Threshold Detection
(Bachelor's thesis)

Hans Alves,
Sjoerd de Jong*, Marco Wiering* and Barry Goeree†

August 30, 2010

Abstract

When an autonomous robot drives around in a house, it will encounter obstacles. Some of these obstacles, like furniture or walls, should be avoided to prevent the robot being damaged. Other obstacles however, have to be passed when a robot has to leave a room, for example a doorstep or a tapestry. In this research we tried to determine which obstacles can be passed, and which cannot. To make this classification, we used a Time Delay Neural Network, an Echo State Network and a Naive Bayes Classifier. The robot we used was equipped with four infra-red sensors, two of which were pointed downwards so even small obstacles could be detected. Our research shows that, of the algorithms tested by us, The Naive Bayes Classifier gives the best results, and that the four used sensors might not always give enough information to make a correct classification.

1 Introduction

When an autonomous robot is driving through a building, that robot will encounter obstacles. Most of these obstacles, for instance furniture, should be avoided to prevent the robot from being damaged. However, when the robot is moving from one room to another, it will usually encounter some kind of threshold in the doorway. This threshold should obviously not be avoided, the same goes for objects like high tapestries. For an autonomous robot it is crucial to be able to distinguish between objects it can pass, and those it should avoid.

For our study we will use a robot provided by Philips Robotics, the robot will be equipped with four infra-red (IR) sensors. Because two of the sensors are pointing downwards we can determine the height of an object as the robot approaches the object. Unless the object is higher than the placing of these IR sensors, in which case the robot will certainly not be able to traverse the obstacle.

The robot will be capable of passing objects with a height up to about 1.5 cm. However, this does not mean that it should pass every object lower than 1.5 cm, as the robot will be used as a vacuum cleaner. Objects that the robot should not drive over include for instance electrical wires and pencils or socks.

Because our robot is equipped with relatively simple sensors it is not easy to distinguish between for example a sock that should be avoided and a tapestry that can be crossed.

Research on this subject has been done by Philips. However, they have not yet found a solution to this problem, and have not published their results. Their research has nevertheless shown that at least some useful information can be drawn from the size of the object; objects that are higher than 1.5 cm should not be passed, but objects that are low but wide and/or deep have a higher chance of being passable than very small objects. As has been said however the heuristics that have been tried by Philips have not yielded the desired results. An other solution for this problem would be the usage of some kind of Machine-Learning to classify the objects our robot encounters.

In this study we will make a dataset in cooperation with Philips Robotics. This dataset will consist of data generated by a robot driving towards sev-

*University of Groningen, Department of Artificial Intelligence
†Philips Consumer Lifestyle

eral objects from different angles. We will try to classify the obstacles in this data set, using several Machine-Learning algorithms. The results that we get from this study will be used to determine which algorithm is best suited for this problem.

The machine-learning techniques that we will use for this study will be a time-delay neural-network, an echo state network and Naive Bayes Classification.

Time Delay Neural Networks (TDNN) are a kind of feed-forward neural-network that can learn using the back-propagation algorithm. TDNN were first introduced by Waibel et al. for the recognition of phonemes (Waibel, Hanazawa, Hinton, Shikano, and Lang, 1990). Later TDNNs have for example been used for Automatic Target Recognition of exo-atmospheric warheads (Lin, 2004).

Echo State Networks (ESN) (Jaeger, 2002) are a type of recurrent neural network that can predict time series (Jaeger and Haas, 2004), and have also been used for speech recognition (Skowronski and Harris, 2007) and localization problems (Broertjes and de Jong, 2009).

Naive Bayes Classifiers (NBC) use a simple probabilistic method for classification. This method calculates the probability that an object belongs to a certain class using the features of the object and the chance that an object in that class will have those features.

In the following sections we will more accurately describe these algorithms and how we will use them to classify obstacles.

2 Background

2.1 TDNN

Time Delay Neural Networks are basically feed-forward neural networks, with the difference that in a TDNN, a neuron bases its output value not only on the current input, but also in the input from the n previous steps (see Figure 1). This way a network can use the information it had a few steps back in time. In a normal network the activation of a neuron is computed by multiplying the inputs with the weights and passing the sum of these weighted inputs to a threshold- or sigmoid function. In a TDNN the same procedure is used, but instead of one weight per input neuron, each of the n delay steps has its own weight. Because a layer 'knows' about the past, a network of time-delay layers can compare inputs from different points in time.

Training a TDNN can be done with an algorithm similar to back-propagation (Rumelhart, Hinton, and Williams, 1986). In back-propagation the weights in the network are initialized randomly. Then, after a training sample has been presented to the network, for each neuron in the output level the desired output and a scaling factor are calculated. This scaling factor, to make the output match the desired output, is the local error. All weights then are adjusted to lower the local error. When the local error is calculated for all neurons at the output level, calculate which neurons of the previous level are to blame for this error, assuming that the neurons with higher weights have more influence on the

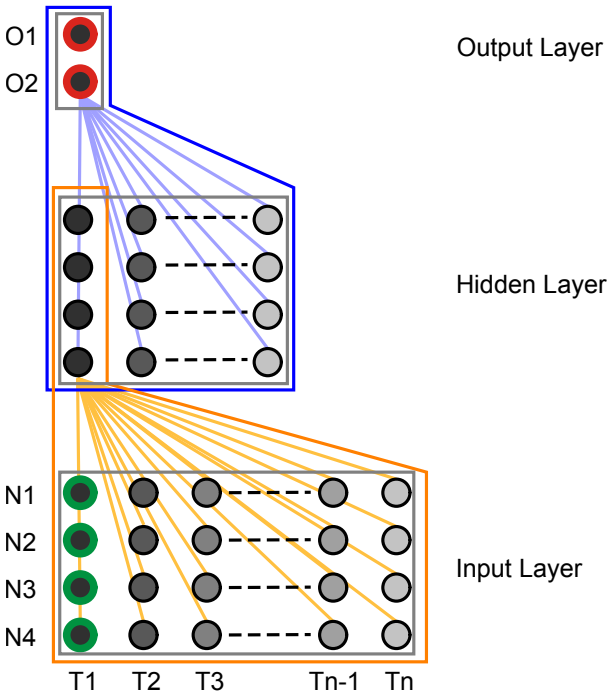

Figure 1: *A schematic view of a TDNN with four input neurons, a hidden layer of five neurons, and two output neurons. The neurons further to the right get their activation from the neuron to their left, thereby creating a time-delay. The yellow box contains the input neurons and their delay, and the neurons that are activated by the input layer. The blue box contains the neurons in the hidden layer and their delay, and the neurons that are activated by the hidden layer.*

error. Then, for the previous level the weights are adjusted according to this 'blame' factor. This is then repeated for all previous levels in the network (see Algorithm 1). The standard back-propagation is used to calculate the error for each input, so a separate error is computed for each time-delay from each input neuron.

2.2 ESN

Echo State Networks are a kind of neural networks, that together with liquid state networks form a group of algorithms used for reservoir computing. The theory of reservoir computing is that an input signal can be fed to the reservoir, in order to map the input to a higher dimension. The reservoir consists of a large recurrent neural network, that behaves like a dynamical system, and responds non-linearly to the input signal.

The neurons in the reservoir are connected to random other neurons in the reservoir. This randomness in the connections can result in circular connections, in the form of one neuron activating itself, or a series of neurons activating each other. These circular connections allow the network to 'remember' previous input signals and compare them with the current input. This makes ESNs fit to process time-series.

In an ESN a number of output neurons are connected to random neurons in the reservoir, which in turn have randomly generated feedback connections back to the reservoir (see Figure 2). These feedback connections make it possible to maintain activation in the reservoir without new input and allow the network to behave dependent on previous output, or handle temporal input patterns.

The ESN receives input from a number of neurons that have connections with random weights to the reservoir. The connections from the input neurons to the reservoir are fixed, and will not change during the training of the network.

There are a few other parameters that can influence the performance of an ESN. The first is the input connectivity (C_{in}), which defines the probability that an input neuron has a connection to each of the neurons in the reservoir. The second is the connectivity in the network (C), which defines the probability that two neurons in the reservoir are connected. And the output connectivity (C_{back}), which similarly describes the probability that an output neuron has a connection back to a certain neuron in the reservoir. These three parameters together define how many connections the network will have. The last parameter is α, and defines the spectral radius of the network; when the weights for the network are first generated, they are scaled to unit spectral radius, and multiplied by α. The spectral radius thus influences how fast the activation in the network dies down, and how much past inputs influence the current output.

Training an ESN is done in two steps, the first one is the sampling step. In the sampling step the input is provided to the network via the input neu-

Algorithm 1 Backpropagation

```
Initialize the weights randomly
repeat
  for each example e in the training set do
    O ⇐ Pass(network, e)
    T ⇐ teacher output for e
    E ⇐ T − O
    Update output weights to reduce E
    for every other layer l_i (in reverse order) do
      Compute 'blame' for all weights from l_{i−1}
      to the previous layer
      Update weights to reduce E
    end for
  end for
until Error is acceptably low
```

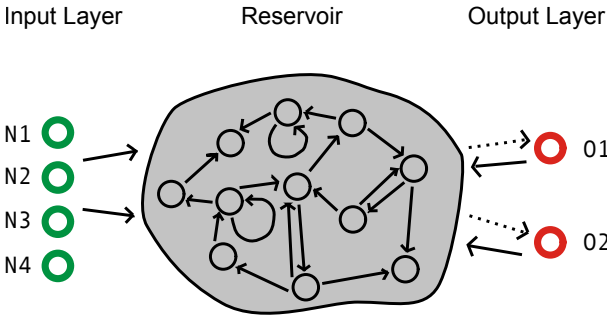

Figure 2: *A schematic depiction of an ESN. To the left is the input layer, with connections to the reservoir. And at the right an output neuron with fixed connections back to the reservoir, and changing connections from the reservoir. The solid arrows indicate the connections that are fixed from the start, the dotted arrows indicate connections that are learned.*

rons, and the desired (teacher) signal is written into the output units, this is often called *teacher forcing*. During the sampling step the internal states of the reservoir are saved in a state-collecting matrix M, the teacher outputs are saved in a matrix T. After all input data has been fed to the network, in the computation step the weights, that the connections to the output neurons should have, can be calculated by computing the matrix-inverse over M, and multiplying that with the teacher outputs: $W^{out} = M^{-1}T$. This way, the teacher time series is approximated as a linear combination of the internal activation time series. Finally, the computed weights are implemented in the network (Jaeger, 2002).

Besides these output weights, all other weights and connections are fixed from the moment they are generated. Because only the weights of the output neurons change during training, the training of an ESN becomes a simple linear regression task.

2.3 NBC

Naive Bayes Classification is a simple probabilistic method of classification. The method depends on the assumption that all features of the object it is classifying are independent of each other, given the class of that object. To classify an object, the following value is calculated for each class $p(C|F_1, F_2, \ldots, F_N)$ where C denotes the class and $F_1, \ldots, F_N$ are the features present in the object. Using Bayes' theorem (Bayes and Price, 1763) and the assumption that all features are independent of each other, give the class, this can be rewritten to:

$$p(C|F_1, \ldots, F_N) = p(C) \prod_{i=1}^{n} p(F_i|C)$$

The classification is then done by choosing the class with the highest chance.

$$C = \max(p(C_1|F_1, \ldots, F_N), \ldots, p(C_n|F_1, \ldots, F_N))$$

Learning is done by calculating $p(F_i|C)$ for all features in a large number of known examples.

3 Method

In this section we will describe how our experiments were done. In the first subsection we discuss the creation of our dataset, the subsequent subsections describe the algorithms we used. The last subsection describes the experiment itself.

3.1 Dataset

For our dataset we used a robot provided by Philips Robotics. The robot was equipped with several sensors, of which we used four IR sensors. The first two sensors were located 80 mm above floor level, and 108 mm on either side of the middle of the robot. These sensors were pointing outwards in an angle of 40 deg. The other two sensors were positioned 100 mm above floor level, and 66 mm on either side of the middle as can be seen in figure 3. These sensors were pointing to the middle in an angle of 33 deg and downwards in an angle of 65 deg. These last two sensors allow the robot to detect a threshold from a distance of $tan(65) * 100 \approx 214$ mm.

The dataset consisted of 52 runs of the robot driving towards an obstacle. The initial distance of the robot towards the obstacle was always 1 meter. For these runs we used 17 different objects, among others one aluminium doorstep, two wooden doorsteps, of which one was too high, a chair and a table and a few rugs. But also a number of other objects a robot could encounter in a living room, like a lamp post, an electrical wire and a potted plant. We let the robot drive at most objects from a number of different angles, as the robot should also be able to classify obstacles if it is not approaching that object in a straight line.

During each of these runs, we have recorded the

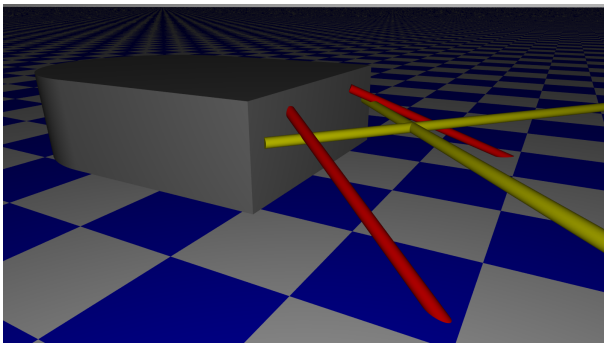

Figure 3: *A schematic view of the robot. In red the beams of the IR sensors that can detect a threshold, in yellow the beams of the sensors that can detect bigger objects.*

values returned of all four sensors ($N_{1:4}$), as well as the position of the robot. For every run we have recorded if the robot would be able to pass the obstacle or not. If the robot could not pass the obstacle, it should stop 5 cm in front of that obstacle, so after driving 95 cm.

To increase the amount of data we could use for training, we duplicated every run five times. For every set of five runs we acquired that way, we removed the data of the first 10 cm from the second run, the data of the first 20 cm from the third run and so on (see figure 4). By doing so we made sure the algorithms would learn to recognize an obstacle, and not just to stop after driving 95 cm. Consecutively we added noise to this data, so the training data would not be exactly the same as the testing data. We did this by computing the standard deviation of the sensor output during the first 50 cm of every run. Then we generated normally distributed random numbers with that same standard deviation, which were then added to the original data.

The resulting dataset however, was flawed in that there were far more time-steps where the robot could drive on, than time-steps where the robot had to stop. Because of this, we made a second dataset with only the last 70, 73, 76, 79 and 82 cm instead of the 100, 90, 80, 70 and 60 cm in the original dataset. This resulted in a somewhat more balanced dataset. Because the sensors were placed to see objects in front of the robot, we could not make the runs much shorter than this, even though the resulting dataset was still not really balanced.

This way we got two sets of 260 runs. To get a better estimate of the algorithms performance we used k-fold cross-validation for our experiments. One in five runs were used in a test set, the others in a training set. We made five test sets and training sets this way, so every run has been used in one test set and in four training sets, for both the short and the long dataset.

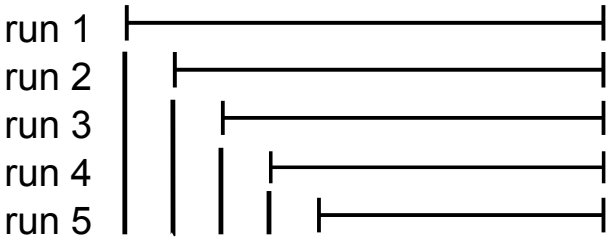

Figure 4: *After we duplicated a run five times we removed the first 10, 20, 30 and 40% from the consecutive duplicate runs.*

3.2 TDNN

Our TDNN had four input neurons $I_{1:4}$. Every time-step each of the sensor values $N_{1:4,t}$ were given to these input neurons. It had three hidden layers, each with five neurons. The hidden layers had time delays of fifty, thirty and twenty steps respectively. We gave our network two output neurons $O_{1:2}$, one (O_1) should be active when the robot could pass an object, the other (O_2) if the robot could not pass the object. The time delay of fifty steps in the first hidden layer was not enough for the network to evaluate a complete object, a doorstep with a depth of approximately three centimetres took about a hundred steps, however, larger time delays give memory problems in Matlab.

For our experiments we used the TDNN implementation provided by the Matlab neural-networks toolbox.

3.3 ESN

Like the TDNN our ESN had four input neurons ($I_{1:4}$) and two output neurons ($O_{1:2}$). To be able to make a fair comparison with the TDNN we used a reservoir of fifteen neurons (the TDNN had three layers of five neurons). We runned several tests to find the optimal values for the connectivity of the

Figure 5: *The making of our data set. Here the robot is driving towards a rug that is low enough, but the robot could get stuck in the long hairs.*

neurons in the reservoir (C), of the input neurons to the reservoir (C_{in}) and of the output neurons back to the reservoir (C_{Back}).

Also like the TDNN, for the ESN the output of the sensors $N_{1:4,t}$ were given to the input neurons $I_{1:4}$ every time-step. O_1 should become active if the robot could drive on, whereas O_2 should activate if the object could not be passed.

Unlike the TDNN, which used backpropagation, the ESN did not have iterative learning. After all training data had been fed to the network the output weights were calculated at once. This raised the problem that we could not use several data-runs. A possible solution would be to just append all runs to make one very long run. Here the problem would be that the transition between the runs would not be realistic, an obstacle would suddenly disappear at the start of the next run. Another solution could be to train for each run separately and use the average of the computed output weights for testing. We have tried both options to see which would yield the best results.

To find out which connectivity values worked best, we tried combinations of different connectivity probabilities. We tested connectivity probabilities of 0.1, 0.5 and 0.9 for each of C, C_{in} and C_{out} resulting to 27 tests in total for each of the two training methods described in the previous paragraph.

In our experiments we used a modified version of the Matlab implementation of an ESN by Broertjes and de Jong (2009).

3.4 NBC

In our experiments we used a number of features: the current sensor input $N_{1:4,t}$, the maximum $max(N_{1:4,t}, N_{1:4,t-1}, \ldots, N_{1:4,t-100})$ and minimum $min(N_{1:4,t}, N_{1:4,t-1}, \ldots, N_{1:4,t-100})$ values for each sensor over the last 100 steps. We also used the average value over the last 100 steps $(N_{1:4,t} + N_{1:4,t-1} + \cdots + N_{1:4,t-100})/100$ and the number of steps since the last time the sensor returned its default value. The default value being the average of each sensor when the robot does not see any object.

We used the Naive Bayes implementation in the Matlab statistics toolbox.

3.5 Experiment

To compare these algorithms we compared the average percentage of correctly classified time-steps. With the making of the dataset we recorded for each run if the object could be passed or not, and if the robot could not pass the obstacle, it should stop on a distance of 5 cm from the obstacle. To determine the percentage of correctly classified time-steps we could just compare the output value of the algorithm to the recorded value. If the algorithm can not make a decision for a certain time-step it was always considered to be incorrect. Since Naive Bayes Classification is a deterministic algorithm it only had to be runned once, the other algorithms were trained multiple times for each train and test set, in order to obtain a reliable measure of correctness.

4 Results

Below we describe the results we got by executing the experiments as described in the previous section. Like in the previous sections we first discuss the Time Delay Neural Network, then the Echo State Network and finally the Naive Bayes Classifier. In the second section we also describe the optimal connectivity values for the ESN.

4.1 TDNN

We have tested the TDNN with both the long and the short dataset. The confusion matrix with the results of the long dataset can be seen in Table 1, the confusion matrix with the results of the short dataset are in Table 2. The columns of the confusion matrix show how the classifications should have been made, the rows show the actual results. Cells with the same classification in the row and column (positive, positive and negative, negative) are correct classifications, the others are incorrect.

In Table 1 it can be seen that although the percentage of correct classifications is quite high (92.02%), no real learning effect has occurred, as the algorithm always answered positive, meaning that the robot could drive on.

In Table 2 we can see that the same is true for the shorter dataset. Because the short dataset is a little less unbalanced the actual percentage of correct classifications has gone down (to 76.70%), however

Table 1: *TDNN Confusion Matrix with long dataset*

		Desired Value	
		positive	negative
Classification	positive	92.02%	7.98%
	negative	0%	0%
	unknown	0%	0%

Table 2: *TDNN Confusion Matrix with short dataset*

		Desired Value	
		positive	negative
Classification	positive	76.70%	23.30%
	negative	0%	0%
	unknown	0%	0%

the algorithm still classified every situation as save to drive on.

4.2 ESN

The ESN has been tested many more times than the TDNN. Not only have we run two different versions of the ESN, but we also tried to determine the optimal connectivity parameters. As expected there were big differences in performance between the different connectivity settings.

The optimal connectivity parameter values for the first ESN (with the runs appended) can be seen in Table 3, the confusion matrix for the run with the best performance is in Table 4. Tables 5 and 6 show the same for the second ESN (that was trained for all runs separately). It can be seen that the second ESN generated better results than the first one, as the top five from Table 5 are all better than the best from Table 3. Although the best run (Table 6) seemed to classify most cases as safe to drive on, the ESN seemed to be a little less troubled by the unbalancedness of the dataset than the TDNN.

The same tests have of course been done with the shorter dataset, and the results of the first ESN can be seen in Table 7. The confusion matrix of the best run can be seen in Table 8. Although the short dataset is more balanced than the long one, the performance of the first ESN has not improved with the short dataset. The performance of the second ESN is shown in Table 9. The confusion matrix of the best run is in Table 10. The second ESN too had better results with the long dataset than with the short, but with the short dataset still performed better than the first ESN.

Table 3: *first ESN performance for different parameter settings with the long dataset*

run	C_{in}	C	C_{out}	correct %	incorrect %
2	0.1	0.1	0.5	59.61%	40.39%
20	0.9	0.1	0.5	57.38%	42.62%
14	0.5	0.5	0.5	55.32%	44.68%
21	0.9	0.1	0.9	55.37%	44.63%
9	0.1	0.9	0.9	54.46%	45.54%
3	0.1	0.1	0.9	54.44%	45.56%
22	0.9	0.5	0.1	52.63%	47.37%
16	0.5	0.9	0.1	51.87%	48.13%
1	0.1	0.1	0.1	44.88%	55.12%
8	0.1	0.9	0.5	44.32%	55.68%
18	0.5	0.9	0.9	42.79%	57.21%
5	0.1	0.5	0.5	42.57%	57.43%
26	0.9	0.9	0.5	41.46%	58.54%
27	0.9	0.9	0.9	40.86%	59.14%
7	0.1	0.9	0.1	37.97%	62.03%
23	0.9	0.5	0.5	36.82%	63.18%
12	0.5	0.1	0.9	30.18%	69.82%
15	0.5	0.5	0.9	27.85%	72.15%
19	0.9	0.1	0.1	20.62%	79.38%
10	0.5	0.1	0.1	17.01%	82.99%
25	0.9	0.9	0.1	16.80%	83.20%
11	0.5	0.1	0.5	14.84%	85.16%
13	0.5	0.5	0.1	14.80%	85.20%
17	0.5	0.9	0.5	7.94%	92.06%
6	0.1	0.5	0.9	7.04%	92.96%
24	0.9	0.5	0.9	6.61%	93.39%
4	0.1	0.5	0.1	6.13%	93.87%

4.3 NBC

With the NBC we have only run two tests, the confusion matrices for the runs with the long and short dataset are shown respectively in Tables 11 and 12. As the percentage of correct classifications with the NBC is $89,65 + 5.26 = 94.91\%$ and $69.77 + 13.99 = 83.76\%$ for the long and the short dataset the NBC has generated the best results of our experiments. A possible reason for this is discussed in the next subsection.

Table 4: *Confusion Matrix of the run with the best connectivity settings for the first ESN with the long dataset*

		Desired Value	
		positive	negative
Classification	positive	58.02%	4.35%
	negative	13.76%	1.60%
	unknown	20.24%	2.03%

Table 5: *second ESN performance for different parameter settings with the long dataset*

run	C_{in}	C	C_{out}	correct %	incorrect %
14	0.5	0.5	0.5	83.60%	16.40%
25	0.9	0.9	0.1	67.45%	32.55%
20	0.9	0.1	0.5	65.11%	34.89%
26	0.9	0.9	0.5	61.21%	38.79%
23	0.9	0.5	0.5	60.28%	39.72%
27	0.9	0.9	0.9	59.93%	40.07%
16	0.5	0.9	0.1	59.51%	40.49%
1	0.1	0.1	0.1	53.29%	46.71%
24	0.9	0.5	0.9	50.62%	49.38%
9	0.1	0.9	0.9	50.20%	49.80%
13	0.5	0.5	0.1	48.89%	51.11%
10	0.5	0.1	0.1	44.07%	55.93%
6	0.1	0.5	0.9	38.77%	61.23%
18	0.5	0.9	0.9	37.80%	62.20%
22	0.9	0.5	0.1	35.67%	64.33%
4	0.1	0.5	0.1	34.95%	65.05%
21	0.9	0.1	0.9	33.71%	66.29%
8	0.1	0.9	0.5	31.82%	68.18%
5	0.1	0.5	0.5	27.37%	72.63%
12	0.5	0.1	0.9	21.13%	78.87%
7	0.1	0.9	0.1	21.07%	78.93%
19	0.9	0.1	0.1	23.27%	76.73%
15	0.5	0.5	0.9	19.16%	80.84%
11	0.5	0.1	0.5	17.72%	82.28%
2	0.1	0.1	0.5	4.74%	95.26%
3	0.1	0.1	0.9	4.71%	95.29%
17	0.5	0.9	0.5	0.39%	99.61%

Table 6: *Confusion Matrix of the run with the best connectivity settings for the second ESN with the long dataset*

		Desired Value	
		positive	negative
Classification	positive	83.55%	6.14%
	negative	0.46%	0.05%
	unknown	8.00%	1.80%

Table 7: *first ESN performance for different parameter settings with the short dataset*

run	C_{in}	C	C_{out}	correct %	incorrect %
21	0.9	0.1	0.9	51.66%	48.34%
22	0.9	0.5	0.1	49.99%	50.01%
3	0.1	0.1	0.9	47.49%	52.51%
14	0.5	0.5	0.5	46.67%	53.33%
2	0.1	0.1	0.5	46.45%	53.55%
20	0.9	0.1	0.5	44.95%	55.05%
26	0.9	0.9	0.5	44.11%	55.89%
9	0.1	0.9	0.9	43.82%	56.18%
18	0.5	0.9	0.9	43.07%	56.93%
16	0.5	0.9	0.1	42.72%	57.28%
5	0.1	0.5	0.5	42.00%	58.00%
8	0.1	0.9	0.5	42.00%	58.00%
1	0.1	0.1	0.1	40.10%	59.90%
15	0.5	0.5	0.9	37.32%	62.68%
6	0.1	0.5	0.9	36.88%	63.12%
27	0.9	0.9	0.9	36.34%	63.66%
7	0.1	0.9	0.1	34.79%	65.21%
17	0.5	0.9	0.5	30.89%	69.11%
19	0.9	0.1	0.1	26.29%	73.71%
24	0.9	0.5	0.9	20.76%	79.24%
25	0.9	0.9	0.1	19.84%	80.16%
23	0.9	0.5	0.5	19.34%	80.66%
11	0.5	0.1	0.5	18.40%	81.60%
12	0.5	0.1	0.9	16.14%	83.86%
13	0.5	0.5	0.1	16.04%	83.96%
4	0.1	0.5	0.1	15.00%	85.00%
10	0.5	0.1	0.1	9.03%	90.97%

Table 8: *Confusion Matrix of the run with the best connectivity settings for the first ESN with the short dataset*

		Desired Value	
		positive	negative
Classification	positive	42.99%	12.66%
	negative	31.63%	8.67%
	unknown	2.08%	1.98%

Table 9: *second ESN performance for different parameter settings with the short dataset*

run	C_{in}	C	C_{out}	correct %	incorrect %
14	0.5	0.5	0.5	60.00%	40.00%
26	0.9	0.9	0.5	59.52%	40.48%
22	0.9	0.5	0.1	57.25%	42.75%
8	0.1	0.9	0.5	57.06%	42.94%
18	0.5	0.9	0.9	53.61%	46.39%
20	0.9	0.1	0.5	51.88%	48.12%
23	0.9	0.5	0.5	49.98%	50.02%
27	0.9	0.9	0.9	48.58%	51.42%
5	0.1	0.5	0.5	45.74%	54.26%
2	0.1	0.1	0.5	46.50%	53.50%
1	0.1	0.1	0.1	46.20%	53.80%
21	0.9	0.1	0.9	44.19%	55.81%
11	0.5	0.1	0.5	41.00%	59.00%
3	0.1	0.1	0.9	40.36%	59.64%
7	0.1	0.9	0.1	39.74%	60.26%
13	0.5	0.5	0.1	38.39%	61.61%
9	0.1	0.9	0.9	35.99%	64.01%
25	0.9	0.9	0.1	35.85%	64.15%
17	0.5	0.9	0.5	35.63%	64.37%
19	0.9	0.1	0.1	33.08%	66.92%
12	0.5	0.1	0.9	32.15%	67.85%
10	0.5	0.1	0.1	31.11%	68.89%
15	0.5	0.5	0.9	30.97%	69.03%
6	0.1	0.5	0.9	29.70%	70.30%
24	0.9	0.5	0.9	28.39%	71.61%
16	0.5	0.9	0.1	25.80%	74.20%
4	0.1	0.5	0.1	6.12%	93.88%

Table 10: *Confusion Matrix of the run with the best connectivity settings for the second ESN with the short dataset*

		Desired Value	
		positive	negative
Classification	positive	59.96%	19.52%
	negative	0.12%	0.04%
	unknown	16.62%	3.74%

Table 11: *NBC Confusion Matrix with the long dataset*

		Desired Value	
		positive	negative
Classification	positive	89.65%	2.52%
	negative	1.04%	5.26%
	unknown	1.33%	0.20%

Table 12: *NBC Confusion Matrix with the short dataset*

		Desired Value	
		positive	negative
Classification	positive	69.77%	8.41%
	negative	1.40%	13.99%
	unknown	5.52%	0.91%

4.4 Discussion

Of all algorithms tested, the best results were, quite unexpected, by the NBC with the long dataset. The reason that the best results were generated with the NBC is probably the unbalancedness of our dataset. Contrasting the neural networks, the learning process of a NBC is not dependent on a balanced dataset as the number of occurrences of a certain class does not change the probability that an object belongs to that class given its features. When a neural network, however, is presented more instances of a certain class during its training, the chance that the network would recognize an object as belonging to that class rises.

Because a NBC does not need a balanced dataset it would do better with a longer dataset, as it has seen more examples and thus has a better estimate of $p(C|F_1, \ldots, F_N)$.

5 Discussion

In this section we summarize our experiment and its results, discuss the practical value of our experiments and give some suggestions for further research.

5.1 Threshold Detection

In our experiments we tested several machine learning algorithms on their ability to determine if a robot could keep driving, or had to stop, given the data from its sensors. The tested algorithms were a Time Delay Neural Network, two variants of an Echo State Network and Naive Bayes Classification. As described in Section 4 the TDNN got a high percentage of correct classifications, but this was mostly due to the unbalancedness of our dataset, as the network classified all points as safe to drive on. The ESN seemed to deal much better

with the unbalancedness of the dataset even though it didn't produce more correct classifications. Of the two ESN's we tested, the one where we trained for each run separately and used the average of the computed weights produced better results than the one with all runs appended and trained at once. The overall best results were from the NBC, this is probably caused by the fact that NBCs are not negatively affected by an unbalanced dataset.

5.2 Practical Value

In its current form, the TDNN would not be of any practical value to solve the problem of threshold classification, it might however perform better if it is trained with a more balanced dataset. The ESN too does not perform good enough to be of practical value yet. The NBC, however, does have the best performance in our test and when trained with more data it would probably improve.

5.3 Further Research

Further research could in the first place of course be done with a more balanced dataset, however, that would require some changes in the sensors of the robot. The reason for this is that if sensors are looking forward, there will almost always be more data-points where the robot should drive on, because the robot should stop in front of the obstacle but not as soon as the obstacle is detected. An other option for additional research would be to use a different time-delay. The Naive Bayes Classification algorithm performed best in our tests, and it might be improved by using more features from the available data. Finally, it would be interesting to see how the algorithms would do if they had more information from more or different sensors.

References

Thomas Bayes and Richard Price. An essay towards solving a problem in the doctrine of chance. by the late rev. mr. bayes, communicated by mr. price, in a letter to john canton, m. a. and f. r. s. *Philosophical Transactions of the Royal Society of London*, 53:370–418, 1763.

Michaja Broertjes and Sjoerd de Jong. Topological localization with omnidirectional images using an echo state network. Bachelor's thesis, University of Groningen, The Netherlands, 2009.

H. Jaeger. Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the "echo state network" approach. GMD- Forschungszentrum Informationstechnik, 2002.

Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304 (5667):78 – 80, April 2004.

Daw-Tung Lin. Target components discrimination using adaptive time-delay neural network. *Journal of Information Science and Engineering*, 20: 959–980, 2004.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, chapter 8, pages 318–362. M.I.T. Press, 1986.

Mark D. Skowronski and John G. Harris. Automatic speech recognition using a predictive echo state network classifier. *Neural Networks*, 20: 414–423, 2007.

A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time-delay neural networks. *Neural Networks*, 3:23–43, 1990.