ASEE 2014 Zone I Conference, April 3-5, 2014, University of Bridgeport, Bridgpeort, CT, USA.

Neural Network Based Obstacle Avoidance Using Simulated Sensor Data

Timothy A. Zimmerman
Department of Electrical and Computer Engineering
University of Hartford
West Hartford, Connecticut
timothy.a.zimmerman@gmail.com

***Abstract*—This study characterized the design and implementation of a low-cost autonomous robot capable of performing obstacle avoidance using neural networks trained with simulated sensor data. The only sensor used for detecting the environment was an infrared distance sensor attached to a hobby servo, allowing for 180° of sensor visibility. In order to train the neural networks, simulated sensor data was created using LabVIEW and presented to a user, who selected the expected robot operation in that specific situation. The simulated sensor data and expected robot operation data was then used to create training data. The trained neural networks were then verified by testing the actual network output with the training data, and additional random sensor data. The robot was controlled wirelessly by a computer running LabVIEW, which processed the sensor data through the networks and controlled the robot's subsequent movements. The networks were able to produce accurate obstacle avoidance actions during the simulated network analysis, and on the test bed, allowing the robot to avoid obstacles while successfully performing its mission.**

Keywords—neural network, backpropagation, infrared, LabVIEW, autonomous robot, dsPIC, microcontroller, obstacle avoidance, extended delta bar delta, simulation

I. INTRODUCTION

Robots and autonomous systems are becoming more commonplace throughout the human work environment, performing many tasks that are considered too mundane or hazardous for humans. However, many of these systems are restricted to movements within a predetermined range of motion. If robots are to "break free" of this restriction and allow autonomous navigation, efficient and accurate control paradigms must be conceived and researched.

One effective paradigm that can be used to successfully perform this type of autonomous operation is the neural network (NN). NNs mimic the way a biological nervous system, e.g. the human brain, processes sensory information. In these networks, simple computations are performed by individual nodes (i.e. neurons) which are then transmitted to other nodes through connections (i.e. synapses) whose connection strength can either enhance or diminish the signal before it reaches the next node. The desired output is calculated from these computations.

The effectiveness of the NN has allowed much research to be performed in regards to its application to robotics. Some of the research being performed includes: navigation and path planning [1], [2], physical orientation determination [3], obstacle and collision avoidance [4]–[6], manipulator and motor control [7], and programming of the robot by demonstration [8].

In this study, the robotic test bed constructed to investigate the obstacle avoidance capabilities of these NN's used a single infrared (IR) distance sensor attached to a hobby servo, giving a full 180º field of view around the front of the robot. The information from this sensor, when rotated through the full range of motion of the servo, was sent to the NN's for processing in order to determine the next course of action the robot needed to perform.

For ease of experimentation, the network was originally stored and processed on a PC running LabVIEW (a graphical programming environment). Once the best performing network is found, the network will be integrated into the robot firmware. This will require some modification of the network code in order to improve processing time.

II. ROBOT TEST BED

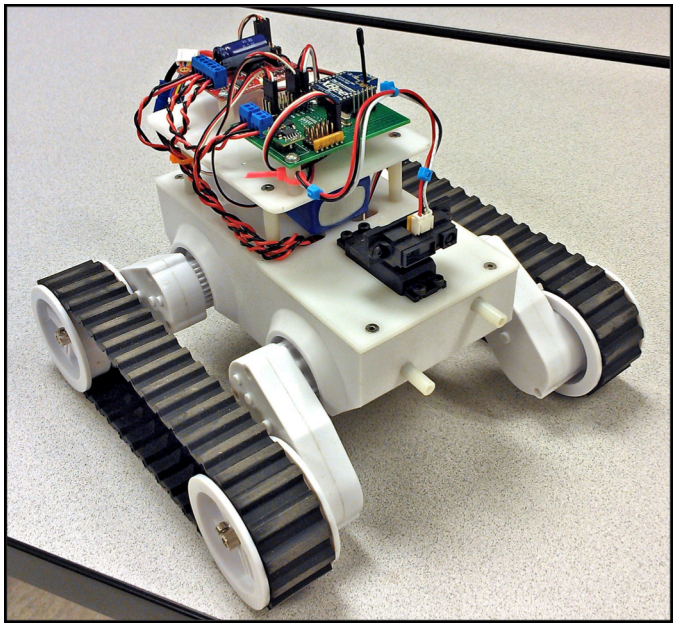
A. Robot Platform

The robot test bed was based on the "Rover 5 Robot Platform" from SparkFun Electronics (ROB-10336). The platform provided all aspects of locomotion required, including: integrated motors, optical encoders, and electromagnetic noise suppression. The platform's tracked differential drive also provided simple and consistent control for steering. Having all of these features available in a single unit greatly reduced development and manufacturing time of the test bed.

Attached to the top of the platform were two levels of Delrin plates. The first level held a 2100mAh 3S 11.1V lithium polymer battery (E-Flite) and a standard hobby servo (900-00005, Parallax, Inc.) which protruded into the Rover 5 body. On the second level were the custom built robot controller, a 300º/second MEMS gyroscope (LISY300AL, SparkFun Electronics), and a two channel 5A motor controller (R0403, Orion Robotics).

Mounted to the servo horn was a single IR distance measuring sensor (Sharp, GP2Y0A21YK0F) capable of accurately measuring distances from 10 to 80 cm. The sensor determined the distance based on the reflectivity of the object, which was converted by the sensor into an analog voltage to be read by the control system.

Fig. 1. Picture of the completed robot platform



B. Control System

The robot test bed was controlled by a custom circuit board, containing a single dsPIC33EP256GP502 digital signal controller, which was operated at a clock speed of 70.0416 MHz. In order to provide a stable clock for high-speed serial communications with minimal timing errors, a 7.3728 MHz crystal was chosen. Also on the control board was an XBee S1 802.15.4 1 mW serial modem, 7-36 V DC to DC converter, 3.3 V voltage regulator, and multi-function / multi-voltage headers for peripheral sensors and devices (i.e. gyroscope, motor controller, or servo).

When the robot performed a rotation, the gyroscope was used to guarantee that the robot executed an accurate rotation. This was done by integrating the rate of rotation over time to determine how much the robot rotated up until that point. The frequency of this calculation was 100 Hz.

C. Robot Test Bed Firmware

At the highest level, the firmware in the dsPIC33 contained a simple looped task structure, directing the robot to perform whichever task was being requested. Tasks were called based on internal and external robot events (i.e. serial packets, detected objects). Even though the robot was wirelessly connected to a computer running the robot control program, the physical navigation was not directly controlled by the computer. All robot control functions were handled within the dsPIC33.

In order to bypass the processing limitations of the dsPIC, the robot firmware would request the computer to perform computationally expensive operations. For example, upon finding an object was within the current path, the robot would automatically stop and scan its environment, and then request the computer to accept the sensor data in order to have the NN process it. The computer would then inform the robot which action it needed to perform next, based on the NN's output. The robot could also ask the computer how it must adjust itself in order to travel to the goal, since the LabVIEW program was monitoring the location of the robot (Section V).

On initial startup of the robot, it waited in an idle state until it was told by the computer to start traveling towards the goal. Once this process begun, it would continue until it received an "all-stop" command from the computer, either initiated by a user, or because the robot reached the goal.

All communication was performed with a custom protocol, defining different messages so they could be easily discerned from each other. The serial modem transmitted 8-bit values, so as to simplify the protocol, all commands were made to operate within these 8-bits. Although only 8-bits in length, in this paper they are referred to as "packets". In the packet, the upper 4-bits controlled the type of message, while the lower four bits controlled the "value" of that specific packet type, if applicable.

III. Neural Network Framework

In order for the robot to learn how to successfully navigate its environment, a neural network was trained to correctly process the sensor data and compute the proper solution. In this case, a back-propagation network was trained using an Extended Delta-Bar-Delta (EDBD) learning algorithm.

In a neural network, the connections between neurons within the layers (input, hidden, and output) have weights associated to them. Data is sent from the input layer to the hidden layer via synapses. And subsequently the hidden layer neurons then send their calculated data to the output layer, which then recalculates before outputting the data for use. Through each synapse, the data is modified by a weight value, which is determined during network learning. All of the previous layer's neuron outputs are connected to each neuron in the subsequent layer, where they are summed and applied to an activation function. This value is then output to each neuron in the subsequent layer, where the process is repeated, until it reaches the output layer.

The output (y_i) of each neuron (excluding the input neuron) can be summarized by Eq. [9]:

$$x_j^{[s]} = A\left(\Sigma_i\left(x_j^{[s-1]} \times w_{ji}^{[s]}\right)\right) \quad (1)$$

where w_{ji} is the weight value from the previous layer to the current layer (which is different for each synapse), x_j is the value of the j^{th} neuron of the previous layer, and A is the activation function.

For this specific application, the EDBD learning algorithm was used to minimize the global error (Eq. 4) of the NN [10]. The algorithm allows each synapse weight to have its own

learning rate, and changes each one independently on each iteration. The Δweight of each synapse is calculated:

$$\Delta w(k+1) = \alpha(k)\delta(k) + \mu(k)\Delta w(k) \quad (2)$$

where $\delta(k)$ is the gradient component of the weight change. From Eq. 2 each weight is then calculated:

$$w(k+1) = w(k) + \Delta w(k) \quad (3)$$

In order for the network to learn using the back propagation algorithm, the global error (e) at the outputs had to be fed back to the previous layers in order to modify the weights. The function used for the global error is described in Eq. 4 below:

$$e_j^{[s]} = A'(I_j^{[s]}) \cdot \Sigma_k (e_k^{[s+1]} \cdot w_{kj}^{[s+1]}) \quad (4)$$

where I_j is the weighted summation of inputs to the jth neuron in layer s [9]. The learning coefficient, $\alpha(k)$, and the momentum coefficient, $\mu(k)$, in Eq. 2 are defined in Eq. 5 and 6.

$$\Delta\alpha(k) = \begin{cases} \kappa_\alpha \exp(-\gamma_\alpha |\bar{\delta}(k)|) & \text{if } \bar{\delta}(k-1)\delta(k) > 0 \\ -\varphi_\alpha \alpha(k) & \text{if } \bar{\delta}(k-1)\delta(k) < 0 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

and

$$\Delta\mu(k) = \begin{cases} \kappa_\mu \exp(-\gamma_\mu |\bar{\delta}(k)|) & \text{if } \bar{\delta}(k-1)\delta(k) > 0 \\ -\varphi_\mu \mu(k) & \text{if } \bar{\delta}(k-1)\delta(k) < 0 \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where κ_α and κ_μ are constant coefficient factors for learning and momentum, φ_α and φ_μ are the constant coefficient decrement factors for learning and momentum, $\bar{\delta}(k)$ is the weighted exponential average of all the previous gradient components, and exp is the exponential function. Ceilings are also used on the coefficients to prevent large changes and oscillations of the weights.

NN learning is deemed complete when the RMS error of the network falls below a certain threshold, or when the network can no longer improve its error.

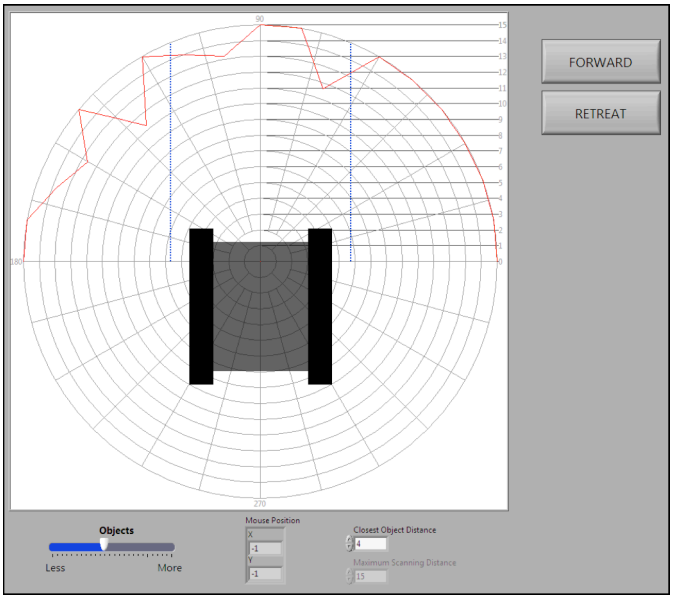
A. Simulation Data for the Neural Network

A NN learns by iterating through training data, which contains the input values and the expected NN output. The network will then process the input signals, producing an "actual" output based on its current weight values. Any deviation between the "actual" output and expected output will be back propagated through the NN. Doing so modifies the weights to correct for the error on the next iteration. The data used to train the NN must contain many unique sets in order to guarantee the NN can produce an accurate output.

In order to obtain this data, the plan was to have the robot learn from a user, who would drive the robot manually while LabVIEW recorded all of the sensor data and driver input. It was quickly determined that the amount of time it took the IR sensor to scan through all 180º (around 3 seconds) would not allow for a user to drive the robot at a sensible pace. If the speed of the robot was reduced to allow user operation, the pace would not allow for a large enough quantity of data sets to be recorded before the user quit from pure boredom.

To solve this dilemma, a simulation of the sensor data plot, shown in Fig. 2, was created in LabVIEW. The amount of objects and the minimum allowable distance from the robot to an object were user selectable to simulate real-world situations (e.g. sparse environments, very crowded environments, and everywhere in between). The computer mouse pointer was then hooked into the sensor plot; its position relative to the origin of the plot.

Fig. 2. Screenshot of navigational decision simulator



On initial start of the simulation, new random sensor data was generated and presented to the user. The user had to determine whether the robot should continue forward, retreat (perform a 180º rotation), or turn at an angle. If it was decided that the robot should turn to a specific angle, the user would click on the sensor plot in the direction the robot should attempt to traverse.

All of the simulation runs were stored in comma-separated value (CSV) files for later use. In all, 1101 data sets were created for the training of the NN. The data sets spanned many different situations that the robot would encounter, allowing the robot to make correct decisions regardless of what obstacles that were presented to it.

B. Neural Network Implementation

The network was created and trained using NeuralWare's Professional II/PLUS (ProfIIPlus) software. After much trial and error testing with different learning algorithms, momentums, learning rates, activations functions, and a plethora of other settings, it was decided that a

backpropagation algorithm with EDBD learning, a sigmoid activation function, and 18 nodes in the hidden layer would be sufficient for accurate training.

The NN was configured to have 19 inputs for each distance value coming from the IR sensor, and originally 22 outputs: 19 angles, forward, retreat, and an angle. If the network decided that the robot should rotate to a specific angle, forward and retreat were expected to be "off", and an angle output would instead be activated. In order to guarantee that only one output was activated, the algorithm tested all network outputs in order to determine which one had the greatest value.

It was discovered early in the network training that the NN was having difficulty trying to account for every different situation it may encounter. RMS error values were not within the expected range for this research so it was decided to split the NN into two separate networks.

Both networks had the same 19 angle inputs, but their outputs differed. The first network simply decided whether the robot should continue forward, retreat, or rotate a certain amount (named the "FRA" network). If the network decided that an angle needed to be selected, the second network was called to process the same input data and produce a rotation angle (named the "Angle" network). Separating the networks greatly improved the RMS error of both networks.

The NeuralWare software allowed for the networks to be exported as C/C++ code for use in other programs and platforms. After the code was exported, it was modified to remove sections of code that were not needed, specifically all sections written for C++.

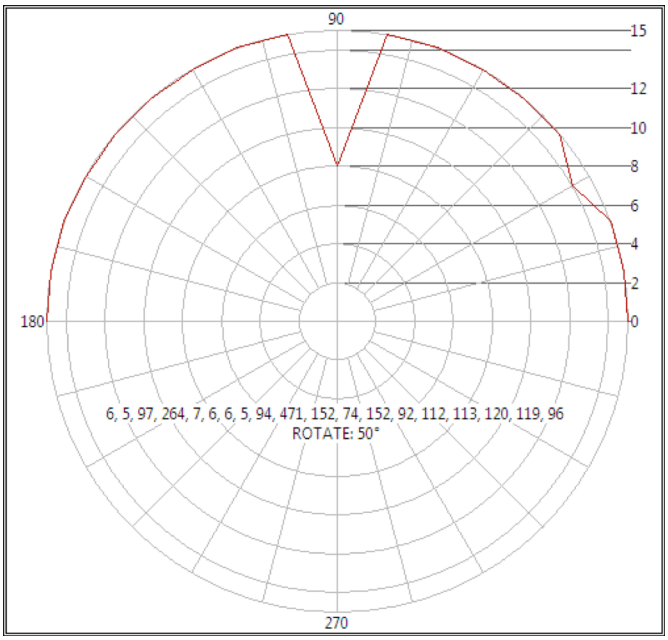
Reading through documentation regarding running external code through LabVIEW virtual instruments (VIs), it was determined that the quickest and most efficient way to import the NN code was to wrap it in a dynamic link library (DLL). To accomplish this, the code was imported into Microsoft Visual Studio C++ 2010 Express where a DLL wrapper file was created. Since the DLL simply wraps the NN C code, it allowed for simple updates to the NN in the future, if the need arose.

To test the functionality of the NN, a NN Data Simulator (NNDS) was created in LabVIEW. This program was a repurposed version of the user driving simulator, modified to send the sensor data to the two NNs, instead of creating simulation data from user input. On each execution, random sensor data was generated and processed through the NNs. After performing many simulations with this program, it was found that the NNs were able to accurately determine the proper robot action to perform.

IV. Obstacle Avoidance

As described in Section II Part C, once the robot was told to begin traversing towards the goal, it automatically initialized the obstacle avoidance process. While the robot was driving forward, the servo directed the sensor to perform a distance scan at three positions: directly in front of the robot (90º), and ±30º from the forward position (60º and 120º). If an object was detected at any of these positions within 22 cm (8.6 inches), the robot would have stopped all movement and perform a 180º scan of the environment.

Fig. 3. Example of distance data from a 180º scan.



In total, 19 distance measurements were performed in 10º increments from right to left. An example of the distance data is shown in Fig. 3. The figure displays an actual robot scan, in which an object was detected 35.5 cm (14 inches) away at 30º, and another object 20.3 cm (8 inches) away at 90º. The plot also displays the raw sensor data received from the robot, and the NN's decision.

The decision of the NN was then sent back to the robot in the form of a requested robot action. If a retreat or angle rotation was requested, the robot would have begun traversing forward once the requested action was complete.

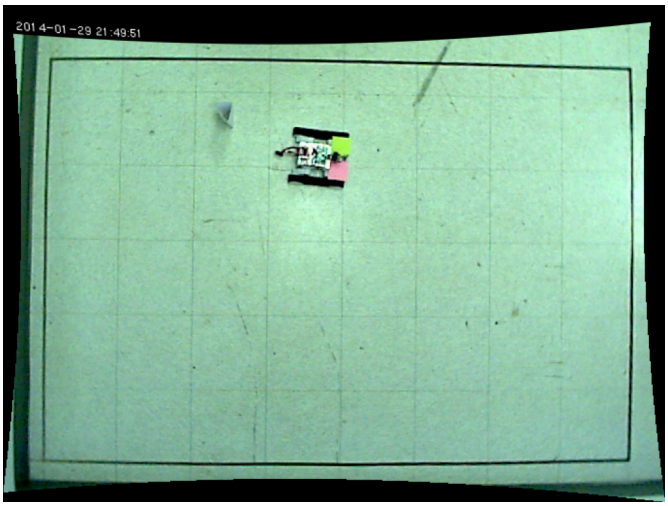
V. Robot Navigation

In order for the robot to have an obstacle to avoid the previously stated method, it would require some form of "objective", or goal. While the goals created in this research were arbitrary and virtual, the real-life application of this navigational goal is limited only to the imagination of the reader.

For the physical testing of the autonomous navigation of the robot, a test area was created which measured 165 cm by 244 cm (5 feet 5 inches by 8 feet). An Axis M1013 network camera was then positioned 3.81 meters (12 feet 6 inches) above the test area, allowing for a complete view of the test area for monitoring robot operations.

In order to translate the robot position within the image to a valid linear system, LabVIEW was used to correct for the barrel distortion of the lens. The image after the distortion correction is shown in Fig. 4. Note the curvature of the edges of the original image.

Fig. 4. Image from the Axis M1013 camera after distortion correction.



While the robot was easily visible to the human observer, the vision system needed an accurate method of determining the position and heading of the robot. This was done by placing two vision targets of different colors on the front of the robot (green and pink bright-colored Post-It notes). With the addition of these targets, the vision system was capable of determining the position and heading of the robot anywhere within the test area.

A. *Navigational Goals*

The navigational goals for the robot to traverse towards were created virtually within the LabVIEW control program. When a goal was created, it was given a random position on the opposite side of the test area from where the robot was currently residing, providing longer trial times and the opportunity to encounter more obstacles.

After a specific amount of time, determined by the robot, it would request an update on the error between its current heading and the heading it needed to traverse in order to accurately travel to the goal. If an adjustment was needed, the robot would perform the adjustment before continuing towards the goal. This allowed the robot to clear any previous obstacles before correcting its heading.

Once the robot got within 10 cm (4 inches) of the goal, the LabVIEW control program automatically generated a new goal location.

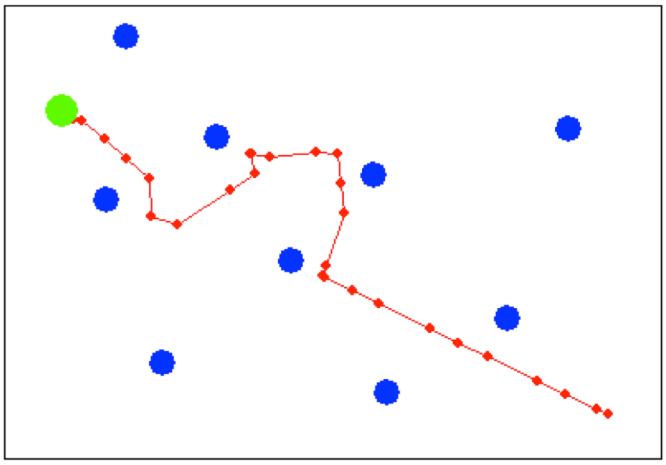
B. *Data Acquisition*

The refresh rate of the test area image within the LabVIEW control software was 1.5 seconds. At each image capture, the robot position and heading were stored within a log file, along with any robot packets that were transferred between the robot and the computer, neural network data, and the original image. This was done not only to document the traversing of the robot throughout the test area, but also to improve the NNs.

For example, if the robot happened to come upon an obstacle and the NNs decided the best course of action was to hit the object head-on, the input data and results of that decision would be stored in the log. That data could then be "scraped" from the log and added to the training data with the correct expected output. When the NN was retrained, this added data set would have prevented the network from performing that action again.

To document the motion of the robot around the test area, a map was generated within the LabVIEW control program to display the robot's travel. On each image capture, the current position of the robot was added to the image, along with a line from the previous data point, to show the path traveled. Upon the robot reaching the goal, the image was stored, and a new one was created. An example of this is shown in Fig. 5. Objects, which were rolled pieces of letter size paper with a diameter of 2.5 cm (1 inch), had blue Post-It notes attached to them, which the control program displayed as blue dots.

Fig. 5. Example of the map created from robot position data.



VI. Experimental Results

In order to analyze the accuracy of the NNs, experiments were performed using sensor data from three different sources: the original training data, simulated sensor scans, and physical autonomous robot operation.

A. *Network Analysis Using Original Training Data*

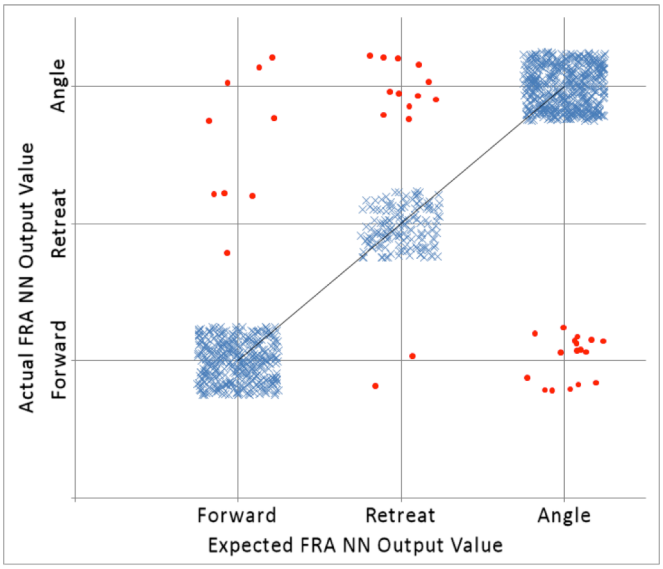
After the NNs were trained in ProfIIPlus, the test data was processed on both networks to determine the error between the expected and actual outputs. The classification results of these two tests are shown in Fig. 6 for the FRA network and Fig. 7 for the Angle network. In order to visualize the large concentration of data points in the figures, a random amount of "jitter" was added to each point in the x and y axis.

Of the 1101 data points tested with the FRA network, only 40 were misclassified (shown in Fig. 6 with circular markers), and 9 of the 40 appear to have been caused by invalid expected values in the training data. While results of the test on the FRA network yielded a classification error of 3.63%, the 9 points of invalid test data were disregarded, resulting in a classification error of 2.82%.

To determine why these 9 sets were misclassified, the original input data was analyzed. All of the "Angle" points that were misclassified as "Forward" contained objects that were directly in front of the robot, but were distant. It is

believed that this was caused by training data which included an attempt at creating robot "curiosity". Some of the simulated sensor data presented by the NNDS lacked enough information to make an accurate navigational decision. In order to gain more information on the objects, it was planned to have the robot move a short distance at a certain angle in order to gather more information about the obstacles. In most of these situations, the robot was instructed to "rotate" to 90° (forward) to get this extra information. When the training data was created, any "rotation" to 90° was converted to a "Forward" command. This is believed to be the source of the misclassification.

Fig. 6. Actual vs. Expected output of FRA network

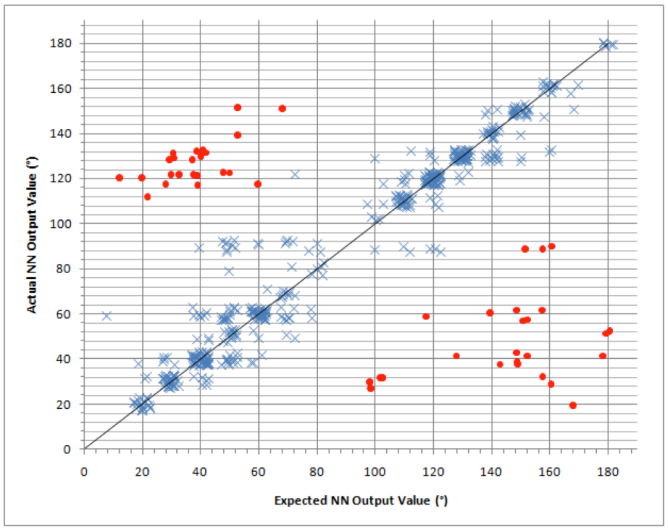


All of the "Retreat" points that were classified as "Angle" appear to have been caused by the sensed objects' distance from the robot. The closest distance of any object in all of the misclassifications was 28 cm (11 inches). Regardless of the rotation the robot would have performed, the proximity of the other objects would have promptly caused a sensor rescan of the environment.

All of the "Forward" points that were classified as "Retreats" appear to have been caused by some sort of pattern matching by the NN. In all of these cases, there were spans of 50° or greater between objects, while the opposing side contained a high population (2-4) of objects. It is believed that the NN tended to classify scans which contained many objects concentrated on either side of the robot as an "Angle" movement, rather than "Forward", as this pattern was typical of most of the correctly classified "Angle" rotations.

Of the 935 data points tested on the Angle network, shown in Fig. 7, the 389 points that made up the 90° category were removed. Of the remaining 546 points, 49 were misclassified, giving an error of 9.18%. After reviewing the data, it was decided to redefine a misclassification as any result that deviated more than 50° from the expected result, and contained an object directly within ±10° of the subsequent path of the robot. The ±10° rule was added after noticing many of the sets of data successfully avoided obstacles, but did not do so at the expected angle in the training data. These modifications reduced the number of misclassifications to 17, lowering the error to 3.11%.

Fig. 7. Actual vs. Expected output of the angle network



Again, each group of misclassified points was analyzed to determine the reason for their misclassification. It is believed the network was more than likely affected by the same robot "curiosity" that caused misclassifications in the FRA network.

B. Network Analysis Using Simulated Sensor Trials

In order to verify the accuracy of the generated NNs, the NNDS program was modified to deliver randomly generated sensor data to the networks. This exactly mimicked how the networks would process data during normal robot operations, and provided data which differed from the 1000+ data sets that were used as training and testing data.

The networks were first tested with a minimal amount of simulated objects (around 2-3) ranging from 28 to 38 cm (11 to 15 inches) away from the robot. Of the 105 simulations, only 3 showed questionable behavior. In each of the three cases, the networks decided that the robot should rotate to angles that would have put the detected objects very close to the robot's direction of travel.

The second round of simulations added more objects (around 4 to 8), at the same range as the previous simulation. After 100 runs, 5 showed questionable behavior. Again, the network was attempting to have the robot rotate to angles that were very close to detected objects.

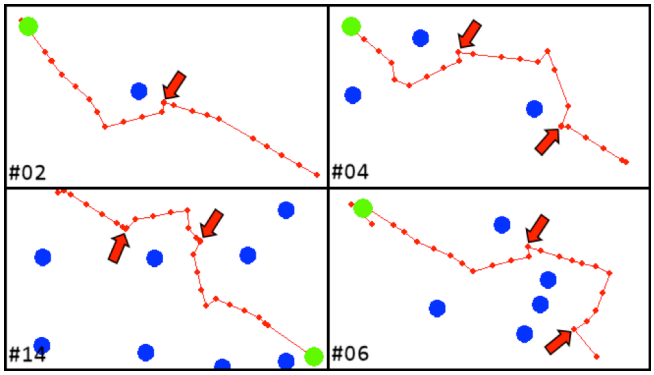
The final simulation used the same amount of objects, but allowed them to be produced within 18 cm (7 inches) of the robot. Doing so produced many "Retreats", as expected. Some of the runs contained gaps between the objects which appeared to be large enough for the robot, but the robot decided instead to retreat. The classification of these situations could be improved upon by adding more training data sets which define the proper navigational decision.

C. *Autonomous Robot Operation Trials*

The robot was tested using a variety of obstacle and goal placements, resulting in 20 trials. Of the 20 trials, 2 caused behavior which resulted in the robot being incapable of finding the goal, and 2 others collided with objects. Fig. 8 shows four of the trials where the robot was able to successfully navigate through the obstacles and find the goal. Each arrow points to the location where a full 180° scan and subsequent NN calculation were performed.

The two trials where the robot was incapable of finding the goal appeared to be caused by the NNs constantly reporting that the robot needed to retreat. In all of these situations, the objects were within 4 inches of the front of the robot as would be expected in environments with a large amount of obstacles. It appeared that the robot was not capable of properly determining the correct course of action when an object was within close range, as well as invalid distance measurements from the IR sensor.

Fig. 8. Map of Robot Trials #02, 04, 14, and 06



The two trials where the robot collided with obstacles appeared to be caused by the top-level navigation program and the lack of side facing sensors to prevent side collisions. In each case, the collisions occurred on the side of the robot, which is where it did not have active sensors to prevent those collisions. It also appeared that in certain situations, the sensor did not accurately inform the control system that there was an object in front of the robot even though it was within view of the sensor. These situations, however, did not result in collisions, as the objects were usually outside of the robot's direction of travel or eventually caused the robot to perform a 180° scan.

VII. Discussion

When analysis was performed on the training data, it revealed the data generated by the NNDS program formed a bimodal distribution (with the 90° data points removed). A comparison of the percentage of population at each angle and the percentage of misclassification at each angle of the training data analysis was performed to verify that the bimodal distribution of training data sets did not cause any of the misclassifications due to some angles having more training sets than others. As evident in Fig. 9, it was concluded there was no correlation between the two variables.

The results from the training data analysis were also reviewed to determine the distribution of expected and actual network output, which is shown in Fig. 10. The actual classifications performed by the network did show a distribution similar to the expected classifications, although it appeared the network preferred certain angles over others. The source of the angle preference was found after analyzing the amount of misclassified results at each angle, which is shown as the third series of data in Fig. 10. This additional series of data shows the extra classifications were a result of the misclassifications of the network.

Fig. 9. Comparison of % Misclassifications and % Population

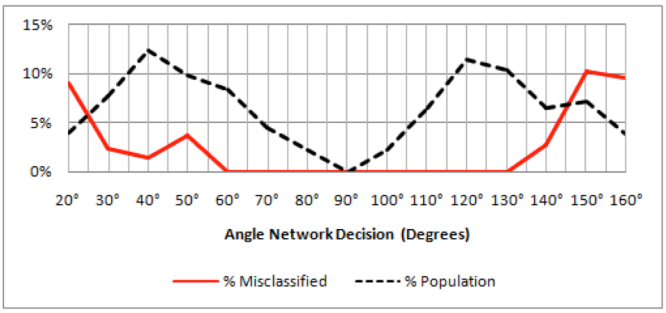


Fig. 10. Comparison of % Classifications and % Population

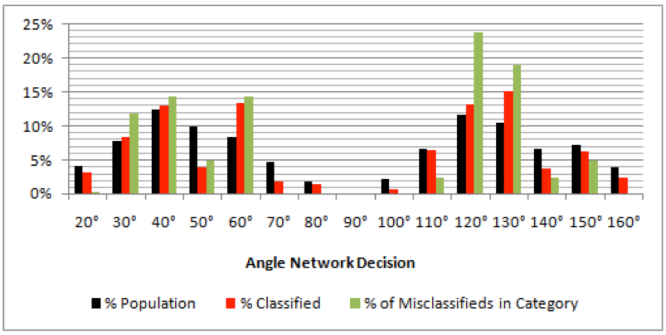
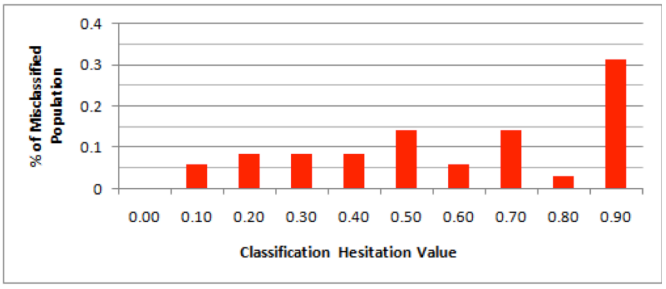


Fig. 11. Percent of misclassified population vs. hesitation value



While analyzing the data, it was noted that the network output value of many of the misclassified data sets had a small value (<0.50). In order to improve the operation of the angle network, it was proposed that the largest output value of the network could be used to determine whether the network was confident in its decision, or if more data needed to be obtained. This value was called the "hesitation" value of the network. After gathering the data, it was determined that the output value was not practical for expressing the amount of

network hesitation because a majority of the misclassifications had values of over 0.50. The distribution is shown in Fig. 11.

It was believed that the IR distance sensor was the primary source of erroneous object detection on the robot test bed. Many sensor scans returned object data that did not appear to be accurate, in regards to the actual distances. During robot traversing, there were also situations when the object detection of the sensor would either delay or sometimes not even sense the object, resulting in the robot colliding with the object, or performing its reading so close to the object that the result of the NN was simply to retreat. After reviewing the datasheet for the sensor, the physical orientation of the sensor appeared to be part of the issue, more than likely because of the position sensitive detector (PSD). Rotating the orientation of the sensor so both the sensors emitter and PSD are aligned vertically may have helped to reduce some of the variability as the light from the emitter transitions from the background to an object. The accuracy may have also been improved by collimating the emitter light into a tighter beam, but this was not tested.

None of the misclassifications by the NNs were affected by the sensor because the sensor was not involved in collecting the training data. If the sensor had been used to collect training data instead of simulating the data, the inaccuracy of the sensor readings would have been introduced training process. Whether or not the network would have been able to learn this error and compensate for it was not tested.

A major limitation of the sensor was also the amount of time needed to obtain an updated sensor value. The analog voltage of the sensor did not change for at least 38.3 ms ±9.6 ms, which made the time required for a 180° scan (with servo position updates) around 2−3 seconds. This not only limited the speed of obstacle avoidance decisions, but also required that the decisions be limited to angular rotations, and also did not allow a user to physically drive the robot while the control system gathered data (hence the simulated sensor data). If an improved sensor was used with a faster update rate (10+ Hz), a form of dynamic control could have been implemented, where the robot would not have had to stop at each object to perform a scan, but instead actively traversed around the object.

Although over 1000 simulated sets of simulated sensor data was created for the training of the NNs, not all situations the robot encountered were included in the data, resulting in some unwanted actions. This was especially evident when the robot found itself close to a wall. The simulation program did not produce situations which imitated a large object directly in front or to the side of the robot, resulting in the FRA NN deciding the best course of action was to "retreat". Many of the situations where the robot test bed detected the wall could have been traversed better with a decision other than "retreat".

VIII. Conclusion

In this paper it has been shown that a back-propagation neural network trained with simulated sensor data is capable of avoiding obstacles in order to accomplish a navigational goal. When combined with more accurate sensors, this low cost system could be added to other devices, allowing autonomous obstacle avoidance capabilities.

References

[1] K. Park and H. Choi, "Neural Network Based Path Planning Plan Design of Autonomous Mobile Robot," in *2006 SICE-ICASE International Joint Conference*, 2006, pp. 3757–3761.

[2] S. Neusser, "Developments in autonomous vehicle navigation," *CompEuro' 92. "Computer Syst. Softw. Eng. Proceedings,"* pp. 453–458, 1992.

[3] M. P. Paulraj, R. B. Ahmad, C. R. Hema, and F. Hashim, "Estimation of mobile robot orientation using neural networks," *2009 5th Int. Colloq. Signal Process. Its Appl.*, pp. 42–46, Mar. 2009.

[4] H. T. Trieu, H. T. Nguyen, and K. Willey, "Obstacle avoidance for power wheelchair using bayesian neural network.," *Conf. Proc. IEEE Eng. Med. Biol. Soc.*, vol. 2007, pp. 4771–4, Jan. 2007.

[5] D. Tsankova, "Neural Networks Based Navigation and Control of a Mobile Robot in a Partially Known Environment," pp. 197–223.

[6] I. Engedy and G. Horvath, "Artificial neural network based mobile robot navigation," in *2009 IEEE International Symposium on Intelligent Signal Processing*, 2009, pp. 241–246.

[7] M. Meng, "A neural network approach to real-time motion planning and control of robot manipulators," *IEEE SMC'99 Conf. Proceedings. 1999 IEEE Int. Conf. Syst. Man, Cybern. (Cat. No.99CH37028)*, vol. 4, pp. 674–679, 1999.

[8] M. Stoica, G. a. Calangiu, F. Sisak, and I. Sarkany, "A method proposed for training an artificial neural network used for industrial robot programming by demonstration," *2010 12th Int. Conf. Optim. Electr. Electron. Equip.*, pp. 831–836, May 2010.

[9] NeuralWare, *Neural Computing - A Technology Handbook for NeuralWorks Professional II/PLUS*, 5.50 ed. Carnegie, PA, 2001, p. 334.

[10] A. Minai and R. Williams, "Back-propagation heuristics: a study of the extended delta-bar-delta algorithm," *Neural Networks, 1990., 1990 IJCNN ...*, pp. 595–600, 1990.