International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:4, 2010

# A Simulator for Robot Navigation Algorithms

Michael A. Folcik and Bijan Karimi

***Abstract*—**A robot simulator was developed to measure and investigate the performance of a robot navigation system based on the relative position of the robot with respect to random obstacles in any two dimensional environment. The presented simulator focuses on investigating the ability of a fuzzy-neural system for object avoidance. A navigation algorithm is proposed and used to allow random navigation of a robot among obstacles when the robot faces an obstacle in the environment. The main features of this simulator can be used for evaluating the performance of any system that can provide the position of the robot with respect to obstacles in the environment. This allows a robot developer to investigate and analyze the performance of a robot without implementing the physical robot.

***Keywords*—**Applications of Fuzzy Logic and Neural Networks in Robotics, Artificial Intelligence, Embedded Systems, Mobile Robots, Robot Navigation, Robotics.

## I. Introduction

THE field of mobile robotics has received considerable attention from researchers during the past two decades. Many methods have been proposed and implemented for robot navigation in various environments [1], and many robot competitions at various experience levels are held that are based on path finding and target identification [2], [3]. One robotic system configuration that has received special attention from researchers is where a neural-fuzzy or fuzzy-neural system is used to guide a mobile robot [4], [5]. In these systems, the data collected about the environment by electro-mechanical sensors is fed into a fuzzy-logic or neural-network system. After the raw data is normalized and fed into the fuzzy-logic (or neural-network) system for pre-processing, the output will be used as input to the second system, the neural-network (or fuzzy-logic), for some form of decision making.

Actual implementation of these or other methods in real robots is expensive, time consuming, and in the case of negative results, wasteful. It is important that a proposed method is simulated to measure its success before the actual investment in developing the system is made. A simulator for a mobile robot must be capable of simulating within various environments, emulating the electromechanical sensors, and moving the robot in specific paths, among other features. The simulator proposed and implemented in this article addresses many of these characteristics.

One important parameter for a robot to make the right decision regarding the next course of action is the knowledge of the relative position of the robot with respect to random obstacles in the environment, in order to avoid them or identify a target. An obstacle in this context is defined as anything that may inhibit the intended actions of the mobile robot. In order to measure the degree of the success in such cases, a simulator capable of accepting the inputs, generating the outputs, and verifying that the generated output is good (for identifying the relative location of obstacles and avoiding them) is needed. The developed simulator has been implemented with a system consisting of a fuzzy-logic layer followed by a neural-network layer. The main focus of this article is on the development and features of the simulator and not the fuzzy-neural system, which has to be discussed and dealt with separately, although the general framework is presented to clearly exemplify the features of the simulator. The data presented to the simulator about the position of the robot with respect to obstacles can be from the given sample system in this article or any other method used for this purpose. This may require some modification to the simulator as the presented simulator also integrates the fuzzy-neural features of the system. The features and the problems faced and solved that are discussed in this article can be used as guidelines for developing similar simulators for evaluating the performance of similarly designed systems.

## II. Sample System under Investigation

Fig. 1 is the general model of a fuzzy-neural system proposed by the authors for which the simulator was developed. Fig. 2 shows the membership function used for the Fuzzy-Logic section. The inputs to fuzzy section are the analog outputs of the robot's sensors which will be preprocessed by the fuzzy-logic and ultimately fed into the neural-network. The system used in this simulator accepts inputs from three distance sensors, in cm. One sensor is placed such that it faces 90º counterclockwise from the front of the robot; the second sensor faces directly forward; and the third sensor faces 90º clockwise from the front of the robot. In general, the number and type of sensors are not restricted to what is used in this article as an example. Table I shows the state format, state names, and description of every possible scenario that this robot with three distance sensors can be in. Fig. 3 is a visual representation of the information in Table I. The output of the neural network can be either binary values or an integer value, and in this work, an integer value is used.

M. A. Folcik, MSEE, was with the University of New Haven, West Haven, CT 06516 USA. He is now with Food Automation Service Techniques, Inc., Stratford, CT 06615 (e-mail: mfolc1@newhaven.edu).
B. Karimi, Ph.D., is with the University of New Haven, West Haven, CT 06516 USA. (phone: 203-932-7164; fax: 203-931-6091; e-mail: bkarimi@newhaven.edu).

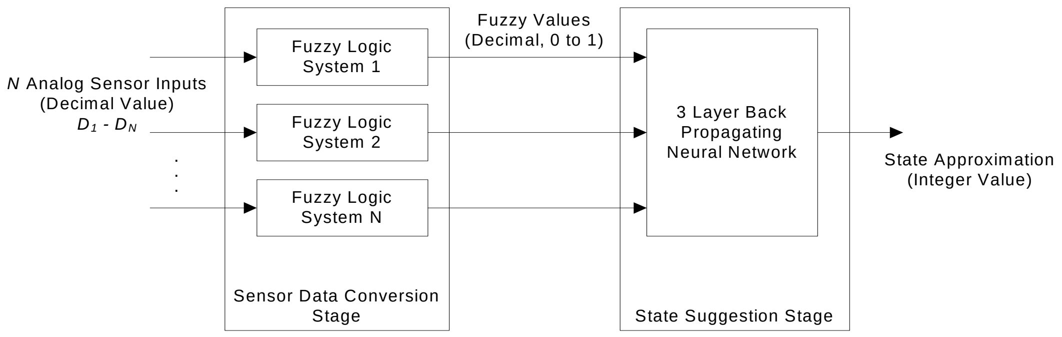


Fig. 1 Block diagram of fuzzy-neural system

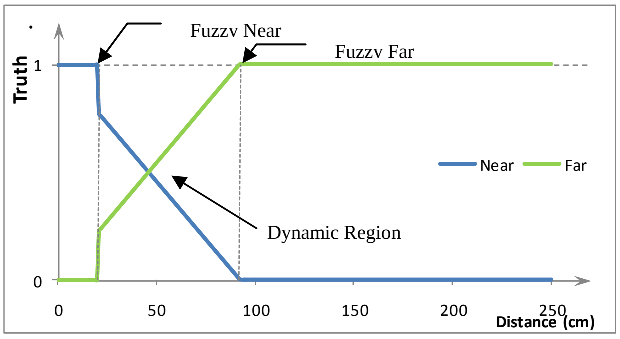


Fig. 2 Membership function for the fuzzy system

The membership function for the fuzzy system is developed such that input values below some constant “fuzzy near” value are simply assigned a fuzzy output of “NEAR”, input values above some constant “fuzzy far” value are assigned a fuzzy output of “FAR”, and inputs between these two constants are assigned some non-crisp value between “NEAR” and “FAR”, representing the truth of the input as it relates to being crisp “NEAR” or crisp “FAR”.

The “Dynamic Region” of Fig. 2 does not have to be a simple linear function. This can be any curve that the designer wishes.

## III. ARMS: A Robot Maze Simulator

A robot simulator utility was produced using C# and Visual Studio 2008 in order to measure the ability of a system to identify the current position of a robot within an environment. ARMS, shown in Fig. 4, is a graphical user interface (GUI) that shows the movement of the mobile robot along with other valuable system information. The robot in this simulator uses the system presented above to navigate within a specified environment. The simulator models the realistic motions and actions of the robot – it can turn left or right by rotating about its center, and it can move forward. Also, the simulator models the distance sensor values by calculating the distance between the sensor and the nearest obstacle in a straight line. The distances from the center of the robot to the front of the sensors are adjustable in order to effectively change the size or shape of the robot.

In this simulator the environment is mapped using a

TABLE I
State Names & Symbols for Describing the Position of a Mobile Robot with Three Distance Sensors

| State (binary) | Description | Symbol |
|---|---|---|
| FFF | No Objects | NO |
| FFN | Object on Right | OR |
| FNF | Object in Front | OF |
| FNN | Objects Front & Right | OFR |
| NFF | Object on Left | OL |
| NFN | Objects Left & Right | OLR |
| NNF | Objects Left & Front | OLF |
| NNN | Surrounded | S |

The binary states above are represented with F = 0 (for FAR) and N = 1 (for NEAR). Each bit represents the value of one of the distance sensors on the example robot.

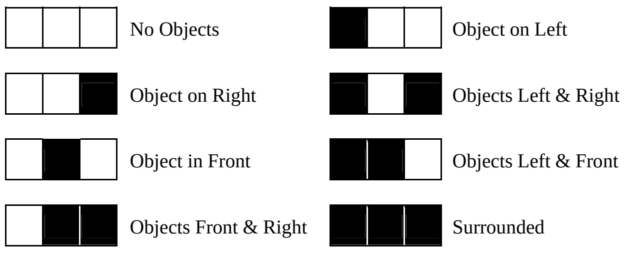
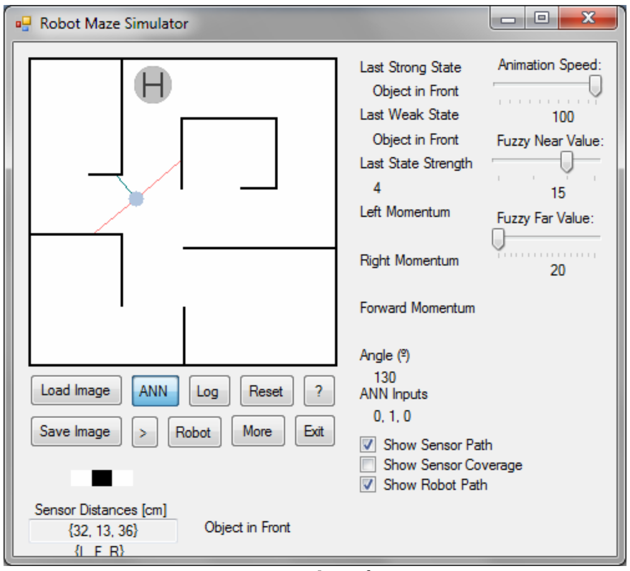


Fig. 3 State Images. Each cell represents one “pixel”. Black means an object is “NEAR”, white means an object is “FAR”.

bitmap, and the robot is moved around within the pixels of the environment. So, the trigonometry is limited to integer math resulting in occasionally losing resolution on some paths. This can cause the robot to “see through” an obstacle. To overcome this problem in the simulator, the obstacles must be made “thick” enough for the robot to see them. In a real scenario, however, this rounding error will not occur and the robot will not be able to violate the boundaries of a physical obstacle.



Last Strong State
Object in Front
Last Weak State
Object in Front
Last State Strength
4
Left Momentum
Right Momentum
Forward Momentum
Angle (º)
130
ANN Inputs
0, 1, 0

Animation Speed: 100
Fuzzy Near Value: 15
Fuzzy Far Value: 20

[✓] Show Sensor Path
[ ] Show Sensor Coverage
[✓] Show Robot Path

Sensor Distances [cm]
{32, 13, 36}
{L F R}
Object in Front

Fig. 4 Screenshot of ARMS

## IV. Features of the Simulator

As stated before, the simulator models the realistic motions and actions of the physical robot – the simulator robot can turn left or right by rotating about its center and it can move forward. Also, the simulator models the sensor distance by calculating the distance between the sensor and the nearest obstacle in a straight line. The distances from the center of the robot to the front of the sensors are adjustable in order to effectively change the size or shape of the robot.

To place the robot and begin simulation, the user must click anywhere on the environment map. Once placed, the robot may be rotated left by pressing the left arrow on the keyboard, rotated right by pressing the right arrow on the keyboard, or moved forward by pressing the up arrow on the keyboard. The user is able to load custom environment maps by pressing the [Load Image] button, and after simulating, the user can press the [Save Image] button to save the image with the robot path or sensor coverage results. The custom environment maps must be 248 by 248 pixels in size in standard bitmap format. The maps can contain color, but only black pixels will be detected as obstacles.

The default algorithm that navigates the robot in ARMS is a non-deterministic algorithm. This was done to show that any algorithm may be used, and even a poor algorithm might produce results. The largest challenge with applying the classic navigation algorithms to mobile robots is no longer the computational power required, but instead it has become identifying where the robot is. By coupling some navigation algorithm with this fuzzy-neural system, a mobile robot may be intelligently navigated around random obstacles with accuracy, repeatability, and confidence.

ARMS is very useful for tuning and tweaking the fuzzy-neural system described in this article. Rather than being concerned with a real robot, loading code onto it, and running it in a maze, ARMS is a starting point where the initial values can be set up for the fuzzy-neural system. It can be easily modified to support different robots with a different amount of sensors or different types of sensors. Since the artificial neural network is written in C and simply called from ARMS, the exact neural network used for training and analysis with ARMS can be used on the physical robot, considering it can be programmed in C.

Below are details for each control and feature in ARMS. This simulator was designed to be a flexible option for many generic two dimensional environments, so more functionality can be added as necessary.

### *A. Load Image button*

This button opens a dialog window that allows the user to select and choose an image to use for the environment map. Upon opening the image, the map will be instantly displayed and all robot path, sensor coverage, and sensor path data will be removed from the environment.

### *B. Save Image button*

Pressing this button will open a dialog window that allows the user to save the current environment map to a file. The saved image will include any robot path, sensor coverage, and sensor path markings that are currently on the map. This feature is useful for keeping track of simulation progress and for reporting.

### *C. ANN (Artificial Neural Network) toggle button*

This feature generates a state identification value from the values of the distance sensors on the robot, by running them through the fuzzy-neural system discussed earlier. Each time the robot is moved, whether by mouse click or an arrow key, this value is recalculated. The sensor values are filtered using the fuzzy logic system, and the resulting values are fed as arguments into a separate executable program, which has the neural network code written in C. This program takes the three sensor values and runs them through the pre-trained neural network with six neurons in the hidden layer. This program was trained to an accuracy of $10^{-8}$ with the eight unique states and the trained neuron weights were saved as constants so that training does not have to occur each time the program is run. The neural network is not computationally intense, so its execution should be transparent on most modern processors.

### *D. Play[>] toggle button*

By pressing this button, ARMS allows the user to run a motion simulation. This will run the navigation algorithm used by the simulator.

### *E. Log option*

One other potentially useful feature in ARMS is the ability to log the sensor data to a file (along with the user's opinion of the current state, if desired). By pressing the [Log] button, this logging is enabled, and the sensor data is printed to the file each time the robot's position changes. This might be useful for data collection or path analysis. For example, this feature was used to generate large amounts of real data which was crucial in designing and testing the fuzzy logic system.

### *F. Robot button*

This button opens up another window that allows customization of the current robot. In this case, it allows the user to adjust the distance between the center of the robot and the front of the distance sensors. This effectively changes the size of the robot.

### *G. Reset button*

Pressing this button clears all robot path, sensor path, and sensor coverage markings on the map and resets the environment to its initial state. The robot remains untouched.

### *H. The [More/Less] toggle button*

This option expands the GUI window to show more advanced options and information. It displays the sensor path, sensor coverage, robot path, and animation speed options, as well as adjustments for the fuzzy near and far values. In addition, the angle of the robot, the inputs to the neural network, the last processed (weak) state, the last strong state and its strength, and the momentum values of the robot. The last weak state is the last state returned from the neural network, while the last strong state is the last state that the robot acted upon. In a simple system, these should be the same state, but this approach is used to filter out incorrect states given by the neural network. If the robot acted upon each state that it came upon, it would frequently get caught in local loops: situations where the robot could never advance from because it toggles between one state and another. Since the robot in this implementation of the simulator only acts upon the last strong state, this means that the robot must have seen this state some number of consecutive times, and each of these consecutive times increases the likelihood of truly being in that state (thereby double checking the neural network which we accept can be wrong). Similarly, the momentum values (forward, left, and right) keep the robot from changing its actions too frequently, and they also help the robot to navigate around difficult obstacles that it would normally avoid. For instance, if the robot has been travelling down a straight hallway and has built up a lot of forward momentum, the robot will want to continue moving forward even if there is an obstacle in front of it. Now, this obstacle is most likely the end of the hallway, but perhaps it is not. If the robot at least attempts to continue forward past this obstacle, it will be able to get closer to it and determine whether it is truly a wall or not. Momentum gets added and subtracted with different weights depending on the severity of the situation; if the robot is extremely close to an obstacle in the direction it is heading, momentum will be subtracted very rapidly to avoid collision. However, if the robot is not in immediate danger of collision with respect to the distance to the obstacle, then the momentum will not be subtracted as quickly.

### *İ. Help [?] button*

Pressing the help button opens up a help window with the program information (revision, author, etc.).

### *J. Exit button*

This button cleanly exits the program.

### *K. Sensor path option*

When the sensor path option is selected, ARMS will show the path of each sensor on the current robot. A sensor path in ARMS is a line that goes directly from each sensor to the nearest obstacle. When the [ANN] system is enabled, the sensor paths vary in color depending on the values being fed into the artificial neural network. When the path is green, it means that the sensor is currently reading a crisp “far” value; when the path is red, it means that the sensor is currently reading a crisp “near” value, and any shade between red and green means that the sensor is in a fuzzy state.

### *L. Sensor coverage option*

Turning this feature on will mark every obstacle that any sensor on the robot picks up with red pixels. This is useful for mapping systems to determine if the robot has seen a certain area of its environment during its lifetime. It does this by drawing a red pixel on the edge of every obstacle that a sensor sees. These red pixels do not become obstacles themselves; they are transparent to the robot’s sensors.

### *M. Robot path option*

This option will draw the path of the robot in its environment in purple. This is useful when running the program for an extended period of time to determine the search coverage of the current algorithm in the current environment.

### *N. Fuzzy near/far value sliders*

Two crucial variables to adjust for optimal performance in a particular environment are the fuzzy logic near and far values. In systems where each sensor uses different values, each sensor must be calibrated individually because each sensor may have a different opinion for what is “near” to it and what is “far” from it. In this implementation of ARMS, however, the “near” and “far” values are global. It should be noted that these two sliders change their possible range with respect to each other so that the “near” value can never be greater than the “far” value.

### *O. Sensor Distances*

The raw analog sensor values are displayed on the lower left of the GUI window. Along with these values (shown as a set) is a grayscale image representing the values in the fuzzy logic system. White represents fuzzy “far”, or “no object”, black represents fuzzy “near”, or “object detected”, and varying shades of gray represent values in between. This image is how the robot sees its world; since its three distance sensors are its only knowledge of the environment, this is the image that it sees at any point in time.

## V. ACCURACY OF THE SYSTEM

The fuzzy-neural system proposed and implemented can be described as a Monte Carlo algorithm, defined as a "*randomized algorithm* that may produce incorrect results, but with bounded error probability”, as opposed to a Las Vegas algorithm, which is defined as a "*randomized algorithm* that always produces correct results, with the only variation from one run to another being its running time" [6]. In other words, Monte Carlo algorithms are fast but not perfect, while Las Vegas algorithms take an indeterminate amount of time, but are guaranteed to be correct. Since the embedded controllers that typically control mobile robots have relatively low performance compared to their desktop cousins, a Monte Carlo algorithm suffices in a situation that does not always require accurate results. Although the system proposed in this text is not a true randomized algorithm, it is also not a crisp classical algorithm.

## VI. IMPLEMENTATION OF THE SYSTEM

The C code for implementing the neural network is based upon the discussion by John Bullinaria [7], and includes the simple feed forward multi-layer artificial neural-network with back propagation for the offline training. This implementation allows for a variable number of neurons in the input, hidden, and output layers. The code can be compiled in training mode which will train the network and print out a block of C source code which will initialize all weight values. The user can simply copy this block of code from the console window in which training was run, and paste it into the robot's source code to be recompiled for use in the robot.

## VII. TESTING THE SYSTEM

Once the state approximation system was implemented and debugged, the system was thoroughly tested with ARMS to prove that it is a good solution. Various environments were made with photo editing software and the robot was run inside each of them. Fig. 9 shows the navigation algorithm used for implementation.

The idea behind these tests is that if the arbitrary navigation algorithm chosen allows the robot to operate in a variety of environments, and that algorithm is based upon the fuzzy-neural system proposed in this article, then the fuzzy-neural system must be feeding useful and correct data to the navigation algorithm. The environment in Fig. 4 represents a real-world scenario. There are many small obstacles scattered randomly around the environment, which consists of oddly shaped and somewhat broken walls. The robot successfully navigated around the maze without colliding with any obstacles and without repeating its path. Although not repeating its path is not a constraint in the design of the fuzzy-neural system, it was part of the goal of the design of the navigation algorithm used in these trials.

Fig. 5 is an exact scale replica of the standard 2009 Trinity Firefighting Competition Maze [3]. The circle with the H represents the "Home" position, where the robot must start and finish its course. The goal is to hunt for a candle in one of the four "rooms" and extinguish it in as little time as possible. The robot was placed on the Home circle, aimed directly south, and run. The robot did not make the most efficient tour around the maze, but it did visit each room, and the likelihood of finding a candle would have been very high.

Fig. 6 shows the path of the robot through an oddly shaped maze, starting and ending in the upper left corner. Since this maze has a lot of tight spaces and the walls are not flat, the robot occasionally gets confused and makes some incorrect turns, but usually manages to overcome this confusion because of the momentum and state weights.

This system is useful for applications that are not maze based, too. Fig. 7 is an example environment with random obstacles and barriers. The robot randomly traverses around the obstacles and provides relatively good coverage of the environment after only a few passes around the perimeter of the environment. This shows that the randomized navigation algorithm is suitable for a wide variety of applications, and the fact that the randomized algorithm works is dependent on the fuzzy-neural system producing proper state identification values. Fig. 8 is yet another example of the robot in a dynamic environment. The robot has relatively good coverage, again, and did not collide with any obstacles.

## VIII. CONCLUSION

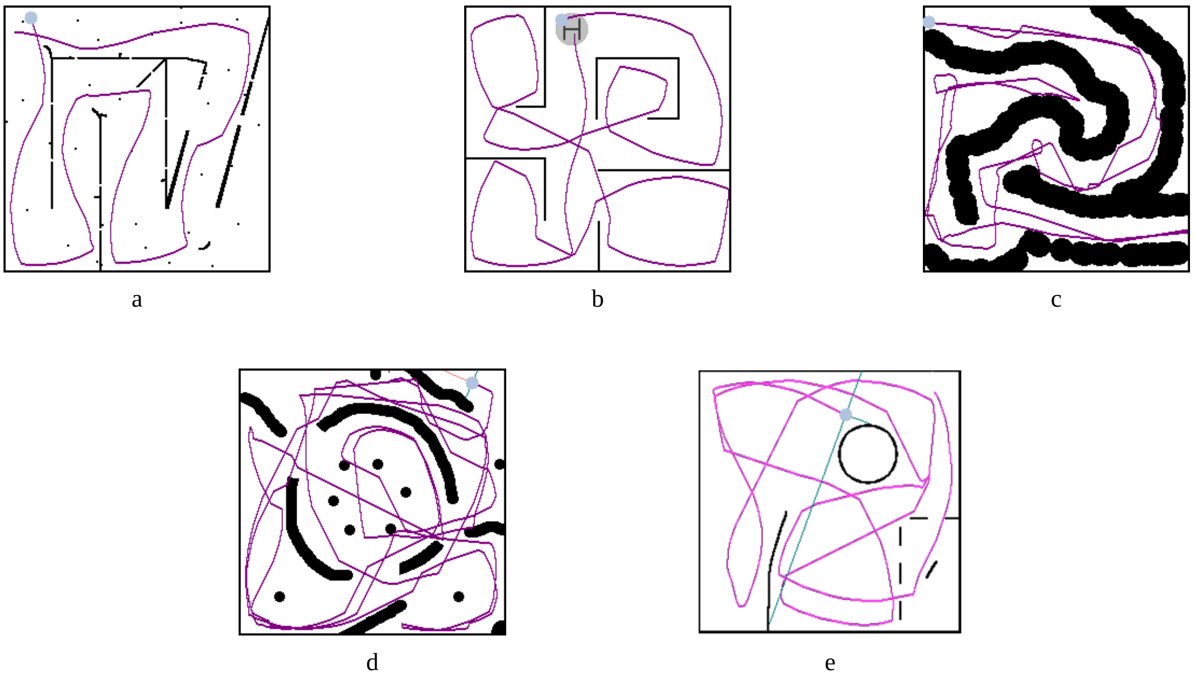
By running these simulations, it is clear that the fuzzy-neural system is taking sensor data and producing usable, and mostly correct, state approximations. The robot can detect and preemptively avoid obstacles, as well as navigate through small hallways, investigate openings, and change its course depending on what is surrounding it. Coupled with a powerful navigation algorithm, this system will be sure to produce excellent results.

## REFERENCES

[1] Mobile Robotics. (2009, September 7). Retrieved September 27, 2009, from Wikipedia: http://en.wikipedia.org/wiki/Mobile_robotics
[2] FIRST Robotics. (2009, May 13). FIRST At A Glance. Retrieved September 27, 2009, from USFIRST.org: http://www.usfirst.org/aboutus/content.aspx?id=160
[3] Trinity College. (2009, September 26). Firefighting Home Robot Contest. Retrieved September 27, 2009, from Trinity College: http://www.trincoll.edu/events/robot/
[4] Tahboub, K. K., & Al-Din, M. S. (2009). A Neuro-Fuzzy Reasoning System for Mobile Robot Navigation. Jordan Journal of Mechanical and Industrial Engineering , 3 (1), 77-88.
[5] Zein-Sabatto, S., Sekmen, A., & Koseeyaporn, P. (2003). Fuzzy Behaviors for Control of Mobile Robots. Systemics, Cybernetics and Informatics, 1 (1), 68-74.
[6] Black, P. E. (1999). Algorithms and Theory of Computation Handbook. Boca Raton, FL: CRC Press LLC.
[7] Bullinaria, J. A. (2002, November 18). Implementing a Neural Network in C. Retrieved September 2, 2009, from School of Computer Science; The University of Birmingham, UK: http://www.cs.bham.ac.uk/~jxb/NN/nn.html

**Michael A. Folcik** graduated from the University of New Haven, West Haven, CT 06516 USA with a Bachelor of Science in computer engineering in 2009, and with a Master of Science in electrical engineering in 2010, with a focus on computer engineering and artificial intelligence. He had experience with designing and implementing various mobile robots while at the University of New Haven, and founded an undergraduate robotics club. He currently works as an Innovation Engineer at Food Automation Service Techniques, Inc. in Stratford, CT 06615 USA, with a focus on robust embedded firmware. His current research interests are artificial intelligence and robotic navigation.

**Bijan Karimi** is a professor of Computer Engineering in the department of Electrical and Computer Engineering and Computer Science at University of New Haven, West Haven, CT 06516. He has conducted research in the areas of Neural Networks, Systolic Arrays, Digital Logic Testing, Digital Image Processing, and Robot Design and Navigation. He is the originator and the supervisor for the Robotics Club at the University of New Haven.



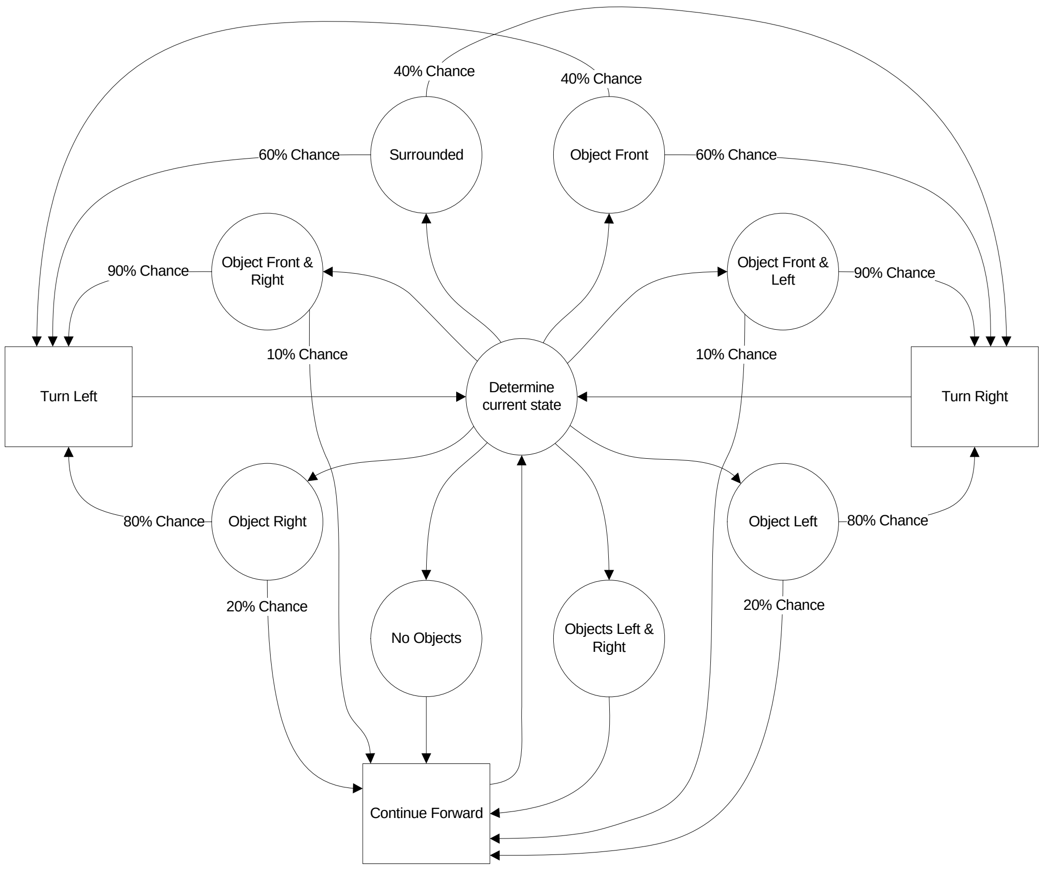


Fig. 9. Default Navigation Algorithm State Machine