CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Masterstudiengang Informatik

Masterarbeit

Probabilistic modeling of human telecontrolled behaviour for sumo robots

vorgelegt von

Michael Möbes

Gutachter:

Prof. Dr. Claus Möbus
MSc. Mark Eilers

Oldenburg, February 10, 2017

Contents

1 Introduction **3**

2 Basics **5**
2.1 Robot sumo 5
2.2 leJOS 5
2.3 Lego robot 6
2.4 Bayesian network 6

3 Sensors **9**
3.1 Lego and third party sensors 9
3.2 Webcam plus OpenCV 11

4 Parameter training **15**
4.1 Recording of training data 15
4.2 Results of the training 16
4.3 Parameter estimation 16

5 Autonomous sumo robot **17**
5.1 Player control and their observations 17
5.2 Discretization 18
5.3 Model 20

6 Implementation **23**
6.1 leJOS and sumo robots 23
6.2 Tracking of sumo robots 23
6.3 Extracting sensor information 24
6.4 Collecting of training data 26
6.5 Programming the Bayesian network 27

7 Evaluation **31**

8 Conclusion **33**

Images **35**

Literature **37**

Index **39**

Abstract

This thesis preoccupies itself with reactive autonomous sumo robots using Lego NXT bricks. The robots are trained using training data gathered by recording human players playing sumo against each other. A Bayesian network is used to make inference about what action the robot should do most likely. A system is built using OpenCV to track the robots and a own Bayesian network is programmed, capable of doing inference and running on a Lego NXT brick. In the end, the trained robot using training data of a stronger human player is able to slightly win over a robot using the training data of a less good human player.

1 Introduction

This thesis tries to teach Lego NXT robots how to fight robot sumo like human players. To achieve this, it starts off explaining some of the basics needed to program such a robot and mentions used hardware and software. Following that the sensors needed are presented and two different approaches shown. Next comes the training and recording of human players to gather data needed for the robots. This is followed by analyzing human robot sumo and finding out important aspects about it that need to be implemented into the robots as well. A mathematical model is introduced needed for calculations and then it is explained how the whole system is implemented. Lastly a short evaluation follows and is followed up by a conclusion and outlook.

2 Basics

2.1 Robot sumo

Robot sumo is based off of real sumo. There is a ring, there are two combatants and there are starting positions. The goal in real sumo is to win a match, consisting out of multiple rounds, by either pushing the enemy out of the ring or making him touch the ring floor with any part of his body other than the bottom of his feet. A judge decides whether either of these conditions are met. The judge also starts the round with a signal, leading both sumo wrestlers to do their initial charge at the enemy, a very important part of the round [sum]. The rules for robot sumo are very similar, but can vary slightly from tournament to tournament. The two robots face off in a sumo ring on their starting positions. Robots can either be remote controlled or autonomous. A human referee gives a start signal and also decides when the round is over. Similar to real sumo, touching any part outside of the ring will earn a loss. Touching the sumo ring with anything other the equivalent of a sumo wrestler's foot, the wheel, does not necessarily mean a loss though, depending on the tournament. One of these tournaments is RoboGames. It is a big and popular event, encompassing many other robot competitions. As such, it is used as a guideline for its rules. Robot sumo takes place for different types of robots including humanoid robots, Lego robots and so called micro or nano robots, each with their own restrictions for ring diameter and robot size. For Lego robots, they sumo ring has a height of 5cm and a diameter of 77cm. The robot has no restrictions for its height, but is not allowed to exceed a width of 15cm, a length of 15cm and a weight of 1kg. A round is won if the enemy robot touches the ground outside of the sumo ring. This can easily be determined, because the sumo ring itself is 5cm high, so the robots drop off the ledge when exceeding the ring. A robot falling over does not mean a loss, but instead the round is continued [rob]. These rules are also used for this thesis.

2.2 leJOS

leJOS is an alternative firmware for Lego Mindstorms EV3, NXT and RCX bricks. It replaces the original Lego firmware and allows the use of programming environments other than the one supplied by Lego. The official Lego programming environment uses an icon-based graphical interface as seen in figure 2.1. The created programs resemble flow charts. It works by dragging predefined blocks and putting them together, creating a program. Each block can represent e.g. a sensor, a motor, conditional statements or even loops. Each of these blocks can have different attributes like motor speed on the motor block. While this might work well for simple programs, it can be restrictive when attempting to create more complex programs, especially when hardware that is not supported by the Lego software is used. There are many alternative firmwares for the Lego bricks that are actively being developed, although not all support all different Lego bricks. Amongst these the ROBOTC, EV3dev, LabVIEW and MonoBrick. The reason leJOS is used is that it uses Java as programming language, is compatible with the Lego NXT brick and is well developed and maintained with an active community. leJOS is a shrunk Java Virtual Machine (JVM) that fits onto the Lego brick. As such it is missing some classes, but still offers features like preemptive threads, arrays, recursion, synchronization, exceptions, Java types and most of the *java.lang, java.util* and *java.io* classes [leJ].

leJOS comes with a few tools, including a debugger for exceptions thrown on the brick, making it easier to handle errors. Additionally it comes with a plugin for the integrated development environment (IDE) Eclipse, making development easier. Programs created in Eclipse can be directly

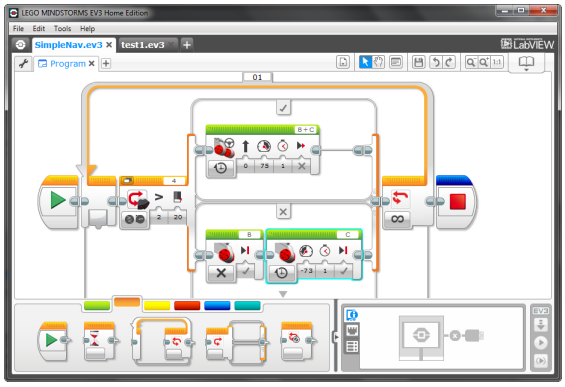

Figure 2.1: Lego Mindstorms programing interface

uploaded to the Lego NXT brick and data can be sent back and forth between the Lego NXT brick and the PC.

2.3 Lego robot

The Lego robot is built out of the Lego NXT set with a Lego NXT brick. The construction design for the robot is based on the Castor Bot [nxt]. The robot has two motorized wheels, able to spin forward and backward independently. A carrying wheel keeps the bot up from the ground. A battering ram like Lego structure is added to the front, acting as primary pushing tool. Additionally a block is added on top to hold an element used for tracking in place. This setup allows for back and forward movements, as well as rotating on the spot and movement in curves.

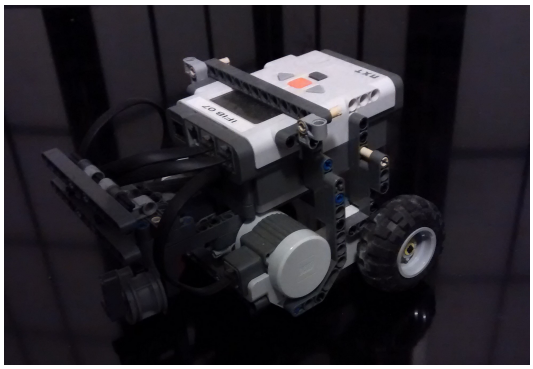

Figure 2.2: Lego NXT Sumo Robot

2.4 Bayesian network

A Bayesian network is a probabilistic graphical model that was coined by Judea Pearl in 1985. It combines classical AI and neural nets, and it was able to break through a lot of problems of systems of the 1960s and 1970s. It became its own study field and has since thrived in further AI research on uncertain reasoning and expert systems (cf. [SR95], 2010, p. 26). Compared to the joint distribution,

it is much more compact by having smaller (conditional) probability tables for each variable, instead of having one big table that rapidly increases in size with increasing number of variables and variable values/features.

A Bayesian network distinguishes itself from models like the Markov networks by being a directed graph. These graphs consist out of nodes, which are connected via edges. For directed graphs edges have a source and a target ([DK09], 2009, p. 5). The full specification as described in [SR95] (2010, p. 511) are as follows.

1. Each node corresponds to a random variable, which may be discrete or continuous.
2. A set of directed links or arrows connects pairs of nodes. If there is an arrow from node *X* to node *Y*, *X* is said to be a parent of *Y*. The graph has no directed cycles (and hence is a directed acyclic graph, or directed acyclic graph.
3. Each node X_i has a conditional probability distribution $\mathbf{P}(X_i|Parents(X_i))$ that quantifies the effect of the parents on the node.

An edge between two nodes means that there is a relation between them. The arrow suggests that one node has a direct influence over the other node, with the parent node (the source of the edge) being the cause of an effect (the target of the edge). These relationships specify conditional independences of the network. Having laid out all the nodes and edges it is only needed to specify a conditional probability distribution for each variable, given its parents ([SR95], 2010, p. 511). A small and often used example seen in figure 2.3 shall illustrate this further. The letters B, E, A, J and M in the conditional probability tables (CPTs) are abbreviations for the variables.

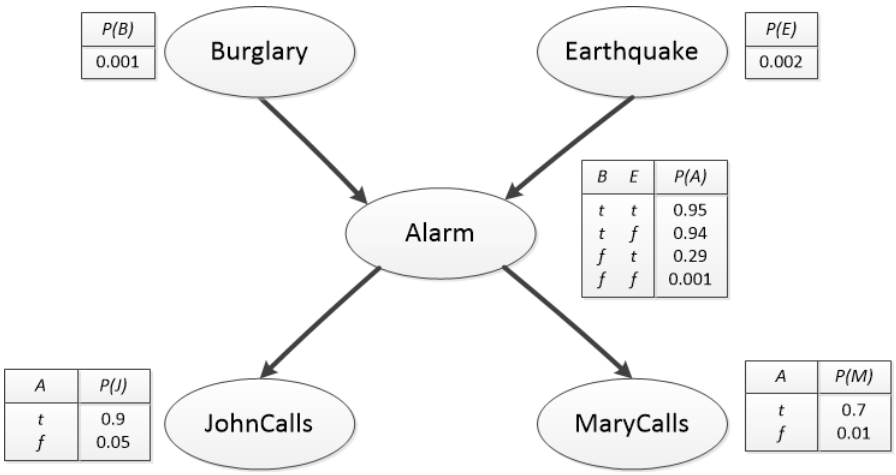

Figure 2.3: A Bayesian network of an alarm system.

In this example there is a house with an alarm system that responds to burglaries with high fidelity, but sometimes is also triggered by earthquakes. Additionally the neighbors John and Mary promise to call whenever they hear the alarm go off, but John sometimes confuses his telephone with the alarm and Mary likes to listen to loud music, missing the alarm completely. The variables *Burglary* and *Earthquake* directly influence the variable *Alarm*, since they are the causes of the effect and thus have an impact on the alarms probability. John or Mary calling on the other hand only depends on the alarm

and nothing else. Based on the topology of the network John and Mary never try to consult each other and also never witness a burglary or notice an earthquake. The conditional probability tables next to the variables show conditional probability for a conditioning case, meaning possible combination of values with the parent nodes. Each row must sum up to 1. In this example the variables are Boolean. There either is a burglary or there is not. Mary either calls or she doesn't. This means that given the conditional probability *p* of Mary calling, the conditional probability of Mary not calling is simply 1-*p*. Variables are not limited to Boolean though and can have many more characteristics. As mentioned in the example, John might confuse the telephone for the alarm. This condition can not explicitly be found in the network though. What happens is that this is summarized in the uncertainty with the link from *Alarm* to *JohnCalls*. Aside from John confusing the telephone, there are many more other factors that could lead to John not calling, but it might not be feasible or very hard to obtain any information. The same applies for Mary listening to too loud music and other elements. The network works with an infinite set of circumstances in which the alarm might fail to go off and even though it might not have enough all information, it can work approximatively. If a more accurate approximation is needed, more relevant information must be added ([SR95], 2010, p. 511-513).

Next we might want to calculate the posterior probability distribution for a set of query variables, given an observed event represented by a set of evidence variables with assigned values. In the following, query variables will be denoted as *X*, a set of evidence variables $E_1, ..., E_m$ as **E**, a particular observed event as **e**, and **Y** the non-evidence or non-query variables $Y_1, ..., Y_l$ also called hidden variables. The complete set of variables is **X** = $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$ and a typical query asking for the posterior probability distribution is $\mathbf{P}(X|\mathbf{e})$ ([SR95], 2010, p. 522-523). A query for the example Bayesian network in figure 2.3 could look like $\mathbf{P}(Earthquake|JohnCalls = true, MaryCalls = false)$

A query $\mathbf{P}(X|\mathbf{e})$ can be answered like $\mathbf{P}(X|\mathbf{e}) = \alpha\mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y})$ ([SR95], 2010, p. 493). α is a normalization constant for the distribution ensuring that it adds up to 1. A Bayesian network gives a complete representation of the full joint distribution. It can be written as products of conditional probabilities from the network. If you take the example $\mathbf{P}(Burglary|JohnCalls = true, MaryCalls = true)$, the hidden variables are the remaining variables *Earthquake* and *Alarm*. Applying the equation for $\mathbf{P}(X|\mathbf{e})$, we get

$$\mathbf{P}(B|j, m) = \alpha\mathbf{P}(B, j, m) = \alpha \sum_e \sum_a \mathbf{P}(B, j, m, e, a)$$

This can then be expressed in terms of CPT entries. For the case of $Burglary = true$ it would look like

$$\mathbf{P}(b|j, m) = \alpha \sum_e \sum_a P(b)P(e)P(a|b, e)P(j|a)P(m|a).$$

A little improvement computing-wise can be achieved by pulling out terms that don't have any part that needs to be summed in front of the summation. In this case, the term $P(b)$ is a constant and thus can be moved out of both summations and the term $P(e)$ only needs to be summed over *e* and can thus be moved out of $\sum_a$ ([SR95], 2010, p. 523-524).

3 Sensors

Two different approaches of sensors have been set up and tried. Initially official Lego NTX sensors and one third party sensor were used to receive different information relevant for the sumo robot. Later on, a system based on a webcam was used instead. In the following, both approaches will be explained as well as their advantages.

3.1 Lego and third party sensors

Lego offers official Lego NXT sensors, providing means to measure a variety of things. Amongst these the distance to objects, touch, sound, light intensity and rotational movement. Aside from official sensors, third party sensors are available too, offering even more capabilities. The sensors used will be shortly introduced.

Lego NXT Ultrasonic Sensor

The Lego NXT ultrasonic sensor offers a way to determine the distance to the closest object straight ahead. It is able to measure distances ranging from 0 to 255 cm with a precision of +/- 3 cm [Leg].

HiTechnic Lego NXT Gyroscope Sensor

The third party NXT gyroscope sensor is able to detect rotation and returns a value for the degrees per second of rotation in one dimension. The rotation rate can be read up to 300 times per second [HiT].

Lego NXT Light Sensor

The light sensor is capable of measuring light intensity, both in the room and on surfaces. This means it can distinguish between different shades of grey, from black to white [Leg].

Mindsensors Sumo Eyes

The Mindsensors Sumo Eyes are a third party sensor for Lego NXT [Min]. They offer a unique feature not offered by official Lego sensors. The Mindsensors Sumo Eyes is an obstacle sensor. It can detect objects in an angle of 60°(see figure 3.1) by using infrared. The detection are consists out of a total of six areas. It's divided into a short (15 cm) and a long range (30 cm), each respectively have a left, front and right area. If an object is within one of the areas, the sensor returns a value indicating one of the six areas.

Sensor setup

The initial setup consists out of one ultrasonic sensor, one light sensor, one gyroscope and one Mindsensors SumoEyes sensor. The ultrasonic sensor is used to determine the distance to the other robot.

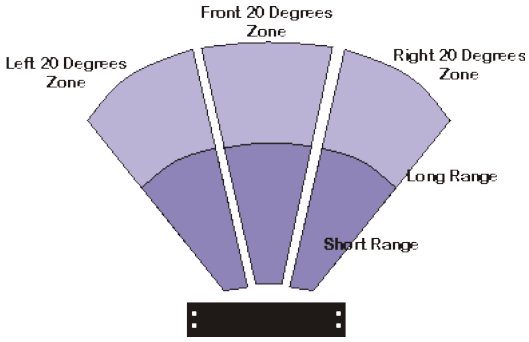

Figure 3.1: Mindsensors Sumo Eyes

Source: http://www.mindsensors.com/ev3-and-nxt/28-dual-range-triple-zone-infrared-obstacle-detector-for-nxt-or-ev3

The light sensor is used to determine whether the robot is stepping on the ring border or not and possibly be about to leave the ring. The gyroscope is used to measure how much the robot has turned away from its starting point. The robots exchange this information with each other to know which way they are facing relative to the other robot. The Mindsensors SumoEyes sensor is used to determine the general position of the other robot in front of him. With its up to 30 cm range it covers about half the diameter of the sumo ring. These sensors in conjunction can offer information about the distance to the other robot when in sight, the other robots position and which way it is facing using the Mindsensors SumoEyes and the gyroscope sensors and roughly the position of the other robot.

Performance

The Lego NXT ultrasonic sensor as well as the Lego NXT light sensor perform well. The measured distance of the ultrasonic sensor is always accurate and mostly fluctuated by 1 cm (maybe add measurement test). The light sensor only needs to distinguish between black and white in the used scenario, which is far less than what the sensor is capable of. The gyroscope on the other hand proves to be highly inaccurate. Remotely controlling the robot and making it spin approximately by 360Â° should lead to a similar value as in the starting position. Instead, the calculated value deviated by 0-30Â°. Additional spins by 360Â° increase the deviation even more, making this sensor too unreliable considering the robots will move and spin a lot more. Equally, the Mindsensors SumoEyes sensors do not meet the expectations. The up to 30 cm detection range can only be met when using big and bright objects. The other robot on the other hand could only ever be detected when it was very close (up to 20 cm), making this sensor unusable as well. These limitations are too great to work in this setup. An adjustment was done by replacing the Mindsensors SumoEyes with two ultrasonic sensors. During experiments it became apparent, that the ultrasonic sensor doesn't detect in a straight line ahead, but actually has a cone like detection area (figure 3.2).

The function of the Mindsensors SumoEyes sensor can be replicated by using two ultrasonic sensors and placing them at such an angle that the cones overlap as shown in figure 3.3. This way three detection areas are available. Left or green if only the left sensor detects something, front or red if both sensors detect something and right or blue if only the right sensor detects something. The distance

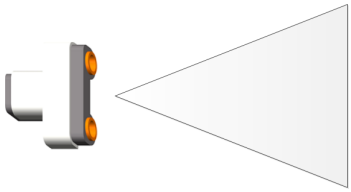

Figure 3.2: Detectable area for the ultrasonic sensor

ranges can be segmented as needed as well, as they are not limited to a short and long range. Yet, this setup still lacks a usable way to determine the orientation of the robots and is thus still insufficient.

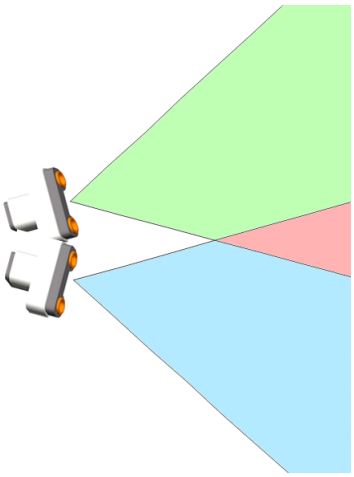

Figure 3.3: Combining ultrasonic sensors

3.2 Webcam plus OpenCV

A different approach is attempted using a webcam and the Open Source Computer Vision library (OpenCV) [Opea]. The sensor information the robots use is extracted out of the webcam's live feed and processed using OpenCV. This is done by tracking the robots and calculating things like their distance and angle from the webcam picture.

OpenCV

OpenCV is an open source computer vision and machine learning software library. It is released under a BSD license and is free to use both academically and commercially. The library encompasses a variety of algorithms, amongst them algorithms to detect and recognize faces, identify objects, classify human actions in video, extract 3D models of objects and track moving objects. It has a large user community and is used by companies like Google and Microsoft, research groups and by governmental bodies. OpenCV is natively written in C++, but has interfaces for C++, C, Python Java and MATLAB and works on Windows, Linux, Android and Mac OS systems [Opeb].

Webcam

The webcam used is a Logitech C525. It supports video of up to 720p (1280 pixels wide and 720 pixels tall).

Figure 3.4: Logitech C525 webcam

Source: https://assets.logitech.com/assets/55374/2/hd-webcam-c525-gallery.png

Physical setup

This setup only consists out of the camera and replaces all the other sensors. The camera is placed on a tripod with an extension and hangs over the sumo ring facing to the ground. The camera's field of view covers the sumo ring and a bit beyond. For the tracking system the robots are marked with colored squares on top, as shown in figure 3.5. Each robot has two different color squares. These colors where chosen using opposing colors from the hue color wheel, seen in figure 3.6, to make detection easier. At the same time, most objects in the camera's field of view are mostly colorless to reduce any possibility to falsely detect and track something that is not the robot.

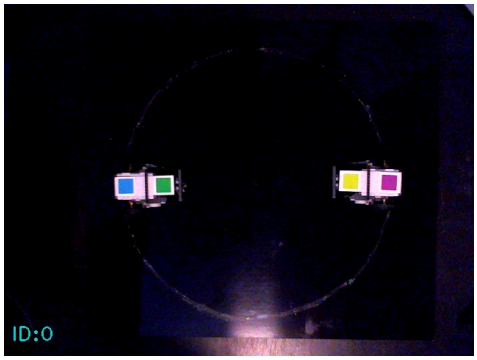

Figure 3.5: Tracking using colored squares

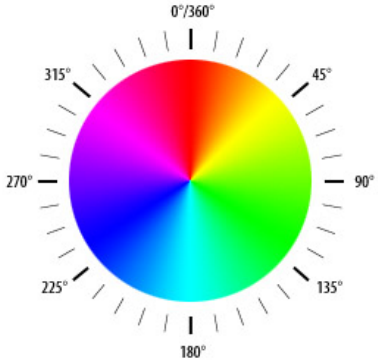

Figure 3.6: Hue colorwheel used to pick colors

Software setup

To be able to track the robots, a software is developed using the OpenCV library. The webcam provides images which are then processed. Using certain features of the OpenCV library, the colored squares are singled out and linked to the robots. A more in depth explanation can be found in chapter 6.

Performance

The camera offers a steady feed which allows the tracking information to be updated at a high rate. Additionally, there are no limitations regarding distance, orientation or movement speed of the robots. The robots are always in the field of view of the camera and thus the distance can always be calculated. Same goes for the orientation of the robots, which can be calculated for every frame. Overall, the information quality and quantity is much greater than with the separate Lego and Third-Party sensors. The only disadvantage is that the sensor is not directly connected to the robot and they don't each use their own sensors. The introduced delay is negligible though.

4 Parameter training

Training of the model parameter is an essential and difficult part when creating an autonomous robot. Supervised learning is used in this case, since for every observation set for a time t, there is a corresponding correct action. Correct action in this case means a decision made by a human player, which is ultimately what is desired to be modeled.

4.1 Recording of training data

Collection of training data is achieved by having two players play rounds of robot sumo against each other. Three players have participated and were used to create three different training data sets. The players were instructed upon first time playing about the goal, namely pushing the other robot sumo out of the ring, prior to starting. The players also had some time to warm up to the controls and to get used to the handling of the robot. To control the robots, players used a keyboard, using four buttons to remote control the robot. A camera is placed hanging over the sumo ring to track the robots and at the same time also to record the actual battle. Sensor information like the distance between the robots, distance to the ring center, direction of the ring center, area and heading is calculated from the video data and written into a data file on the computer, along with the current control command of the player (the action). Every 50ms a new set of sensor information and current action is added to the data file on the computer with a timestamp. This allows whole rounds to be recorded. The sensor information is raw and not discretized, which means that the discretization can always be changed if needed and the rounds can be reevaluated. The start of the round is voiced to the two players, but recording of both sensor data and video only happens the moment a player presses a button. This prevents that data about the robots idling is recorded as well, since it's not actually part of the match. The end of a round is determined by the computer recording it. Since it calculates the distance to the ring center for both robots as it is recording, it can determine when a robot has exceeded the rings perimeter by comparing its ring center distance to the radius of the sumo ring. If a robot goes beyond the sumo ring, all action commands by the players will be blocked, signalizing to the players that the round has ended and the recording of the round is immediately ended.

Another thing that needs to be considered is the human reaction time. Unlike robots, humans are not able to act instantaneously without any delay. It takes time for a visual impulse to be processed in the brain and then be evaluated before the result gets transformed into a finger press. Prof. Dr. Claus Möbus has presented an Example (Example 9) [Möb] for a simple reaction time test (Card, Moran & Newell, Ex 9, 1983, p.66). In this test, a user sits in front of a display terminal and is supposed to press the space bar whenever any symbol appears before him. The reaction time between the appearance on the display and the pressing of the space bar by the user is measured. A fast user is able to do this within 105ms, a medium fast user within 240ms and a slow user within 470ms. A similar test was done with the participants used the recording of training data, where they had to click with the mouse on a screen as soon as it turned green. The users all averaged at around 200ms. For this reason, a shift of about 200ms should be imposed on the action commands when evaluating the training data, since the actions of the human player are responses to observations they made about 200ms before.

4.2 Results of the training

The results of the training data can be seen in the figures 4.1, 4.2 and 4.3. It becomes quite clear that one of the players dominates the other two. This particular participant mentioned to have been an avid fan of a show called "Robot Wars", but whether this gave them an advantage could not be determined. The weakest player did not get to play as much as the second player and might have adapted more after playing for a longer time. The overall win percentage shows a clear favorite in participant 1.

Times played vs	P1	P2	P3
P1	-	39	12
P2	-	-	10
P3	-	-	-

Figure 4.1: Times each player played against another player

win\loss	P1	P2	P3
P1	-	27	9
P2	12	-	7
P3	3	3	-

Figure 4.2: Times a player won/lost against another player

Players	P1	P2	P3
Overall win %	70.5882353	38.7755102	27.2727273

Figure 4.3: Overall win percantage for a player

4.3 Parameter estimation

Parameter estimation is done using a maximum likelihood estimator (MLE). This means we basically count the number of relative occurrences in the training data and use that as the parameters. Something that needs to be taken into consideration is that certain occurrences might never happen in our training data. This can happen when the training data set is too small. Such a case is not desirable though, since it would lead to a probability of 0 in the relevant conditional probability table. Since it still actually is a possible outcome and nothing should ever be impossible, a simple plus-1-smoothing is done. This means that occurrences that never happened in the training data will now at least happen once, giving it a small probability.

5 Autonomous sumo robot

Creating an autonomous sumo robot that acts like a human player means that the way a human player plays and his circumstances need to be analyzed. In this chapter these are being taken into consideration, leading to the discretization of sensor information as well as how a sumo robot model could be modeled.

5.1 Player control and their observations

A player controls the robot remotely with a keyboard on a computer, which connects to the robot via Bluetooth. Since the robot has two motorized wheels and one carrying wheel as seen in figure 2.2 of section 2.3, it is possible to move the robot backward, forward, rotate on the spot and in curves. These possible movements are mapped to keyboard inputs. The human player uses the arrow keys, or alternatively the w, a, s and d keys to steer the robot. Pressing up or down (respectively w or s) makes the robot move forward and backward, pressing left and right (respectively a and d) tells the robot to rotate on the spot and pressing a combination of either up and down, and either left and right makes the robot drive in a curve. Not pressing any button will immediately stop the robot's movement. The movement speed cannot be manipulated by the player. Every movement has a set speed, equal for both robots. To attempt to push the other robot out of the ring, the player simply drives towards the enemy, with either his front or backside.

Both players sit close to the ring, always being able to observe the whole sumo ring and every movement of their own and the enemy robot. The players act and react based on what they observe happening in the sumo ring. To be able to create a reactive autonomous bot that acts like a human player, it is necessary to analyze what information a human player can gather from observing the two sumo robots in the ring. The following extractable information is chosen for the development of the robot.

Distance between robots: The distance between the two robots. Whether the robot is very far away or really close makes a difference in how the human player acts.

Distance to the ring center: The distance between the robot and the sumo ring center. The distance between the robot to the sumo ring edge is a more intuitive way to describe how close the robot is to being outside of the ring, but it is easier to determine the distance to the center of the ring. Essentially, knowing the distance to the ring center also describes how close the robot is to the sumo ring border, since the diameter is known. Both his own distance and the other robot's are gathered. If the enemy robot is close to the sumo ring border, a player might want to push out the robot. Likewise, when the own robot is close to the border, the player might want to flee from the ring edge to not risk being pushed out.

Direction of center: The direction of the sumo ring center relative to the robots own direction. Knowing e.g. that the ring center is behind the robot and that the enemy robot is in front, indicating that the positioning might be dangerous for the player.

Area: The position of the other robot relative to the own robot. This means, whether the other robot is currently positioned e.g. in front of the own robot or behind to the right. Having the

enemy robot right in front of you might be ideal for an attack, while having the enemy positioned at a side might make the player vulnerable.

Heading: The direction the enemy robot is facing in relation to the own robot. This could mean both robots are facing the same way or opposing directions, but doesn't say anything about their position.

These different aspects cover a lot of the information a player has while fighting against another player in robot sumo. Thus these are used as information for the robot. The information is gathered using the Sensors described in chapter 3. In the next section it is explained how these different aspects are extracted from the sensor information.

5.2 Discretization

The sensor information gained, such as the distance or the angle, is continuous and as such has an infinite number of possible values. This makes giving every possible outcome a probability impossible [SR95]. Instead the infinite amount of possible values is divided into intervals, e.g. having the distance measured between the bots be divided into ranges like 0-10cm, 11-20cm and 21-30cm. This makes it easier to work with the sensor information, but can also mean a loss of accuracy. In the following, the different aspects mentioned in 5.1 will be presented regarding their discretization.

Distance between two robots

The distance between two robots is divided into three intervals as seen in figure 5.1. The white block depicts the sumo robot and the three circles around it the different intervals. The intervals range from 0 to <15cm, 15 to <30cm and >=30. The last interval is open ended, but considering the sumo ring has a diameter of 77cm and a robot loses as soon as it exits the ring, it is still limited.

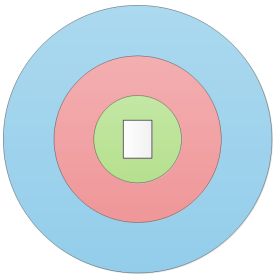

Figure 5.1: The three distance intervals

Distance to the ring center

The distance between the robot and the ring center is divided into three intervals as well. The first interval reaches from 0 to <= 12cm, the second one from 12 to <= 24cm and the last one is 24cm and upwards.

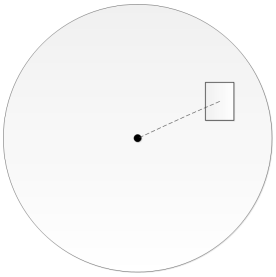

Figure 5.2: The distance between the robot and the ring center

Area

The area for the robots position is split up into 8 evenly big sectors as seen in figure 5.3, with each being 45°big. They range from front (337.5°to <22.5°), frontRight (22.5°to <67.5°), right (67.7°to <112.5°), backRight (112.5°to <157.5°), back (157.5°to <202.5°), backLeft (202.5°to <247.5°), left (247.5°to <292.5°) to frontLeft (292.5°to <337.5°).

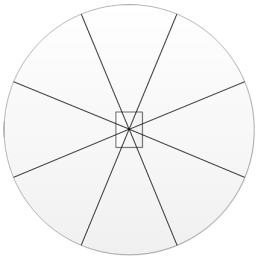

Figure 5.3: The 8 sectors for area

Direction of the center

The direction to the ring center has the same sectors and works the same way as *Area*. The only difference is that the object of which the relative positional area is to be determined is the ring center instead of the other robot.

Heading

The heading for the robot is divided into 3 sectors. The figure 5.4 shows the different sectors. In this example we can see the other robot is facing its backside to the center robot.

To help with finding suitable sectors, a javascript library called plotly for data visualization was used to visualize the parameters for each variable. An extract of this can be seen in figure 5.5. This makes it easier to identify sectors that might be too small when probabilities are very low or intervals that are too big if probabilities are very high.

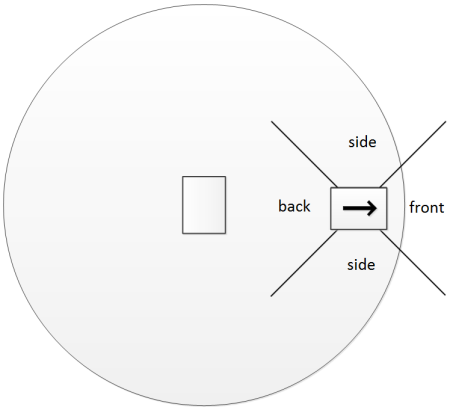

Figure 5.4: The 3 sectors for heading

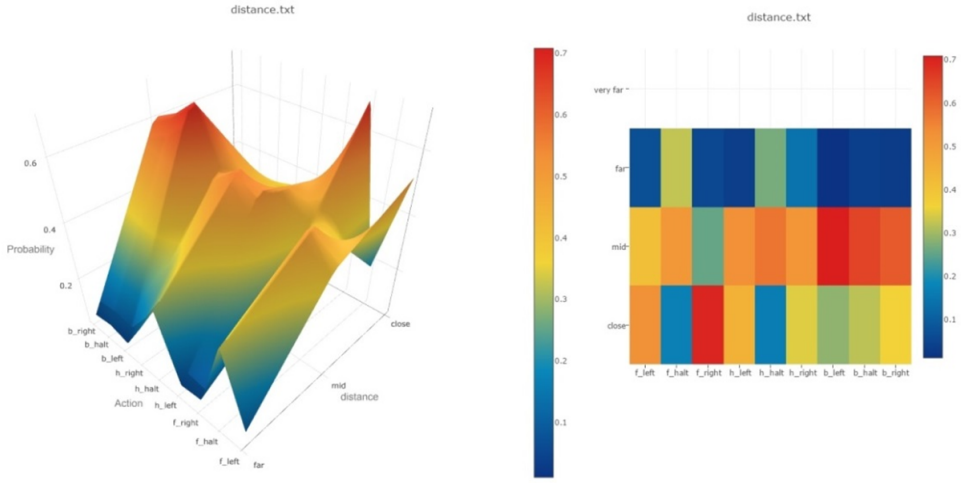

Figure 5.5: Visualizing data using plotly

5.3 Model

After learning about Bayesian networks and identifying important elements and information in robot sumo which should be represented in a Bayesian network, it is time to model these. The first and most simple one consists out of one parent node and multiple child nodes as seen in figure 5.6. *Action* is the query variable for which we want to know the probabilities of, so the robot can decide on an action to execute. *Action* is also the parent of all the other variables, creating a dependency to the parent variable. The dependent child nodes are all evidence variables for which the robot gets observations for.

The inference for *Action* given all the evidence variable values from observations would look as follows (To save space Action, Distance, Area and Heading will be abbreviated to Act, D, A and H, respectively).

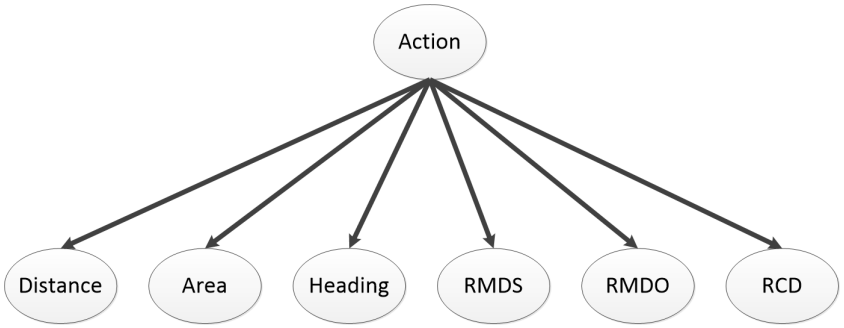

Figure 5.6: Simple Bayesian network (RMDS, RMDO and RCD stand for ringMiddleDistanceSelf, ringMiddleDistanceOther and ringCenterDirection, respectively)

$$\begin{aligned}&\mathbf{P}(Act|d,a,h,rmds,rmdo,rcd)\\ &= \alpha\mathbf{P}(Act,d,a,h,rmds,rmdo,rcd)\\ &= \alpha\mathbf{P}(Act)\mathbf{P}(d|Act)\mathbf{P}(a|Act)\mathbf{P}(h|Act)\mathbf{P}(rmds|Act)\mathbf{P}(rmdo|Act)\mathbf{P}(rcd|Act).\end{aligned}$$

A different model that was created upon observing the robot react and the probability distribution for action can be seen in figure 5.7. The action variable, which contains actions like forward, forwardLeft and left is split into movements in two dimensions. The regular action variable might e.g. have the action *right* the the most likely action, while at the same time the actions *forwardLeft, forward* and *forwardRight*, all having forward movement, would have a higher probability if it weren't split up. That's why in this model there is one variable (FHB) for forward, halt and backward movement and a variable (LHR) for left, halt and right movement. The inference is done for both variables and then combined, meaning if FHB has the action forward as highest probability and LHR left as the highest, then a left forward movement is done.

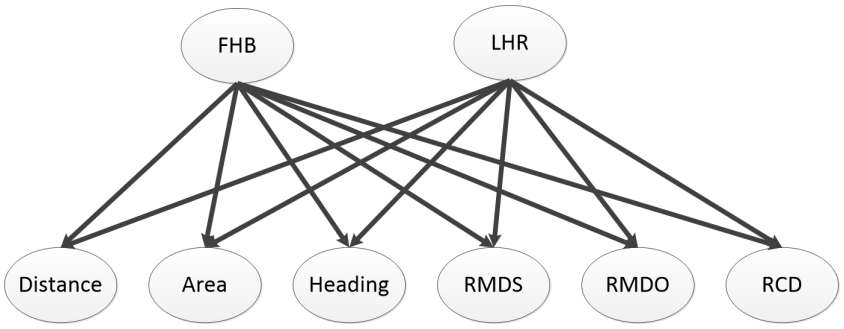

Figure 5.7: Action variable split into horizontal and vertical movement

Another attempt at a model that proved to not bring much benefit was using the first model, but making it dynamic by adding a dependency of $action$ to the new variable $action_(t-1)$. The behaviour of the robot was too unchanging. It would stay stuck in an action, because the probability to ever change was always too low.

6 Implementation

After having designed a model and the discretization of the sensor information, the actual implementation begins. In the following sections the programming of the robots using leJOS, the tracking of the robots and other elements are presented.

6.1 leJOS and sumo robots

leJOS is the framework used on the Lego NXT robots. It's a tiny Java Virtual Machine stripped from a lot of classes to be able to fit onto the small memory of the NXT brick. Some functions had to be reimplemented, e.g. a split function for Strings used to split incoming data from the PC, since they were absent in leJOS. The robots connect via Bluetooth, although a USB connection is possible too. The NXT bricks need to be paired with the computer in order to be able to exchange data. After having paired the devices, a connection can be established at any time, providing the devices are turned on and ready. To establish a connection the NXT brick is told to listen using *Bluetooth.waitForConnection()*. After that, the computer can initiate a search for the device and establish a connection. Once this is done, both ends are able to exchange data using input and output streams. The supported data types are boolean, byte, unsigned-byte, byte-arrays, int, short, long, double, float, char, string and specially encoded strings. For the exchange of data, strings are used that contain all the necessary sensor information as numbers, split with a delimiter (e.g. "5;3;22;9;2"). Each number corresponds to certain sensor information and the value itself is an encoded variable value. These are basically the values of the evidence variables.

The program running on the NXT brick is a separate program managing the Bluetooth connection, the inference given the evidence and the control of the robot. Programs can be uploaded onto the NXT brick using a plugin for the integrated development environment Eclipse. The program contains the Bayesian network and conditional probability tables for its variables to be able to do the inference using the incoming observations for the evidence variables.

6.2 Tracking of sumo robots

The tracking of the sumo robots is done using the developed program which makes use of the Java interface for the OpenCV library. A webcam is connected to the computer and the program fetches every frame from the camera to process it. The tracking is color-based and uses the HSV color space, which consists out of hue, saturation and value. Hue means the color, saturation the intensity and value the brightness. The HSV color space is used because it gives some robustness to lighting changes over other color spaces like e.g. RGB ([PS08], 2008). To track an object or in this case a color, ranges for hue, saturation and value need to be defined which correspond to the color. To facilitate finding those ranges, sliders for each of them have been implemented. A live view of the webcam feed and a filtered view using the HSV slider ranges help to identify the objects that are desired to be tracked and how well the current slider settings work. This can be seen in figure 6.1 with the live webcam image on the left and the HSV filtered images on the right. The white part corresponds to the color we want to track and the black everything that got filtered out.

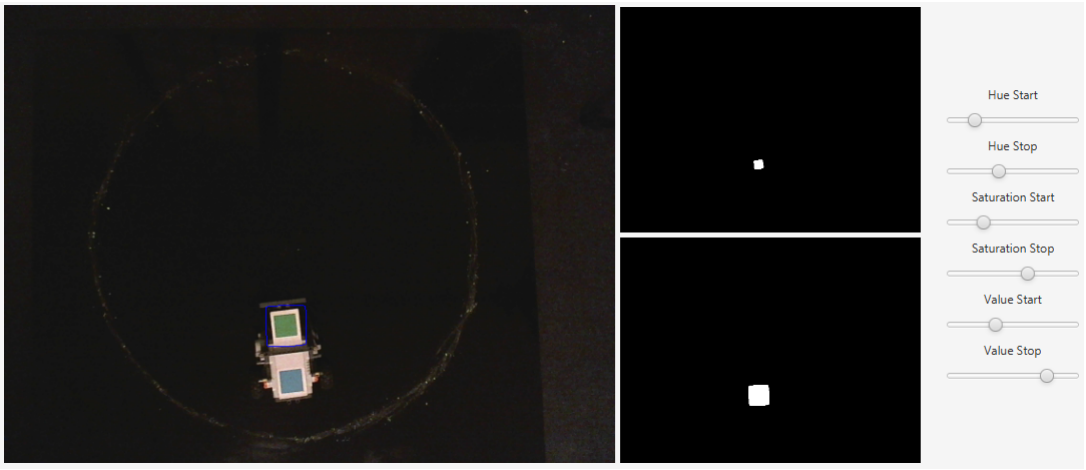

Figure 6.1: Webcam live view and HSV filtered views with adjustment sliders

To achieve the result seen in the lower right image, more steps than just HSV filtering are required, using the OpenCV library. First, a blur function is used to remove some noise from the original image. Next the image is converted from the RGB color space into the HSV color space. After this, the threshold from the sliders is applied and the result of this is what is seen on the upper right view in figure 6.1. At this stage, the white part is usually very jagged and a bit smaller than the real colored part in the live view. It can also have stray small dots around the desired element. This is why in the next step white elements in the image are first dilated, meaning the outer most layers are stripped off, and then eroded, which does the opposite and adds some layers of pixels. The dilation happens with a bigger square-shaped kernel, while the erosion is done with a half the size square-shaped kernel. The result of these operations is show in the lower right view. Lastly, the contours of the white element is calculated and drawn onto the live view on the left in shape of blue circles. Since there are four colored squares, two on each robot, that need to be tracked, a calibration function is implemented that allows to set custom HSV ranges for each colored square. The calibrated HSV ranges are printed into the console upon successful calibration. After having calibrated all four color square, hitting the *Use calibration* check box makes the program track all four squares at once as it can be seen in 6.2. For each colored square it is possible to determine its center. This can then be used to calculate things like the distance or the angle between the two robots. The position of the ring center is determined by placing a colored square temporarily into the middle of the sumo ring, adjusting the HSV sliders to detect the colored square and then saved by clicking on the *Calibrate Ring Mid* button on the programs interface.

6.3 Extracting sensor information

Not only for the training but also to provide the robots with the needed observations for the inference in their Bayesian networks it is needed to extract the information through the webcam. Being able to track the colored squares means that the center of their position can be calculated. The position equals the pixel distance in width and height from the origin in the top left corner of the image. The required sensor information are the distance between the two robots, the distance to the ring center for each robot, the relative area of position of the other robot for both, the heading for both robots and

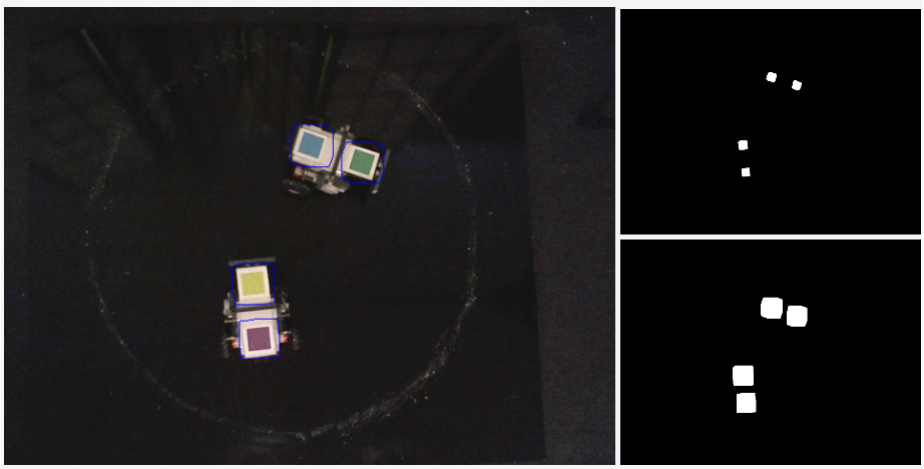

Figure 6.2: Four color squares being tracked at the same time

the relative direction of the center for each robot. Distance is converted from pixels into cm. This is done by using the distance in cm between the two colored squares on a robot as a scale. The distance between the centers of the two squares is 10cm. The calculated pixel distance between those two colored squares is then used to convert any pixel distance into cm.

Calculating the distance between the robots

The distance between both robots is calculated by first calculating the center of the two colored squares on the robot. Then, using the centers of both robots, the distance is calculated and converted into cm.

Calculating the distance to the ring center

The calculation for the distance to the ring center is done similarly. Again, the center of the robot is used with the saved ring center position to calculate the distance. This distance too is then converted into cm.

Calculating the relative area of position

First, the angle of the robot itself, using the two colored squares and their distance, to a vertical line is calculated. Depending on which way the robot is facing, a correction is applied as to receive values from 0°to 359°. This means, that if the robot faces upward, from the webcam's aerial perspective, the robot has an angle of 0°/360°. If the robot faces to the right from the webcam's perspective, the robot has an angle of 90°and if the robot faces to the left, the robot has an angle of 270°. This is done for both robots. After that, the angle between the centers of the robots is calculated for each robot. The result of this is then subtracted by the already calculated own angle of the robot and results in a relative angle to the other robot. The angles can end up being negative, but in such a case the correct result can be achieved by adding 360 to it. This means that if the other robot is right in front of the

robot, the angle is 0°. If the robot is to the right of the robot from the robots perspective, the angle is 90°and if the other robot is to the left of the robot from the robots perspective, the angle is 270°. The 360°around the robot are split into areas into which these angles then fall.

Calculating the heading

The heading is calculated by using the angles of each robot to a vertical line like before and then subtracting the robot's own angle from the other robot's angle. Again, if the result is negative, the correct result can be achieved by adding 360 to it.

Calculating the relative direction of the center of the ring

The calculation of the relative direction to the center of the sumo ring basically works the same way as calculating the relative are of position of the other robot, but instead of trying to locate the other robot, the position of the relative area of the ring center is calculated.

6.4 Collecting of training data

The general procedure was already presented in section 4.1. The details of it will be explained here further. Since the robots are now trackable and all the necessary information can be retrieved through the webcam and further processing, all the gathered data can be saved and used to parametrize the variable nodes in the Bayesian network. The collection of this training data is done automatically. Once a round starts, a timestamp, the action command for the robots given by the players and the retrieved sensor information through the webcam is stored every 50ms into arrays. Once one robot exceeds the sumo rings perimeter, the data collection is immediately stopped and the collected data is saved onto the hard disk of the computer. Gson is used to save the data. It has the advantage that the data can easily be loaded into an object containing all the arrays for later processing. Video is also recorded for the duration of the round. A timestamp helps linking the timestamped data and the video together. The recording is done using an OpenCV function that writes all the frames that are used to retrieve the data onto the disk. It is somewhat problematic to record the video in such a way that it is exactly as long as the round actually was, thus making the timestamp a necessity to link it with the data. The video is recorded at 24 frames per second, since that gives the most consistent framerate.

Visualization of the inference

Views have been added that allow for live comprehension of which action was how probable as the sumo robot is fighting against another robot. Such a view can be seen in figure 6.3.

It's possible to slow down the speed of the robots with a button, making it possible to read the visualization and understand how it's changing better. This helps understanding how a robot is reacting given certain circumstances.

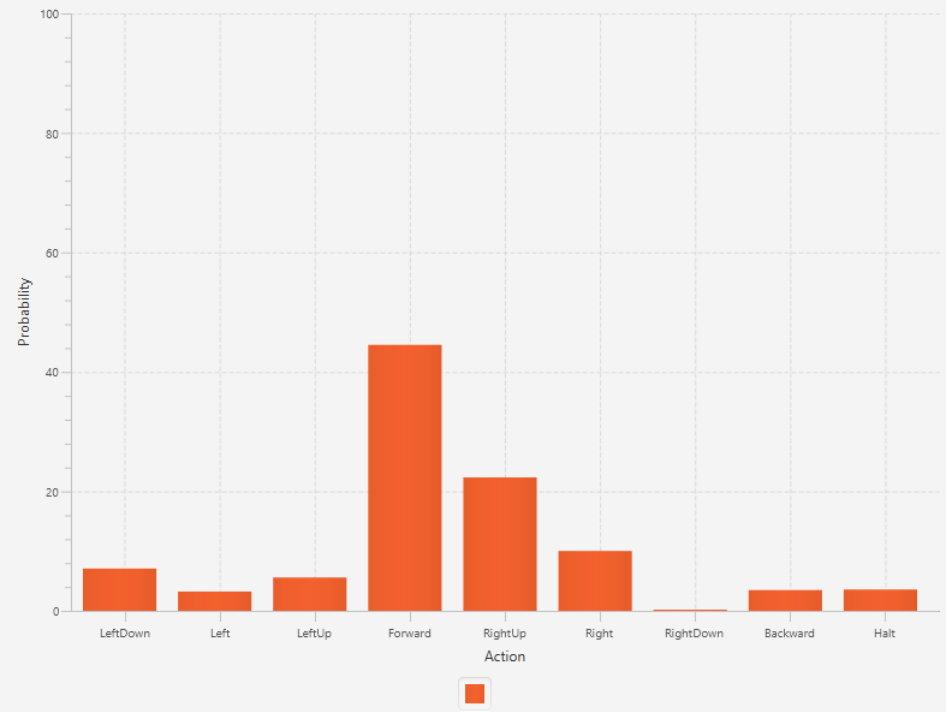

Figure 6.3: Live view of the probability distribution of the variable Action.

6.5 Programming the Bayesian network

Frameworks for Bayesian networks exist for all kinds of languages, including the one being used here, Java. The problem with them though is, that they all are too big for the small memory of the NXT brick and use classes not supported by the tiny JVM provided by leJOS. So it was needed to write an own system meeting all requirements and being small enough to fit onto the NXT brick. The usage is simple. An example on how to construct and set up a Bayes Network can be seen in figure 6.4.

```
Net net = new Net();

Node a = new Node("A", 2);
net.addNode(a);
a.setCpt(new double[]{
        0.3,
        0.7
});
```

Figure 6.4: Setting up the network and filling it with a node.

First an empty net is initialized. This will hold all the nodes and information about dependencies. Next a node *A* is created. This node variable has to values or features like a Boolean. Thus the node is initialized with its name and the number of features it has. Next the node is added to the network and then the probability table is filled. Every column should add up to 1 if it is properly formatted. Next, another node is created and added to the network as seen in figure 6.5.

```
Net net = new Net();

Node a = new Node("A", 2);
net.addNode(a);
a.setCpt(new double[]{
        0.3,
        0.7
});


Node b = new Node("B", 2);
net.addNode(b);
net.addDependency(a, b);
b.setCpt(new double[] {
        0.8, 0.4,
        0.2, 0.6
});
```

Figure 6.5: Setting up another node and its dependencies.

The node's name *B* and the number of features is added. Then the node is added to the network as well. Unlike *A*, *B* has a dependency. This dependency is added by telling the net to make *A* the parent of *B* with the method *addDependency(parent, child)*. Afterwards the conditional probability table for *B* is set. If this is nicely formatted, the columns should sum up to 1. The graphical representation of this programmed net can be seen in figure 6.6.

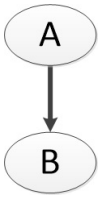

Figure 6.6: Graphical representation of the programmed net.

Now that the Bayesian network is set up, we can do some inference. Figure 6.7 shows how this is done. For a better overview, the variables *A* and *B* have received names for their variable values, in this case *true* and *false*. It is not necessary to set the variable names and a print function expressing those needs to be written manually. After setting up the network, it is possible to set observations for nodes. This is done by using *setObservation(index)* on the node. The index is the variable value or feature index. In this case *0* would equal *true* and *1* would equal *false* for both nodes.

Lastly we can do the inference using the last function with the parameters being the net, the query node and a list of evidence nodes. If no observation is set, then the list of evidence nodes is left empty. The function then returns the probabilities for all variables values or features of the query node, in this case $\mathbf{P}(A|B = false)$ or $\mathbf{P}(A|B = 0)$. For this example, the program calculate $\mathbf{P}(A = true|B = false) = 0.46$ and $\mathbf{P}(A = false|B = false) = 0.54$ as seen in figure 6.8.

For the calculation of the inference, the equation from section 2.4 are used. Sums and terms are moved to optimize performance. After that follows the computation of the probability of a query variable given none or any amount of evidence variables. This is done using a recursive function that

```
Node a = new Node("A", 2);
net.addNode(a);
a.setValNames(new String[] {"T", "F"});
a.setCpt(new double[]{
        0.3,
        0.7
});

Node b = new Node("B", 2);
b.setValNames(new String[] {"T", "F"});
net.addNode(b);
net.addDependency(a, b);
b.setCpt(new double[] {
        0.8, 0.4,
        0.2, 0.6
});

b.setObservation(0);

double[] probs = net.estimateVariableNormalizedExt(net, a, new Node[] {b} );
```

Figure 6.7: Example program code to do inference.

```
[T]:    0.46153846153846145
[F]:    0.5384615384615383
```

Figure 6.8: Example inference result

begins with the most right sum and carries over interim results as it goes through the sums. In the end, the normalized probabilities for each value or feature of the query variable given none or any evidence variables is returned within a double array.

7 Evaluation

For the evaluation of the Bayesian networks and their parameters, the strongest competitors data (participant 1 with a win percentage of 70%) versus the weakest competitors data (participant 3 with a win percentage of 27%) were made to compete against each other. Both robots using the same simple Bayesian network, but each with different training data. Even though the participant 1 had a much higher winning percentage, this could not be seen in the same amount when the robots were fighting against each other using the training data of their respective participants. The win percentage of the robot using participant 1's training data had a win percentage of merely 58%, just hinting at an advantage over the other robot, suggesting that the training data could not be properly transformed. The model might also be too simple for something as hectic as robot sumo.

8 Conclusion

In this thesis, a system was developed to create Bayesian networks, train its variables with training data which was gathered from human players and finally have sumo robots use those systems to duel each other. leJOS was used as a framework on the robots and although the NXT brick doesn't have much memory, it was possible to fit a Bayesian network on it with all the necessary mathematical functions and data. Different models have been created and tested. There were a lot of hurdles trying to achieve this. Starting with the limited possibilities of the NXT brick and the sensors all functioning way under expectation, through to creating a completely different and own system to replace the unusable sensors. The system for the calculations of the inference works rather well, but maintaining the views that show a live preview of the robots current action probability distribution proved to be a bit difficult. Adding new elements or new models with new variables always meant a restructure ending with the program being a bit convoluted. The evaluation was sadly not as extensive as initially planned. Problems with the software after adding new variables were only solved very late leading to an evaluation that would have deserved more depth and more models. The models itself also still seem to be too simple to describe something like a sumo battle. A behaviour variable for attack or defense/escaping would make sense and might improve the robot even more. The robot with the training data from the stronger competitor was able to pull ahead in victories, but there still are ways to improve the results.

Images

2.1 Lego Mindstorms programing interface 6
2.2 Lego NXT Sumo Robot 6
2.3 A Bayesian network of an alarm system. 7

3.1 Mindsensors Sumo Eyes 10
3.2 Detectable area for the ultrasonic sensor 11
3.3 Combining ultrasonic sensors 11
3.4 Logitech C525 webcam 12
3.5 Tracking using colored squares 12
3.6 Hue colorwheel used to pick colors 12

4.1 Times each player played against another player 16
4.2 Times a player won/lost against another player 16
4.3 Overall win percantage for a player 16

5.1 The three distance intervals 18
5.2 The distance between the robot and the ring center 19
5.3 The 8 sectors for area 19
5.4 The 3 sectors for heading 20
5.5 Visualizing data using plotly 20
5.6 Simple Bayesian network (RMDS, RMDO and RCD stand for ringMiddleDistance-Self, ringMiddleDistanceOther and ringCenterDirection, respectively) 21
5.7 Action variable split into horizontal and vertical movement 21

6.1 Webcam live view and HSV filtered views with adjustment sliders 24
6.2 Four color squares being tracked at the same time 25
6.3 Live view of the probability distribution of the variable Action. 27
6.4 Setting up the network and filling it with a node. 27
6.5 Setting up another node and its dependencies. 28
6.6 Graphical representation of the programmed net. 28
6.7 Example program code to do inference. 29
6.8 Example inference result 29

Literature

[DK09] DAPHNE KOLLER, Nir F.: *Probabilistic Graphical Models - Principles and Techniques*. 2009. – ISBN 978–0–262–01319–2

[HiT] HITECHNIC: *Hitechnic Lego NXT Gyro Sensor*. https://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NGY1044

[Leg] LEGO: *Lego NXT Ultrasonic Sensor*. http://cache.lego.com/downloads/education/9797_LME_UserGuide_US_low.pdf

[leJ] LEJOS: *leJOS*. http://www.lejos.org/nxj.php

[Min] MINDSENSORS.COM: *Mindsensors Sumoe Eyes Sensor*. http://www.mindsensors.com/ev3-and-nxt/28-dual-range-triple-zone-infrared-obstacle-detector-for-nxt-or-ev3

[Möb] MÖBUS, Prof. Dr. C.: *Example from Simple Reaction Time - The Psychology of Human-Computer Interaction*. http://www.uni-oldenburg.de/en/computingscience/lcs/probabilistic-programming/church-a-probabilistic-scheme/simple-reaction-time-the-psychology-of-human-computer-interaction/

[nxt] NXTPROGRAMS.COM: *nxtprograms.com*. http://www.nxtprograms.com/castor_bot/steps.html

[Opea] OPENCV: *Open Source Computer Vision*. http://opencv.org

[Opeb] OPENCV: *Open Source Computer Vision*. http://opencv.org/about.html

[PS08] PATRICK SEBASTIAN, Richard C. Yap Vooi Voon V. Yap Vooi Voon: The effect of colour space on tracking robustness. In: *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference* (2008)

[rob] ROBOGAMES.NET: *robogames.net*. http://robogames.net/rules/all-sumo.php

[SR95] STUART RUSSELL, Peter N.: *Artificial Intelligence - A Modern Aproach (Third Edition)*. 2010, 2003, 1995. – ISBN 0136042597

[sum] SUMOTALK.COM: *sumotalk.com*. http://www.sumotalk.com/rules.htm

Versicherung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Oldenburg, den February 10, 2017

Michael Möbes