

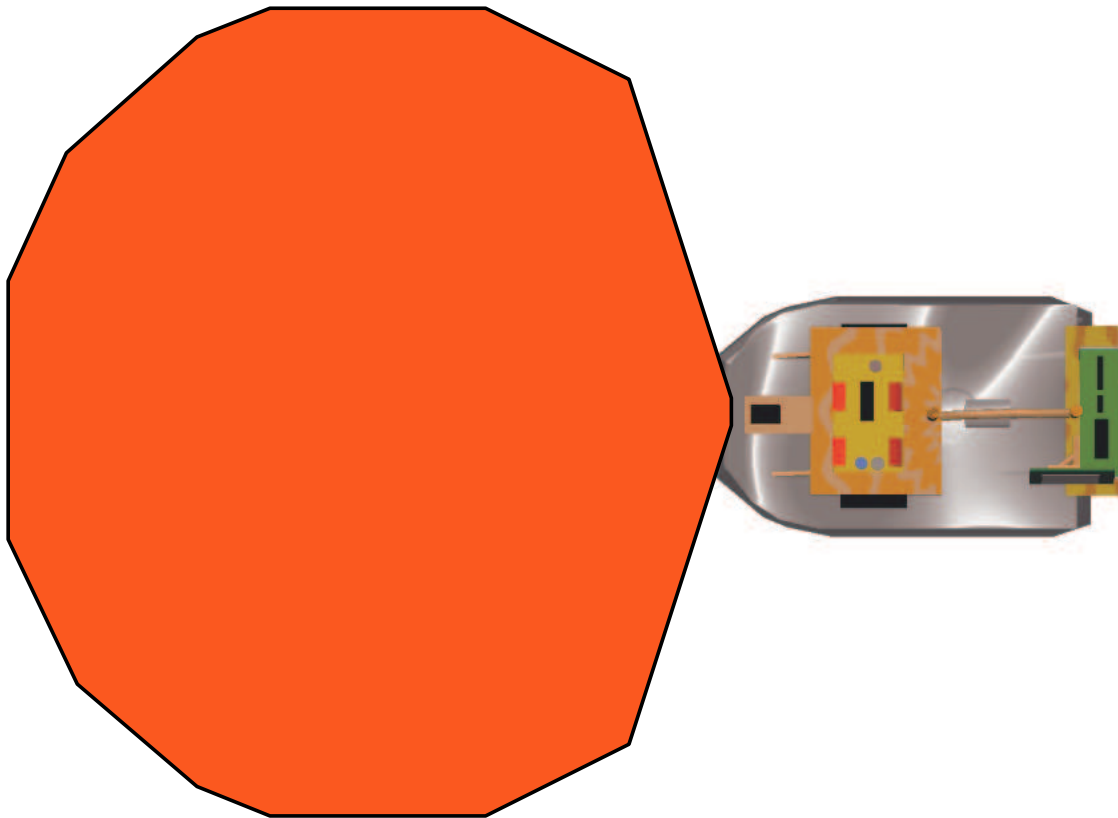
Future Improvements

The hovercraft is an excellent platform for future improvements. There are three major improvements to be made to the hovercraft:

1. Additional sensors.
2. The ability to learn on its own.
3. Improved training file by expert system training.

1. Additional sensors

Before any of the improvements listed below could be completed, the sensors would have to be updated. Currently, the hovercrafts sonar pattern is similar to this:

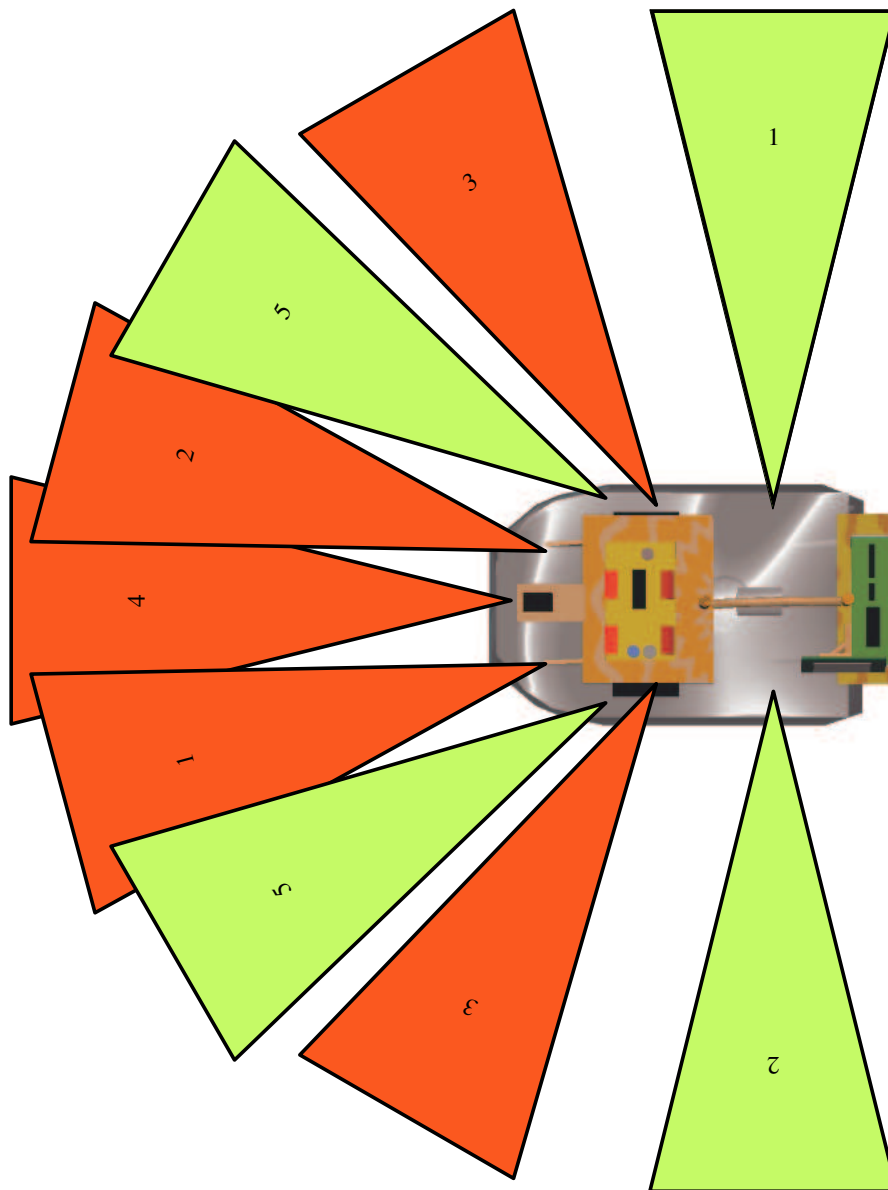


(Note: This is not a representation of actual sensing distances, only an approximation of the coverage area)

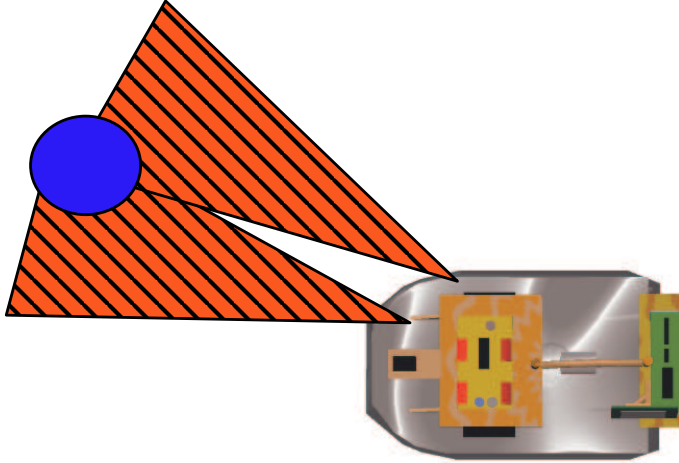
One transducer mounted on a servo covers this large area. This means that the servo has to be scanned, which is a relatively slow process. In addition the hovercraft would not be able to see obstacles to the side of it, which is a serious disadvantage if it had a wall beside it.

The new sensor pattern is shown below, and uses several different transducers. The yellow areas show transducers that are not necessary, although the hovercraft's performance would be further improved if they were included.

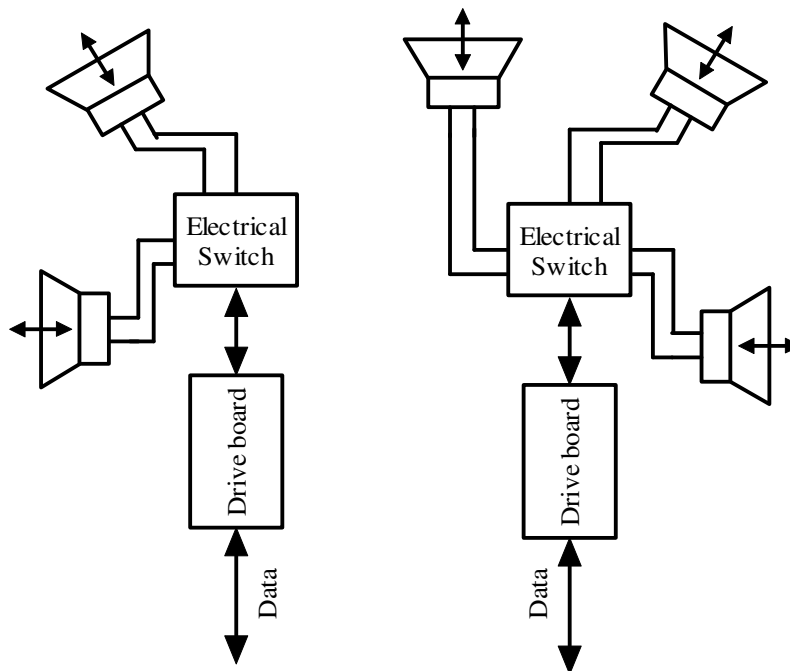
The firing order is also shown. This layout was chosen to allow the hovercraft to detect a wider variety of obstacles, and also detect approximately where they are and how big they are. The firing order is designed to eliminate the possibility of crosstalk. It is important that when two sensors fire at the same time, it would not be possible for the reflection of one transducer to make its way into another transducer.



In the diagram below, the obstacle sits between the two beams. Since both are activated, the hovercraft knows that the obstacle has to be within the activation “cone” of both transducers. Determining the exact size of the obstacle would be difficult though, as it could be a coffee cup or a beach ball. This isn’t of great importance, as the hovercraft can just mark that entire section “not passable”.



The new transducer arrangement would require anywhere from four to eight ultrasonic transducers, and two driver boards. One driver board would drive the left side of the hovercraft, and the other driver board would drive the right side, as well as the middle transducer.



This block diagram shows how the hardware would be implemented.

Enhancements to the software would greatly increase the flexibility of the hovercraft. For instance, the current software only computes the distance to an obstacle, which is effective as long as the obstacle is in front of the hovercraft. However, this cannot be used on the side sensors. The hovercraft could be driving beside a wall, and be only 40cm from the wall. However, if the hovercraft had a constant forward velocity, this would not be a problem. On the other hand, if the hovercraft was sliding towards the wall, calculating the speed of the hovercraft referenced to each sensor would be extremely useful and allow the hovercraft to take corrective action. For instance, the speed of the hovercraft measured from the left sensor could be 6 km/h, from the forward sensor it is 1km/h, and from the right sensor it is -7 km/h. In this case the hovercraft could discover that its velocity is not forward, but instead sideways! A simple control algorithm that only takes into account the distance to an obstacle in front of the hovercraft would fail. Instead the hovercraft must know where it will most likely hit an object. This data would be derived from the difference of the current sensor reading from the previous, and the distance to an obstacle. The following inputs for every sensor would be communicated to the control logic (in this case a neural network):

- 1) Difference of current sensor reading from last sensor reading
- 2) Current distance measurement

In this case, if the speed was high and distance low, the hovercraft would realize that a collision was immanent, and it needed to try to apply power in the opposite direction that the hovercraft is moving.

Additional sensors would also provide a more robust platform. For instance a tilt sensor could be added that would allow the hovercraft to detect when it is on a slant, and needs to correct for this slant by using less (or more) power than it normally would.

This could be implemented using an ADXL202 sensor. This is an accelerometer manufactured by Analog Devices, and can be set up to measure the tilt in two axis. The micro controller would simply measure the signal outputted by the sensor, perform some slight conditioning, and read the value of tilt.

Another useful sensor would be collision sensors. These would be simple switches mounted on the perimeter of the hovercraft so it would be able to detect a collision. These would be combined with an accelerometer as well. The tilt sensor is actually an accelerometer, and can perform double-duty. This would allow the hovercraft to detect the intensity of the crash.

2. The Ability to Learn on its Own

(Note: Please read the document on how an artificial neural network works first)

The primary benefit of a neural network is its ability to adapt and learn. The current neural network running on the hovercraft does not have the ability to modify its own weight file. A neural network that had the ability to modify its own weight file would be a future improvement.

Training is the process of teaching the neural network how to respond to new situations. For example, if the original neural network was trained to start turning when an obstacle is 600cm away, and it was found that this wasn't sufficient, the neural network would have to be retrained.

However, knowing when the hovercraft has made an error, and what correction will be needed is difficult to accomplish. The additional sensors are a requirement to be able to detect this though, as well as more memory.

The system would constantly log sensor data. As the object of the hovercraft is to avoid a collision with an obstacle, a collision would be when the obstacle failed to be avoided. When it detects a collision, it would save some of the logged data. For instance it would log 10 seconds of sensor data leading up to the crash. Once it has collected a few crashes and the data leading up to them, it could analyze the data. It would look for patterns in this data to detect situations that are the same. This would let it attempt to solve one problem at a time. This example data log shows various sensor data (next page).

There are distances to obstacles in CM, as well as the collision sensors and accelerometer data. The accelerometer data does not have a scale. Also, note that these are not actual logs, simply “possible scenarios” The power setting is a percent of maximum power. The rudder setting shows the direction the hovercraft should be going (C = center, L = left, R = right) as well as how much it will turn in each direction (1 = small turn 10 = max turn).

Left side distance to obstacle CM	Front distance to obstacle CM	Right side distance to obstacle CM	Left side collision	Front collision	Right side collision	Power Setting (0-100)	Rudder Setting
500	500	500	0.02 – n	0.04 – n	0.03 – n	80	C
500	432	500	0.05 – n	0.11 – n	0.05 – n	80	C
500	323	500	0.14 – n	0.12 – n	0.14 – n	60	R-1
500	150	500	0.42 – n	0.22 – n	0.42 – n	100	R-9
460	98	500	0.54 – n	0.34 – n	0.59 – n	100	R-9
300	123	500	0.23 – n	0.23 – n	0.16 – n	100	R-8
140	130	500	0.30 – n	0.30 – n	0.23 – n	80	R-7
89	154	500	0.24 – n	0.04 – n	0.33 – n	80	R-6
43	196	500	0.12 – n	0.02 – n	0.15 – n	80	R-6
0	203	500	4.64 - y	1.55 – n	4.48 – n	70	R-6

Left side distance to obstacle CM	Front distance to obstacle CM	Right side distance to obstacle CM	Left side collision	Front collision	Right side collision	Power Setting (0-100)	Rudder Setting
500	500	478	0.03 – n	0.14 – n	0.01 – n	80	C
500	493	456	0.04 – n	0.23 – n	0.03 – n	80	C
500	345	425	0.16 – n	0.11 – n	0.14 – n	70	C
500	189	367	0.36 – n	0.15 – n	0.42 – n	100	R-8
452	108	489	0.59 – n	0.21 – n	0.51 – n	100	R-9
256	165	346	0.21 – n	0.09 – n	0.20 – n	100	R-9
120	157	234	0.33 – n	0.52 – n	0.15 – n	95	R-8
65	190	456	0.27 – n	0.23 – n	0.28 – n	80	R-7
23	234	234	0.15 – n	0.02 – n	0.19 – n	70	R-5
0	283	500	5.24 - y	2.25 – n	5.45 – n	70	R-3

First, the learning algorithm would look up which side the collision occurred on. As there was no collision on the other side of the hovercraft, it is safe to assume that the sensor data does not matter for that side of the hovercraft. Then the records are searched to look for similar events. In this case there are two records, and both contain similar data. This means it is a reoccurring problem that must be dealt with.

A type of “trial and error” method would be used to test new strategies. First, the hovercraft would attempt to calculate the velocity of the hovercraft moments before the crash. It then calculates the velocity that the hovercraft is supposed to be going, and these are compared. If the hovercraft is supposed to be going forward but instead is going sideways at 5 km/h before the crash, this is a problem. This means the crash was caused by a lack of proper control. To counter this, the hovercraft would start testing “ideas” it has. These would be a combination of random, pre-programmed, and discovered ideas. For instance, first it would attempt to counter this sideways movement by applying more forward power and more rudder correction (turn harder). It will then assess how well this works. Even if it fails to avoid a collision, it can still check whether it lessened the impact. If it did, then it knows it is able to lessen the impact, but not avoid it completely. If the problem gets worse, it would try doing the opposite. If this fails to make any headway, then it might try other things such as low power and lots of rudder correction. If it is unable to avoid the crash entirely using the above method, it would continue onto the next stage (even if it made no progress using the above steps).

If completely avoiding the loss of control isn't possible, then the next step would be to predict when this loss of control will happen, and what undesirable effect it will have. This means it no longer randomly loses control of the hovercraft, but instead it will know when the hovercraft will not respond as it is being told. The software will know that it is not turning properly, as when it is supposed to be going forward it is going sideways. This means it takes longer to turn than it expects, and crashes. It will then attempt to turn earlier than before. This is because it realises that it hit the wall even though it was supposed to have turned. It then tries to turn earlier to see if the problem gets better or worse.

Making the proper training files is part of the solution; the other part is the actual training. It won't do much good if the hovercraft knows what it is doing wrong if it can't change its behaviour. The retraining is the other part of the solution.

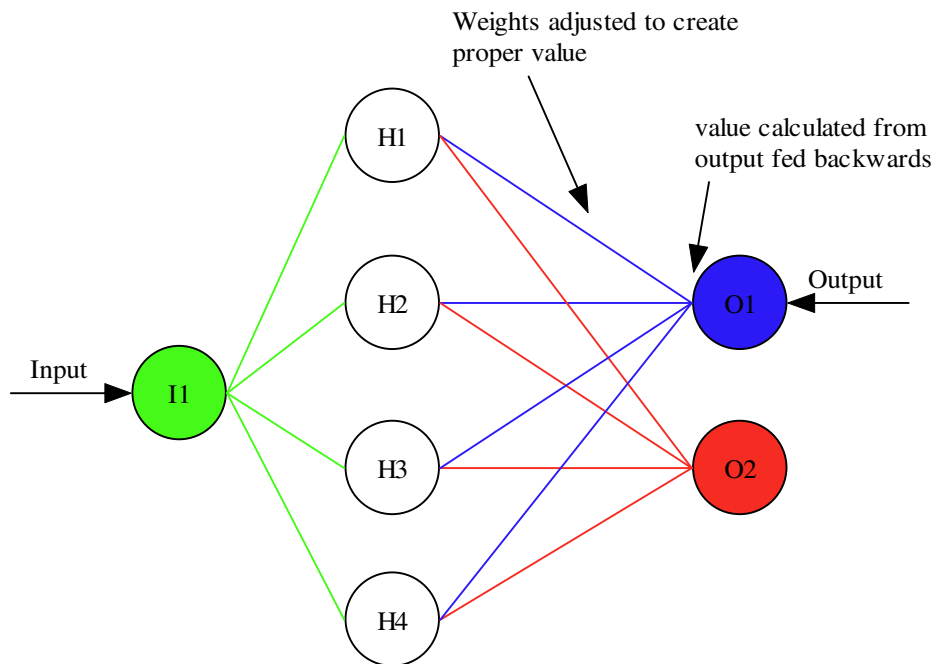
2.2 Retraining the ANN

The artificial neural network needs to be trained so it is able to respond to situations. Completely retraining a neural network would be extremely slow on a small microcontroller, as training a neural network is a slow and long process. However, “tweaking” the neural network would be possible. This would be making small changes to it dynamically, such as adjusting when the hovercraft would start to turn.

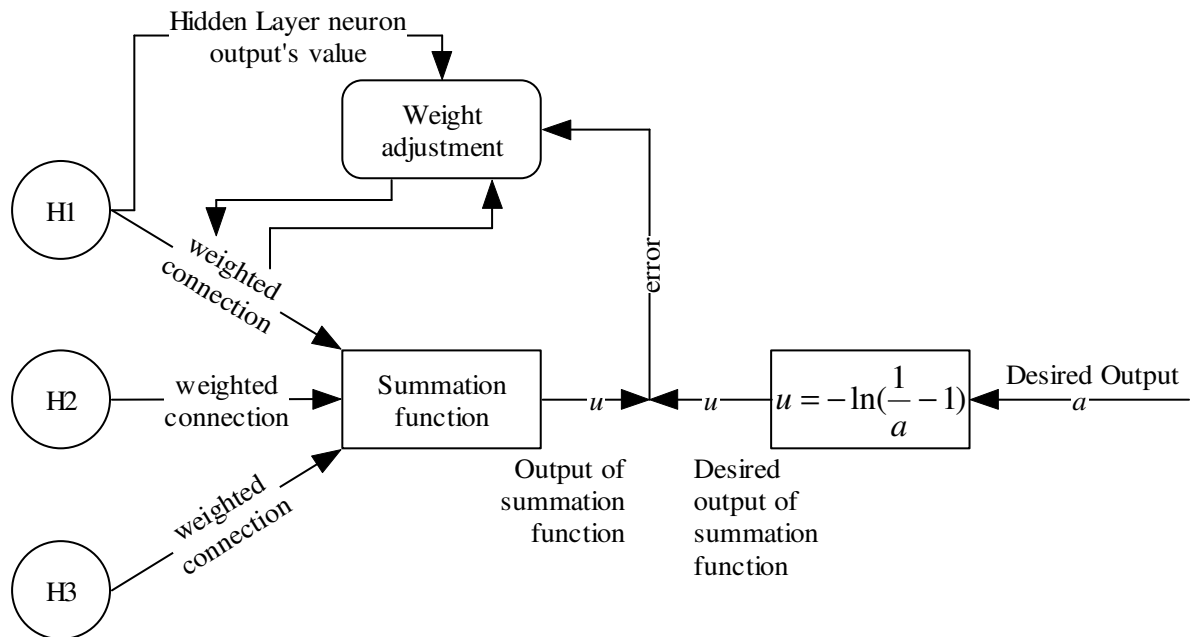
There are many ways to train a neural network. Back propogation is one of the most popular. The technique described here is not quite back propogation, instead more of a self-developed technique. It is designed to work with small neural networks, as large neural networks would not run on a small microcontroller. The lesson file is checked to see what the output should be with a given input. Then this input is run through the neural network. The outputs of the hidden layer are recorded. Then the desired output is run through the rearranged activation function where a is the desired output

$u = -\ln\left(\frac{1}{a} - 1\right)$ Using this formula, the output for the summation function can be found.

Once the output for the summation function is found, the weights can be “tweaked” to fine-tune the output. This would go through the list of weights and look for ones that can be changed.



Here is a more detailed view:

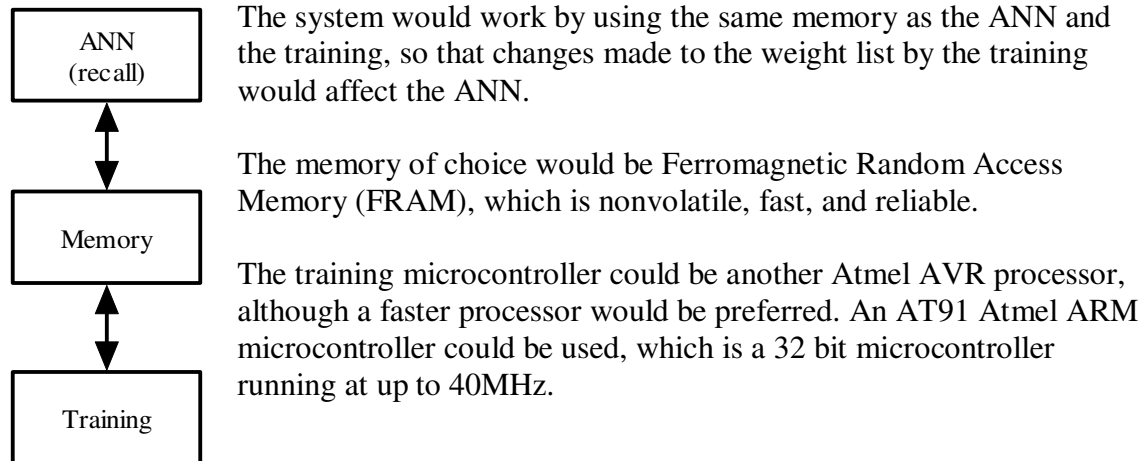


Note: only three of the nine connections (not counting error signal) are shown to the “weight adjustment”. Each hidden neuron’s output would be connected to the weight adjustment, as well as each weight.

The weight adjustment block is the key to the training. It takes the error, which is how much the output of the summation function must be increased (or decreased) and it calculates the required change to the weights based on the current weight, the output of the hidden layer neuron, and the required output of the summation function. This training algorithm is not designed to completely re-train a neural network, only make small adjustments. The hidden neurons that are most active would have their weighted values connecting them to the output neuron adjusted first.

The problem with the above training algorithm is convergence. It may be difficult for the neural network to be properly set up to allow all the inputs to generate the proper output. The training algorithm is fairly simple. However a more advanced method of selecting which weights to adjust and how much to adjust them should help with this problem.

Training is a very processor intensive task, and would likely need a dedicated microcontroller to achieve maximum performance.

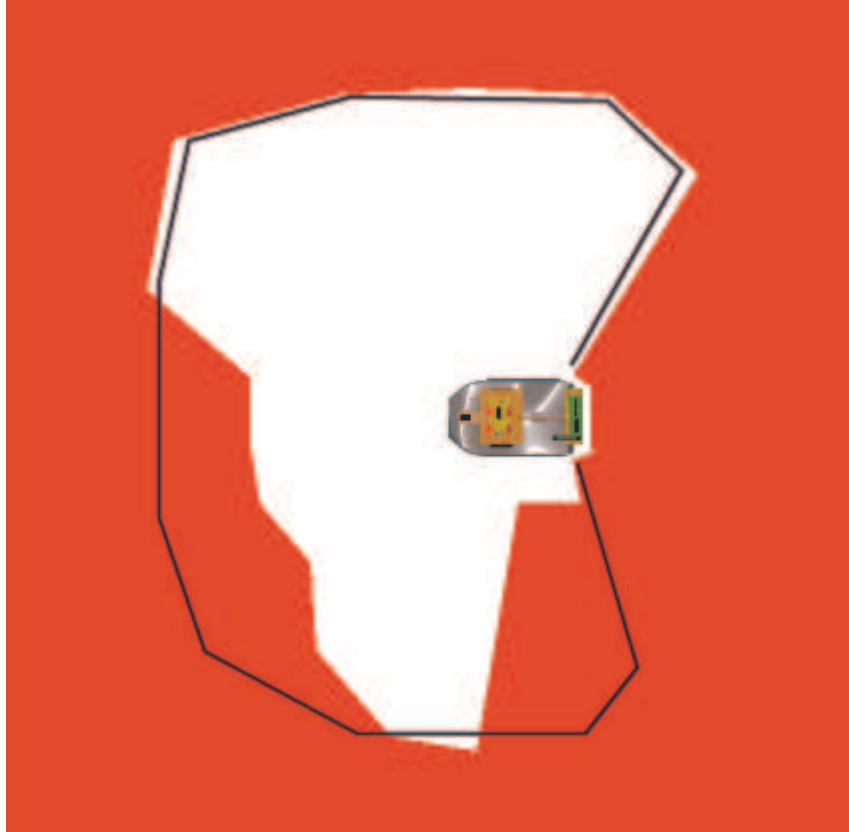


3. The Expert System

Even if the hovercraft can learn on its own, it still needs the starting lesson file. This lesson file is what it modifies when it learns. The current method used to create this file is just to use numbers that appear to be correct. For instance when an obstacle is 300cm away, turn a fair amount and apply full power. However not only is this prone to errors, there is the problem that it becomes more complex when more sensors are added. The expert system training would be the preferred method of training.

In the expert system, an expert drives the hovercraft around an obstacle. While this is happening, the sensor readings and the expert's response is recorded. From these the training file can be created.

The expert system would work best if the expert only sees the data the hovercraft sees. For example the sensor information is transmitted to a laptop. These sensor readings are converted into a graphical display for the expert. Then the expert guides the hovercraft around based on what he sees on the graphical display. The graphical display would simply have an image of the hovercraft, and the sensor data is graphically draw on.



The red areas are areas where sound cannot pass through, and there is either an obstacle, no sensor data, or the maximum distance has been reached. The black line indicates where the maximum distance that the sensors are reliable for. If an area inside the black line is red, there is an obstacle there. The area can be white outside the black line, although the data isn't reliable.