSF13] where it takes an image after executing every command. This results

in an image before (I_f) and after (I_c) executing each command. These images are compared to determine how much shifting and zoom is required to simulate the concerning command.

For every image are its feature points detected with Accelerated-KAZE (A-KAZE) [ANB13]. This method is used, instead of alternative methods such as SIFT, SURF and KAZE, due to its performance to computation time ratio. The method makes use of a nonlinear scale space to detect the features of an image. Then, with the use of a Modified-Local Difference Binary, it constructs a description for each feature based on its gradient. In contrary to the low resolution images used by the DSN, the configuration process utilises higher resolution images such that more feature points can be detected.

Then, a feature matching method, Fast Library for Approximate Nearest Neighbour (FLANN) [ML09], is used to match the feature points from both images. For every matched set i is its translation vector $\vec{t}_i$ computed. A sum over all translation vectors, divided by the total amount of matched sets k, results in the general translation vector $\vec{T}$ for the concerning command. Every command has its own $\vec{T}$ and it contains the amount of pixels for the transformations to simulate the concerning command.

$$\vec{T} = \left(\frac{\sum_{i=0}^{k} \vec{t}_i}{k} \right) / a \;\text{ with }\; a = \frac{r_c}{r_{DSN}} \tag{3}$$

Here, r_c and r_{DSN} are the resolutions used for respectively the configuration and the DSN.

The y-component of $\vec{t}$ is only considered when the feature point lies in the upper halve of the image. Otherwise, the translation of feature points in the lower halve of the image would cancel out the translation of the feature points in the upper halve, and thus resulting in a zoom approximating zero.

Figure 7 visualises the feature matching between two images. Here the left image is before the command `Left` was executed, and the right image is after the execution.

Figure 7: Feature matching of two images with A-KAZE and FLANN. The red dots represent feature points that did not have a match with the other image. The green dots and lines are matched pairs of feature points.

The feature matching method and Equation 3 make it possible to configure the DSN based on the user-specified commands. After the DSN is configured, it is able to construct a prediction which approximates the current state as closely as possible. The following figure shows an example of a prediction, with its I_f and I_c as reference.

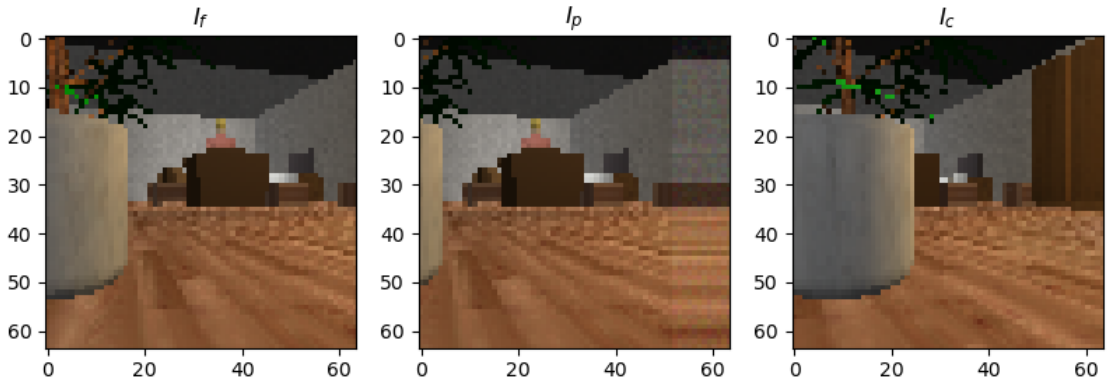


Figure 8: The prediction I_p constructed with I_f. The command that results in I_c from I_f is `Right`.

4.2 Merging the command vectors

After constructing the prediction, both I_p and I_c are inserted into the CNNs. The CNNs will determine $\vec{C}_p$ and $\vec{C}_c$, which are used to determine whether there is a moving obstacle in front of the robot. If $\vec{C}_p$ and $\vec{C}_c$ are significantly different, then this indicates that the images I_p and I_c are significantly different and thus the presence of a moving obstacle. Figure 9 gives a possible situation in which the DSN will be able to detect a moving obstacle, with the corresponding command vectors below.

However, not all commands will be taken into account when comparing the command vectors. The commands which have a confidence greater or equal to a threshold value τ are not considered. These commands are interpreted as free space where the robot can drive into, so they are not of interest when detecting a moving obstacle. Moreover, `Backwards` is left out for the same reason when constructing the prediction.

We take $\tau = 0.35$ with as consequence that at most two commands in the command vector exceed the threshold value. Table 1 gives the six combinations of directions, with this τ, that are occupied by an obstacle. Here, the '■' means that an obstacle is present in that direction and the '-' means that the space in that direction is free (and thus that its command vector confidence greater is than τ).

Then, the combination is encoded with $e \in \mathbb{Z}$, indicating where the obstacles are located relative to the robot. e is computed with the following equation.

$$e = \left(\sum_{k \in \{l,r\}} v_k a_k \right) \cdot (1 - v_s a_s) \text{ with } a_x = \begin{cases} 0 & \text{if } c_x \geq \tau \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

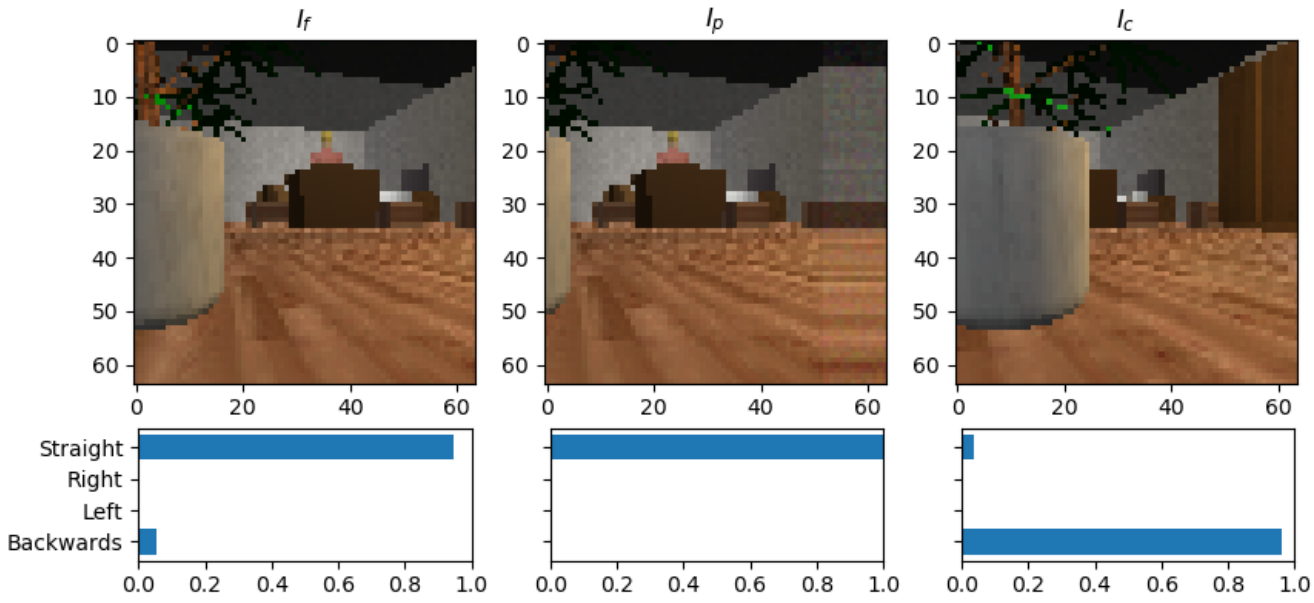


Figure 9: Command vectors for I_f, I_p and I_c from Figure 8. It shows a significant difference between I_p and I_c, indicating that there is a moving obstacle in front of the robot.

Here, v_x is a factor defined in $\vec{V} = [v_l, v_s, v_r] = [-2, 0.5, 2]$. The encoding e is also included in Table 1.

	Left	Straight	Right	e
1	■	-	-	-2
2	■	■	-	-1
3	-	■	-	0
4	■	-	■	0
5	-	■	■	1
6	-	-	■	2

Table 1: All possible combinations with $\tau = 0.35$ and the corresponding encoding e. The combinations are ordered based on e.

As can be seen in Table 1, the combinations are ordered by their encoding. A negative encoding states that there is an obstacle present to the left of the robot. Moreover, the lower the negative e, the further to the left the object is located relative to the robot. Likewise for $e > 0$, but the greater the value, the more to the right the object is located. If e is zero, then the obstacle is right in front of the robot, with the exception of combination 4. Here, an obstacle is on both sides of the robot.

The encodings e_p and e_c, for respectively I_p and I_c, are used to determine the movement of the obstacles. Since a negative e represents an obstacle to the left and $e > 0$ an obstacle to the right, a

comparison can be made between the two.

$$\text{Movement} = \begin{cases} \text{Left} & \text{if } e_p < e_c \\ \text{Right} & \text{if } e_p > e_c \\ \text{Static} & \text{if } e_p = e_c \end{cases} \tag{5}$$

The equation becomes clear with Table 1; from combination n to $n+k$, with $k \in \mathbb{N}_1$ and $n+k \leq 6$, an obstacle is moving to the right. From combination n to $n-k$, with $k \in \mathbb{N}_1$ and $n-k > 0$, an obstacle is moving to the left. If both encodings are the same, then there is no moving obstacle detected.

There is however an exception, if combination 3 and 4 apply to respectively I_p and I_c (or the other way around), then there is a moving obstacle in front of the robot while the encodings are identical. Since there is no way to determine in which direction the object moved with this method, $\vec{C}_c$ will be adopted as the final command vector.

If $e_p = e_c$, then $\vec{C}_c$ is taken as final command vector. If a movement is detected, then the command corresponding to the detected movement is set to zero to avoid the moving obstacle. So if a movement to the right is detected, then $c_r = 0$ in the final command vector. The other entries in the final command vector are adopted from $\vec{C}_c$. Likewise for a movement to the left.

At last, the entry in the final command vector with the highest confidence is selected as the next command.

5 Experimental setups

This section contains the description of the experimental setup. The baseline is trained on a machine with the following specifications:

- Intel Core i7-5820K 3.3 GHz (12 cores)
- Nvidia GeForce Titan X
- 64 GB of main memory

Multiple instances of the baseline are trained, with each an unique subset of the below described datasets. Multiprocessing is enabled in Keras to speed up the retrieval of images.

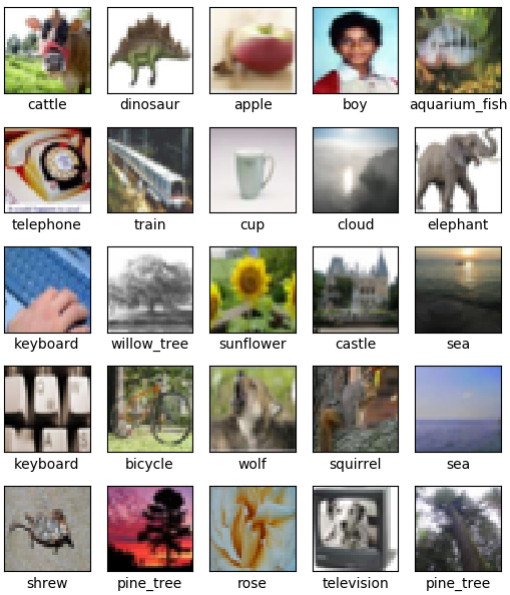
5.1 Datasets

Four datasets are used to train the baseline with. Two datasets are aimed at image classification, while the other two are aimed at the obstacle avoidance problem.

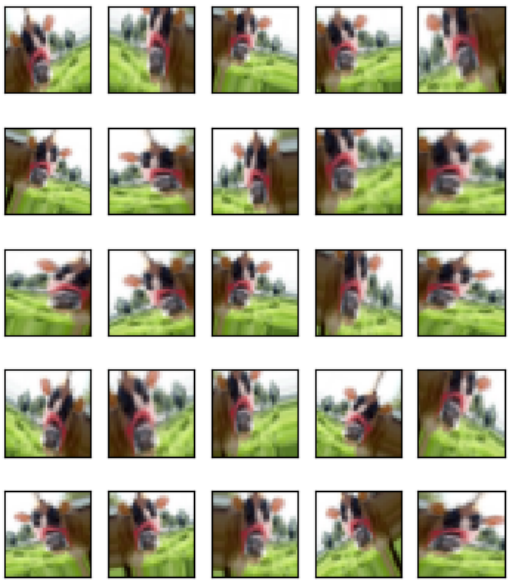
CIFAR-100

As mentioned in Section 3.3, we used the CIFAR-100 dataset in contrary to CIFAR-10. This dataset contains 100 disjoint classes which are subdivided into 20 superclasses. We opted for CIFAR-100 over CIFAR-10 due to it containing more classes, resulting in the network learning more shapes. Consequently, this is beneficial for the robustness of its obstacle detection and thus avoidance.

Figure 10a shows the first 25 images from the CIFAR-100 dataset, with their corresponding classes. The resolution is 32×32, identical to the CIFAR-10 dataset.



(a) Subset of CIFAR-100.



(b) Augmented instances.

Figure 10: A subset of the images with their corresponding classes visualised in (a), with data augmentation applied to the first image in (b). It shows that after the augmentation is applied, the images remain recognisable as a cattle.

However, this dataset comes with a drawback: it contains the same amount of images as CIFAR-10. As a consequence, the dataset contains significantly less images per class in comparison to CIFAR-10, 600 instead 6.000 per class. This resulted in overfitting.

Data augmentation is applied to the training set of CIFAR-100 to reduce overfitting. It consists of the following transformations: horizontal flip, rotation (30), width/height shift (both 4), zoom (0.3) and channel shift (0.1). The value between the brackets is the parameter of a transformation (when applicable). These transformations and parameters were chosen such that the images remain recognisable but are vastly different to the original. The nearby pixels are used to fill in the empty pixels after width/height shifting. Figure 10b visualises the possible augmented instances of the first image in Figure 10a.

Open Images Dataset v6

The next dataset is a subset of the Open Images Dataset v6 (OIDv6) from Google [KRA+20]. This dataset is chosen due to the freedom it provides with regards to creating a subset; it can be filtered on the desired classes. Our desired classes appear in an urban, office and/or household environment. These classes are the most likely to appear in the environments wherein a robot finds itself. For example, a smartphone, cup, pen and a pair of headphones are classes of the subset. The complete list of classes can be found in Appendix A.

In total, the subset contains 436.950 images and 64 disjoint classes. The images are downscaled to a resolution of 64×64, in contrary to the lower resolution of CIFAR-100. The same data augmentation as for CIFAR-100 is also applied to the OIDv6 subset.

Due to an unequal distribution of classes in the subset, class weights are used such that every

instance from under-represented classes are more important than the instances from over-represented classes.

Moreover, these classes are initially not disjoint; a multitude of images belongs to multiple classes. These images are removed from the subset with the annotated files provided by Google, such that every image belongs to exactly one of the selected classes.

OAR and OAH datasets

The final datasets are specifically developed for the obstacle avoidance problem and referred to as the OAR (Obstacle Avoidance Room) and OAH (Obstacle Avoidance Hallway) datasets. It contains images that are mapped to one of four possible commands; `Left`, `Right`, `Straight` and `Backwards`. The images have a resolution of 64×64.

The OAR and OAH datasets contain images taken in respectively a household and hallway environment. In total, they contain respectively 34.513 and 39.262 images. The images were collected in circa 14 hours and were horizontally mirrored to double the amount of images in the datasets. So if an image belongs to `Left`, then the image mirrored horizontally belongs to `Right`. The same applies for `Right` to `Left`. With regards to `Straight`, images belonging to this command will still belong here when mirrored. The same applies to `Backwards`. No further data augmentation is applied to either dataset. Figure 11 shows an example for each command.

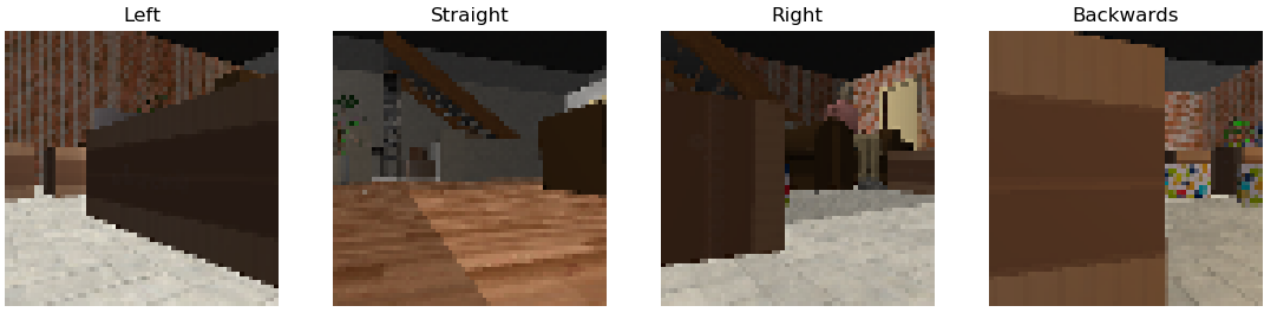


Figure 11: An example for each command. These images are taken in a virtual environment and have a resolution of 64×64 pixels.

The two virtual environments can be seen in Figure 12. Textures and lighting have been applied to the environments to make them as realistic as possible. We also used textures to create a view out of the windows, which would otherwise remain plain and thus adding to the realism.

Moreover, Figure 12a visualises the base environment for the OAR dataset. Multiple environments are used to create a diverse dataset, with every environment being slightly different than the base. Lighting conditions were altered and objects were added/moved/removed to make the dataset as varied as possible, but the main layout remained unchanged for every environment. Thereby, the texture of the floor and walls are changed once such that the network would not learn to follow the texture of the floor and avoid the texture of the wall. See Figure 11 for the difference in textures (e.g. `Left` vs. `Straight`). This does not apply to the hallway environment.

The datasets are created by driving a robot in a virtual environment, where an image is mapped to its corresponding command every 100 ms. CoppeliaSim is utilised as the simulator for the creation

(a) Household environment.

(b) Hallway environment.

Figure 12: The environments which are used to create the datasets. However, the household environment is the base environment for the OAR dataset. Multiple environments are created to cover a broad spectrum of scenarios.

of the dataset and the robot is a Pioneer p3dx. This robot is not manually controlled, but makes use of nine ultrasonic sensors to observe its surroundings. To avoid the obstacles, the robot makes use of the Braitenberg algorithm with as input the ultrasonic sensors. This algorithm computes the velocity of the left and right motors with the use of the following equation:

$$v_k = v_0 - \sum_{i=1}^{n} \beta_{k,i} \cdot D_i \tag{6}$$

Here, i is an ultrasonic sensor, n the amount of ultrasonic sensors on the robot, k the left or right motor, $\beta_{k,i}$ a Braitenberg factor, v_0 the starting velocity and D_i the measured distance by the sensor. If the sensor did not find any object, then $D_i = 0$. Otherwise, D_i is computed with the following equation:

$$D_i = 1 - \frac{\max(d^i_{meas}, d_{max}) - d_{max}}{d_{nodetect} - d_{max}} \tag{7}$$

Here, d^i_{meas} is the measured distance of sensor i, d_{max} the maximum distance at which the measurement is capped and $d_{nodetect}$ indicates from which distance objects no longer can be detected. So increasing $d_{nodetect}$ results in the robot earlier taking evasive actions to avoid a collision.

The above equations do not apply to `Backwards`. If sensor 9, which points directly in front of the robot, detects an object within a distance a, then v_l and v_r are both overwritten with -1 as their velocity.

The algorithm is already implemented for the Pioneer in CoppeliaSim, but has been tweaked such that it had the desired behaviour. For instance, it would originally turn at a very short

distance from the obstacle while we want the robot to take actions at an earlier stage. Moreover, the original algorithm made use of 8 ultrasonic sensors, but most of these are outside of the FoV of the monocular camera. So these sensors are redirected to be in the FoV, with every sensor at a different angle with regards to the center of the robot. Furthermore, one sensor has been added to the Pioneer which points directly in front of it.

Every image its command is determined based on the velocity of both motors v_l and v_r.

$$\text{Command} = \begin{cases} \texttt{Left} & \text{if } v_r \geq v_l + 1 \\ \texttt{Right} & \text{if } v_l \geq v_r + 1 \\ \texttt{Backwards} & \text{if } v_l < 0 \text{ and } v_r < 0 \\ \texttt{Straight} & \text{otherwise} \end{cases} \tag{8}$$

Tweaking the algorithm resulted in the following values for the parameters of equations 6 and 7: $v_0 = 3$, $d_{max} = 0.2$, $d_{nodetect} = 1.2$, $a = 0.3$. The values for $\beta_{k,i}$ are described in Table 2.

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
β_r	-2.0	-1.6	-0.4	-0.2	0.0	0.0	0.0	0.0	-2.0
β_l	0.0	0.0	0.0	0.0	-0.2	-0.4	-1.6	-2.0	-2.0

Table 2: The Braitenberg values used to compute v_r and v_l. Here, sensor 1 is the leftmost sensor, while 8 is the rightmost sensor. Sensor 9 points directly in front of the robot.

These values are chosen such that the velocity of a motor is only affected if an obstacle is detected at its side of the robot. This way, the robot would turn sharply to the other side.

We opted for this way of creating the dataset since the behaviour of the robot remains consistent with the use of an algorithm, in contrary to manually controlling the robot. A human would not be able to interpret similar situations the same way every time and this would cause the dataset to be inconsistent.

Overview datasets

The following table contains an overview of all the datasets that are used to train the CNN with.

	CIFAR-100	Open Images Dataset V6	OAR dataset	OAH dataset
Training set	45.000	433.430	29.687	34.189
Validation set	5.000	1.280	2.423	2.423
Test set	10.000	2.240	2.403	2.272
Total:	**60.000**	**436.950**	**34.513**	**39.262**

Table 3: Amount of images per dataset, divided into their training, validation and test sets.

5.2 Experiment 1: OA dataset performance

First will be investigated with which dataset(s) the baseline is able to achieve the highest accuracy on the OAR/OAH dataset. Three instances of the baseline will be trained: one with CIFAR-100, one with CIFAR-100 + OIDv6 and one with OIDv6. Then, the baseline is finetuned on the OAR or OAH dataset. Here, all layers are frozen with the exception of the fully connected layer.

As for the model which is trained with both CIFAR-100 and OIDv6, it is first trained with CIFAR-100. Then a new model is created for the OIDv6, which is initialised with the weights from the CIFAR-100 model. No layers are frozen in the new model, so the whole model is trained further on the OIDv6 dataset.

A fourth baseline will be assessed on the OAR and OAH datasets. This baseline has the highest performance on the other obstacle avoidance dataset. That is, the network with the highest performance on the OAH dataset will be assessed on the OAR dataset. Likewise for the other way around. This indicates whether the baseline would be able to safely manoeuvre in an unknown environment.

The batch size and amount of epochs is set to respectively 64 and 100 for all datasets.

Since the CIFAR-100 dataset contains classes which are not relevant to obstacle avoidance (e.g. insects, rockets and fish), its performance on the OA datasets might be worse compared to training it with the OIDv6 dataset. Furthermore, training the baseline on both CIFAR-100 and OIDv6 might result in the best performance out of the three training methods.

Furthermore, the fourth baseline trained on the OAR dataset is expected to perform better in an unknown environment than the fourth network trained on the OAH dataset. This is due to the fact that the OAR dataset contains images from multiple environments, in contrary to the OAH dataset.

5.3 Experiment 2: DSN-n vs. baseline

The following experiment is conducted to compare the DSN architecture with the baseline. Both have to traverse in environments containing static and/or moving obstacles. The moving obstacles have horizontal movements and are located on the ground.

To ensure a fair comparison, the experiment is conducted in a virtual environment due to the fact that both networks will find themselves in an identical environment. The moving obstacles are objects which are programmed to follow a predetermined path with an uniform velocity. The virtual environments are created with CoppeliaSim.

In contrary to the development of the OA datasets, a Lumibot is used for the experiments for future research purposes. The Lumibot is equipped with a camera and ultrasonic sensor. This ultrasonic sensor is used to measure the distance to obstacles during the runs. It points directly in front of the robot and does not influence the trajectory in any way. The range of the ultrasonic sensor is set to 2 meters.

Aside from the ultrasonic sensor data, the amount of collisions is also denoted per architecture.

The experiment consists of two components. The first is to verify that the DSN does not lose any performance in comparison to the baseline with regards to avoiding static obstacles. The second is to assess the DSN and baseline purely on the avoidance of moving obstacles.

Considerations
Not all collisions are taken into consideration, only the collisions where the obstacle is in the FoV of the camera are counted. In other words, an obstacle colliding with the robot from behind is ignored. Moreover, if the robot is not able to recover from a collision during a run (e.g. the robot keeps driving into the wall), then the robot will be manually re-positioned to continue the run.

Furthermore, there occurred situations where the robot would be stuck. It finds itself in a loop where it continuously drives backwards and then immediately forwards. In order to avoid these situations, the execution of backwards is randomised. The command will be defined as $[x, x]$ where $x < 0$ for the left and right motor of the Lumibot. Then, the left or right motor will be randomly set to $-x$ while the other remains x. This way, the robot stationary turns left/right and avoids the described situation.

Experiment 2.1: Static obstacles

Firstly, we have to determine whether the DSN loses any performance when avoiding static obstacles. In order to determine this, the architectures will traverse in environments with predominantly static obstacles. The environments can be subdivided into the room and hallway environments. Examples for both can be seen in Figure 13.

(a) Room environment.

(b) Hallway environment.

Figure 13: A subset of the environments. The red lines represent the trajectories of the moving obstacles, the yellow line represents the range of the ultrasonic sensor. The FoV of the camera is represented by the blue lines.

Five configurations are constructed for the room environments. The differences between the configurations consist of the following:

- The velocity and the path of the moving obstacles are altered.
- Obstacles are added/moved/removed.
- The initial orientation and position of the robot is altered.

This way, a broad amount of scenarios are covered in the experiment, resulting in a fair comparison. Important to note is that these configurations are not part of the environments with which the OAR dataset is developed.

The robot traverses in every configuration for five minutes during which the amount of collisions is counted and the data from the ultrasonic sensor is gathered.

As for the hallway environment, there is only one configuration which can be seen in Figure 13b. The robot has to complete a lap both clockwise and anti-clockwise. The same data-gathering from the room environments applies here.

In total, 25 minutes and 2 laps of data is gathered per architecture.

A comparison can be made with this data and be determined which architecture opted for the overall safest trajectory. Say we have trajectories t and u, then t is safer than u if its amount of collisions is lower than u. The data from the ultrasonic sensor is taken into consideration when the amount of collisions is equal. For this measurement applies that the safer trajectory t kept an overall greater distance to the obstacles than u.

Experiment 2.2: Moving obstacles

For the second part of the experiment, small environments will be created in which moving obstacles are present. There are no static obstacles present other than walls. The objective for the robot is to reach the other side of the environment, by passing the moving obstacles. These environments also contain textures and lighting effects. Figure 14 contains the two environments. For both environments applies that multiple configurations are used to cover a broad amount of scenarios. The configuration consists of the initial orientation and position of the robot, and the velocity of the moving obstacles.

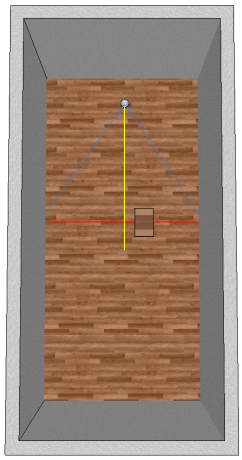
We opted for the construction of several small environments over large and extensive environments since the large environments also contain static obstacles. Consequently, a significant portion of the run would be occupied by the robot avoiding static obstacles, which is not desired.

Only the amount of collisions is denoted per architecture.

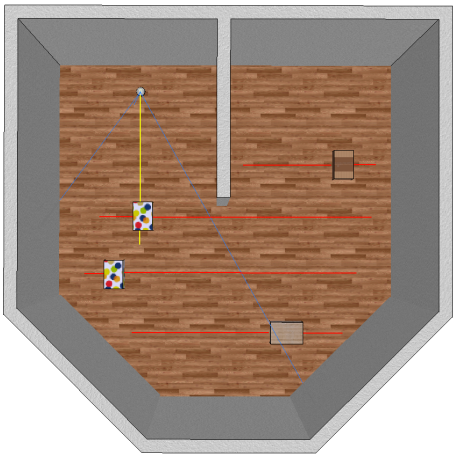
6 Experimental results

This section contains the results of the conducted experiments, as described in the previous section. There are three instances of the baseline: one trained on CIFAR-100; one trained on CIFAR-100 and OIDv6; and one trained OIDv6. The performance and training times are denoted in Table 4.

The table shows that the performance on the OIDv6 dataset is significantly worse in comparison to CIFAR-100, this is due to the distribution of classes in the training data. CIFAR-100's distribution



(a) Configuration 1.



(b) Configuration 2.

Figure 14: Two configurations for this experiment. Similar to the previous figures, the yellow line is the ultrasonic sensor while the red lines are the trajectories of the moving obstacles. Blue represents the FoV.

Baseline trained on:	Loss	Top-1 error rate	Top-5 error rate	Recall	Precision	Time (h)
CIFAR-100	1.606	0.366	0.088	0.532	0.766	0.3
CIFAR-100 + OIDv6	4.368	0.792	0.538	0.116	0.406	5.5
OIDv6	4.309	0.799	0.560	0.413	0.413	5.3

Table 4: Performance and training time per baseline instance. The third instance contains one entry corresponding to the performance on the OIDv6 dataset.

is equal (each class 600 images), while OIDv6 contains classes with 200 to 30.000 images in the training set.

6.1 Experiment 1: OA dataset performance

OAR dataset

We will first discuss the performance on the OAR dataset. The first three networks are trained on the OAR dataset, while the fourth network is trained on the OAH dataset and had the highest performance on that dataset. This network is pretrained with CIFAR-100. Table 5 denotes the performance per network.

The networks, with the exception of the network from the OAH dataset, are alike performance wise. The first network has the lowest loss, while the third has the lowest error rate and highest precision. As for fourth network, it performs significantly worse than the other networks. This is probably due to the environments which are in the OAR dataset. As mentioned in Section 5.1, the environment is changed several times. Even more, the texture of the floor and wall were changed once. All in contrary to the one environment which is used to develop the OAH dataset with, indicating that the network trained on the OAH dataset is not able to achieve a high performance in unknown

Baseline trained on:	Loss	Top-1 error rate	Recall	Precision
CIFAR-100	*0.875*	0.184	*0.811*	0.824
CIFAR-100 + OIDv6	1.006	0.180	0.800	0.832
OIDv6	1.004	*0.174*	*0.811*	*0.840*
OAH dataset	3.367	0.536	0.456	0.470

Table 5: The performance of the four networks with an average training time of 0.13 hours. In *italics* are the best results over the four networks.

environments.

Figure 15 shows the top-1 error rate training trajectory for the three network instances. As can be seen in the figure, the network trained on CIFAR-100 overfits significantly more to the training data than the other networks. But with regards to the top-1 error rate on the validation set, the three networks are closely matched.

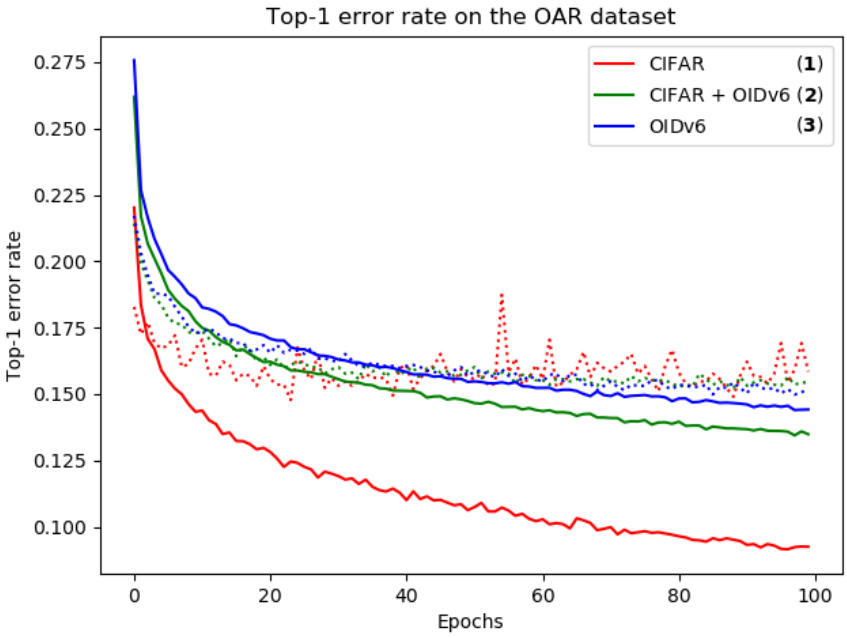


Figure 15: Top-1 error rate per baseline instance. The solid and dotted lines are for respectively the training and validation set.

OAH dataset

Likewise to the OAR experiment, the three networks are trained on the OAH dataset, while the fourth is the network with the highest performance on the OAR dataset. The best performing network from the OAR experiment is the network trained on both CIFAR-100 and OIDv6. Table 6 gives the performance and training time.

The first network achieves the highest performance on the basis of all metrics. Notably, the fourth network performs significantly better than the fourth network of the OAR experiment.

Baseline trained on:	Loss	Top-1 error rate	Recall	Precision
CIFAR-100	*0.538*	*0.059*	*0.940*	*0.941*
CIFAR-100 + OIDv6	0.764	0.079	0.908	0.926
OIDv6	0.772	0.075	0.918	0.930
OAR dataset	1.267	0.272	0.713	0.743

Table 6: The performance of the four networks, where the text in *italics* is the best result over the networks. The average training time for the OAH dataset is 0.16 hours.

They have a top-1 error rate of respectively 0.272 and 0.536. This indicates that the change in environment, in particular to the change in texture of the floor and walls, positively influences the performance of a network in an unknown environment.

Figure 16 shows the development of the top-1 error rate during training. Similar to the OAR experiment, the first network achieves the lowest error rate on the training set. Furthermore, the second network always outperforms the third, although slightly for both the OAR and OAH experiments. Thus, training the baseline on both CIFAR-100 and OIDv6 results in an increase in performance in comparison to training solely on the OIDv6 dataset.

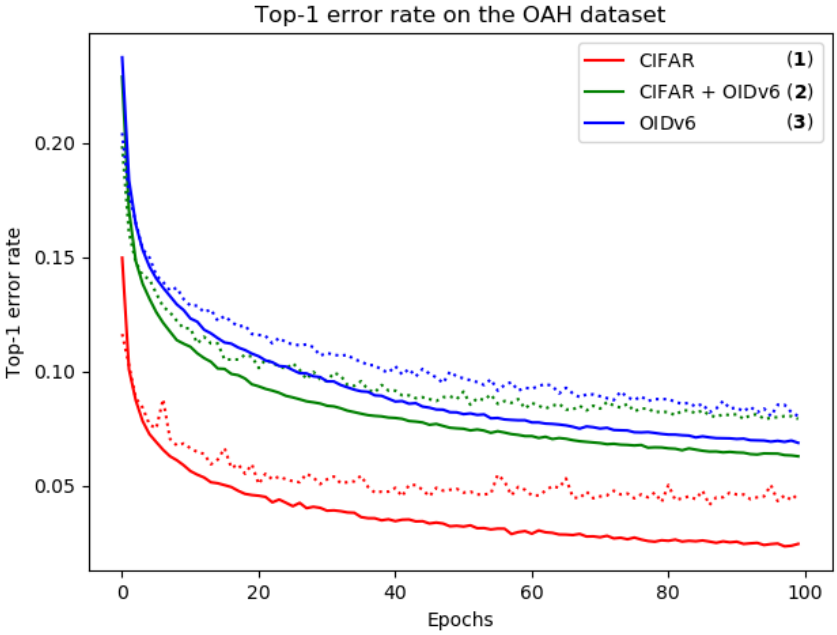


Figure 16: Top-1 error rate per baseline instance. The solid and dotted lines are for respectively the training and validation set. All three networks are slightly overfitting to the training set, but the first network significantly less in comparison to the OAR experiment.

ROC space

At last, every baseline instance is plotted in the ROC space, which can be seen in Figure 17. The figure shows that the first three networks from the OAH dataset are excellent, approximating a perfect score. The network trained on the OAR dataset performs worse than all three. As for the OAR dataset, the three networks trained on the dataset are almost identical, while the fourth network approximates the dashed black line. This strengthens the claim that training on multiple environments increases the performance of a network in unknown environments.

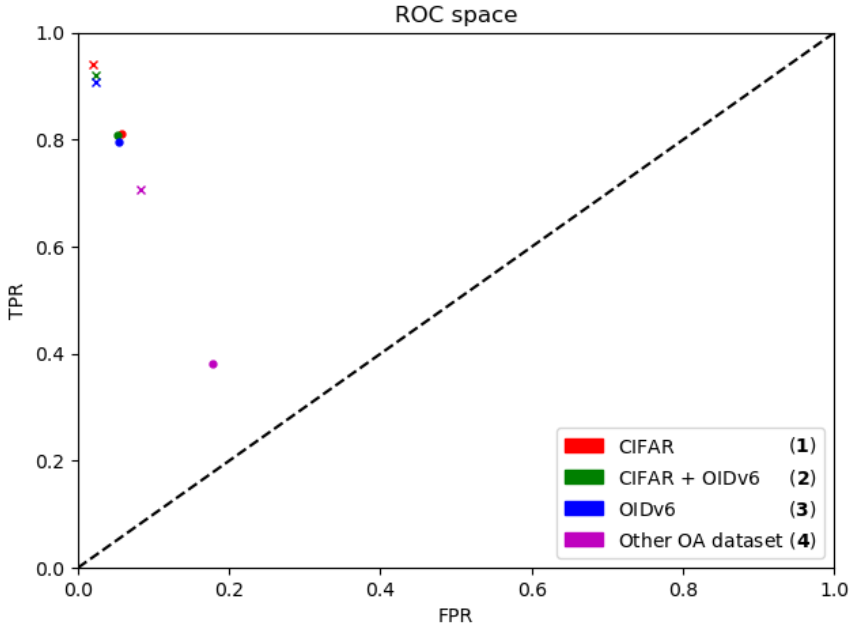


Figure 17: All networks plotted in the ROC space. The '×' and '•' represent respectively a network from the OAH and OAR experiments. The network from the other OA dataset always performs significantly worse than the networks which are trained on that dataset.

6.2 Experiment 2: DSN-n vs. baseline

When taking the results from the preceding experiment into consideration, the network with the highest performance for the OAR and OAH dataset are respectively networks trained on CIFAR-100 + OIDv6 and CIFAR-100. These networks will be used to assess the DSN-n and baseline. So, the network with the highest performance on the OAR dataset is used in the household environments, while the network with the highest performance on the OAH dataset is used in the hallway environment.

Experiment 2.1: Static obstacles

With this experiment will be determined whether the DSN's performance in environments with predominantly static obstacles is significantly worse than the baseline.

The data from the ultrasonic sensors will be subdivided into five ranges, which are: $[0, 0.25]$, $(0.25, 0.5]$, $(0.5, 1.0]$, $(1.0, 2.0]$ and $(2.0, +\infty)$. Since the range of the ultrasonic sensor is set to 2 meters, everything outside of this range will belong to the fifth range.

Furthermore, the average distance from the Lumibot to the obstacles is computed. Only the data points which belong to one of the first four ranges are included in the average. This, in addition to the distribution of the data points in the five ranges, is visualised in Figure 18.

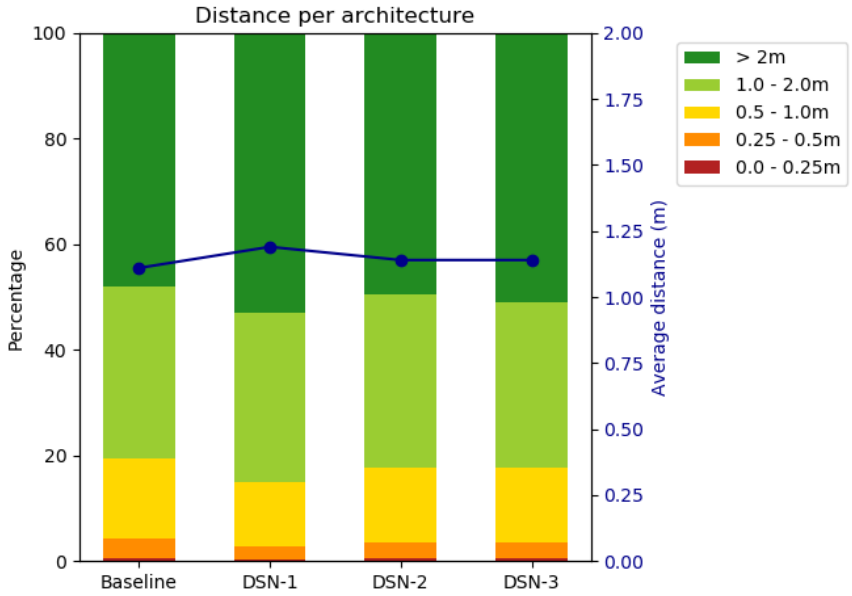


Figure 18: The distribution per architecture. The average distance is visualised by the blue line. All four architectures are similar in both aspects. Notably, all architectures rarely come in the $[0, 0.25]$ range.

From the figure can be concluded that the architectures are similar in both regards. The distribution varies little between the architectures and the average distance is at most 0.05 m different. Moreover, the architectures find themselves rarely in the first range of $[0, 0.25]$, meaning that they all keep their distance to the obstacles.

The amount of collisions per architecture is also denoted. As mentioned in Section 5.3, collisions where the obstacle is not in the FoV of the camera are ignored. These collisions could not have been avoided. Table 7 gives the amount of collisions per architecture.
The table indicates that the DSN-n is able to avoid the moving obstacles more in comparison to the baseline. Furthermore, all architectures collided twice with static obstacles during the runs in the room and hallway environments.

Notably, the vast majority of the collisions was with one type of object, namely a chair. All architectures collided with the chair, both when it was static and moving. It seems that the chair is not recognised by the network, due to the fact that the chair did not appear in the OAR nor OAH dataset. It shows that, although the high variety of objects in the datasets, the network struggles with never seen before objects.

Moreover, situations occurred where the robot would not be able to drive out of a corner. It

Collided with:	Static	Moving
Baseline	2	5
DSN-1	2	0
DSN-2	2	3
DSN-3	2	3

Table 7: The amount of collisions with static and moving obstacles. This is over both the room and hallway environments. As can be seen in the table, the amount of moving obstacle collisions is lower for the DSN-n in comparison to the baseline.

would become stuck there in a loop, similar to the loop described in Section 5.3. But the method to avoid these situations did not have any effect here, since the commands `Left` and `Right` were executed alternately (instead of `Backwards` and `Straight`). We have manually rotated the robot in these situations to continue the run.

Experiment 2.2: Moving obstacles

With this experiment is determined whether the DSN opts for the safer trajectory than the baseline. The second network (CIFAR-100 + OIDv6) from the OAR experiment is used. Likewise to the preceding experiment, collisions where the obstacle is outside of the FoV are ignored. In total, there are 8 configurations of the small environment with which the experiment is conducted. The amount of collisions per architecture can be seen in Table 8.

	Collisions
Baseline	1
DSN-1	0
DSN-2	0
DSN-3	1

Table 8: Amount of collisions per architecture during the runs in the small environments.

From the experiment came forth that the baseline and DSN-1 were similar in behaviour. If an obstacle came from the left, then both architectures would turn right to avoid it. But since the obstacle kept moving to the right, the architectures would have to avoid the obstacle again and thus turned right once more. This resulted in the robot turning very sharply towards the wall. For instance, this situation occurred in the environment of Figure 14a, here the robot would drive towards the wall. As for the DSN-2 and DSN-3, they also behaved this way, but significantly less frequent and severe. They would often opt for the more efficient and safer command, `Left` in the case of Figure 14a.

All in all, the DSN-2 and DSN-3 drove the more efficient trajectories in comparison to the baseline and DSN-1. The difference between DSN-1 and the other two is due to the time in between the prediction and current state of the environment. The movement of the obstacle is more often detected by the DSN-2 and DSN-3, resulting in the optimal command.

7 Conclusion

For the avoidance of moving obstacles, we propose the Differential Siamese Network (DSN) architecture using a monocular RGB camera as its sole input sensor. The architecture constructs a prediction of the current state of the environment with a previous image, assuming that every object is static. If the prediction and current image result in significantly different command vectors, then can be concluded that a moving obstacle is present in front of the robot.

A baseline is utilised to create the DSN with, and also serves as a benchmark for the experiments. This baseline consists of a convolutional neural network (CNN), which maps a command to an image. A few modifications are applied to the baseline in order to increase its performance. The CNN is deepened and a new command is added to the set of outputs, namely `Backwards`.

Multiple instances of the network are trained with combinations of two image-classification datasets; two networks are trained on one of the datasets while the third network is trained on both datasets. Next, all instances are trained on one of two obstacle avoidance datasets where every image is mapped to one command. One dataset contains images from a household, while the other contains images from a hallway.

The DSN is assessed against the baseline in two experiments. Both experiments are conducted in virtual environments to ensure that the architectures find themselves in identical situations. The first experiment verifies whether the performance of the DSN in an environment, with predominantly static obstacles, is deteriorated in comparison to the baseline. This is not the case, the amount of collisions and average distance to the obstacles are for all architectures nearly equivalent.

With the second experiment is the behaviour of the DSN, when approaching moving obstacles, assessed against the baseline. The architectures had to traverse in an environment with exclusively moving obstacles. From the experiment came forth that the DSN-1 shows similar behaviour to the baseline, while the DSN-2 and DSN-3 often opt for the more efficient trajectories around the moving obstacles.

For future research, we would like to verify whether the measured performance also applies to a real world environment. Since the architectures are trained on images acquired in the virtual environments, its ability to detect obstacles might suffer in real world environments.

Moreover, the current DSN has four commands (`Left`, `Right`, `Straight` and `Backwards`), an expansion of these commands can result in more refined trajectories and can aid in the avoidance of moving obstacles. For instance, the next direction of the robot can be split up into an angular and longitudinal component. It offers new challenges with regards to the DSN, which is specifically designed for the current four commands.

At last, a dynamic DSN might be interesting to investigate. Instead of taking a predetermined state to construct the prediction with, an earlier state can be taken under the circumstances that the prediction is still usable (not too much shifted). So, letting the selection of a preceding state be dependent on the commands that occurred between that state and the current state might result in an improvement in performance.

References

[ANB13] Pablo Fernández Alcantarilla, Jesús Nuevo, and Adrien Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. In Tilo Burghardt, Dima Damen, Walterio W. Mayol-Cuevas, and Majid Mirmehdi, editors, *British Machine Vision Conference, BMVC 2013, Bristol, UK, September 9-13, 2013*. BMVA Press, 2013.

[BBPW04] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In Tomás Pajdla and Jiri Matas, editors, *Computer Vision - ECCV 2004, 8th European Conference on Computer Vision, Prague, Czech Republic, May 11-14, 2004. Proceedings, Part IV*, volume 3024 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2004.

[CGSC13] Andrea Cherubini, Boris Grechanichenko, Fabien Spindler, and François Chaumette. Avoiding moving obstacles during visual navigation. In *2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013*, pages 3069–3074. IEEE, 2013.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255. IEEE Computer Society, 2009.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[KD12] Jeongdae Kim and Yongtae Do. Moving obstacle avoidance of a mobile robot using a single camera. *International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012)*, Daegu University, Major of Electronic Control Engineering, South Korea. 2012.

[KP19] Mohammad O. Khan and Gary B. Parker. Vision based indoor obstacle avoidance using a deep convolutional neural network. In Juan Julián Merelo Guervós, Jonathan Garibaldi, Alejandro Linares-Barranco, Kurosh Madani, and Kevin Warwick, editors, *Proceedings of the 11th International Joint Conference on Computational Intelligence, IJCCI 2019, Vienna, Austria, September 17-19, 2019*, pages 403–411. ScitePress, 2019.

[KRA+20] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *IJCV*, 2020.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, Department of Computer Science, Canada, 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.

[LRB+16] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *Fourth International Conference on 3D Vision, 3DV 2016, Stanford, CA, USA, October 25-28, 2016*, pages 239–248. IEEE Computer Society, 2016.

[MCVC16] Michele Mancini, Gabriele Costante, Paolo Valigi, and Thomas A. Ciarfuglia. Fast robust monocular depth estimation for obstacle detection with fully convolutional networks. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9-14, 2016*, pages 4296–4303. IEEE, 2016.

[MKR16] Iaroslav Melekhov, Juho Kannala, and Esa Rahtu. Siamese network features for image matching. In *23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016*, pages 378–383. IEEE, 2016.

[ML09] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In Alpesh Ranchordas and Helder Araújo, editors, *VISAPP 2009 - Proceedings of the Fourth International Conference on Computer Vision Theory and Applications, Lisboa, Portugal, February 5-8, 2009 - Volume 1*, pages 331–340. INSTICC Press, 2009.

[RSF13] E. Rohmer, S. P. N. Singh, and M. Freese. Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013. www.coppeliarobotics.com.

[WJQ+17] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6450–6458. IEEE Computer Society, 2017.

[XWMT17] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *CoRR*, abs/1706.09829, 2017.

A Open Images Dataset classes

Here are all 64 classes for the subset of the Open Images Dataset listed.

Alloy wheel	Aluminium can	Bag	Ball
Barrel	Battery	Bicycle	Book
Bottle	Bowl	Box	Cable
Calculator	Calendar	Camera	Candy
Cardboard	Chair	Clock	Closet
Computer	Computer keyboard	Computer monitor	Computer mouse
Cup	Desk	Door	Fan
Fence	Frying pan	Game controller	Games
Glass	Headphones	Helmet	Jewellery
Key	Lamp	Lantern	Laptop
Light bulb	Locker	Medicine	Money
Outdoor bench	Pen	Pencil	Pillow
Power plugs & sockets	Present	Radio	Refridgerator
Remote control	Scissors	Shoe	Shopping cart
Smartphone	Stairs	Umbrella	Vending machine
Washing machine	Watch	Wheelchair	Whiteboard

Table 9: All selected classes from the Open Images Dataset. Every class can be found in an household, office and/or urban environment.