# Towards Sensor and Motion Measurements Databases for Training Models for the NAO Robot

Kholud Alghamdi, Jesse Torres, Justin Burden, Venkata Maddineni, Roba Alharbi, Tejeswar Neelam, Peter Tarsoly, Abdullah AlHaif, Joseph Nke, Soham Jadhav, Sai Sohana Dodle, Marius Silaghi

Florida Institute of Technology

## ABSTRACT

High quality probabilistic models for a complex robot like Aldebaran's Nao humanoid depend heavily on details from its environment, involving multiple parameters. Building such models requires significant effort with data gathering and data cleaning. We propose to create a public database of NAO sensor data that can be used by researchers and engineers training models for localization, mapping, and planning in controlled environments. Here we report on our release of a database with sensor data, parameterized by environment *configuration* and *nature*. The database contains structured folders with *documentation*, *measurements*, *Nao AI feedback*, and *data management tools*. The measurements can include transition, sonar, and image data while the internal AI feedback contains results of default landmark detection.
Keywords: Humanoid, framework, model, AI, Nao.

## 1. Introduction

In order to support high level intelligent moving tasks for Nao, high quality models of its sensors and transition models have to be prepared. A database is designed and released with recorded sensor and transition data, as reported here, to support the construction of such models. The models are not solely a function of the electromechanical design of Nao, but also a function of the environment in which Nao acts. Both sonars and vision produce measurements that strongly depend on the materials, textures, and colors of the surroundings. Further, the friction coefficient and stability of the robot depends on the adherence, elasticity, and softness of the floor. The recognition rate of the landmark detection module is sensitive to the Nao-Marks sizes, colors, and positions.

In a previous work on this direction involving some of the authors [1] we have have proposed a model and framework for constructing probabilistic models for the problem of exploration of a labyrinth. These models are designed for integration with high level artificial intelligence techniques, usable for advancing towards a long term goal of providing a software architecture for supporting Nao's motion and localization activities for general tasks. However, the models built in that work used only small amounts of data for training and therefore the error rate of the algorithms using them was too high for achieving a usable behavior.

Other areas of Artificial Intelligence, such as speech recognition and vision, exhibited spectacular improvements in quality after significant work was laid over decades into the gathering of data and creation of clean databases for model training, such as the TIMIT corpus from the Language Data Consortium (LDC) [2]. While the basic techniques for speech recognition and vision (HMMs and ANNs) have largely existed for many decades, it was the creation and wide availability of these databases that enabled the development of the high quality probabilistic models that can handle reasonably the challenges posed by real applications.

A similar situation is present now in robotics, where advanced algorithms are available for planning, localization, and mapping, but where accurate probabilistic models of reasonable robots are absent. Nao is a skilled robot which has advanced mechanical and computational resources, but where quality models of its interaction with the environment are essential for the advancement of the technology towards a status that is sufficiently usable to be brought to the large public.

Aldebaran's Nao is a humanoid robot that can be a human companion and is appropriate for household environments. It has small dimensions which do not offer him the strength, height, and dexterity to open doors or perform other mechanical tasks. In home, hospital, and office environments it can provide, for example, entertainment or emergency support for children and seniors. While its communication and emotional intelligence is meeting many expectations, its movement and ability to localize with standard software is reduced, and insufficient for most other tasks. A number of research efforts have led to the description of successful applications of *simultaneous localization and mapping* (SLAM) to certain tasks such as wall following in mazes and localization in spaces filled with landmarks [3]. However there is a lack of general software packages applicable to new scenarios. In this work we take additional steps towards designing such a general mapping and localization support software architecture.

## 2. Nao Sensors and transition

Nao robots have 25 joints for robot motion that include HeadYaw, HeadPitch, ShoulderPitch, ShoulderRoll, ElbowYaw, ElbowRoll, HipYawPitch, WristYaw, Hand, HipRoll, HipPitch, KneePitch, AnklePitch, and AnkleRoll. Each one has two sides (right and left) except for three joins, namely HeadYaw, HeadPitch, and HipYaw-Pitch. These joins are responsible for the movement of the robot and, for introspection, each of them has five sensors which include position, temperature, electric current, stiffness of the join sensors, and position of the actuators. However, these are not the only sensors that the NAO robot has. There are two HD cameras, four microphones, and ten touch sensors.

The sonar sensors are particularly relevant to us, as they are important for NAO's navigation. NAO robots have two sonar sensors that help to calculate the distance between the robot and the obstacles ahead of the robot. These have the ability to detect and measure distance to objects in the range between 20 cm and 80 cm (according to Aldebaran website for NAO robots v3 and v4).

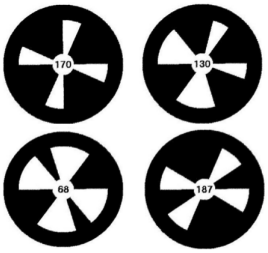


**Figure 1.** Nao Marks

Above or under this detection range, the object distance cannot be calculated.

There are four different events generated by the sonar sensor drivers. First, the SonarLeftDetected event, which signals that there is an obstacle in front-left side at less than 50 cm, so NAO may need to stop and turn right to avoid the obstacle. Next, the SonarRightDetected event signaling that there is an obstacle in front-right side and NAO may need to stop and turn left to avoid hitting the object. Further, SonarNothingLeftDetected signals there is nothing in the front-left side, and SonarNothingRightDetected, which signals there is nothing in the front-right side.

Additionally, we use ALLandMarkDetection, which is a vision module that uses a NAO camera to recognize special patterns called landmarks or NaoMarks (see Figure 1). Those landmarks are used to get the accurate robot position. The landmark is a dark disk that has white triangle fans inside it. The white triangle sizes are different to distinguish between landmarks.

## 3. Framework for Sensors and Transition Data

The studied labyrinth consists of connected square cells with interior sides of 20 inches. Each wall is 2 inches thick. For recording a sensor model, we program Nao to take various positions is a labyrinth cell with 3 walls and an open side. Each wall has a NAOmark pair at the level of the head. We save the sonar and landmark detection readings at each position. Each position is taken 10 different times, recording each time separately the corresponding measurements. At the moment of each measurement, the head is looking straightforward, as the landmark detection is associated with the direction of the head, and the sonar reading is associated to the orientation of the body.

The floor of the cell is split into a safely reachable and an unreachable area. The unreachable area is within 6 inches from the walls. The central reachable area is 8 inches wide and 16 inches long, namely in the direction of the cell opening. The safely reachable area is split into squares with sides of 2 inches. The sensor measurements are made in the centers of each square. The robot is oriented once in each direction that forms angles that are a multiple of 5 degrees with respect to the sides of the squares.

The conditional probabilities of each of the sensor readings given each position an orientation can be estimated from these measurements. Measurements of the accuracy of Nao movements are also taken as measurements of results of walking movements intended to be between such adjacent squares.

## 4. Procedure For Sensor Data Gathering

In order to obtain sufficient data for localization, we programmed the robot to take ten measurements at each position and orientation within a single labyrinth 20in x 20in cell. As previously explained,

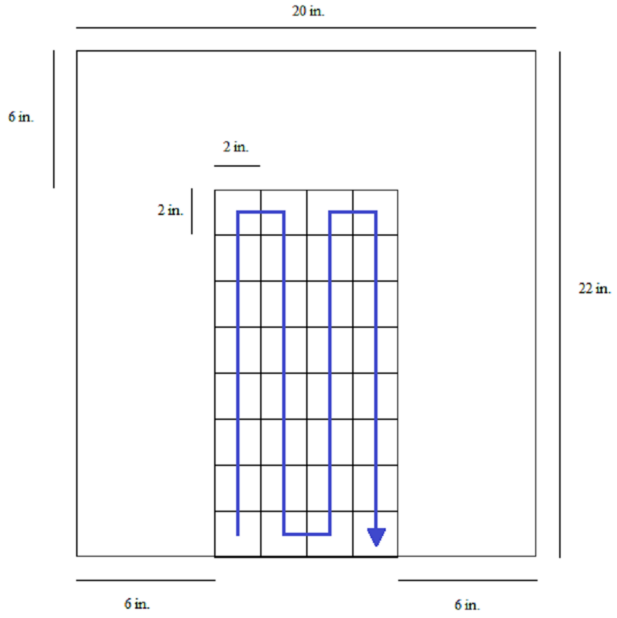


**Figure 2.** Data Gathering: The Nao robot travels on this path taking measurements at each 5° to 11.25° orientation in each of the squares.

the cell is divided into 2in x 2in squares, in which the center of the NAO robot cannot be within 6 in of a wall.

An example path the NAO robot took through the 2 in squares in the labyrinth cell is displayed in Figure 2. In each of the 2 in squares, the robot makes a 360-degree rotation, taking sonar and landmark readings every five or degrees. Due to inaccuracy in the robot's movement, the robot may move more than ten degrees on a rotation. Based on our measurements, our program asks the NAO library that the robot should rotate only a total of 320 degrees, request that will have it rotate 360 degrees in practice. For each measurement, the position and orientation are stored in the database with the sensor readings.

The recorded measurements include readings from both the left and right sonars and the landmark detector. The program includes several instances of the NAOqi module ALProxy, which allow it to take these measurements, in addition to movement functionality. Among these instances are a memory proxy, which gets data from NAO's subdevices such as the sonars and head rotations, and a landmark proxy which gets data on which landmarks, if any, are detected. The memory proxy obtains the data of the left and right sonars, which represent measured distances between the robot and the wall, and stores these in the database. Then, a boolean value is stored indicating whether a landmark was detected, and if any were detected, then measurements on the ID, dimensions, and direction of view for each landmark are stored in the database.

## 5. Procedure For Transition Data Gathering

We represent the current square and the adjacent squares as a 5x5 matrix, with the initial position is in the center of the matrix. This allows to record the case where a requested move of one square ends up being a move of two squares.

The robot was programmed to rotate clockwise, and counterclockwise, and to move to adjacent squares. We start giving in-

structions by selecting the orientation and specifying the goal position, to which robot has to go. Then we record the actually obtained position.

## 6. Database Folders

The Transition data and Sensor data is organized in our database[1] for which robot type according to the following hierarchy:

```
/<Env>/<Model>/<Collection>/Type[_run].extension
```

Where,

```
Env := Env1 | Env2
Model := Sensor | Transition
Collection := Code | Data | Documentation
File := readings|cpt
Extension := .txt | .csv | .py | .c
Examples :
/Env1/Sensor/Code/Method/SensorFusion/
/Env1/Transition/Code/ParticleFiltering.c
/Env1/Transition/Data/cpt_01.csv
/Env1/Sensor/Data/readings_05.txt
```

The Env segment identified the physical environment described in the documentation. For the data described so far, Env1 is described as a NAO v5 in a labyrinth with walls of plastic cloth, and hard linoleum floor.

Our database contains transition data and sensor data. Transition data is defined as when the robot tries to move between two adjacent squares, each 2inx2in, in the labyrinth. The sensor data consist of measurements about landmarks and sonars detecting walls. Each move/turn is recorded multiple times.

As an example, the data formats in the Env1 are described below:

*Transition Data.* The transition data is a 270 number dataset. The dataset is given by four columns:

- Given_Position is defined as the position the robot requests to reach
- Given_Orientation is the requested orientation of the robot at the destination cell.
- Recorded_Position is where the robot moves from the center of the cell to the destination cell and
- Recorded_Orientation is the final orientation of the robot at the destination cell (see Table 1).

*Sensor Data.* Sensor data is a 5120 number dataset defined by 32 columns. Column and row are the position of the robot in the cell. Angle is the degree of the rotation. Head Pitch and Head yaw represents the Orientation of the Head. Sonar left and Sonar Right are the sensor reading of the sonar. Landmark detection represents whether the landmark is detected or not. Two Landmarks are placed on each wall. So, the robot detects two landmarks: Landmark1, Alpha1, Beta1, Width1, Height1, landmarkID2, Alpha2, Beta2, Width2, Height2.

For each landmark position robot detects width and height of the landmark. Alpha and Beta are the line of sight angle of landmark from the position of the robot. If the Robot is at the corner of a labyrinth cell, it sometimes detect two other landmarks on the other wall so we have LandmarkId3, Alpha3, Beta3, Width3, Height3, LandmarkID4, Alpha4, Beta4, Width4, Height4. Sample measurements are shown in Table 2, where only one out of the 4 landmarks is observed.

Delta shows the horizontal and vertical angle under which the landmark is seen (the value reported by NAO being always equal to the largest of the two).

In a set of measurements runs, we used three additional columns, namely for the requested head pitch and yaw values, which can help learn discrepancies between requested and actual values for NAO actuators.

[1] `github.fit.edu/RoboDB`

## 7. Data Formats for Sensors

In order to programmatically access the sensors and transitions on the NAO humanoid, numerous SoftBank NAO SDKs were considered. However, due to its widespread usage and adoption within the AI and Machine Learning communities, Python was chosen as the primary language and SDK for development. Using the Python SDK, the main python module utilized was ALProxy, alluding to the effective Proxy design pattern implemented within the SDK. ALProxy is passed parameters regarding the IP address of the NAO, the port number of the broker, and the exact name of the NAO module being utilized. The usage of NAO modules can be classified into two separate categories: modules used for sensors and transitions.

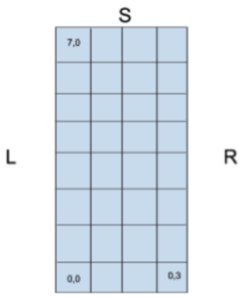


**Figure 3.** The layout of the area where the sensor data was collected. Each square was two inches in length and width, with each of the boundaries six inches away from a labyrinth wall. The letters represent the orientation seen in the data; additionally, each letter represents a wall with two NAO Landmarks.

Each CSV file has the following headers: runid, column, row, orientation, headOrientationYaw, actualYawU, actualPitchU, actualYawL, actualPitchL, leftSonar, rightSonar, alpha1U, beta1U, dalU, db1U, nb1U, alpha2U, beta2U, da2U, db2U, nb2U, tU, NU, alpha1L, beta1L, dalL, db1L, nb1L, alpha2L, beta2L, da2L, db2L, nb2L, tL, and NL.

The runid corresponds to the run number of the specific trial. Column and row correspond to the numbers seen in the boxes in Figure 3. However, there is a caveat to this system; these numbers are relative to each of the orientations S, L, and R. For example, 7,0 corresponds to the top left square when the orientation is S, or straight. If the orientation is L (left), for instance, 0,0 is where 0,3 is written in Figure 3. If the orientation is R (right), 0,0 is where 7,0 is written in Figure 3.

The headOrientationYaw field is the position of the head, measured in degrees away from the head facing the same forward position as the body. The other yaw and pitch values are recorded when ALLandMarkDetection is called. For these values, and other values

| Given_Position | Given_Orientation | Recorded_Position | Recorded_Orientation |
|---|---|---|---|
| 0 | left | 17 | left |
| 0 | right | 2 | right |
| 0 | still | 0 | still |
| 1 | left | 3 | left |
| 1 | right | 2 | right |
| 1 | still | 1 | still |

**Table 1.** Sample Transition Data

| Measurement | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Column | 0 | 0 | 0 | 0 |
| Row | 0 | 0 | 1 | 1 |
| Angle | 0 | 10 | 10 | 20 |
| HeadPitch | N/A | N/A | N/A | N/A |
| HeadYaw | N/A | N/A | N/A | N/A |
| Sonar Left | .46 | .47 | .45 | .45 |
| Sonar Right | .48 | .51 | .47 | .30 |
| LandmarkDetected | False | False | True | True |
| LandmarkID1 | 0 | 0 | 68 | 68 |
| Alpha1 | 0 | 0 | -.0089 | .2294 |
| Beta1 | 0 | 0 | -.0649 | -.0935 |
| Width1 | 0 | 0 | .098 | .103 |
| Height1 | 0 | 0 | .098 | .103 |
| Delta | 0 | 0 | .01 | .008 |

**Table 2.** Sensor Data

associated with landmark detection, the U and L denote measurements taken from the upper and lower cameras built into NAO's head.

The fields for the sonar distances, leftSonar and rightSonar, are measured in meters. The measurements for landmark detection make up the remainder of the fields. The numbers 1 and 2 for each of these fields correspond to the values associated with each of the landmarks detected by NAO, as two landmarks were placed adjacent to each other on each of the three walls. The alpha and beta fields correspond to the angles of the landmarks, and the da and db fields respectively correspond to the height and width of the landmarks, all with respect to the specific NAO camera (upper or lower). The t field corresponds to a timestamp in seconds and microseconds that the respective landmarks were found. Finally, the N field corresponds to the number of landmarks detected by the respective camera.

All previously mentioned sensors have associated NAO modules that can be utilized to gather sensor data. In order to collect data from each of the sonar sensors, ALSonar was used. To collect information regarding landmark detection, ALLandMarkDetection was used. However, sensor data is not directly collected by simply calling the aforementioned modules prior to gathering any sensor data. After calling the desired module, via ALProxy, to capture the sensor data, the collection of data was completed through calling ALMemory. The resultant proxy object generated from calling the ALMemory module from ALProxy allowed for the collection of data using the getData function.

In order to make transitions, ALMotion was utilized. The combination of the different functionality within this module allowed for control over the orientation and position of both the robot body and robot head. In order to move the robot, the moveTo function was used, where the movement desired along with desired end orientation was passed in. Additionally, prior to each movement, the moveInit function was called. Similarly, directly following each move, the killMove function was called. Finally, in order to make transitions with respect to the head orientation, the angleInterpolationWithSpeed function was called in order to effectively set the head orientation. Utilizing the angleInterpolationWithSpeed and moveTo functions allowed for transitions defined by inches of translation and degrees of rotation.

As seen, each run successfully resulted in the collection of sonar values. However, there is a sparsity in the number of landmarks recorded; this is because the NAO simply did not detect landmarks during a majority of samples detected. If a landmark was not recorded, every value after rightSonar in each respective CSV row is populated with a zero.

## 8. Data Formats for Transitions

The amount of the data that has been collected for evaluating movements is 270 measurements, which consist of 10 rounds where each round has 27 measurements, which are the outcome of 9 squares times 3 orientations. There are three orientations that are considered, which are straight (S), left (L), and right (R). The map that is used in this experiment contains 9 destination squares, and these squares were numbered from 0 to 8. The robot has to go to every square from 0 to 8 to complete one round. The data that was collected for the transitions were saved in the csv files by using the following function. As a result, the data is stored in csv files.

```
with open('motion_data.csv', 'a') as csvfile:
    fieldnames = ['directedSquareNumber',
                  'directedEndOrientation',
                  'actualSquareNumber',
                  'actualEndOrientation']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

**Figure 4.** The function that is used in saving the csv files.

When the program is run, some steps were followed, which start with entering a square number to move to and the orientations of the robot. Then, waiting until the robot moves and after that other two questions are showed up, which ask about the square number and the orientation that the robot ended up in.

After collecting these data, one may calculate the transition probabilities between states, given action decisions. That is calculated by summing up the counts that ended up in each target square/orientation, and then dividing the result by the total number of runs, for the given action. These transition probabilities predict the end state statistically.

## 9. Sample use of data for localization

Particle filtering allows tracking nonlinear systems and dealing with non-Gaussian problems, and is utilized for simultaneous localization and mapping. It can be used for large dimensional

```
Square numbers (4 is the starting square)
0  1  2
3  4  5
6  7  8
What square number to go to? (0-8): 0
What should the ending orientation be? (L, S, or R): S
Moving up
Upward movement should now be complete
Moving left
Left movement should now be complete
What square number did the robot end up in? (0-8): 0
What ending orientation did the robot end up in? (L, S, or R): S
```

**Figure 5.** The execution of the program.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | 0 | S | 0 | S |
| 2 | | | | |
| 3 | 0 | L | 17 | L |
| 4 | | | | |
| 5 | 0 | R | 2 | R |
| 6 | | | | |

**Figure 6.** Transitions Map

state spaces. Here we give an example of implementing localization for our model using particle filtering with data trained from our database. It works by producing samples "particles" from $p(x_i|e_1,\ldots,e_i)$ for approximating $p(x_{i+1}|e_1,\ldots,e_{i+1})$, where $x_i$ are random variables describing the state at time $i$, and $e_i$ are evidence variables measured at time $i$.

We show how localization is applied to maintain robot situation awareness during a predefined sequential plan.

At each step of the predefined sequential plan, the robot is given a new action, such as going straight, or turning left or right, at a certain angle. Then, we read the current sensor measurements: sonar, landmarks list and position.

Based on the map and model that was trained from data, a probability distribution is initialized and maintained over the possible states of the robot.

A set of particles is generated based on the current probability distribution. Each particle has a state (the position of the robot) and a weight. The particles are updated for the performed transition based on the taken action. The measured sensors are used to associate a weight with each particle, based on its likelihood. Further, a new distribution is inferred from these weighted particles, effectively making the robot aware of the new situation.

The repetition of the process is also possible, for the next action in the sequential plan.

The first thing we do, we sample a particle from the current distribution. A distribution can even be represented with a set of weighted particles: $< x_t, w_t >$, interpreted as a Gaussian mixture.

Then, one can sample a new states $x_{t+1}$ using the previous sample $< x_t >$ and the control (action) $u_t$. After that, one re-weights that sample by measuring how likely the measurement have been if that $x_{t+1}$ is actually where the robot was. This become the particle's new weight. The sample weights are further normalized and identical states are aggregated.

We assume the robot has to go through three cells of the labyrinth, and it has to localize itself while walking through this environment. That is done by utilizing information from sonars and landmarks spotted on the walls of the cells. As discussed previously, the space inside that cells is divided into small 2 in squares. We have 3 cells of the labyrinth, looking like in Figure 8.

The sample sequential plan used in this example is [ F F F F F F F F RL45 D D D RL45 F F F F F F F F ], where $F$ is the action

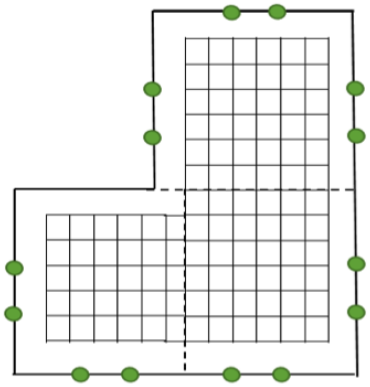

**Figure 7.** Model of the world where the robot acts.

**Figure 8.** Model of the world where the robot is supposed to be.

```
for row in sensorDataList:
    colNum = int(row[0])
    rowNum = int(row[1])
    left_samples[rowNum].append(
        float(row[5])*meter_to_inch_conversion)
    right_samples[rowNum].append(
        float(row[6])*meter_to_inch_conversion)
```

**Figure 9.** Parsing Sonar Data

for going forward 2 inches, RL45 is the action for turning left 45 degrees, and $D$ is the action for going forward $2\sqrt{2}$ inches.

Based on this plan the robot should go 8 steps forward, then turn right 45 degree, then make 3 steps forward on diagonals, then turn right 45 degree, then go 7 more steps forward. We keep running this plan and as we are running the plan the robot is measuring the sonars and landmarks. This is would be a map and a configuration of the problem.

At the very beginning, the robot knows where it is, and in our case the robot starts at (3,12) which on an edge with the back to the wall. The hidden variable is a position and a rotation. The position is given by the current square. The possible rotations are multiples of 5 degree. We have approximately 100 cells *360/5 =7200 possible states.

The whole problem is a dynamic belief network (DBN), To start with particle filtering, all the particles at the beginning are in (3,12). We started by generating 1000 particles which is a rather small number for the given space size.

If based on the collected database we have a transition matrix with 0% probability of transiting to an adjacent node, we subtract

```
sonarLeft = memoryProxy.getData("Device/SubDeviceList/US/Left/Sensor/Value")
sonarRight = memoryProxy.getData("Device/SubDeviceList/US/Right/Sensor/Value")
row.append(sonarLeft)
row.append(sonarRight)
```

**Figure 10.** Fetching Sonar Data

```
def gaussian(x, mu, var):
    return np.exp(-np.power(x - mu, 2.) / (2 * var))/np.sqrt(2*np.pi*var)
```

**Figure 11.** Regression on Sonar Data

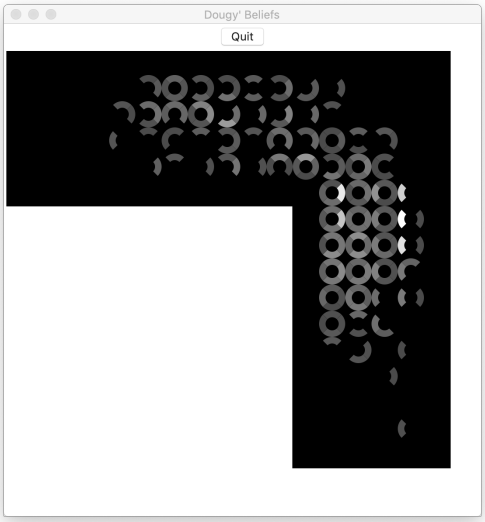


**Figure 12.** Belief Map Localizing NAO

a fraction (i.e. $\alpha = 10\%$) of the probability mass of existing target states, to distribute to all states within 4 inches distance. When we sample the first transition, with approximately 90% likelihood the next node is (3,11) with 0 degree for orientation, and approximately 100 particles are distributed on the surrounding areas which are left, right, and diagonally, with adjacent orientations.

The Figure 12 shows a snapshot of the belief map where whiter doughnut segments correspond to location and orientations of higher likelihood, white darker area illustrate lower location probability. In the example of the figure, the NAO robot "Dougie" is most likely close to the middle of the right wall, facing left.

## 10. Data Processing

CSV data files containing the sensor and transition data were generated by using a writer object from the Python CSV module (Python v2.7). The algorithms used for generating the sensor and transition data collection are described in Sections 4 and 5 respectively. The CSV data formats are described in detail in Sections 7 and 8. Figure 10 shows an example of how the writer object is used to hold data obtained from the Nao sonar sensor into the "row" object, which is a Python list. Similar methods were used to capture landmark detection data from the Nao ALLandMarkDetection API, and to capture movement transition data using manual input.

An example of the way the data can be used is described below. The sonar data is read from the CSV file and the mean and variance of the sonar readings are calculated for each row of the 2x2 inch squares, using the Python numpy library. Data is first loaded into numpy arrays "left_samples" and "right_samples" as shown in Figure 9.

The numpy "average" and "var" functions can then be used to calculate the mean and variance for each row of squares. For example, for one subset of data this method calculated the mean and variance for the second row of squares from the wall (9 inches from the wall) to be 8.31015 and 3.26334. This data can then be used in a Gaussian function, shown in Figure 11, to perform sensor fusion to estimate the probability of the Nao robot's actual location being in the second row of squares given a sonar measurement.

Additional conditional probability tables can be computed using the landmark detection module data by summing the landmark detection counts to compute discrete probability tables for Bayesian inference.

## 11. Related Work

The problem of simultaneous localization and mapping (SLAM) for a Nao robot placed in a room with multiple Nao marks placed at random locations, at the height of the cameras of Nao, was addressed in [3]. The effort led to new SLAM approaches for real-time incorporation of new landmarks in exploration.

Probabilistic reasoning for localization has been used with mobile robots to address various types of problems. If the map of the explored world is known, then Monte Carlo Localization can employ particle filtering [4].

Motion planning in partially observable non-deterministic environments of this type can be modeled with Partially Observable Markov Decision Processes (POMDPs), for which various techniques have been proposed to tame the complexity challenges. All these techniques can employ a dynamic belief network representation of the problem [5].

## 12. Conclusion

In this work we propose to build a database of sensor and transition data describing Nao's behavior in a standardized environment, together with documentation and tools, that can support the training of high quality probabilistic model of the humanoid robot Nao sensors and actions, enabling the application of high level intelligent algorithms for tasks such as localization, mapping, and planning.

In previous work we had reported that sensors and movements of Nao have significant noise and it was difficult for him to reliably move using the models obtained with preliminary measurements. The database proposed and released in this report is a further step towards the development of reliable models of the type that large and high quality speech databases enabled for the area of speech recognition.

## References

[1] A. Hasanain, T. Weekes, M. Person, K. Paul, Y. Chang, A. Rothman, Z. Rui, R. Rahil, M. Syed, C. Wodd, and

M. Silaghi, “Models and framework for supporting humanoid robot planning & exploration,” in *FCRAR*.

[2] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, “Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1,” *NASA STI/Recon technical report n*, vol. 93, 1993.

[3] Y. Zhang, *Real-time SLAM for Humanoid Robot Navigation Using Augmented Reality*. PhD thesis, Applied Sciences: School of Mechatronic Systems Engineering, 2014.

[4] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, “Robust monte carlo localization for mobile robots,” *Artificial intelligence*, vol. 128, no. 1-2, pp. 99–141, 2001.

[5] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.