

# **Important Copyright Notice:**

The provision of this paper in an electronic form in this site is only for scholarly study purposes and any other use of this material is prohibited. What appears here is a near-publication draft of the final paper as appeared in the journal or conference proceedings. This is subject to the copyrights of the publishers. Please observe their copyrights.

# Map Building for Mobile Robots, Using a Hybrid Neural-Bayesian Fusion Approach

Reza HoseinNezhad  
Ph.D.

Department of Elec. & Compt. Eng  
Faculty of Engineering  
University of Tehran  
Tehran, IRAN  
rhnezhad@ut.ac.ir

Behzad Moshiri  
Associate Professor

Department of Elec. & Compt. Eng  
Faculty of Engineering  
University of Tehran  
Tehran, IRAN  
moshiri@ut.ac.ir

Mohammad Reza Asharif  
Professor

Department of Information Engineering  
Faculty of Engineering  
University of the Ryukyus  
Okinawa, JAPAN  
asharif@ie.u-ryukyu.ac.jp

School of Intelligent Systems (SIS)  
Institute for studies in theoretical Physics and Mathematics (IPM)  
Niavaran, Tehran, IRAN

## Abstract

Environment perception is a well-known problem to be solved in AI. In this paper a hybrid fusion approach for mapping the environment in to an occupancy grids map is introduced. The robot which is applied for mapping experiments, is the miniature Khepera robot, equipped with only 8 infra red proximity detectors. The sensory data are firstly fed into the inputs of a feed-forward perceptron neural network and the output of this neural fusion is a local map of the environment, around the robot. The different local maps that are generated by the robot, locating in different positions, are fused by the Bayesian fusion method. This combination of neural and Bayesian fusion is a global map of the environment. Such a map will be useful for planning and localization purposes.

## Keywords

Sensor data fusion, Environment understanding, Intelligent perception, Bayesian fusion, Neural fusion

## I. INTRODUCTION

THE “traditional” approach for developing mobile robot control systems is to design them carefully by hand, using knowledge about robot, world and using task solution strategy. Thrun [1] lists three bottlenecks faced by these non-learning strategies: 1- Knowledge bottleneck: Acquiring knowledge about the robot 2- Engineering bottleneck: Acquiring knowledge about the robot, its world and the task to solve can be an insurmountable difficulty. The environment and the robot’s sensors and motors are never totally predictable. 3- Tractability bottleneck: The computational complexity of the control program can be a last obstruction.

Advantages of the non-learning approaches are that the programmer, through forced to, learns about the problem and its solution and development time still seems to be shorter than for the learning methods. In the last decade machine learning has been applied with some success in robotics (e.g. Maes and Brooks [2], Floreano and Mondada [3], Lund and Hallam [4] on admittedly, very simple problems. ML<sup>1</sup> strives to overcome the difficulties of the traditional approaches by, in its purest form, collecting knowledge through interaction with the world (using multiple sensors) and including appropriate models and behaviours. Since the overall goal of this research program is to study and examine sensor fusion methods and their applications in machine learning for robotic tasks, it would be necessary to implement some environment modeling and path planning experiments on a real mobile robot. We have used a simple miniature mobile

Prof. Asharif is also a member of academic staff in Department of Electrical and Computer Engineering, Faculty of Engineering, University of Tehran, Tehran, IRAN

<sup>1</sup>It stands for *Machine Learning*

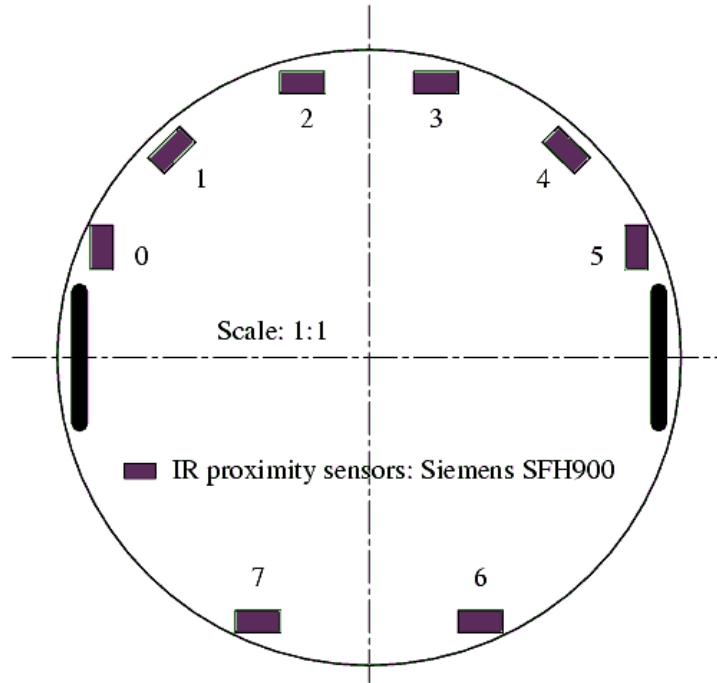


Fig. 1. The physical shape of Khepera robot. ( sensors, two wheels and two motors are located as in this figure. It shows Khepera, seen from above.

robot, called **Khepera**. In this paper we will explain the structure of this mobile robot and environment mapping for Khepera, using a combination of neural and Bayesian fusion of IR<sup>2</sup> sensory data.

## II. ARCHITECTURE OF THE KHEPERA ROBOT

In this section the structure and specifications of the Khepera mobile robot are discussed. Data in this section are gathered from the references [5], [6] and own measurements. The Khepera robot, depicted in Fig.1 is circular in shape, measuring 58 mm in diameter, and with the vision turret mounted on it (It has not been used in our experiments yet) is 70 mm high and weigh 140 grams. Two independent controllable wheels are located on the sides 53 mm apart and the robot is further supported by two small Teflon balls. The wheels can be controlled in units of 1/12 mm and the actual distance traveled, can be read through the wheels encoders as position counters. It is equipped with a 16 MHz Motorola M68331 processor, 256 KBytes RAM and 512 KBytes ROM containing low level I/O functionality, motor control and sensor scanning. An RS232 serial port can be used for connecting the robot to a computer either for remote control or transfer of the control program.

The robot has 8 infrared sensors that can measure both reflected and ambient light giving *proximity* and *brightness* values.<sup>3</sup> Proximity values are integers between 0 and 1023 and measure approximate distances between 2 and 6 cm.

Khepera is built for research not for practical purposes. The sensors are cheap, noisy and shortsighted.<sup>4</sup> Most of research groups use robots equipped with higher precision sonar sensors with much larger range, approximately 10 m compared to Khepera 's 6 cm, and they have a narrow sensing field of 2°-15° compared to Khepera 's 140°. Thrun [1] uses laser range finders with a 0.5° view ([7] and [8] are other two examples).

<sup>2</sup>Infra Red

<sup>3</sup>In our experiments, only *proximity values* were found useful for for map building. Thus brightness values have been left unused and by *sensor values* we mean proximity values.

<sup>4</sup>That 's why we have chosen this robot for our experiment. Sensor fusion methods will be engaged with very noisy and fluctuated sensory data!

TABLE I

AVERAGE, MINIMUM, MAXIMUM AND STANDARD DEVIATION VALUES AT DISTANCES FROM 10 TO 70 MM FROM A LIGHT GRAY WOODEN WALL. THE LAST COLUMN IS THE AVERAGE OVER ALL OF THE 8 STANDARD DEVIATION VALUES OF THE 8 PROXIMITY SENSORS.

Distance in mm	Average (Sensor 2)	Minimum (Sensor 2)	Maximum (Sensor 2)	Average (Sensor 7)	Minimum (Sensor 7)	Maximum (Sensor 7)	$\bar{\sigma}$ (All)
10	1023.0±0.0	1023	1023	1023.0±0.0	1023	1023	0.0
15	1023.0±0.0	1023	1023	1023.0±0.0	1023	1023	0.0
20	1023.0±0.0	1023	1023	1012.4±14.1	909	1023	4.9
25	935.0±8.6	910	977	617.6±13.0	576	674	12.4
30	629.8±12.5	566	672	397.8±9.3	377	459	10.0
35	425.2±6.9	393	446	245.6±9.1	226	296	7.9
40	291.9±8.1	268	334	141.9±8.1	106	163	7.5
45	186.9±7.7	153	213	72.3±7.7	50	100	5.5
50	103.3±6.4	86	124	17.7±7.9	0	36	5.5
55	54.3±9.8	25	105	0.1±0.5	0	4	4.6
60	10.0±9.4	0	37	0.1±0.3	0	1	2.4
65	0.1±0.3	0	1	0.1±0.2	0	1	0.5
70	0.0±0.2	0	1	0.1±0.2	0	1	0.2

### III. REAL ROBOT ISSUES

In the view of an examination of the Khepera robot, some of the special problems that arise when developing software for physical robots are summarized here. Harvey et. al. [9] points to this matter, saying that using only simulators, researchers “... have not had to face the exponential increase in complexity that follows with progress from toy worlds (i.e. simulators) into the real world.”

In embedded systems, timing is an important factor. Proper synchronization is required between internal modules in the control algorithm, the sensors and the motors, the feedback time, the robot velocity, and the serial link. Also these timing problems can be avoided by operating the robot in discrete steps and bringing it to a full stop after each step.

Another consideration is the heavy map computations and path planning that have to adapt to the limited processing power of the M68331. In many real life applications e.g. hazardous or life saving tasks, such computations must be done in real time for the robot to be of any use. The Khepera 's CPU is not of today 's standard which makes it necessary to use computationally cheap algorithms.

#### A. Sensor noise

In the official Khepera 's simulator [5] the sensor noise level is fixed at  $\pm 10\%$  of the value of the underlying mathematical model but on the real robot the class of the distribution of the sensor noise is unknown. An experiment was performed to estimate the noise at distances 10-70 mm from a light gray wall. 100 measurements were recorded in steps of 5 mm for each sensor turned  $90^\circ$  to the wall in order to achieve maximum reflection.<sup>5</sup> Table I lists key figures for some of the measurements.

The noise is the largest in relative magnitude for small sensor values i.e. large distances. Figure 2 shows all readings for sensor 5 at 35mm from the wall. When using the values for estimation of the distance, the biggest problem caused by noise is spikes, not the small deviations. Identified noise sources are the lamps in the laboratory, sun light varying over the day, reflections from the walls and floor, the battery charge level and differing surface structures of the walls in the mini-world. By reducing their influence as much as possible we obtain the results presented here.

<sup>5</sup>Preferably this should have been the maximum over angles about  $90^\circ$  as the sensor might be imprecisely adjusted and have the maximum reflection at diverting angles.

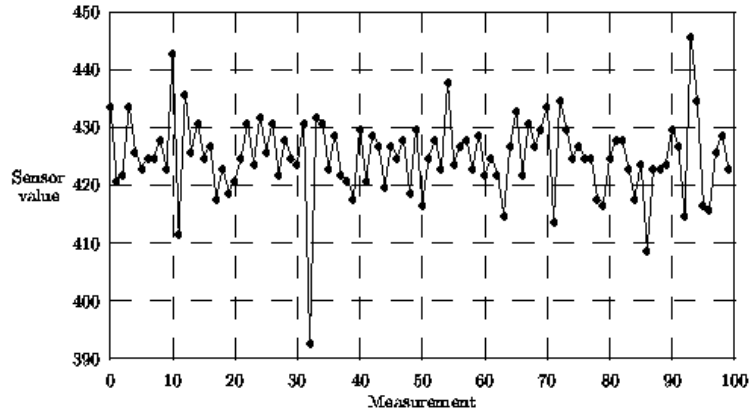


Fig. 2. Proximity values at  $90^\circ$  and 35 mm from a light gray wooden wall for sensor 5. Standard deviation is 5.6

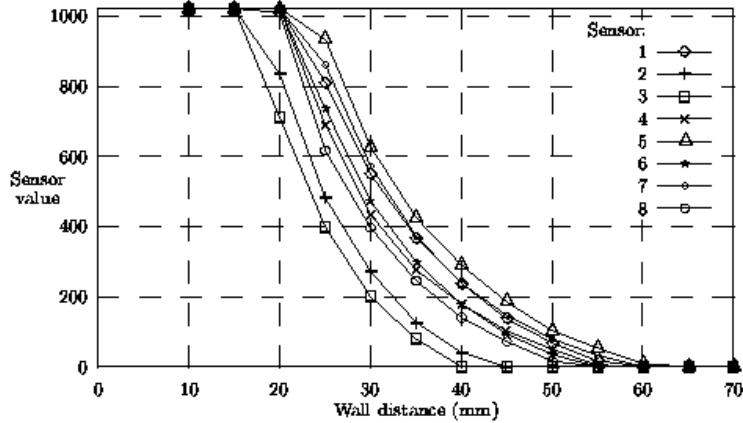


Fig. 3. Proximity values at  $90^\circ$  and 35 mm from a light gray wooden wall for sensor 5. Standard deviation is 5.6

A further difficulty is that the sensors have different characteristics. Figure 3 shows the average sensor values at varying distances. A theoretically important observation concerns the **independence** of noise in sensor values over time. If they are dependent, using an average of for example 3 consecutive measurements might not be an improvement over a single value. Independence of readings is also theoretical basis for sensor fusion using Eq. ?? or any pseudo-information measure formulas discussed in chapter 3. To examine this important feature, Khepera was set to collect sensor values as fast as possible. The results are summarized in Table II concluding that sensor readings are independent over short periods of time (1 minute).

### B. Wheel noise

When tracking the robot's position, we must use a wheel model built on odometric information, which is inherently noisy. In case of a model built on kinematic equations on the odometric information, noise sources are best understood when split into *systematic* and *stochastic* (or *non-systematic*) errors. The systematic errors include unequal wheel diameters, imprecise measurement of wheel distance, and differing motor stop times. The stochastic errors comprise wheel slippage, uneven floors and cord pull. For a learned model, instead of a mathematical one, the errors are more properly split into *imperfect learning* and stochastic errors.

Borenstein et. al. [10] present a procedure, the *bidirectional square-path experiment*, for estimating and reducing systematic odometry errors caused by:

- Unequal left and right wheel diameters  $d_l$  and  $d_r$ , defined as  $E_d = d_r/d_l$

TABLE II

SENSOR INDEPENDENCE MEASUREMENTS. EACH ROW REPRESENTS 70 READINGS. THE DISTANCE TO A POLYSTYRENE WALL IS VARIED TO GIVE A COMPACT SLICE OF ALL COMBINATIONS OF SENSORS AND DISTANCES. IN TWO SEPARATE COLUMNS, THE AVERAGE OF THE ABSOLUTE DIFFERENCE FOR ALL 69 SUCCESSIVE PAIRS OF READINGS AND THE AVERAGE ABSOLUTE DIFFERENCE BETWEEN ANY TWO READINGS IN THE SET, ARE SHOWN. THE VALUES OF THE TWO COLUMNS ARE CLOSE AND THEREFORE WE CAN DEDUCE ABOUT THE **independence** OF SENSOR READINGS IN TIME.

Sensor Number	Distance in mm	Average Sensor Value	Averaged Successive Absolute Difference	Average of All Absolute Differences
1	20	1023.0	0.0	0.0
2	25	1023.0	0.0	0.0
3	30	811.5	20.3	18.1
4	35	550.5	12.4	11.8
5	40	706.5	8.4	9.5
6	45	425.2	8.5	8.0
7	50	370.0	6.8	6.2
8	55	200.7	5.4	5.3

- Uncertainty of the wheel distance  $b$ , defined as  $E_b = b_{actual}/b_{nominal}$

In order to separate systematic errors from stochastic ones the robot is set to drive a number of square paths, returning to its origin. The error is then calculated as the average of differences between start and end positions including orientation. To separate  $E_d$  from  $E_h$  the path is traversed both clockwise and anti-clockwise. When  $E_D$  and  $E_h$  have been found, the kinematic equation of the robot can be adjusted accordingly. This was done in our experiments and will be briefly in the next section. Details of the method can be found in [10]. The remaining noise (mainly its stochastic or non-systematic portion) gives rise to an average small error.

### C. World model

In the present task the world model serves two purposes: it is the final goal of the running application and it is the basis for navigation. Occupancy grids are simple, but fulfill these demands. Further, the map computation must not be too intensive in order to operate on Khepera robot (this is also fulfilled by our choice of occupancy grids and using Bayesian or Pseudo-information measure formulas). The local maps should be fused into a global map as often as it improves the map, the sensor values are less than 1023 (this is important because if the sensor values are 1023, the distance may have an infinite number of possible values) and as often as it is computationally tractable. The construction of local maps will be discussed later in section V.

As the error, existing in the localization process grows, the local map will be integrated into the wrong place, but as pointed out by Thrun et.al. [11] most approaches does not use the past sensor data for revising the map when a position error estimate has been corrected. Thrun et. al. [11] propose a way to do just that, but we abstain from this sensible extension to save time.

## IV. LOCALIZATION

The robot uses information about its position in the world to properly integrate local maps into the global map. That's why *localization* is an important issue in any robotic task.

### A. Position representation

We have used the *sample model* which estimates only the most likely location  $\chi = (\mathbf{x}, \mathbf{y}, \theta)$  in a global coordinate system. Some measure of the quality of the position estimate is needed. We use the distance that was traveled since last calibration. Such a distance, consists of two parts: translational movement and rotational movement. The former part is proportional with the addition of the left and right wheels movements and the latter part is proportional with the negation of the left and right wheels.

Another possibility is was Cowley 's suggestion [12] suggestion to maintain the position *confidence*, which is expected difference between the estimate and the true position. This is the statistical accumulation of the position confidences for single steps of the motion. The confidence value is useful input to e.g. Markov localization or map matching, but the disadvantage is more complex calculations.

Another untried alternative was the *distribution model*  $P(\chi)$  of the position which can be maintained as a discrete approximation or with the parametric densities. Either way it is much more complex than the sample model which is our choice.

### B. Wheel model

As the basis for position tracking, the *wheel model* must map the variation in the wheels states  $\Delta \mathbf{c} = [\Delta \mathbf{c}_r \ \Delta \mathbf{c}_l]^T$  into the most likely position change  $\Delta \chi = [\Delta \mathbf{x}, \Delta \mathbf{y}, \Delta \theta]^T$ . On the real motor it is more suitable to use the change  $\Delta \mathbf{c}$  of the *position counter* (wheel encoders) than the speed command. The position counter gives the distance traveled by each wheel in units of approximately 0.08 mm. There are two fundamentally different ways to establish a wheel model:

#### IV-B.A: Learning the model

Learning the model would ideally generalize through stochastic errors. The natural approach is to observe the position change resulting from various  $\Delta \mathbf{c}$ . For the real robot, the position change is too small to be observed for small  $\Delta \mathbf{c}$  and larger  $\Delta \mathbf{c}$  can represent many different paths and accordingly represent too different position changes, to be useful. If learning of the mapping process is insisted, one approach would be to apply temporal difference learning to propagate the error correction back from the end of a period of e.g. 100 "standard size" steps. But we will not insist.

#### IV-B.B: Constructing the Kinematic Equations

We will construct a wheel model mapping  $\Delta \mathbf{c} = [\Delta \mathbf{c}_r \ \Delta \mathbf{c}_l]^T \mapsto \Delta \chi$  in local coordinates for the real robot by looking at Fig.4 where the left and right wheels have traveled the increased position counter values  $\Delta c_l$  and  $\Delta c_r$  respectively. If  $|\Delta c_l - \Delta c_r|$  is small enough (i.e. for a small motion period) we can simply formulate the motion equations as follows:

$$\Delta x = \frac{\Delta c_l + \Delta c_r}{2} \times \sin(\theta) \quad (1)$$

$$\Delta y = \frac{\Delta c_l + \Delta c_r}{2} \times \cos(\theta) \quad (2)$$

$$\Delta \theta = \frac{\Delta c_l - \Delta c_r}{b} \quad (3)$$

These equations are build on kinematic equations similar to what can be found in Borenstein 's and Crowley 's papers [10], [12].

Now it is the time to tune the above kinematic equations, using the method of Borenstein et. al. [10] (called as UMB Mark). Adjusting the wheel base and are done by:

$$E_b \times b \rightarrow b \quad (4)$$

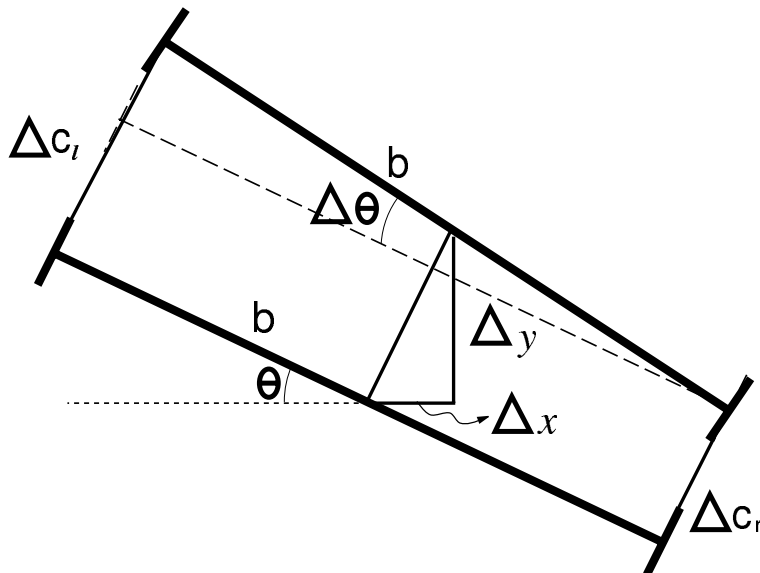


Fig. 4. The track of the center of the robot and its left and right wheels in a very short period of motion.

$$\frac{2E_d}{E_d + 1} \times \Delta c_r \rightarrow \Delta c_r \quad (5)$$

$$\frac{2}{E_d + 1} \times \Delta c_l \rightarrow \Delta c_l \quad (6)$$

Note that tuning for wheel traveling values  $\Delta c_r$  and  $\Delta c_l$ , have been formulated in such a way that not only their ratio equals  $E_d$ , but also their sum does not change.

## V. INVERSE SENSOR MODEL

The inverse sensor model is basic in the mapping functionality as it provides the local maps to integrate into the global map. The local map is an occupancy grid with the same center and orientation as the robot and has the shape of a circle with radius set to the average sensor range plus Khepera's radius.

For general tasks, the desired resolution of the map can vary or otherwise be difficult to determine before the application task is initiated or depend on unknown performance of hardware, therefore a nice quality of the solution method would be independence of resolution. Further, when integrating the local map into the global map, the two maps will have different orientation and cell boundaries will not align. If map resolution is low this can cause a poorer result than with perfectly aligned grids. This is another reason for preferring mapping with unlimited resolution.

When referring to a grid cell  $c_{x,y}$  the coordinates  $(x, y)$  of a point in local space are understood to be rounded to fit the resolution of the (global) map.

The only information that is relevant for the ISM-mapping is, the current sensor values  $\mathbf{s} = [s_1, \dots, s_8]^T$  which are integers between 0 and 1023. As mentioned before, reading of sensor values and estimation of  $\chi$  are simultaneously, making integration of  $C_{x,y}(\mathbf{s})$  well-defined for  $(x, y)$  in the local space.

To have a better idea of the task to be solved, the reader can try to use his biological neural network to convert the sensor values  $\mathbf{s} = [59, 0, 0, 1, 2, 1, 1023, 1023]$  into a local map. One solution is given in Fig.5.

We have trained a neural network with the topology shown in Fig.6 to approximate the value  $C_{x,y}(\mathbf{s})$  of the inverse sensor model.

Thrun et. al. [11] use a similar topology on a robot with 25 sonar sensors, but they only use the values of the 4

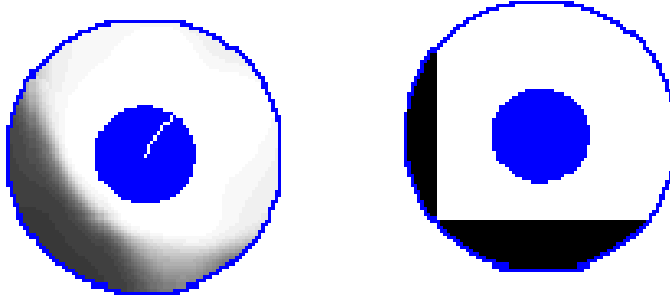


Fig. 5. The real (right) and the mapped (left) local maps for which the sensor values are:  $\mathbf{s} = [59, 0, 0, 1, 2, 1, 1023, 1023]$

sensors closest to the point  $(x, y)$  as input. Doing so without introducing much error requires that:

1. Sensors are alike.
2. They are evenly distributed around the robot.

Neither of these demands are met by Khepera and therefore reducing input-size by using the same input neuron for varying sensors in separate propagations is not feasible.

One advantage of this structure is the fact that the resolution of the grid need not be specified. The network can be trained independently. Of course, training with a very low resolution will not provide for good generalization.

Further, using all of  $s_1, s_2, \dots, s_8$  as input makes a very general model that will adapt to the location of sensors on the robot. The only Khepera-specific knowledge employed in this method is the average sensor range, which is used indirectly by only training on points within sensor range. A disadvantage caused by using all sensor values as input, is that the training data must represent all the positions in which the robot will find itself in the application phase: at edges, wall ends, corners etc. with many different angles.

Here are some of pre-processing that have been executed before the training of the network:

- Most of the researchers in neural network domain, recommend normalizing input (for example Bishop in [13] offers normalizing to zero mean and unit standard deviation). We simply scale all inputs to  $[0,1]$  with no consideration of the distribution within the interval. Even though all input values are scaled, they have been referred to their original values in this text.
- For training the neural network,  $N = 78$  points were selected in a simple environment with only one rectangular obstacle. The location of these points were chosen in such a way that the neural net can distinguish many cases such as a straight wall, a concave corner, a convex corner etc. in many different distances. About the directivity, in each of the  $N$  points the robot was rotated  $5^\circ$  step by step. Thus the total number of patterns, applied for training of the network is:  $M = \frac{360}{5} \times N = 2808$
- The network was trained using online training not batch training. But for performance measurement in the training phase, in each  $D$  steps, the following MSE expression:

$$\text{MSE} = \frac{1}{D} \sum_{i=1}^D (T_i - O_i)^2 \quad (7)$$

was used. Here  $i$  accounts the number of locations in a *block* of  $D$  samples taken at random.  $T$  and  $O$  are the values of the **T**True and **O**Output of the network. We could reach the average measure of 0.03 for MSE expression and stopped training at this point.

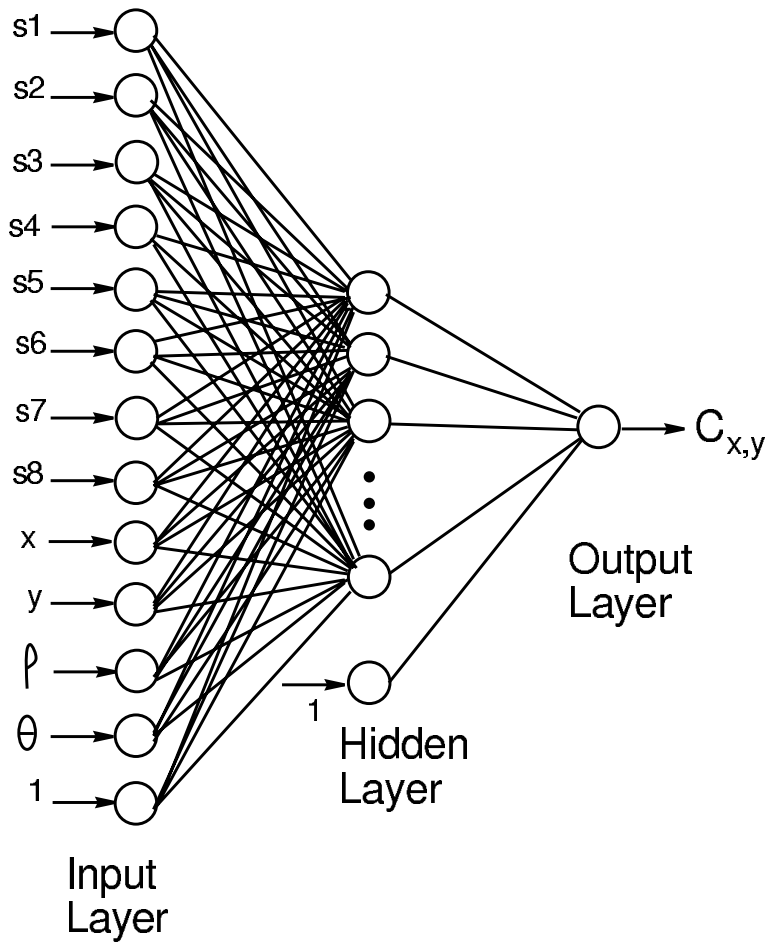


Fig. 6. The structure of the neural network, applied in this research work for inverse sensor model learning. The single output  $C_{x,y}$  is a probability value between 0 and 1, judging about the cell  $c_{x,y}$  to be occupied or empty.

- Neural networks have been said to ‘handle’ noise. This only means that the learning process will not be disturbed by a moderate noise level, but generalize through it. It does not imply that application performance will not be improved by a noise reduction. In subsection III-A the sensor noise level was examined and it is natural to try to reduce the effect of noise by averaging a number of measurements (10 measurements in our experiments) before using them for input.
- Representing the coordinates  $(x, y)$  of grid cells in the local coordinate system is obvious and intuitively it is better to represent the cell with polar coordinate  $(\rho, \theta)$  additionally. This will lead the network to train faster and have more power for generalization. This happens because of the robot’s morphology and the sensors’ characteristics.

#### A. Training Results

Figure 7 shows local maps for 10 test points. Each couple contains the local map generated by the best 12-10-1 network trained on real robot data and the true world image with the walls and corners that generate the 8 sensor values. The outer circle is the limit of the local map. On (e) and (f) the space on the other side of the wall can be glimpsed to the left. The robot orientation is marked with a thin white line pointing outwards from the center of the robot.

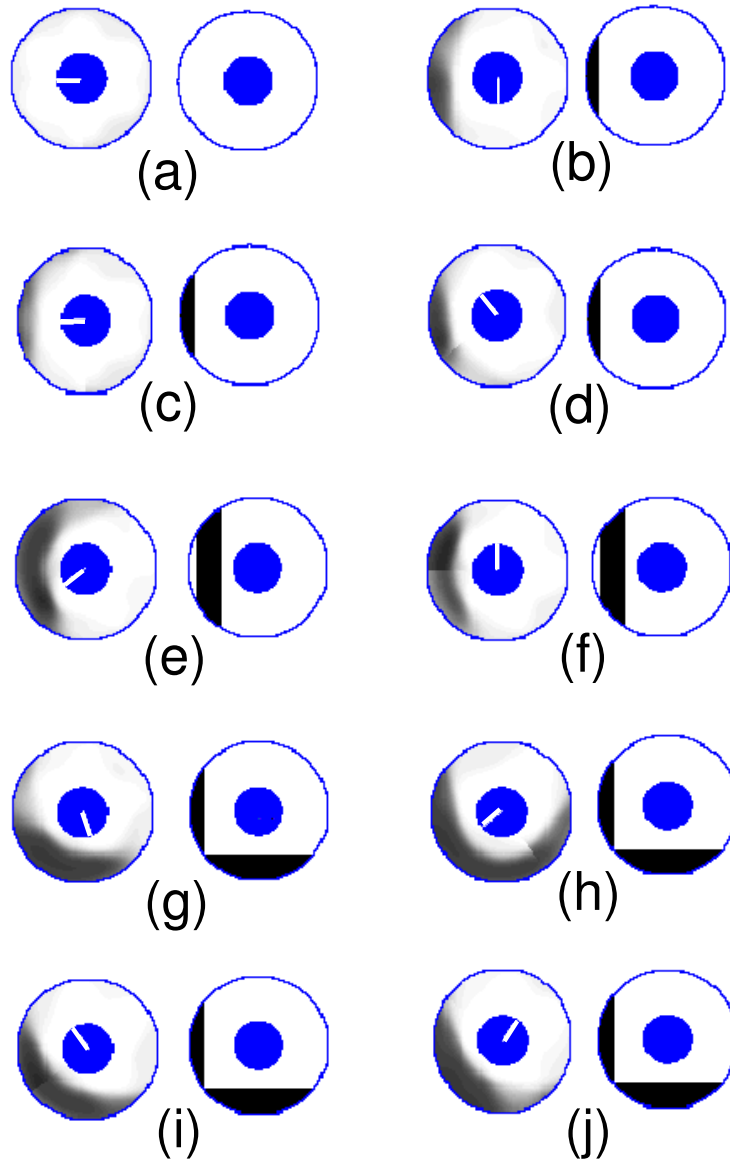


Fig. 7. Local maps for 10 test positions

## VI. MAP GENERATION

The neural network that was explained in the previous section, was applied for mapping multiple environments. Two of the environments are brought and their maps are discussed here.

The first environment is a simple four-wall environment with four rectangular obstacles inside. Figure 8 shows a photo of this environment.

As depicted in the figure, the environment has four walls and four rectangular obstacles. While exploring the environment for mapping, using this neural net, it is necessary to calibrate its position, because in spite of using UMB mark technique explained in section III-B (and with more detail in [10]) there is still some error in the wheel model and the applied dead reckoning technique requires calibration. To do that, we stop the robot periodically and correct its location  $\chi$  manually. These periods of time are the time instances when a special criterion is satisfied and about this criterion we chose it as "a linear combination of translational and rotational motion of the robot to go over than a threshold". It is reminded that the translational and rotational parts of the robot's motion are proportional to  $|\Delta c_r + \Delta c_l|$  and

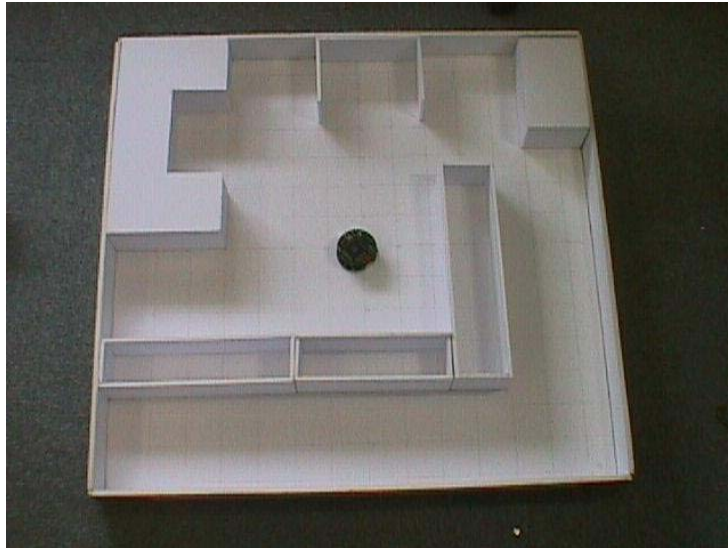


Fig. 8. Local maps for 10 test positions

$|\Delta c_r - \Delta c_l|$  respectively ( $\Delta c_r$  and  $\Delta c_l$  are the variation in the right and left wheel encoder values, respectively).

Here we give the brief algorithm of navigation and mapping process:

1. **Initialization:** The robot is placed in a known position.
2. **Exploration:** The robot starts stochastic navigation using a simple obstacle avoidance algorithm, named “Braitenberg algorithm” [14]. While exploration, the robot checks its traveled distance (translational and rotational) and after a while it stops for calibration.
3. **Local mapping:** After a specific number of iterations of exploration, robot stops, senses its environment, and a local map around the robot is created using the neural network.
4. **Fusion:** The local map is inserted inside the global map (a big  $800 \times 800$  matrix) using a fusion method. In our experiments, Bayesian and 3 of Pseudo-Information measure methods were selected for map fusion.
5. Go to step 2.

In Fig. ?? the final map, created using Bayesian fusion (after 1000 iterations of local map generation and then fusion) is shown. In Fig. ?? and ?? and ??, three maps are shown. They are the final map, created using Pseudo-information measure fusion (after 1000 iterations of local map generation and then fusion) by three definitions for  $J(\cdot)$  function.

#### ACKNOWLEDGMENTS

This work was partially supported by SIS (School of Intelligent Systems) at IPM, Tehran, IRAN under contract numbers 0177-115 and 0180-3 and also financially supported by the ministry of industries and mines under contract no. 782015168. The grant provided by Japanese MONBUSHO scholarship is also kindly appreciated.

#### REFERENCES

- [1] S. Thrun, “An approach to learning mobile robot navigation,” *Robotics and automation systems* 15, 301-319, 1995
- [2] R. A. Brooks, P. Maes, “Learning to Coordinate Behaviours,” *Proceedings of the Eighth National Conference on Artificial Intelligence*, Cambridge, MA, 796-802, 1990
- [3] D. Floreano, F. Mondada, “Evolution of Homing Navigation in a Real Mobile Robot,” *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, Volume 26, 396-407, 1996
- [4] H. H. Lund, J. Hallam, “Sufficient Neuro Controllers can be Surperisingly Simple,” *Technical Report 824*, Department of Artificial Intelligence, University of Edinburgh, 1996
- [5] K-Team S. A., “Khepera User Manual(5.0 ed.),” Lausanne, Switzerland, 1998
- [6] K-Team S.A. Home page, <http://www.k-team.com/>, 1999

- [7] S. Mahadevan, J. Connell, "Automatic Programming of Behaviour-Based Robots, Using Reinforcement Learning," *Artificial Intelligence* 55(2-3), 311-365, 1992
- [8] J. Buhmann, W. Burgard, A. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, S. Thrun, "The mobile Robot RHINO," *AI Magazine* 16(2), 31-38, 1995
- [9] I. Harvey, P. Husbands, D. Cliff, "Issues in evolutionary robotics," In *From Animals to Animats: Proceedings of the Second International Conference on Simulation of Adaptive Behaviour*, Edited by J. A. Meyer, H. Roitblat, S. Wilson, MIT Press, 364-373, 1993
- [10] J. Borenstein, H. R. Everett, L. Feng, "Where am I? Sensors and Methods for Mobile Robot Positioning," Technical Report, University of Michigan, <ftp://ftp.eecs.umich.edu/people/johannb/pos96rep.pdf> , 1996
- [11] S. Thrun, W. Burgard, D. Fox, "A Probabilistic Approach to Concurrent Mapping and Localization for mobile robots," *Machine Learning* 31, 29-53, 1998
- [12] J. L. Crowley, "Mathematical Foundations of Navigation and Perception for an Autonomous Mobile Robot," In *Reasoning with Uncertainty in Robotics* Edited by L. Dorst, M. Van Lambalgen and F. Voordraak, Volume 1093 of *Lecture Notes in Artificial Intelligence*, 9-51, Springer Verlag, 1995
- [13] C. M. Bishop, "Neural Networks for Pattern Recognition," Oxford University, 1995
- [14] V. Braitenberg, "Vehicles," Kluwer Academic Publishers, 1984