# NAO RoboCup: Classification of Sound Localization

Jasper van Eck

University of Amsterdam

# NAO RoboCup:
# Classification of Sound Localization

Jasper van Eck
6228194

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

*Supervisor*
Dr. Arnoud Visser

Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 28th, 2019


## Acknowledgements

I would like to express my appreciation and gratitude to my supervisor, Dr. Arnoud Visser. I would like to thank him for his excellent feedback and for taking the time out of his busy schedule for weekly meetings.

I would also like to thank my fellow students supervised by Dr. Visser, for the support, new insights, and moral support.

Lastly, I would like to thank my proofreaders for giving supporting feedback and providing encouragement.

**Abstract**

The Directional Whistle Challenge study is a study aimed at solving an important task, the localization of a whistling sound inside or outside a standard RoboCup field. Solving this task would allow for a faster kick-off during a game. This study uses data gathered in an Unreal Engine 4 simulation. Three NAO robots are used to hear the sound and determine the time delay between hearing. The time delays, together with the location of the NAOs is used for the classification of the sound signal. This study aims to clarify how the amount of noise influences the accuracy with which the sound signal is classified.

# Contents

# 1 Introduction

For many years, the RoboCup challenge has been running, allowing for an accessible and fun way for researchers and students to step into the field of robotics and its peripheral software fields [9]. For example, previous challenges have included the recognition of a whistling sound by an NAO robot [1]. This study is based on one of those challenges [3].

To be more specific, this study has the aim of designing an algorithm that allows for the classification of a whistle sounds location, whether it originates from inside or outside the playing field. With the use of this algorithm, this study seeks to pinpoint the amount of noise that can be added to the measured data before the classifier can no longer reliably predict classifications. This study also wants to show the accuracy, precision, recall and other scores of the classification models for each amount of noise that is added to the data.

This classification of whether a sound comes from inside or outside the field is important for two reasons. Firstly, during a RoboCup tournament, there may be multiple fields in the same space, relatively close to each other. This may cause the NAO robots to pick up whistle signals from other fields and act on those signals, which is not desirable. Therefore, a distinction must be made between whistle signals originating from inside or outside of the playing field. Secondly, it is possible that whistle or whistle-like signals originate from the audience viewing the matches. Similarly, acting on those signals is unwanted behavior.

However, as there has not been done something similar for the NAO robots as of yet, it is the intention of this study to allow for further expansion of the robot's capabilities in future challenges and studies.

To complete the study as laid out by the RoboCup Challenge, several topics are necessary to address. First, more research must be done into how the localization of sound is achieved mathematically, biologically and mechanically [12], so that this can be replicated with an algorithm. Secondly, an investigation into the current software of the NAO robots is required, so that a new algorithm will work properly with the existing code. Another example of existing software to be studied includes communication protocols, as the challenge allows for up to five robots being used. The third and final area of research examines coordinate systems and the conversion of coordinates between them. This is because the robots will be able to determine locations and directions based on their relative position, rather than their global

position, requiring the need to investigate coordinate system conversions.

This study has the objective of constructing an algorithm, which is able to determine the location from the source of a whistling sound, with the aim of determining whether the source originates from inside the playing field or originates from outside it. More specifically, the study aims to clarify the accuracy with which the algorithm can determine the angle and distance of the sound source and its reliability of classifying whether a sound originates from inside or outside the playing field. The next sections of this thesis will describe the methods and results of this study.

# 2 Related Research

Localization of a sound source electronically has been a topic of study for over 80 years. Starting with radar, moving onto passive sonar and, currently, studies and companies are moving into the VR domain to create dynamic 3D audio[1] [14, 8]. This study builds on the principles discovered in those studies. More specifically, this study applies the mathematically derived function for determining the location of a sound with two microphones [12]. Eq. 1 shows the derived function. The variable $AB'$ is the distance the sound wave has traveled in the computed time difference between the two microphones. This distance is equal to the time delay multiplied by the speed of sound. The $x_B$ is the distance from point $(0, 0)$ to microphone B along the $x$-axis. Since the origin of the coordinate system is exactly halfway between the two microphones, the value of $x_B$ is half the distance between the two microphones. The $x$- and $y$-values are the values along which the function is defined, for a given value of $x$, there are two corresponding $y$ values. Determining three functions between three microphone pairs allows for calculating the intersection between those three functions. These functions are Eq. 1, with different values for $x_B$ and $AB'$. For certain values of $x$ the function is not defined; Eq. 2 shows for which values of $x$ the function is not defined. The intersection of those three functions models the sound source location.

$$y = \pm\sqrt{\frac{AB'^2}{4} - x_B^2 + x^2(\frac{4 * x_B^2}{AB'^2} - 1)} \tag{1}$$

[1] `https://spectrum.ieee.org/consumer-electronics/audiovideo/vr-for-your-ears-dynamic-3d-audio-is-coming-soon`

$$x \geq \sqrt{-\frac{AB'^2(AB'^2 - 4X_B^2)}{4(4X_B^2 - AB'^2)}} \tag{2}$$

A previous study showed that it is possible to recognize a whistling sound with high precision and accuracy [1]. It used three different preprocessing methods, and it tested four different classifiers. However, due to the high precision and accuracy of the models generated by the classifiers, it could not reliably determine the best performing option. This present thesis is a continuation, in that both studies can be used for the classification of whistle sounds. Not only recognizing the sounds itself but also determine the location of the sound. Both classifications are used with the intent of allowing the robot to correctly alter its behavior based on stimuli present on and around the playing field. This study focuses on location classification.

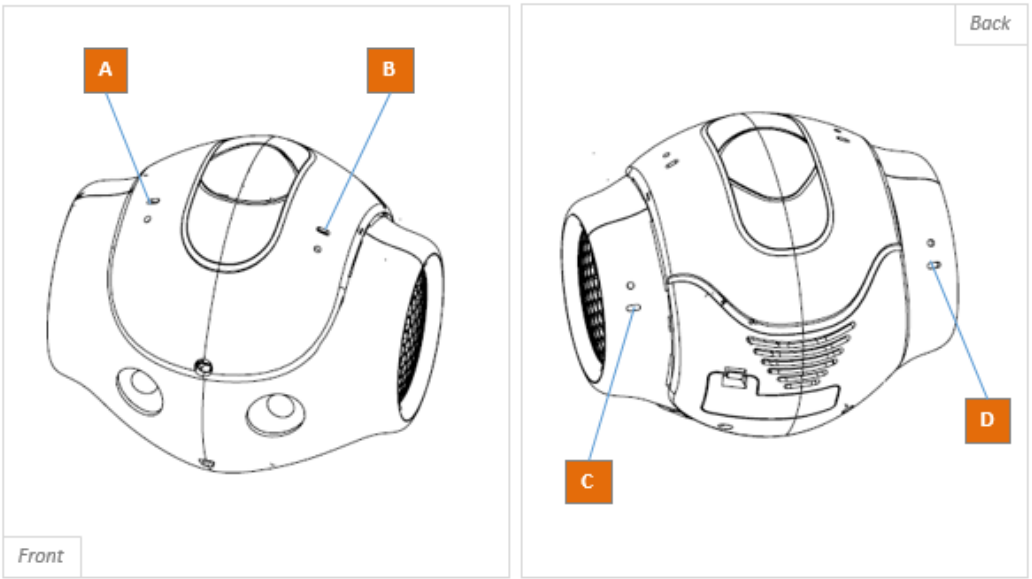


Figure 1: The locations of the NAO robot microphones on the head.

The NAO robot is the main subject of this study. The robot's four microphones will be used to listen for the whistle[2]. Two microphones are at the top front of the head, and two are on the middle of the back. Fig. 1 shows the locations of the microphones on the head of the NAO robot. Even though

[2]http://doc.aldebaran.com/2-1/family/robots/microphone_robot.html

the NAO robot has 4 microphones, they are on different heights. This forces the use of only two pairs of microphones, the two back microphones, and the two front ones or the use of multiple NAOs, for the function mentioned above.

Determining the actual location of the robot is an important aspect of this study. For determining the actual location we need the ground-truth data [11]. This data can be obtained with the use of cameras mounted above and around the playing field, allowing for a complete view of the field[3]. With this camera setup, the ground-truth location on the field of the NAO can be determined. Fig. 2 shows a top-down representation of the playing field.

The location can also be determined with simulation. Using the Unreal Engine 4.13.3 [4] it is possible to simulate the NAO playing field and the NAOs themselves[5]. With this simulation, it is possible to retrieve the locations and the orientation of the robots in the simulated environment [10]. This data of locations and orientations is ground-truth data. The location and orientation are not the only possible data attainable through simulation. It is also possible to render scenes with NAO robots in random or predetermined positions. These scenes can be used for training object, opponents or balls, for example, classifiers [6]. In this study the focus is on sound.

There are many different classifiers and machine learning algorithms that can classify whether the source of a sound is located inside the playing field or outside of it. In this study three classifiers are used, logistic regression, K-Nearest Neighbor and AdaBoost [7, 4, 5]. Each of the three classifiers is quite distinct in how they operate, this diversity in abilities should cover the range of possible data structures generated for this study. Logistic regression is a regular supervised learning algorithm, and only able to handle polynomial datasets. K-Nearest Neighbor is a supervised learning algorithm that can operate on non-polynomial datasets. AdaBoost is a supervised machine learning algorithm that combines multiple weak-learner algorithms so that they can increase their overall learning scores.

The next section of this thesis will demonstrate the methods executed to achieve the classification. It also describes how the data for this study was obtained, and how it will be used.

[3] `https://optitrack.com/products/flex-13/`

[4] `https://www.unrealengine.com/en-US/what-is-unreal-engine-4`

[5] `https://github.com/TimmHess/UERoboCup`

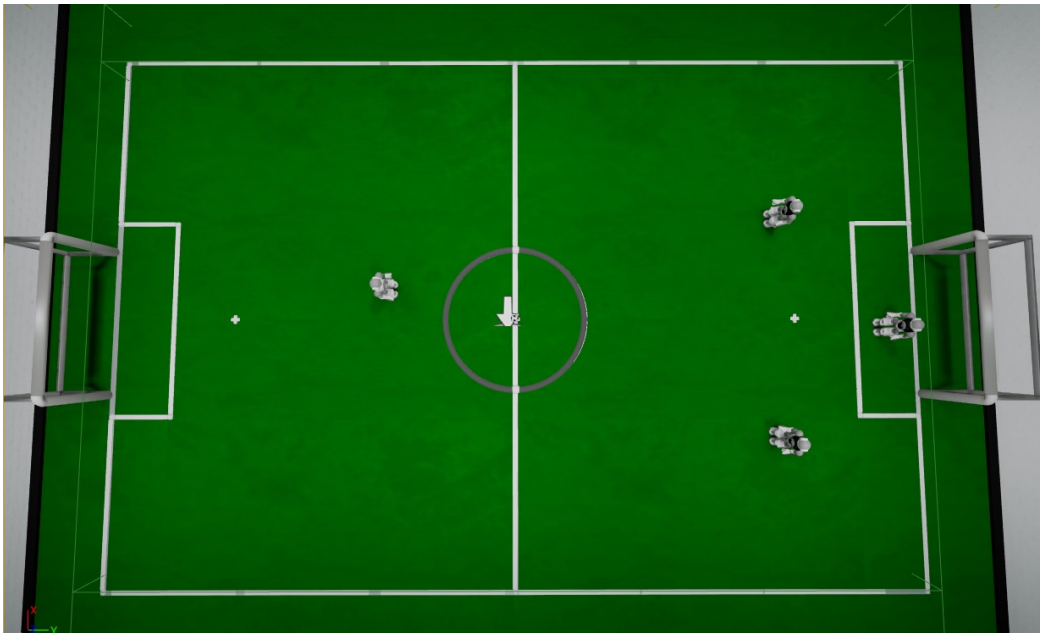

Figure 2: Top-down representation of the playing field.

# 3 Research Method

This section describes all the methods, and the details of implementation, that were used for this study. The first, subsections will describe how the data was collected. Further subsections will describe the algorithm used for this study and how it is used. The last subsection discusses the classification of the location from the sound source. The main focus of this study is on the data collection and the classification of the acquired data.

## 3.1 Data Collection

This subsection discusses the data collection done for this study. It will show how the data was created. Later sections will describe how the data is used. To collect the data for this study, a simulation was created with the use of Unreal Engine 4.13.3[6]. All data generated by the simulation, the NAO robot locations, the sound source location, and the time delays, are written to text files.

[6] `https://www.unrealengine.com/en-US/what-is-unreal-engine-4`

### 3.1.1 The Unreal Simulation

The created simulation[7] in the Unreal Engine, is based on a previously made simulation[8] [6]. The alterations made for this study pertain mainly to the underlying blueprint, which defines the dynamics of the simulation environment, shown partially in Fig. 3, of the simulation, and small alterations to the level. The version of UE4 that is used for this simulation is 4.13.3[9]. The Unreal Engine also has the ability to use plugins. For this simulation two plugins are used, Materials provided by Allegorithmic[10] and the substance plugin that is available on the Unreal marketplace.

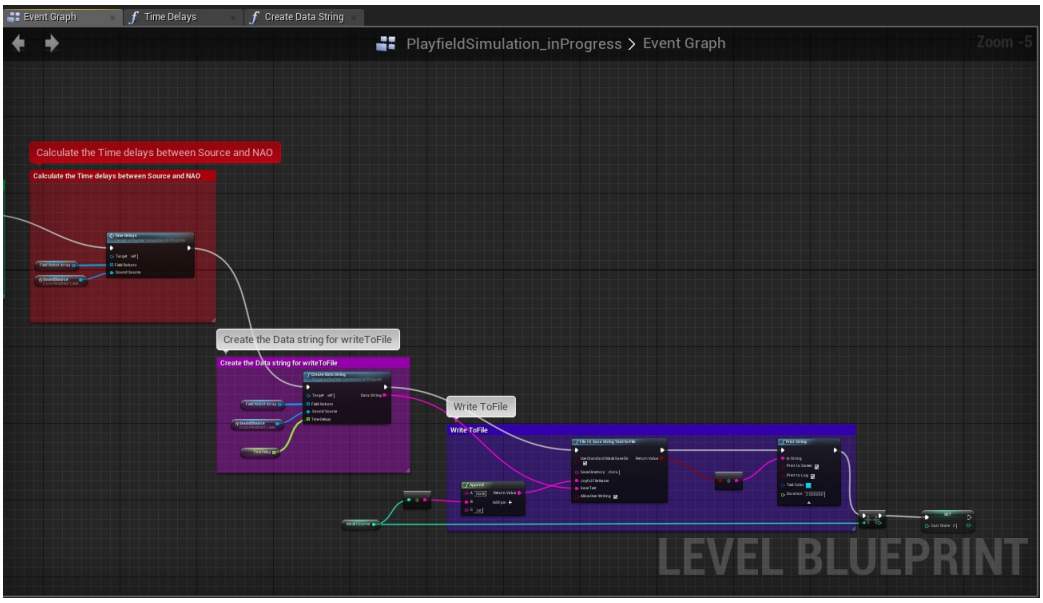


Figure 3: Screenshot of partial Blueprint

Fig. 2 shown the top down view of the simulation, Fig. 4 shows the set up of the simulation and Fig. 5 shows the view of the camera when the simulation is executed. The three robots with the black jerseys are the agents that hear the sound. The single robot without a jersey is the origin of the sound source. The section Sound Delays describes the actual method

[7] https://github.com/JasperVanEck/AfstudeerProject2019
[8] https://github.com/TimmHess/UERoboCup
[9] https://www.unrealengine.com/en-US/what-is-unreal-engine-4
[10] https://share.allegorithmic.com/

of determining the sound delays from the sound source to the NAO agents. The last NAO robot, the one with the camera behind its head and floating in the air, is the cameraman. The cameraman shows the field when running the simulation, allowing us to keep track of the simulation whilst it is running. It does not perform any other tasks and remains static during the simulation.

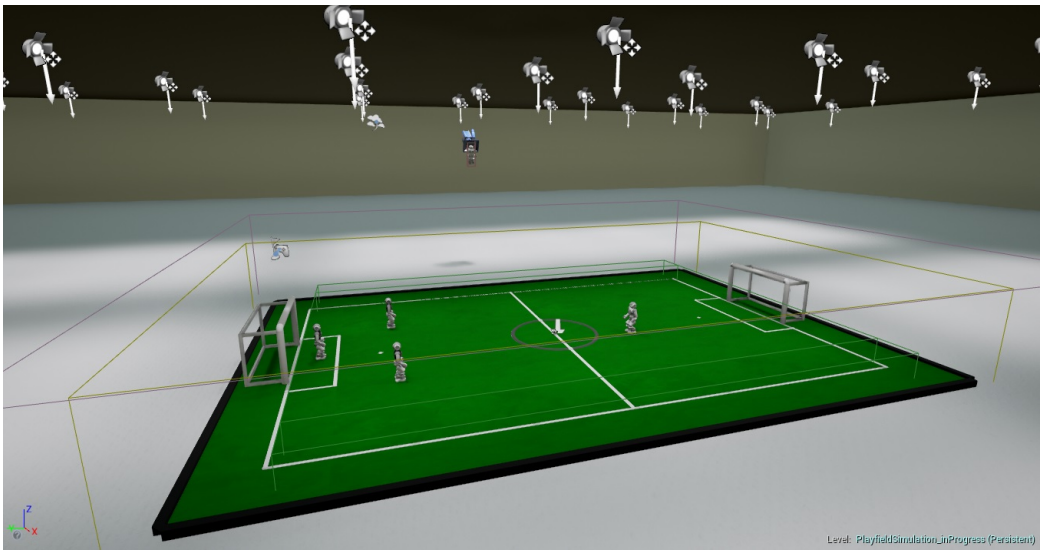

Figure 4: Screenshot of simulation level

The level blueprint contains the logic of the simulation. When the simulation is started, it runs the level blueprint each game tick that occurs. The level blueprint contains a state machine. Each game tick the state is determined based on starting variables or on variables assigned in the previous state. There are four states the simulation can run, the start or set up state, the actual simulation state, a dummy state, and the shutdown state. The setup state ensures that all the lighting in the simulation is working properly, and it also sets up the camera view, so that one may see what is occurring during the simulation. The simulation state takes all the simulation actions required, such as the placement of the hearing NAOs and the sound source location. A mask counter keeping track of the number of game ticks, that after a predetermined amount, the shutdown state of the simulation will be executed, after which the simulation will shut down. The dummy state exists to ensure a smooth transition between states. Without the dummy state, it occasionally happened that the simulation state was not executed properly,

for example, the NAO hearing agents did not get placed correctly. The shut down state ensures that the data is correctly recorded, and the simulation shuts down correctly.


Figure 5: The view of the camera when running the simulation.

### 3.1.2 Location Data

The location data is the data containing the locations of the NAO robots. The robots remain static during each game tick. They are only moved during the set up of a new measuring event. This data is recorded directly by the simulation. The gathered data is ground-truth data. The exact location of the NAO is known and recorded, making the location data the ground-truth. Furthermore, the data is recorded in Unreal Engine units, one unit is equal to one centimeter, making the recorded data in centimeters.

### 3.1.3 Sound Delays

The sound delay is calculated from dividing the distance between the sound source and each robot by the speed of sound. Just as the robots placed, the sound source location is static as well. The elapsed time between the sound source and each robot is then used to calculate the time delay between the robots themselves. After this calculation, an error margin is added to represent real-life conditions. This error is a Gaussian distribution based on the mean and standard deviation of the time delays. To determine how much noise can be present on the time delays before the accuracy of the classifier drops below an acceptable level, different amounts of Gaussian noise are added to the time delays. These amounts range from no noise, meaning the original data, to five times the standard deviation of the time delays. This time delay with the error is used to estimate the sound source location. The next subsection describes the algorithm used for the estimation of the location.

## 3.2 Algorithm

This subsection describes the processing of the data generated in the simulation to prepare it for classification. To proceed with the classification certain data is required. The required data comprises of the NAO listening agent locations, and the estimated location of the sound source location. The coming paragraphs will describe the actions performed on the gathered data, so that it may be used for classification. The algorithm can be viewed in more detail in appendix A.

The first step for the estimation of the sound source location is to determine the function along which the sound source is located. See Eq. 1 in a previous section. $x_B$ is equal to half the distance between the two NAOs, and $AB' = C * \tau$, where C is the speed of sound at $343m/s$ and $\tau$ is the time delay between the NAOs. With these values and a chosen $x$ value, a corresponding $y$ value is generated.

The second step is creating a line that can be intersected with another line. This is accomplished by using two $x$ values to generate their corresponding $y$ values. The minimum value for $x$ is dependent on the distance between the NAOs and the time delay of the soundwave [12]. This is because the Eq. 1 is not defined for certain values of $x$. The values of $x$ for which the function is not defined is expressed at Eq. 2. The negatives of the $y$ values

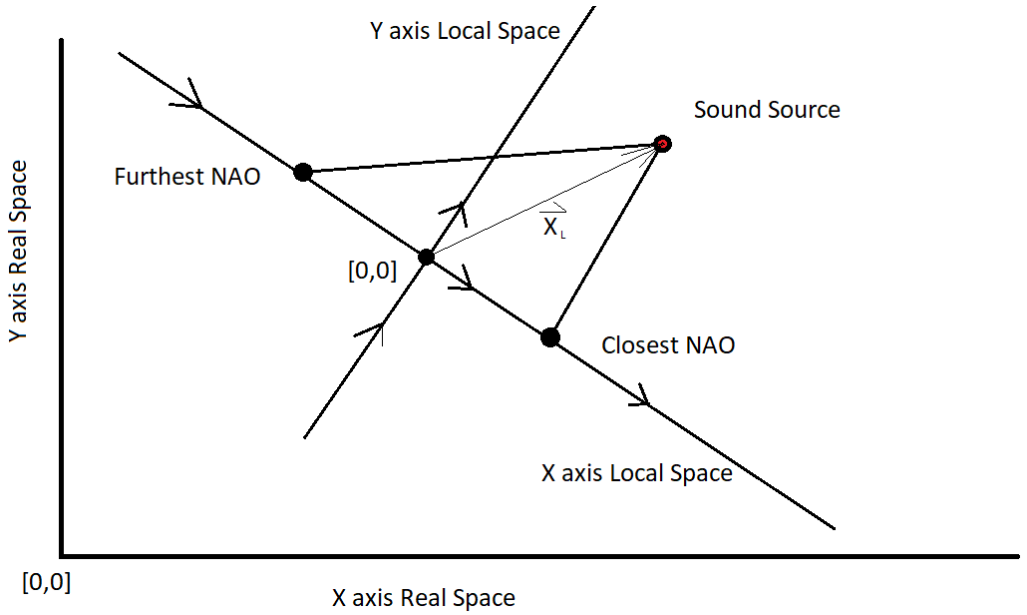


Figure 6: Shows the direction of the vector through the NAO robots in local space.

are also saved since we cannot resolve the direction yet with just two NAOs. With coordinate pairs determined for each NAO pair, the next step can be performed.

$$\begin{bmatrix} X_R \\ Y_R \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \sin\theta \end{bmatrix} \begin{bmatrix} X_L \\ Y_L \end{bmatrix} \tag{3}$$

The third step is the conversion of the coordinate pairs from their local coordinate system to the real space coordinate system in which the NAO locations are represented. This is accomplished with a rotation and then a translation of the coordinate pairs. See Eq. 3 and 4 for the rotation and translation functions, where $X_L$ and $Y_L$ are the coordinates in local space, $X_R$ and $Y_R$ are the coordinates in real space, $M_{Rx}$ and $M_{Rx}$ are the the mid point coordinates in real space between the NAO robots and that is equal to the coordinate $0, 0$ in local space. The angle of the rotation is the angle between the $x$ axis of the real space coordinate system and the vector between the two NAOs, it is represented by $\theta$ in Eq. 3. The direction of the vector is chosen by calculating which NAO is closest to the sound source, based on the

time it takes the soundwave to travel from the sound source to the NAO, see Fig. 6. The direction of the vector is from the furthest NAO to the closest NAO. With the transformed coordinates of the original coordinate pairs, the next step can be taken.

$$\begin{bmatrix} X_R \\ Y_R \end{bmatrix} = \begin{bmatrix} M_{Rx} \\ M_{Ry} \end{bmatrix} + \begin{bmatrix} X_L \\ Y_L \end{bmatrix} \tag{4}$$

The fourth step is the creation of a line representation and the intersection of those lines. The Eq. 1 becomes a straight line by approximation after a certain value of $x$ [12]. Due to this, we can create the representation of a straight line based on the coordinates obtained in the previous paragraph. Line representations are created for the positive $y$ values, and for the negative $y$ values. These line representations can then be intersected, but not all line intersections estimate the sound source location. The lines are all intersected with each other, except for themselves and their negative or positive counterpart. The next step describes the actions taken to ensure the three correct line intersections are used.

The fifth step is determining which three intersections to use for the average. Each intersection is a potential location of the sound source. The average of the intersections gives the best approximation of the sound source location. In the previous step, all the lines were intersected with each other. However, not all intersections represent the sound source location closely, as can be seen in Fig. 7, where the light blue area shows the correct area for intersection. This is due to either a negative or positive $y$ value coordinate pair that does not have the sound source location on its line, which may have been intersected with another coordinate pair. Deciding which intersections are relevant for the average is achieved by applying heuristics. The heuristics applied is the limits of the area in which the sound source could be randomly placed during the simulation. The sound source location was bound by a minimum of -1200cm, and a maximum of 1200cm for both the $x$ and $y$ axis. This means that any intersection coordinate greater than twelve meters or smaller than minus twelve meters of either the $x$ or $y$ axis can be discarded for the average, as the data has been converted to meters from centimeters. The leftover intersections can then be averaged, giving an estimated sound source location. It is not possible yet to determine the three exact lines to intersect; this is further discussed in the Discussion section.

The final step of the data preparation, before the classification, is the addition of a bias, as $x_0$, and the normalization of the data. Normalization is

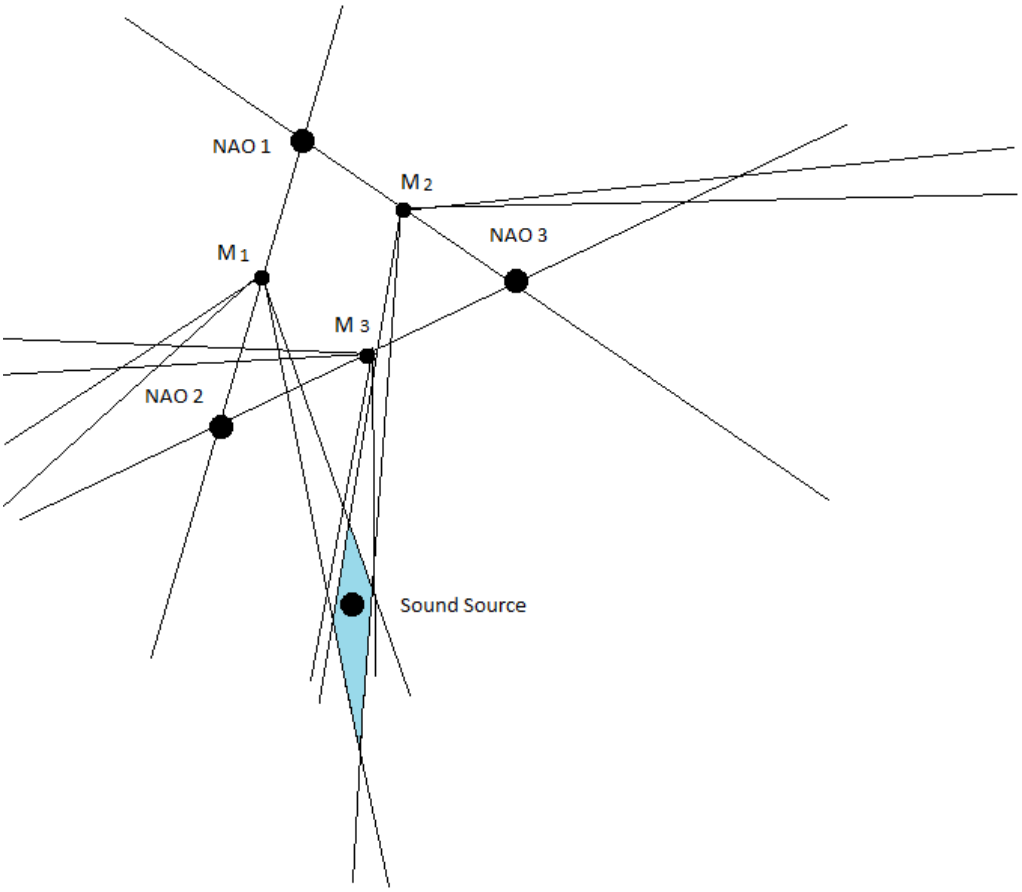


Figure 7: A simplified representation of the uncertainty of intersecting the lines.

not strictly necessary to perform on this data set, as the NAO location data and the estimated sound source location data are both in the same range. However, normalization is still performed to ensure that the model created during the classification stage, can still classify data with a sound source location that is placed much further away than the current sound source locations are, thereby increasing the robustness of the generated model.

## 3.3 Classification

This subsection describes the chosen classifier, the used parameters of that classifier, and the explanation for choosing that specific classifier. Three classifiers have been tested to determine which classifier gives the best results. The three classifiers are Multiple Logistic Regression [7], AdaBoost [5] and K-nearest neighbor(KNN) [4]. For each classifier different parameters were tested, so that the most desired results could be determined. The best parameters were used to compare the three classifiers to determine the most effective one. The resulting scores of the comparison are recorded in the tables located in the Results section.

Each classifier has to be initialized with parameters to form a starting model. These initializations are each started with a different call. The classes the classifiers have to classify are class 0, these are the locations inside the field, and class 1, those are the locations outside the field. The following paragraphs will explain the parameters of each call.

For the call of the Logistic Regression model[11], five parameters are required, see listing 1. The parameter *class_weight*, which is set to $'balanced'$, prevents bias from occurring when the dataset is imbalanced in its classes. The second parameter is *solver*. The solver refers to the algorithm used in the optimization problem. The third parameter is *multi_class* and is set to $'multinomial'$. This parameter allows for the loss to be minimized across the entire probability distribution. The fourth parameter is *penalty*, with the value of $'l2'$. This parameter allows for a regularization penalty to be applied. The final parameter of this call is $fit\_intercept$, set to $False$. Setting this parameter to $True$ would add a bias column to the data. As a bias has already been added to the data, this is not required. To determine the best parameters, all different solvers were tested.

Listing 1: Logistic Regression call.

```
modelSK = LogisticRegression(class_weight ='balanced',
                             solver='lbfgs',
                             multi_class='multinomial',
                             penalty='l2',
                             fit_intercept=False)
```

There are two parameters required for the call of KNN[12], see listing 2. The first parameters is $n\_neighbors$. This parameter determines the amount of its

[11] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[12] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

closest neighbors that are compared against to determine the classification. The second parameter is *weights*, with a value of $'distance'$. This parameter allows the model to use the inverse of the distance between data points as weights, making other points that are nearer have greater influence than points further away. To determine the best parameters, multiple values of *n_neighbors* are tested.

Listing 2: KNN call

```
modelSK = KNeighborsClassifier(n_neighbors=N,
                               weights='distance')
```

There is one parameters required for the call of AdaBoost[13], as shown by listing 3. The parameter required is *n_estimators*. This parameter determines the maximum amount of estimators after which boosting is terminated. If an optimal solution is achieved before all the estimators have been used, the algorithm will terminate. To determine the best parameters, multiple numbers of estimators are tested.

Listing 3: AdaBoost call

```
modelSK = AdaBoostClassifier(n_estimators=N)
```

Another aspect to discuss for the testing of the classifiers is the split of training and test data. The training data is a set of 250 entries, randomly shuffled. The test data is a set of 100 entries, that are randomly shuffled as well. The use of more data entries caused the classifiers to overfit, class 0 was no longer being predicted.

In Table 1 the results of the three best parameter sets are compared. The choice for the best parameter sets is examined in the subsection Classifier Comparison, of section Results. The best choice based on the previously set criteria is between KNN and AdaBoost. The accuracy of Logistic Regression is too low to compete with the other two. The accuracy score of AdaBoost is higher than the accuracy of KNN, however, the F1-score of class 0 is higher than the F1-score of AdaBoost for class 0. Due to relatively low average accuracy, it is deemed that the higher F1-score weights heavier. The more correct class 0 classifications are possible to achieve, the better the NAO robot will be able to operate during a match. Therefore the choice made for which classifier to use is KNN. How the best parameters were determined for each classifier is shown in the results section, more specifically subsection Classifier Comparison.

[13]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

| Classifiers | Acc. | Prec. 0 | Recall 0 | F1 0 | Prec. 1 | Recall 1 | F1 1 | MSE |
|---|---|---|---|---|---|---|---|---|
| LogReg | 0.440 | 0.28 | 0.41 | 0.33 | 0.60 | 0.45 | 0.52 | 0.560 |
| AdaBoost | 0.620 | 0.40 | 0.17 | 0.24 | 0.66 | 0.86 | 0.75 | 0.380 |
| **KNN** | **0.559** | **0.42** | **0.40** | **0.41** | **0.64** | **0.65** | **0.65** | **0.441** |

Table 1: The results with Accuracy, Precision, Recall, F1-scores, Mean Square Error and the amount of test entries for class 0 and 1, for LogReg, AdaBoost and KNN.

# 4 Results

This section describes the results from the parameter comparisons chosen classifier, and the results from noise increase comparison. For the classifier comparison, there will be a brief explanation of how the decision to choose the parameters and each classifier was achieved. For the noise increase comparison, there will be an explanation of the results achieved.

## 4.1 Classifier Comparison

The classifier choices were based on three scores the accuracy, the F1-score for class 0 and the F1-score for class 1 a distant third. Since the average accuracy is rather low, being below 0.6, the accuracy score weights a lot heavier, than if the scores were at least 0.8. As for every small increase in accuracy, the F1-score tends to rise as well, this can be seen in any of the result tables. The F1-score for class 0 is chosen as a second predictor, as it combines the scores of the precision and recall, thereby not requiring the use of those separately. It is also the class we want to predict most accurately, since the sounds classified as inside the playing field, are the ones that must be acted upon. The F1-score for class 1 is used mainly as a tie-breaker when the accuracy and F1-scores for class 0 are equal.

The best parameter based on the accuracy and F1-score for Logistic Regression is a choice between $lbfgs$ and $newton - cg$ solvers, both have the same scores of accuracy and F1. Due to having the same score, the choice was made to continue with $lbfgs$ as the solver to compete against the other 2 classifiers. The results can be viewed at Table 2. The best parameter for KNN, is the use of $n\_neighbors = 2$. Whilst not having the highest accuracy, it does have the highest F1-score. It has almost double the F1-score, compared to other high accuracy parameters. The results can be viewed at

| Solvers | Acc. | Prec. 0 | Recall 0 | F1 0 | Prec. 1 | Recall 1 | F1 1 | MSE |
|---|---|---|---|---|---|---|---|---|
| **lbfgs** | **0.440** | **0.28** | **0.41** | **0.33** | **0.60** | **0.45** | **0.52** | **0.560** |
| sag | 0.420 | 0.20 | 0.24 | 0.22 | 0.57 | 0.52 | 0.54 | 0.580 |
| saga | 0.420 | 0.29 | 0.37 | 0.33 | 0.54 | 0.45 | 0.49 | 0.580 |
| newton-cg | 0.440 | 0.27 | 0.42 | 0.33 | 0.61 | 0.45 | 0.52 | 0.560 |
| liblinear | 0.400 | 0.27 | 0.39 | 0.32 | 0.54 | 0.41 | 0.46 | 0.600 |

Table 2: The results of LogReg, with different solvers, with Accuracy, Precision, Recall, F1-scores, Mean Square Error and the amount of test entries for class 0 and 1.

Table 3. The best parameter for AdaBoost is where $n\_estimators = 16$. It again does not have the highest accuracy, but it does have a observable higher F1-score for class 0. The F1-score is almost five times as high. The results can be viewed at Table 4. The results from these three parameter sets will compete against each other. The results of this comparison have been shown in the subsection Classification of the section Research Method.

## 4.2 Noise Comparison

This subsection will describe the results obtained from the different amounts of noise applied to the data. These results are intended to show how the accuracy changes when the amount of noise, that is added to the time delays between microphones, is increased by multiples of the standard deviation.

The Tables 5 and 6 show the results for class 0 and class 1 respectively. Each table has the precision, recall, F1-score and number of classifications for each respective classification. The tables also contain the accuracy and mean-squared error, both of those are the same for each level of noise in both tables. Fig. 8 has all the values from both tables represented, except for the number of classifications.

In Fig. 8 it is shown that there is no observable decline when the amount of added noise is increased. The only decrease that can be seen in the figure is the decrease of including noise at all. For the increasing amounts of noise, there is no visible decrease, as would be expected. An increase in noise makes data more unreliable, ensuring the model is less representative of the real world experience, which in turn would cause a drop in accurately classifying test data. Why this decrease is not present in the results of this study, could be due to the fact the current classification is quite poor. The average

| K | Acc. | Prec. 0 | Recall 0 | F1 0 | Prec. 1 | Recall 1 | F1 1 | MSE |
|---|---|---|---|---|---|---|---|---|
| **2** | **0.559** | **0.42** | **0.40** | **0.41** | **0.64** | **0.65** | **0.65** | **0.441** |
| 3 | 0.540 | 0.31 | 0.28 | 0.30 | 0.64 | 0.68 | 0.66 | 0.460 |
| 4 | 0.547 | 0.39 | 0.32 | 0.35 | 0.62 | 0.69 | 0.65 | 0.453 |
| 5 | 0.552 | 0.34 | 0.29 | 0.31 | 0.65 | 0.69 | 0.67 | 0.448 |
| 6 | 0.555 | 0.39 | 0.29 | 0.33 | 0.62 | 0.72 | 0.67 | 0.445 |
| 7 | 0.556 | 0.38 | 0.28 | 0.33 | 0.63 | 0.72 | 0.67 | 0.444 |
| 8 | 0.540 | 0.36 | 0.27 | 0.31 | 0.61 | 0.71 | 0.65 | 0.460 |
| 9 | 0.581 | 0.41 | 0.26 | 0.32 | 0.64 | 0.77 | 0.70 | 0.419 |
| 10 | 0.580 | 0.38 | 0.24 | 0.30 | 0.64 | 0.77 | 0.70 | 0.420 |
| 11 | 0.576 | 0.32 | 0.20 | 0.25 | 0.65 | 0.77 | 0.70 | 0.424 |
| 12 | 0.567 | 0.36 | 0.19 | 0.25 | 0.62 | 0.80 | 0.70 | 0.433 |
| 13 | 0.562 | 0.38 | 0.21 | 0.27 | 0.61 | 0.79 | 0.69 | 0.438 |
| 14 | 0.567 | 0.41 | 0.23 | 0.29 | 0.61 | 0.78 | 0.69 | 0.433 |
| 15 | 0.575 | 0.33 | 0.16 | 0.21 | 0.63 | 0.71 | 0.81 | 0.425 |
| 16 | 0.595 | 0.40 | 0.21 | 0.28 | 0.64 | 0.82 | 0.72 | 0.405 |
| 17 | 0.589 | 0.35 | 0.16 | 0.22 | 0.64 | 0.83 | 0.72 | 0.411 |
| 18 | 0.591 | 0.36 | 0.18 | 0.24 | 0.64 | 0.82 | 0.72 | 0.409 |
| 19 | 0.585 | 0.37 | 0.20 | 0.25 | 0.64 | 0.81 | 0.71 | 0.415 |
| 20 | 0.559 | 0.34 | 0.17 | 0.22 | 0.61 | 0.80 | 0.69 | 0.441 |

Table 3: The results of KNN, with different Ks, with Accuracy, Precision, Recall, F1-scores, Mean Square Error and the amount of test entries for class 0 and 1.

accuracy of the three competing classifiers is only 0.54, which is marginally better than flipping a coin. Further discourse on this subject is available in the section Discussion.

| n | Acc. | Prec. 0 | Recall 0 | F1 0 | Prec. 1 | Recall 1 | F1 1 | MSE |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.650 | 0.50 | 0.03 | 0.05 | 0.65 | 0.98 | 0.79 | 0.350 |
| 2 | 0.600 | 0.22 | 0.06 | 0.09 | 0.64 | 0.89 | 0.74 | 0.400 |
| 3 | 0.600 | 0.22 | 0.06 | 0.09 | 0.64 | 0.89 | 0.74 | 0.400 |
| 4 | 0.610 | 0.25 | 0.06 | 0.09 | 0.64 | 0.91 | 0.75 | 0.390 |
| 5 | 0.610 | 0.25 | 0.06 | 0.09 | 0.64 | 0.91 | 0.75 | 0.390 |
| 6 | 0.610 | 0.25 | 0.06 | 0.09 | 0.64 | 0.91 | 0.75 | 0.390 |
| 7 | 0.610 | 0.25 | 0.06 | 0.09 | 0.64 | 0.91 | 0.75 | 0.390 |
| 8 | 0.610 | 0.25 | 0.06 | 0.09 | 0.64 | 0.91 | 0.75 | 0.390 |
| 9 | 0.610 | 0.25 | 0.06 | 0.09 | 0.64 | 0.91 | 0.75 | 0.390 |
| 10 | 0.610 | 0.25 | 0.06 | 0.09 | 0.64 | 0.91 | 0.75 | 0.390 |
| 11 | 0.610 | 0.25 | 0.06 | 0.09 | 0.64 | 0.91 | 0.75 | 0.390 |
| 12 | 0.580 | 0.27 | 0.11 | 0.16 | 0.64 | 0.83 | 0.72 | 0.420 |
| 13 | 0.610 | 0.25 | 0.06 | 0.09 | 0.64 | 0.91 | 0.75 | 0.390 |
| 14 | 0.590 | 0.31 | 0.14 | 0.20 | 0.64 | 0.83 | 0.72 | 0.410 |
| 15 | 0.610 | 0.33 | 0.11 | 0.17 | 0.65 | 0.88 | 0.75 | 0.390 |
| **16** | **0.620** | **0.40** | **0.17** | **0.24** | **0.66** | **0.86** | **0.75** | **0.380** |
| 17 | 0.590 | 0.29 | 0.11 | 0.16 | 0.64 | 0.85 | 0.73 | 0.410 |
| 18 | 0.620 | 0.40 | 0.17 | 0.24 | 0.66 | 0.86 | 0.75 | 0.380 |
| 19 | 0.620 | 0.40 | 0.17 | 0.24 | 0.66 | 0.86 | 0.75 | 0.380 |
| 20 | 0.590 | 0.25 | 0.09 | 0.13 | 0.64 | 0.87 | 0.73 | 0.410 |

Table 4: The results of KNN, with different $n$ estimators, with Accuracy, Precision, Recall, F1-scores, Mean Square Error and the amount of test entries for class 0 and 1.

| | Accuracy | Precision | Recall | F1-score | MSE | N |
|---|---|---|---|---|---|---|
| No noise | 0.59 | 0.35 | 0.49 | 0.41 | 0.31 | 35 |
| 1x Stdr. dev. | 0.50 | 0.33 | 0.43 | 0.38 | 0.50 | 35 |
| 2x Stdr. dev. | 0.54 | 0.37 | 0.43 | 0.39 | 0.46 | 35 |
| 3x Stdr. dev. | 0.48 | 0.33 | 0.46 | 0.38 | 0.52 | 35 |
| 4x Stdr. dev. | 0.55 | 0.38 | 0.46 | 0.42 | 0.45 | 35 |
| 5x Stdr. dev. | 0.51 | 0.35 | 0.49 | 0.41 | 0.49 | 35 |

Table 5: The results with Accuracy, Precision, Recall, F1-scores, Mean Square Error and the amount of test entries for class 0.

| | Accuracy | Precision | Recall | F1-score | MSE | N |
|---|---|---|---|---|---|---|
| No noise | 0.59 | 0.68 | 0.71 | 0.69 | 0.31 | 65 |
| 1x Stdr. dev. | 0.50 | 0.64 | 0.54 | 0.58 | 0.50 | 65 |
| 2x Stdr. dev. | 0.54 | 0.66 | 0.60 | 0.63 | 0.46 | 65 |
| 3x Stdr. dev. | 0.48 | 0.63 | 0.49 | 0.55 | 0.52 | 65 |
| 4x Stdr. dev. | 0.55 | 0.67 | 0.60 | 0.63 | 0.45 | 65 |
| 5x Stdr. dev. | 0.51 | 0.65 | 0.52 | 0.58 | 0.49 | 65 |

Table 6: The results with Accuracy, Precision, Recall, F1-scores, Mean Square Error and the amount of test entries for class 1.

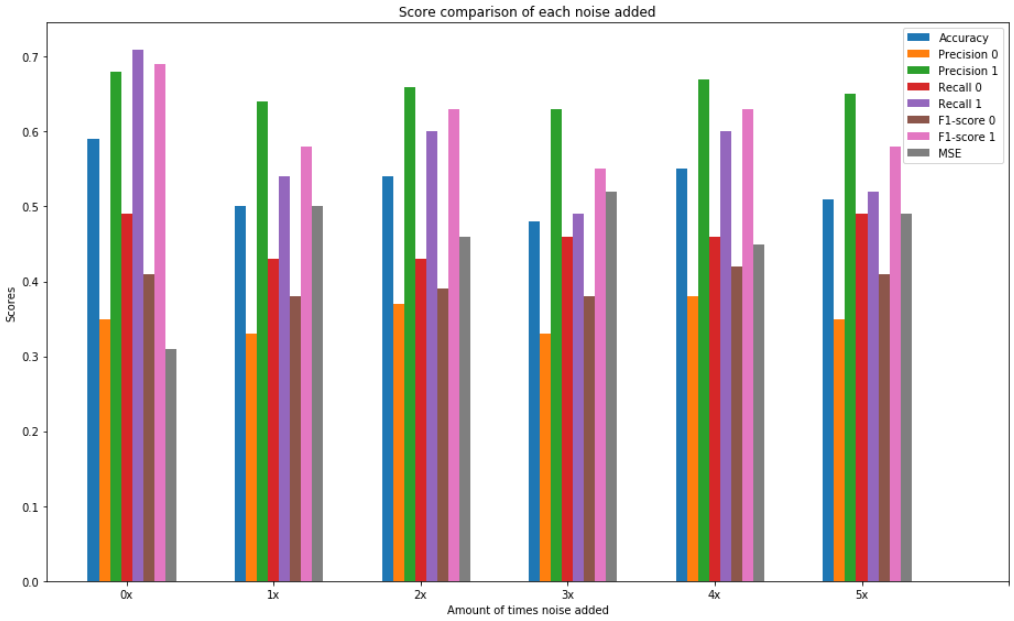


Figure 8: Comparison of all the scores in a bar chart.

# 5 Error Sources

This section will describe errors that have and may have occurred in the data set, influencing the accuracy of the generated model. They will also describe hardware limitations, some of which may introduce errors into the dataset. Other topics are errors that have been avoided due to the use of a simulation rather than real-world experiments.

One of the potential errors that could occur in a real world set up, is the difference in height between the front and back microphones of the NAO. The microphones on the back are placed lower than the microphones on the front. When using multiple NAOs to determine the sound location, this problem will not occur, due to the fact that when measuring sound in general on a NAO, it uses a combination of the two front microphones. Furthermore, this problem will have to be taken into account when estimating the sound source location before using the classifier. When only using the time delays between microphones the model created by the classifier should represent the height difference.

Another error that would occur when using a real world set up, is system time errors. Since the NAOs have to use the internal system time to measure the time delay, a certain amount of noise will occur within the data. Synchronizing the internal system times between the robots would already have eliminated a discrepancy in time measurement. However, the usage of system time will remain problematic. This is because it is unclear what code is exactly running on the CPU of the robot. Meaning it might be possible that the robot is running code from the OS during the hearing as well, creating a delay in the time measurement. This delay is noise in the data. This noise may range from microseconds to milliseconds, making it difficult to measure small time delays [2].

The last two errors are partially related, and actually enhance each other. The first is the sample rate of the microphones on the NAO. And the second is the distance between the microphones on the head of the NAO. The sample rates of the microphones on the NAO are 48k Hz. This means that for every sample taken, the sound has traveled at least seven millimeters, as $\frac{1}{48000} * 343 = 0.00714583$. Meaning there would be at least an error margin of seven millimeters. For larger distances, over several meters, such an error is not significant enough to cause worry. Over smaller distances, it will be more of a cause to worry.

The distance between the microphones on the NAO is small, it is just

over six centimeters[14]. This means that the seven millimeters mentioned previously turns out to be significant. This combination of problems was in this study not a complication, due to the use of three NAOs separately on the field. For future studies and use, it could potentially form a problem. A change in the NAOs themselves would be required to completely overcome the problem. An increase in the sample rate would lessen the error margin, allowing for smaller distances between microphones to be used. And a larger distance between the microphones on the head of the NAO would also decrease the impact of this problem. By, for example, placing them at the outer edge of the head of the NAO. However, it would not be as significant as increasing the sample rate, due to the fact that the NAO is of a limited size. Increasing the size of the NAO, or its head is not a reasonable solution to this problem.

# 6 Discussion

After completing this study, there are several topics that need to be discussed. These topics relate to discussions about choices made in this study. The following paragraphs will detail these topics.

One of the main topics, if not the biggest, topic to discuss in this study is the choice of using a simulation to gather the required data over the use of conventional real-world data gathering. There are three reasons for this choice. The first reason is that it will allow future studies to use and build on the current simulation, allowing for more intensive research into the topic of this study. Secondly, a simulation allows for the possibility to gather more data relatively quickly. Real-world data acquisition requires the set up of the experiment every time the experiment is performed. Also, a real-world experiment might, and most likely will, require manual actions to be performed. Those manual actions require time and effort to complete. This makes it easier to gather more data, faster with simulation and for classification, and for machine learning in general, more data is better. Thirdly, hardware constraints could have made gathering accurate data difficult. A simulation does not have those restraints. In further paragraphs, a more detailed overview of hardware constraints will be given. As a final reason, the replication of the study is eased immensely as well. The creation of new

[14] `http://doc.aldebaran.com/2-1/family/robots/microphone_robot.html`

data comes down to a click of the button with a simulation, rather than a complete set up of an experiment.

A second topic for discussion is the use of version 4.13.3[15], rather than the, at the moment of writing, newest version of the Unreal Engine, 4.22.2. A sample of features that have been added since 4.13.3 are NVidia PhysX improvements, for better physics modeling, improved asset animation and audio improvements[16]. As mentioned in the previous paragraph, the choice for a simulation was partially so that others may continue to build upon it. This study has done the same. It has built upon the simulation of another study [6]. This previous simulation was built using version 4.13.3, and this study continued the use of that simulation. It might be wise and useful to upgrade the simulation to a newer version when a future study continues to work with it. Support for the current version will most likely stop at a point in time, making it difficult to continue developing it. Also newer, and future versions might gain features that are desirable to include in the simulation.

# 7 Conclusion

This study has used the time delay between NAO robots of the arrival of the soundwave, and the ground-truth location of the NAOs to estimate the location of the sound source. The location of this sound source has been classified as either inside or outside the playing field of the NAO robots, with an accuracy of 0.559%. The objective of this study was to show how much noise needed to be applied to the time delay between microphones, for the accuracy to observably decrease.

The results show there is no observable decrease in accuracy when the amount of noise is increased on the time delays between microphones. The only noticeable drop in accuracy is from the no noise model to the one standard deviation noise model, but the no noise model is not representative of real-world conditions. Different classifiers might improve the accuracy, and the F1-scores to a lesser degree, of the resulting classifications.

There are several topics that can be discussed for future works based on this study. Those topics range from several elements that may improve on this study, allowing the current approach to improve, and elements that expand on the approach of this study.

[15] `https://www.unrealengine.com/en-US/what-is-unreal-engine-4`

[16] https://docs.unrealengine.com/en-US/Support/Builds/index.html

One of the additional elements that may improve this study, is the use of different machine learning algorithms. In this study, only multiple logistic regression, KNN and AdaBoost are used. Different algorithms may potentially improve the results obtained in this study. Other algorithms may model the data better than, allowing for a better fit of the generated data. Especially when the sound source location is no longer going to be estimated, and the direct time delays between microphones will be used as features.

Another element that could expand on this study is the use of 3D space. This study only uses 2D space. The NAO hearing agents and the sound source location are all on the same plane, remaining on the same height. Having the sound source location be on different heights would more accurately represent real-world conditions. In a real-world situation, the sound source location will almost never be on the same plane as the microphones of the NAOs. A referee would have to waddle around on his or her knees to be on the same height, which is not practical. Also, whistling sounds may originate from the viewing audience, who are also not likely to be at the same height as the microphones of the NAOs. Another aspect of using 3D space is that not all of the NAOs microphones are at the same height. The microphones on the back of the NAOs head are placed lower than the ones on the front. See Fig. 1 for info on the NAO microphone locations.

A third element that could improve this study is the use of a better heuristic for determining which intersections to use in the average location. In the current algorithm, the only heuristic applied are the limits of the area in which the sound source location could be randomly placed. It is a good first step, but it is not a perfect predictor. An additional heuristic may be getting the average of the leftover intersections after applying the limits heuristic and using this average to determine the distances to the intersections. After having obtained the distances, the largest distances can be discarded, whilst ensuring at least 3 intersections remain for the estimated average. A better estimate of the sound source location will improve the accuracy of the classification. Better heuristics, such as the example mentioned, will improve the estimate made by the algorithm.

# References

[1] Niels W Backer, Arnoud Visser, et al. Learning to recognize horn and whistle sounds for humanoid robots. In *Proceedings of the*

*26th Belgian-Netherlands Conference on Artificial Intelligence (BNAIC 2014).(November 2014)*, 2014.

[2] Randal E Bryant, O'Hallaron David Richard, and O'Hallaron David Richard. *Computer systems: a programmer's perspective*, volume 281. Prentice Hall Upper Saddle River, 2003.

[3] RoboCup Technical Committee et al. Robocup standard platform league (nao) technical challenges, 2019. *https://spl.robocup.org/wp-content/uploads/downloads/Challenges2019.pdf*, 2019.

[4] Thomas M Cover, Peter E Hart, et al. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[5] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[6] Timm Hess, Martin Mundt, Tobias Weis, and Visvanathan Ramesh. Large-scale stochastic scene generation and semantic annotation for deep convolutional neural network training in the robocup spl. In *Robot World Cup*, pages 33–44. Springer, 2017.

[7] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[8] FV Hunt. The past twenty years in underwater acoustics: Introductory retrospection. *The Journal of the Acoustical Society of America*, 51(3B):992–993, 1972.

[9] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents*, AGENTS '97, pages 340–347, New York, NY, USA, 1997. ACM.

[10] Sébastien Negrijn. Exploiting symmetries to relocalise in robocup soccer. Master's thesis, Universiteit van Amsterdam, 2017.

[11] Tim Niemüller, Alexander Ferrein, Gerhard Eckel, David Pirro, Patrick Podbregar, Tobias Kellner, Christof Rath, and Gerald Steinbauer. Providing ground-truth data for the nao robot platform. In *Robot Soccer World Cup*, pages 133–144. Springer, 2010.

[12] Carlos Fernández Scola and Maria Dolores Bolaños Ortega. Direction of arrival estimation: A two microphones approach. Master's thesis, Blekinge Tekniska Högskola, 2010.

[13] Skipper Seabold and Josef Perktold. Statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[14] Merrill I Skolnik. Introduction to radar. *Radar handbook*, 2:21, 1962.

# A Code: Location Estimation and Classification

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 28 13:54:57 2019

@author: Jasper van Eck
"""

import sys
import numpy as np
import math
import random
import getData
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import statsmodels.api as sm

#Constants
SPEEDOFSOUND = 343#m/s

#For 2 NAOs coords, give the distance in m([X,Y],[X,Y])
def naoDistance(nao1, nao2):
    return math.sqrt((nao1[0]+nao2[0])**2+(nao1[1]+nao2[1])**2)

#For a given delay, distance between mics and a given X coordinate, returns the Y coordinate
def locationFunction(X, delay, micDistance):
    ABaccent = SPEEDOFSOUND * delay
    Xb = micDistance / 2
    return math.sqrt((((ABaccent**2)/4)-(Xb**2)) + (X**2 * (((4*(Xb**2))/Xb**2) - 1)))

#Determine the minimum viable X coord to use in the locationFunction
def minXCoord(delay, micDistance):
    ABaccent = SPEEDOFSOUND * delay
    Xb = micDistance / 2
    return math.sqrt(-1*((ABaccent**2 * (ABaccent**2 - 4 * Xb**2))/(4*(4 * Xb**2 - ABaccent**2))))+5

#function to shift coordinates from NAO coords to world/field coords
def coordinateShift(XYnao, XYtarget):
    new = []
    new.append(XYnao[0] + XYtarget[0])
    new.append(XYnao[1] + XYtarget[1])
    return new

#Line intersection coordinates, for linear only, sourced from stackOverflow([[],[]],[[],[]])
#https://stackoverflow.com/questions/20677795/how-do-i-compute-the-intersection-point-of-two-lines-in-python
def line(p1, p2):
    A = (p1[1] - p2[1])
    B = (p2[0] - p1[0])
    C = (p1[0]*p2[1] - p2[0]*p1[1])
```

```
    return A, B, -C

def intersection(L1, L2):
    D  = L1[0] * L2[1] - L1[1] * L2[0]
    Dx = L1[2] * L2[1] - L1[1] * L2[2]
    Dy = L1[0] * L2[2] - L1[2] * L2[0]
    if D != 0:
        x = Dx / D
        y = Dy / D
        return [x,y]
    else:
        return False

#calculate the angle of the sound source[X,Y]
def angleSoundSource(location):
    return 90 - np.arctan((location[1])/(location[0]))

#Calculate the distances between the NAOs
def micDistances(naoLocations):
    distances = [0, 0, 0]
    distances[0] = naoDistance(naoLocations[0:2], naoLocations[2:4])
    distances[1] = naoDistance(naoLocations[0:2], naoLocations[4:])
    distances[2] = naoDistance(naoLocations[2:4], naoLocations[4:])
    return distances

#Determine the mid points of the mics
def midPoints(naoLocations):
    midPoints = []
    midPoints.append([(naoLocations[0] + naoLocations[2])/2, (naoLocations[1] + naoLocations[3])/2])
    midPoints.append([(naoLocations[0] + naoLocations[4])/2, (naoLocations[1] + naoLocations[5])/2])
    midPoints.append([(naoLocations[2] + naoLocations[4])/2, (naoLocations[3] + naoLocations[5])/2])
    return midPoints

#Delays between Mics rather than source and a mic
def delayMics(delays):
    delayMic = []
    delayMic.append(abs(delays[0]-delays[1]))
    delayMic.append(abs(delays[0]-delays[2]))
    delayMic.append(abs(delays[1]-delays[2]))
    return delayMic

#Average the coordinates
def averageCoords(coordsArray):
    x_total = 0
    y_total = 0
    n = len(coordsArray)
    for i in range(n):
        x_total += coordsArray[i][0]
        y_total += coordsArray[i][1]
    x_avgr = x_total / n
    y_avgr = y_total / n
    return [x_avgr, y_avgr]

#Determine which mic of pairings is closests
def closestsMic(delays):
    closests = [0, 0, 0]
    if delays[0] > delays[1]:
```

```
        closests[0] = 1
    else:
        closests[0] = 2

    if delays[0] > delays[2]:
        closests[1] = 1
    else:
        closests[1] = 2

    if delays[1] > delays[2]:
        closests[2] = 1
    else:
        closests[2] = 2

    return closests

#create the nao combo 12, 13, 23
def naoCombo(naoLocations, n):
    naoCombo = []
    if n == 0:
        naoCombo = [[naoLocations[0],naoLocations[1]],[naoLocations[2],naoLocations[3]]]
    elif n == 1:
        naoCombo = [[naoLocations[0],naoLocations[1]],[naoLocations[4],naoLocations[5]]]
    else:
        naoCombo = [[naoLocations[2],naoLocations[3]],[naoLocations[4],naoLocations[5]]]
    return naoCombo

#Sourced from: https://stackoverflow.com/questions/2827393/angles-between-two-n-dimensional-vectors-in-python
def unit_vector(vector):
    """ Returns the unit vector of the vector.  """
    return vector / np.linalg.norm(vector)

#Sourced from: https://stackoverflow.com/questions/2827393/angles-between-two-n-dimensional-vectors-in-python
def angle_between(v1, v2):
    """ Returns the angle in radians between vectors 'v1' and 'v2'::

            >>> angle_between((1, 0, 0), (0, 1, 0))
            1.5707963267948966
            >>> angle_between((1, 0, 0), (1, 0, 0))
            0.0
            >>> angle_between((1, 0, 0), (-1, 0, 0))
            3.1415926535897793
    """
    v1_u = unit_vector(v1)
    v2_u = unit_vector(v2)
    return math.radians(np.arccos(np.clip(np.dot(v1_u, v2_u), -1.0, 1.0)))

#Rotate a vector by certain degrees counterclockwise
def rotateVector(vector, theta):
    c, s = np.cos(theta), np.sin(theta)
    rotationM = np.array(((c,-s), (s, c)))
    #print(rotationM)
    rotated = rotationM @ vector
    #print(rotated)
    return rotated

#translate & rotate from local to realworld coords
```

```
def localToReal(local, naoCombo, midPoint, closests):
    real = []
    angleVec1 = []
    if closests == 1:
        angleVec1 = [naoCombo[0][0]-naoCombo[1][0],naoCombo[0][1]-naoCombo[1][1]]
    else:
        angleVec1 = [naoCombo[1][0]-naoCombo[0][0],naoCombo[1][1]-naoCombo[0][1]]

    #print(angleVec1)
    theta = angle_between(angleVec1, [1,0])
    tmp = rotateVector(local, theta)
    #print(tmp)
    real = coordinateShift(tmp,midPoint)
    #print(real)
    return real

#Create noise to be applied to time delays and add it.
def addGaussianNoise(timeDelays):
    noise = []
    for i in range(len(timeDelays[0])):
        noise.append(np.random.normal(np.mean(timeDelays[:,[i]]), np.std(timeDelays[:,[i]]), len(timeDelays[:,
    return timeDelays + np.array(noise).T

#Create noise to be applied to time delays with multiplier.
def gaussianNoise(timeDelays, mult):
    noise = []
    for i in range(len(timeDelays[0])):
        noise.append(np.random.normal(np.mean(timeDelays[:,[i]]), np.std(timeDelays[:,[i]])*mult, len(timeDela

    #noise = [[i*mult for i in r] for r in noise]
    return np.array(noise).T

#Create model using sklearn
def trainModel(X, Y):
    modelSK = AdaBoostClassifier(n_estimators=100, random_state=0)
    #modelSK = KNeighborsClassifier(n_neighbors=20, weights='distance')
    #modelSK = LogisticRegression(class_weight = 'balanced')
    modelSK.fit(X, Y)

    return modelSK

#Create model using SM
def trainModelSM(X, Y):
    X = sm.add_constant(X)
    #modelSM = sm.OLS(Y, X).fit()
    modelSM = sm.Logit(Y, X).fit()

    return modelSM

#Main Function; calls all other functions & stuff
def main(argv):

    #Retrieve Data & Seperate it in usable arrays
    data = getData.getData(25000)
    random.shuffle(data)
    delays = np.array(getData.getTimeDelays(data))
    classification = getData.getClassifications(data)
```

```
#soundSourceLoc = getData.getSoundSourceLocations(data)
#print(soundSourceLoc)
naoLocations = getData.getRobotLocations(data)

timeDelays = []
for i in range(len(delays)):
    timeDelays.append(delayMics(delays[i]))

multiplier = 0
timeDelays2 = timeDelays + gaussianNoise(np.array(timeDelays), multiplier)


predictedSoundSource = []
#Determine functions of possible locations
for i in range(len(naoLocations)):
    distances = micDistances(naoLocations[i])
    midPoint = midPoints(naoLocations[i])
    #delayMic = delayMics(delays[i])
    delayMic = timeDelays2[i]
    closests = closestsMic(delays[i])
    yPlus1 = []
    yPlus2 = []
    yMin1 = []
    yMin2 = []
    for j in range(len(distances)):
        naoCombos = naoCombo(naoLocations[i], j)
        X_coord = minXCoord(delayMic[j], distances[j])
        X2_coord = X_coord + 20
        shift1 = [X_coord]
        coord1 = locationFunction(X_coord, delayMic[j], distances[j])
        shift1.append(coord1)
        yPlus1.append(np.array(localToReal(shift1, naoCombos, midPoint[j], closests[j])))

        shift2 = [X2_coord]
        coord2 = locationFunction(X2_coord, delayMic[j], distances[j])
        shift2.append(coord2)
        yPlus2.append(np.array(localToReal(shift2, naoCombos, midPoint[j], closests[j])))

        #Negative Part
        shift3 = [X_coord]
        shift3.append(-coord1)
        yMin1.append(np.array(localToReal(shift3, naoCombos, midPoint[j], closests[j])))

        shift4 = [X2_coord]
        shift4.append(-coord2)
        yMin2.append(np.array(localToReal(shift4, naoCombos, midPoint[j], closests[j])))


    #Arrayify the coordinates of possible sound source locations
    intersections = []
    yPlus1 = np.array(yPlus1)
    yPlus2 = np.array(yPlus2)
    yMin1 = np.array(yMin1)
    yMin2 = np.array(yMin2)

    #Create the lines to use for intersection
    lines = []
```

```
    for j in range(len(yPlus1)):
        lines.append(line(yPlus1[j], yPlus2[j]))
        lines.append(line(yMin1[j], yMin2[j]))

    #Intersect all the lines, except for itself and the minus version
    for j in range(len(lines)):
        l = 0
        if j%2 == 0:
            l = j+2
        else:
            l = j+1
        for k in range(l, len(lines), 1):
            intersections.append(intersection(lines[j], lines[k]))

    goodIntsect = []
    for j in range(len(intersections)):
        if (intersections[j][0] < 12 and intersections[j][0] > -12) and (intersections[j][1] < 12 and inte
            goodIntsect.append(intersections[j])

    #print(goodIntsect)
    #print("----")
    #Average the intersection Positions
    predictedSoundSource.append(averageCoords(goodIntsect))

#print(predictedSoundSource)

#Append time delays to locations to form complete matrix of data
#train = np.append(naoLocations, timeDelays2, axis=1)
train = np.append(naoLocations, predictedSoundSource, axis=1)
#train = np.append(naoLocations, soundSourceLoc, axis=1)
#train = predictedSoundSource

#Split data in test and training sets.
testLength = 1000
pre_test = train[-testLength:]
pre_train = train[:len(train)-testLength]
classTest = classification[-testLength:]
classTrain = classification[:len(classification)-testLength]

#Normalize test & training data
norm_train = preprocessing.normalize(pre_train)
norm_test = preprocessing.normalize(pre_test)

#Use calculated sound source location & own location for multiple linear regression
model = trainModel(norm_train, classTrain)
#modelSM = trainModelSM(norm_train, classification)

#Do prediction on test data & print report
classPred = model.predict(norm_test)
#classPredProb = model.predict_proba(norm_test)

#Print the Results
print(model.__class__.__name__)
print("The noise multiplier: " + str(multiplier))
print("The accuracy: " + str(accuracy_score(classTest, classPred)))
print(classification_report(classTest, classPred, target_names=['Inside','Out of bounds']))
print("The Mean Squared Error: " + str(mean_squared_error(classTest, classPred)))
```

```
    print("Cofusion Matrix: ")
    print("(tn, fp, fn, tp)")
    print(confusion_matrix(classTest, classPred).ravel())

if __name__ == "__main__":
    main(sys.argv)
```