

ON DATA FUSION METHODS USING NEURAL NETWORKS, FROM A PRACTICAL IMPLEMENTATION POV*

ing. Radu Bogdan Rusu¹

Robotics Research Group,
Faculty of Automation and Computer Science,
Technical University Of Cluj-Napoca
Barițiu Str. no.26-28, Cluj-Napoca, Romania
Website¹: <http://www.rbrusu.com> , <http://www.robotux.info>

Abstract:

Instead of the usual abstract, I'm using this section to express my gratitude to the open source community. This world truly is a better place to live and study because of them. Inspired by the "open source" path, I'm starting a series of tutorials, targeted at the people who are just starting out in the field of mobile robotics. The "unpublished papers series" are just basic tutorials on things that should already be open sourced, but are not (or I couldn't find them). Their purpose is not to be used as a source of copy&paste, but as a beginning point. Some of them might seem trivial, but remembering how I started to learn, they are not.

Keywords: mobile robots, data fusion, neural networks, artificial intelligence, Player/Stage, Javaclient, Joone.

Note: This tutorial is released under GPL (GNU General Public License - <http://www.gnu.org/copyleft/gpl.html>)

1. BRIEF INTRODUCTION

Mobile robotics is an emerging field of study. Several methods and algorithms have been developed and studied over the years, with the purpose of controlling the way mobile robots navigate in their environments. This paper is presenting one of them.

2. FEED FORWARD NEURAL NETWORKS

Artificial Neural Networks are mathematical algorithms that are able to learn mappings between input and output states through supervised learning, or to cluster incoming information in an unsupervised manner[5].

Every neural network has two components:

- **nodes**, also known as *neurons*
- **connections** between nodes, also known as *synapses*

.----=== “unpublished papers” series ===----

The neurons are connected with each other via synapses. Each synapse has a weight attached. The output of a neuron is usually calculated with a function such as:

$$y_k = f\left(\sum_{j=0}^m w_{kj} * x_j\right)$$

Where y_k is the output of the neuron, f is an activation/transfer function, w_{kj} is the weight attached to synapse j for the neuron k , and x_j is the input signal of the neuron.

The literature is abundant on explanations on how an artificial neural network works. Basically, ANNs are much like our brains, but simpler in structure. While a human brain consists of about 20 to 50 billions of neurons, you won't usually need more than 100 neurons in your artificial neural network.

One of the simplest type of neural networks is the *perceptron*. The basic perceptron consists of two layers of units: the input layer and the output layer. The perceptron is also known as a “single layer neural network”.

A “feed forward neural network” contains at least 3 layers: an input layer, at least one hidden layer and an output layer. A simple FFNN is shown in Figure 1.

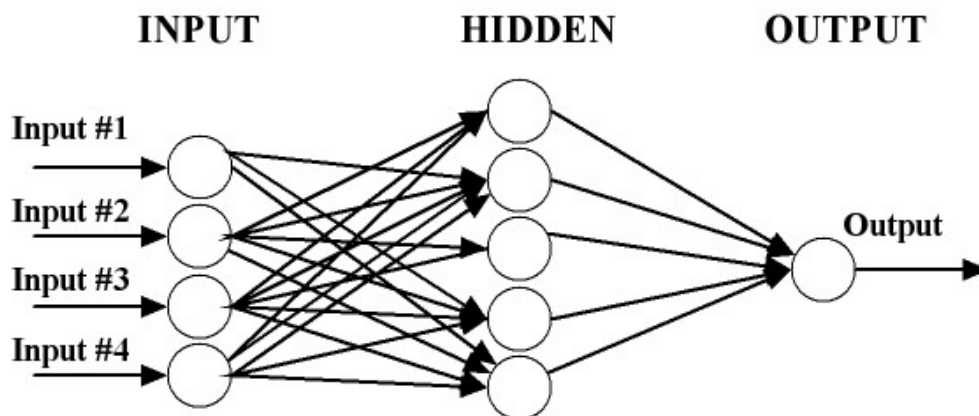


Figure 1. A simple feed forward neural network

To make a long story short, the feed forward neural network is probably the simplest form of an artificial neural network (apart from the perceptron) found in practice. For other types of ANNs, consult the adequate literature.

3. SENSOR DATA FUSION

One of the problems with mobile robots is that they are equipped with different types of sensors. The goal of the control algorithm is to map the sensory input to motor commands, in the best possible way. But different sensors, “sense” the world in a different way. For

.----=== "unpublished papers" series ===----

instance, let's suppose we have a robot with the following configuration:

- 8 ultrasonic (sonar) sensors
- 8 infrared (ir) sensors (placed in the same configuration as the sonar sensors)
- odometry shaft encoders

Let the robot wander around, and gather data from the environment, using for starters just the sonar and ir sensors. Build a graphic from the values of each sensor type, and you will see that they differ. That's normal, since their working principle is different. While the sonar sensors use sound waves, the infrared ones use light beams. The waves and the beams bounce off objects and return to the robot, and the sensor will give you the "distance" between the robot and the object(s). Problem is, that different objects respond differently to the above sensors.

But, you probably know this already if you had a course in basic robotics or sensors. Getting back to the problem though, who should you trust? If the ultrasonic sensors say that an object is 1 meter away, while the infrared sensors say that it is just 70cm away, which value should you consider? One safe bet would be to consider the smallest value, but that will most certainly limit your robot's working space.

As always, there are a lot of techniques which could be implemented in order to solve this problem. This tutorial/paper uses data fusion with neural networks techniques to achieve the desired behavior.

Assuming that you can assign some predefined weights to each type of sensor, you could calculate the distance using a formula such as:

$$d = \sum_{j=0}^n \text{Sensor}_j * \text{Weight}_j$$

Problem solved. Apparently. The only thing we haven't considered yet is that the weights could change over time, or from object to object. So an algorithm that can automatically change the weights is desired.

A simple solution is to apply the same principle on a feed forward neural network. For the sake of experimentation, we will consider a robot with:

- 16 ultrasonic (sonar) sensors (Pioneer2-like)
- Odometry information from both motors

.-----= "unpublished papers" series =-----.

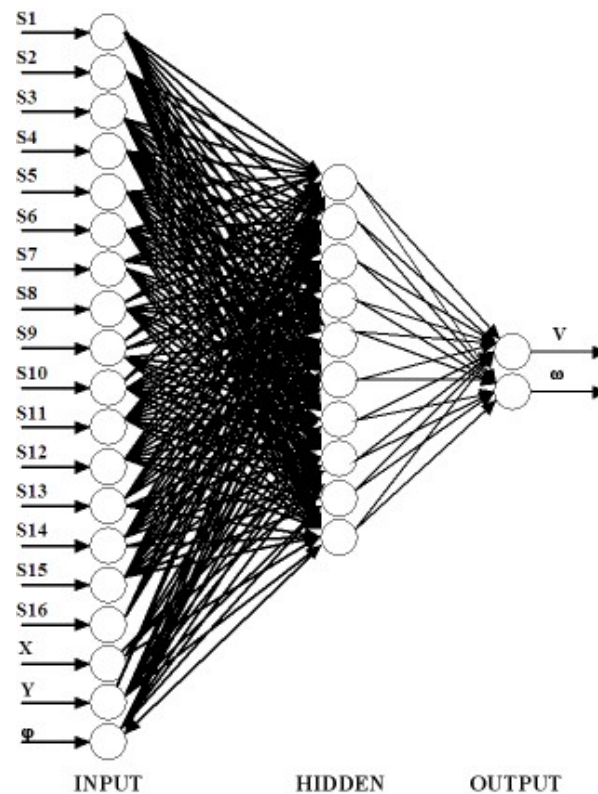


Figure 2. 19-10-2 FFNN for our robot

A simple feed forward neural network with three layers has been chosen (see Figure 2). The network has 19 input neurons (16 sonars + X,Y, Φ position from odometry), 10 hidden neurons and 2 output neurons (Xspeed and YawSpeed). See [7] for a similar example.

The activation functions are sigmoids, thus on a [0,1] domain. The learning rate was set to 0.3, the momentum to 0.3 and the number of training cycles to 500.

4. PLAYER/STAGE, JAVACLIENT, JOONE

The above pieces of software need no presentation for those of you who are already a part of the robotics worldwide community. Still, for those readers who are just starting out, here's a brief introduction:

- ▶ Player/Stage (<http://playerstage.sf.net>) is currently the best open source robot server-simulator available. As the name suggests it has two major components (well actually 3 if we count the 3D simulator, Gazebo, too): Player and Stage. Player is the robot server, supporting a variety of robot hardware. It has a modular architecture (which makes it easy to add support for new hardware), and an active user/developer community. Stage is a 2D simulator for mobile robots and sensors. The user writes robot controllers and sensor algorithms as 'clients' to the Player 'server'. The best feature of Player/Stage is that it was designed to be language and platform independent. We currently have

.----== "unpublished papers" series ==----.

clients for C++, Java, Tcl, Python, Lisp, Octave, Ruby and C#.

- ▶ Javacient (<http://java-player.sf.net>) is the Java interface between the Player/Stage project and the Java programming language. Its threaded design makes it a powerful tool for robot programming, using the simplicity and gracefulness of Java.
- ▶ Joone (<http://www.jooneworld.com>) is the Java Object Oriented Neural Engine, a free neural network framework, which permits the user to train and test artificial neural networks.

Using these three tools, everyone can implement and test neural networks using the Java programming language for as many physical or simulated robots he/she wants.

For more information, please check the "source codes" section.

5. EXPERIMENT

Following the very simple idea in [7], I built a small map (see Figure 3) for the Stage simulator. My goal was to control the robot using a neural network to walk around the central box, without user intervention, and without bumping into the walls.

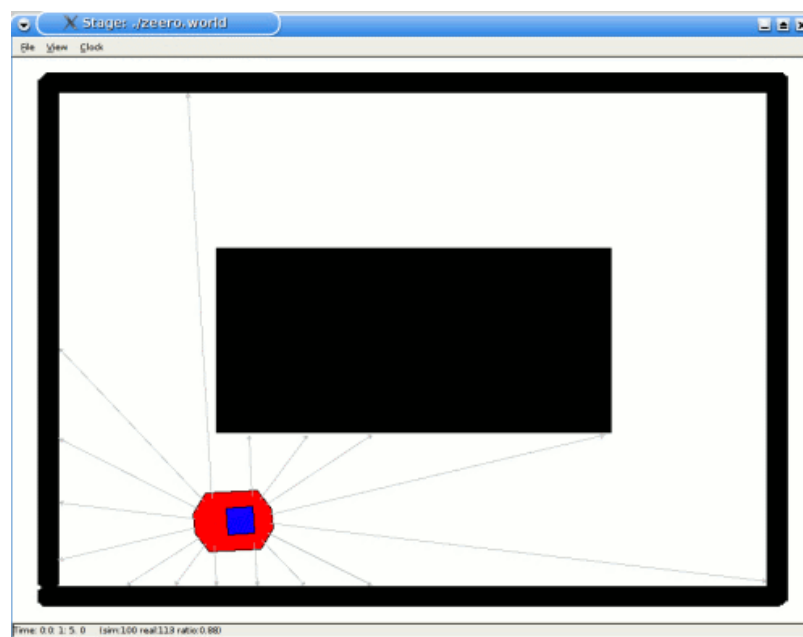


Figure 3. A simple simulated Stage map

Training the neural network was a cumbersome process. First, I moved the robot around the box two times, in order to gather the sensor and odometry data. Then I had to normalize the values. The training data was fed to the FFNN, and after around 500 epochs, the resulting neural network was saved to a file. Finally, the neural network was opened by another piece of software that controlled the

.----- "unpublished papers" series -----.

simulated robot using Javaclient. The basic architecture of the entire process is shown in Figure 4.

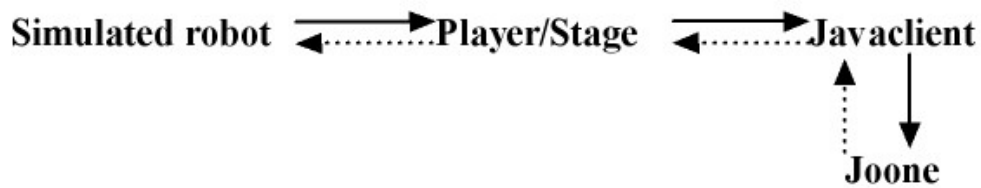


Figure 4. The used architecture

After some attempts, I finally got it right. While the robot was moving around controlled by the ANN, I saved its odometry data to a file, and then I used the (x,y) information to compare it with the trained data. The results can be seen in Figure 5. The original information training (x,y) set is shown with blue and the resulting (x,y) position with red.

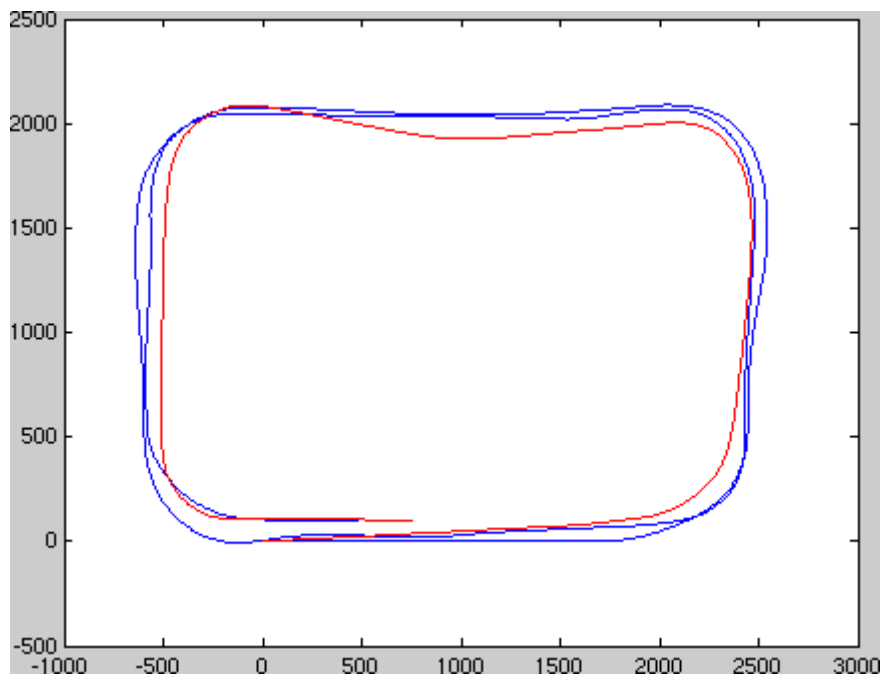


Figure 5. The results

6. CONCLUSIONS

This tutorial presented a very simple way to implement and use neural networks for the purpose of controlling your mobile robot. The chosen feed-forward neural network works quite well for the simulated map. You are of course encouraged to play with it, change the number of neurons in the hidden layer or train it with different data.

Some other nice implementations can be found in [1], [4] and [7].

They say that a picture is worth a thousand words. Following the same path, in my opinion, a piece of source code is worth a thousand papers. That is why I'm including the full source code used for this

.----== "unpublished papers" series ==----.

small experiment. See the source code section for more information on how to compile and run it. Just remember: if you manage to understand this tutorial, you're in. There is only a small step between this and using unsupervised learning Kohonen neural networks (Self Organizing Maps) or using data gathered from video cameras or laser sensors as inputs to your network.

7. REFERENCES

- [1]U.R. Zimmer, E.von Puttkamer, *"Realtime e-learning on an Autonomous Mobile Robot with Neural Networks"*, Euromicro '94 – Realtime Workshop, Vaesteraas Sweeden
- [2]B.Barshan, B. Ayrulu, S.W.Utete, *"Neural Network-Based Target Differentiation Using Sonar for Robotics Applications"*, IEEE Transactions on Robotics and Automation, vol.16, no.4, August 2000
- [3]G. Dudek, M. Jenkin, *"Computational Principles of Mobile Robotics"*, Cambridge University Press
- [4]G. Chronis, M. Skubic, *"Experiments in Programming by Demonstration: Training a Neural Network for Navigation Behaviors"*
- [5]U. Nehmzow, *"Mobile Robotics: A Practical Introduction"* second edition, Springer-Verlag London, 2003
- [6]R.C. Arkin, *"Behavior-Based Robotics"*, MIT Press, 1998
- [7]P. Martin, U. Nehmzow, *"Programming by Teaching: Neural Networks Control in the Manchester Mobile Robot"*
- [8]D. Janglova, *"Neural Networks in Mobile Robot Motion"*, Neural Networks in Mobile Robot Motion, pp. 15-22, International Journal of Advanced Robotic Systems, Volume 1, Number 1, 2004

Note: The source codes are available on the www.robotux.info web site, under the Artificial Intelligence->Neural Networks section.