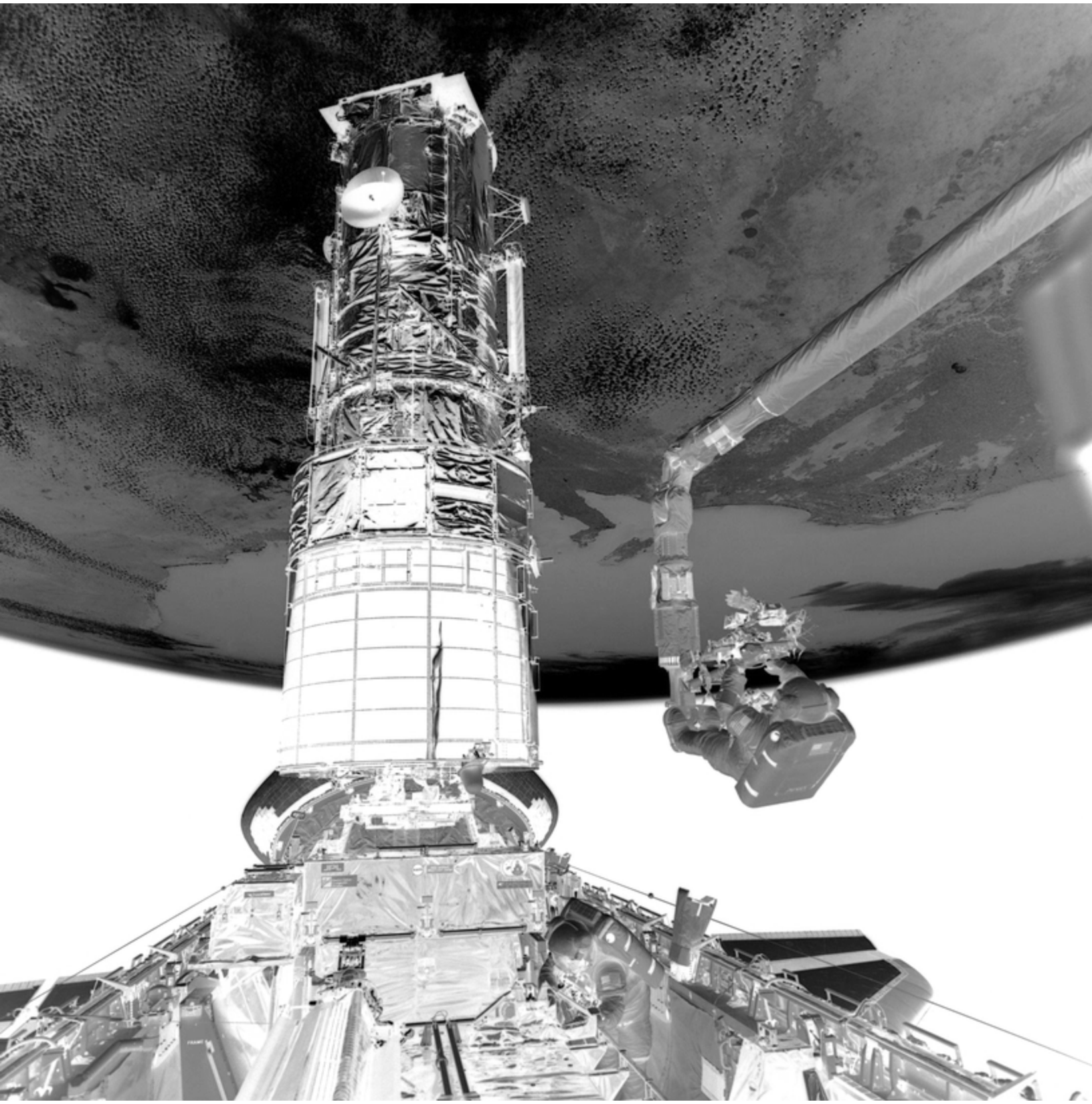


Manned or Robotted: An argument for the latter

Jovonni Pharr

Jovonni pharr

Manned or Robotted: An argument for the latter

This project is on the comparison between artificially controlled space missions versus manned space mission. As a result of many failed attempts at manned space missions, some with fatal consequences, the scientific discussion about artificial intelligence vs human intelligence has taken on new frontiers.

Alan Turing was a British pioneering computer scientist, mathematician, logician, cryptanalyst, philosopher, and mathematical biologist. He is most famous for being the scientist that cracked secret military communications during World War 2 from the famous Enigma Cipher Machine. This gave the western powers a significant advantage over their enemies. The cracked communications were morse-coded radio communications, and Dwight D Eisenhower dictated that this served as a decisive event for the allied victory. Alan Turing was also the person who created an examination framework for a system that is claimed as “Intelligent”; this exam is well known as the Turing Test. The Turing Test, has become the primary goal of computer scientists that are aiming to ever give birth to Artificial Intelligence. It is the current state of this journey, to create an Artificially Intelligent agent, that enables this paper to convey an argument, serving as a proponent of Artificially Controlled space missions -- opposed to Human Controlled Missions.

There have been many scientific, and computational breakthroughs that allow unmanned space missions to become more reliable; thus, allowing more trust in completely unmanned expeditions. Using these breakthroughs, there have been very few space missions that have taken advantage of the new capabilities. Although all space missions are not completely manned, as computer systems are abundant on board, the purpose of this paper

is to highlight the specific breakthroughs in computational science, that enable intelligent systems to perform far above results yielded by its human counterpart. This paper is founded upon three concepts that support the potential success of artificially controlled space expedition; these include Fuzzy Logic, Neural Networks, and Evolutionary Computing.

In the real world, everything is not so black and white. In the computational world, there is no exception. Throughout the journey for Artificial Intelligence, there has been a need for machine logic, to be expanded to further resemble the “gray-area” logic of the real world. To accomplish such a task, new paradigms had to be adopted; for example, a paradigm that possesses the flexibility to not only decide whether, or not a space shuttle is in danger, but also determine the degree of “danger” at any given moment.

The methodology known as Fuzzy Logic is a paradigm that emphasizes approximations, instead of fixed-value absolute numbers. This will highlight the capabilities to expand boolean logic, into “truth values” -- in order to reproduce human like decision making. This allows computational processes to make even more flexible, and real-world decisions. Fuzzy Logic is a concept in artificial intelligent systems, and the concept of Fuzzy Logic is built around the notion of having “fuzzy” truth values. In essence, a truth value is usually determined by whether something is true, or false. However, in Fuzzy Logic, a truth value, is no longer represented by boolean logic, but adopts a value that is between 0 and 1. So the truth value becomes

$$0 \leq truthvalue \leq 1$$

Using the Fuzzy Logic paradigm, a Fuzzy Logic Controller is initiated with a crisp, or “fuzzy” number. The fuzzy inference system is a popular computing framework based on the concepts of fuzzy set theory, fuzzy if-then rules, and fuzzy reasoning. It has found successful

application in a wide variety of fields, such as automatic control, data classification, decision, expert systems, time series prediction, robotics, and pattern recognition (Jang, Jyh-shing 73).

Let's say we want to build a system that can decide whether or not our spaceship is in danger of being hit by a particular object, $SpaceDebris$. Also, we would like to decide what our space ship should do next. For this Fuzzy Logic problem, let's assume the following inputs x :

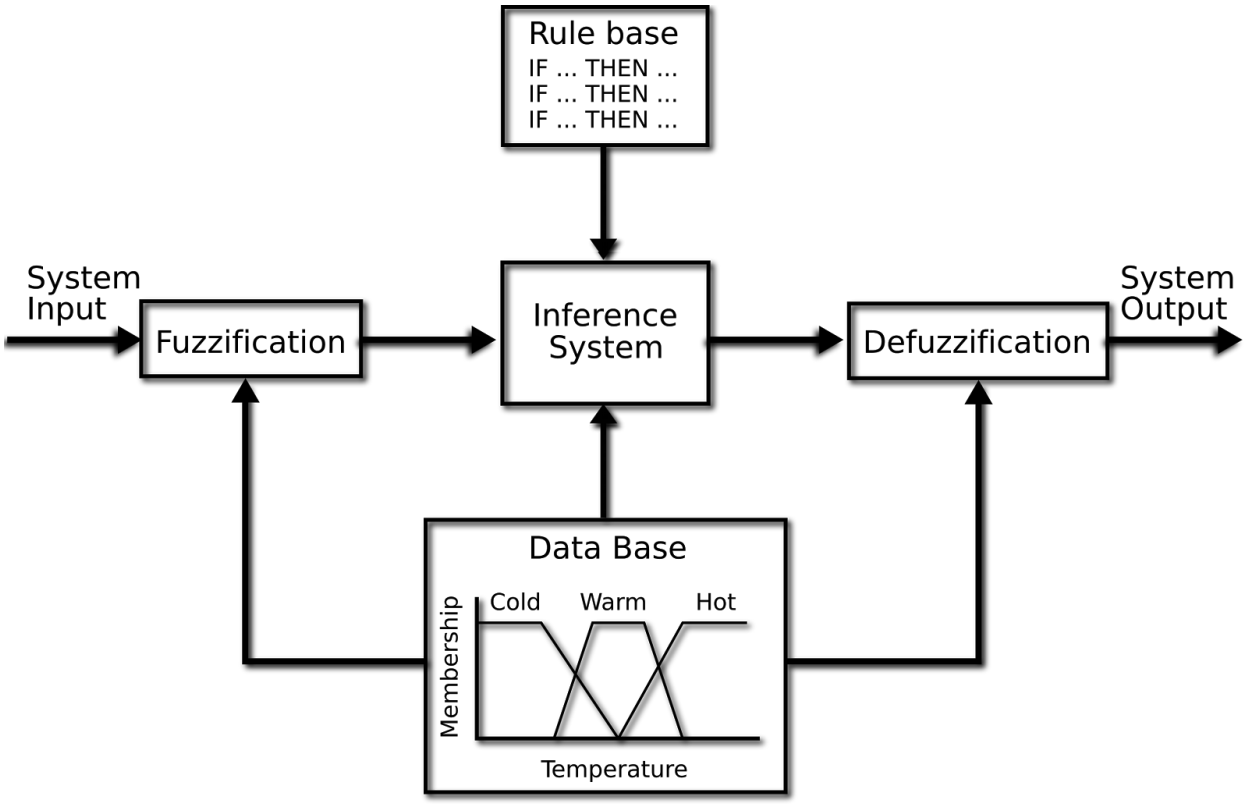
$$x = (SpaceDebris_{Proximity}, SpaceDebris_{ApparentSize}, SpaceDebris_{Density})$$

A valid output can then be $Danger_{Level}$, and here would be the rules for such problem:

$SpaceDebris_{Proximity}$ **rules**	$SpaceDebris_{ApparentSize}$ **rules**	$SpaceDebris_{Density}$ **rules**
If $SpaceDebris_{Proximity}$ is $close$, then $Danger_{Level}$ is $high$	If $SpaceDebris_{ApparentSize}$ is big , then $Danger_{Level}$ is $high$	If $SpaceDebris_{Density}$ is $high$, then $Danger_{Level}$ is $high$
If $SpaceDebris_{Proximity}$ is $medium$, then $Danger_{Level}$ is $medium$	If $SpaceDebris_{ApparentSize}$ is $medium$, then $Danger_{Level}$ is $medium$	If $SpaceDebris_{Density}$ is $medium$, then $Danger_{Level}$ is $medium$
If $SpaceDebris_{Proximity}$ is far , then $Danger_{Level}$ is low	If $SpaceDebris_{ApparentSize}$ is $small$, then $Danger_{Level}$ is low	If $SpaceDebris_{Density}$ is low , then $Danger_{Level}$ is low

If a Fuzzy Logic Controller was built to deal with this decision making framework, it would need real world inputs to represent each rule, in each of it's states. Doing so allows the controller to "know" examples of each rule & state. In theory, everytime the Fuzzy Logic Controller encounters a new input that is has not seen before, it converts the input (fuzzy), to an output (crisp), using the $inputs \rightarrow outputs$ of all of the training cases that were used to build

the system. This is how the Fuzzy Logic Controllers can figure out a real world output, from an ambiguous human-understandable input.



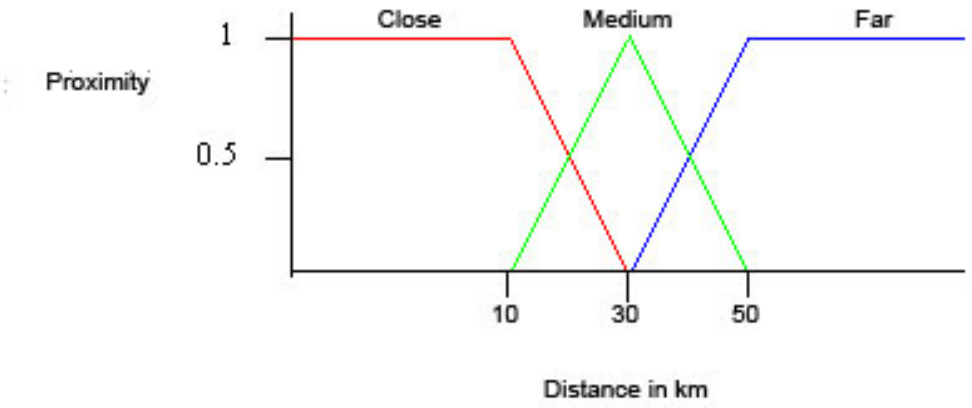
A real world example of what “medium proximity” means could be $30km$ away from the ship, “small apparent size” could be represented as an object with $200cm$ in length, and “low density” could be represented as an object with density of 25g. Fuzzy logic allows a system to take advantage of an ambiguous term like, “small apparent size”. The values that represent a correct example of each rule, are initially up to the discretion of the engineers building the system, and what training cases they feed into the system. To actually run an instance of this Fuzzy Logic Controller, let’s assume the system received an input of:

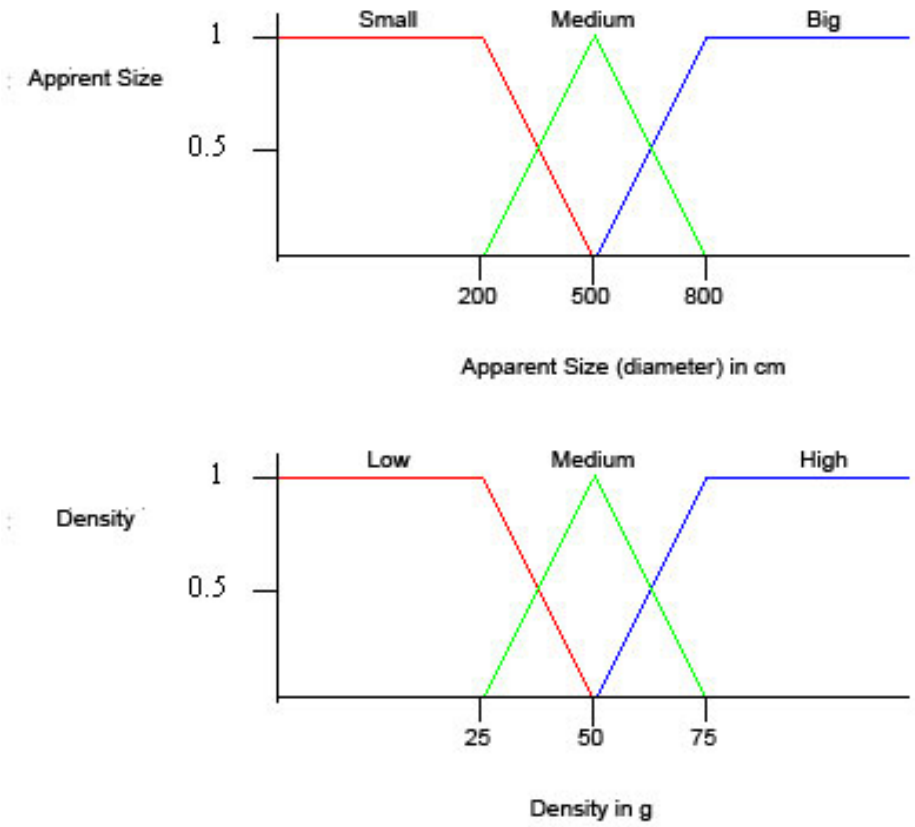
$$x = (18_{Proximity}, 214_{ApparentSize}, 25_{Density})$$

Each input is processed through the Fuzzy Logic Controller to produce defuzzified outputs. This method allows the system to calculate what the most likely classification of each input type. So the input of $18_{Proximity}$ would result in:

$$y = (.7_{close}, 3_{medium}, 0_{far})$$

Here we have a stronger output of $close$, with .7. Remember, If $SpaceDebris_{Proximity}$ is $close$, then $Danger_{Level}$ is $high$. If the system is to make a decision based off of this input alone, then the system would determine that the "Ship is at a high danger level", for the input value of $proximity$. The system will take into account the other input values, and their inferenced outputs; however, we only examined one input value in this example. Ideally, the system will take the greatest value per each category, and make its decision based on the highest $Danger_{Level}$ value given by the Fuzzy Logic Controller. Here is what the membership function graph, per each $SpaceDebris_x$, would look like in the real system, for this example:



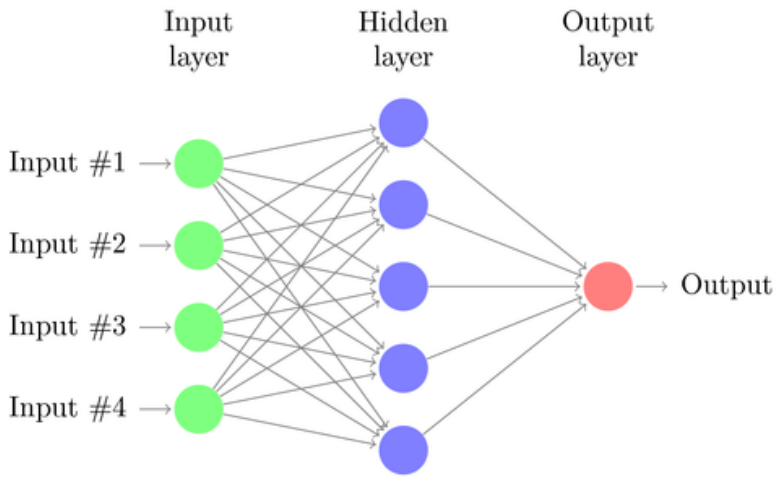


Fuzzy Logic Controllers work well by themselves, as there have been various large scale projects made using this approach. Moreover, Fuzzy Logic works well when used in tandem with other methodologies in Artificial Agent Computing.

Another example of a concept that supports Artificially Controlled Space Missions is called a Neural Network. This allows a program to make decisions, based on a method that is closely modeled after the human brain, and how the human brain makes decisions through neural processes. There several components to a Neural Network, but we shall examine from high level -- because of the purpose of this paper.

The benefits of a Neural Network, because it is modeled behind the processes of the human brain, allows for the system to take inputs, and map them to predict what the outcome should be, versus what it is. This involves many different steps, and that is a full paper in and of itself. The difference between a Fuzzy Logic Controller, and Neural Network is in the

method of which is used to calculate an output. Similar to a Fuzzy Logic Controller, an input is given, but sometimes there are no arbitrary rules written by the engineer. Once a value is passed into an input layer node, it then outputs a total output value for its “decision unit”; it can also be the case where those input layer neurons send their output to be used as input for a single hidden layer of neurons, or multiple -- thus making it a “Deep” Neural Network. Each Neuron can run its own internal calculations to adjust the input as it passes through the network. The following is an example of a Neural Network:



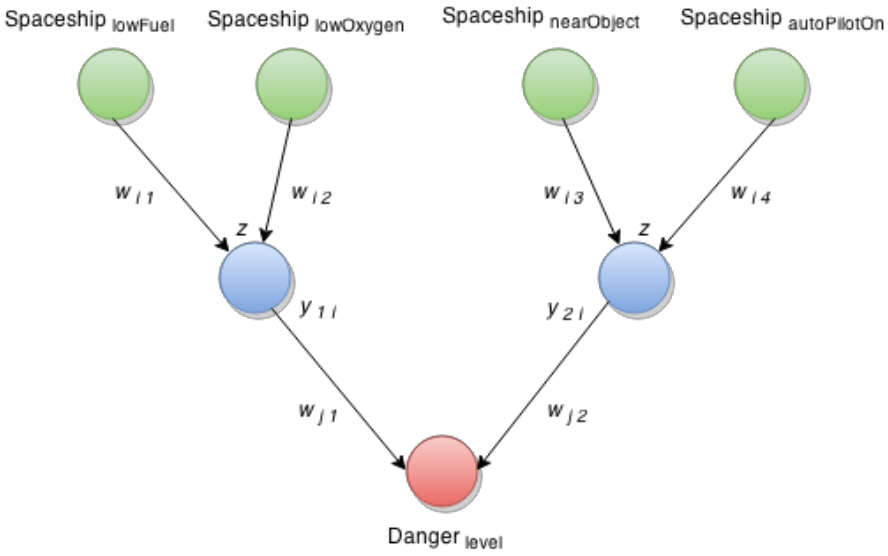
As an input is given, it is adjusted by a weight before reaching the next layer. Each branch between nodes, can have an associated weight that adjusts the output/input traveling through. In the output layer of the network, the system takes the total output, and utilizes this output to then decide a final output for the entire network.

Let’s look at an example of a Neural Network being used to decide the $Danger_{Level}$ of our spacecraft, and let’s use the following inputs:

$$Spaceship_{lowFuel},\ Spaceship_{lowOxygen},\ Spaceship_{nearObject},\ Spaceship_{autoPilotOn}$$

For this example, we can represent each input value as a boolean value -- as a true or false.

$$1 = true,\ 0 = false$$



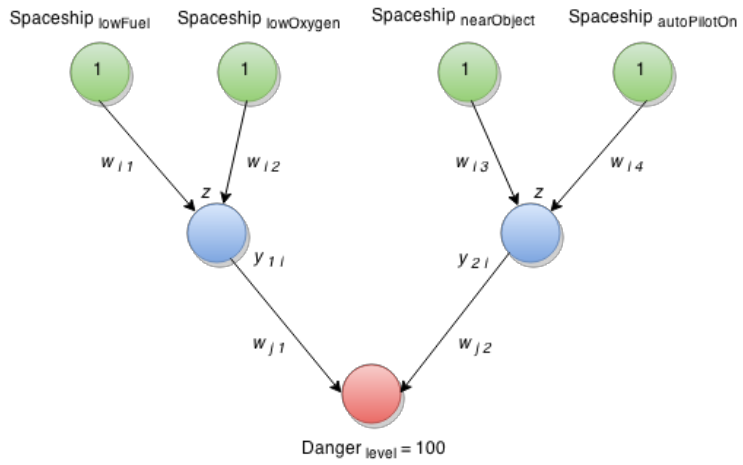
After any Neural Network is architected, the network must then be trained with sample data, and examples of correctly classified training cases. These training cases allow the network to "learn" what features assist in generating a correct classification for any new test case. This just means that the network learns from a set of training data, and then makes an attempt at predicting what output will be yielded with a new set of input data that is has not encountered -- by comparing the new input, with the training cases it has been given. Lets look at the following training cases. There are many ways to evaluate a case, here are two:

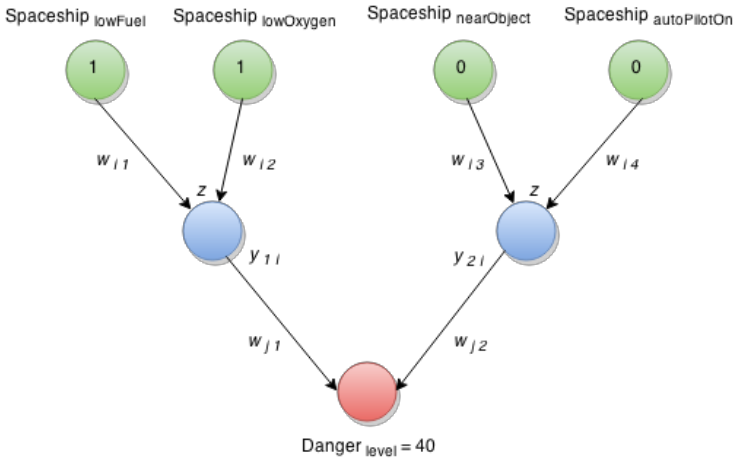
$$z = \sum_i x_i w_i \qquad z = b + \sum_i x_i w_i$$

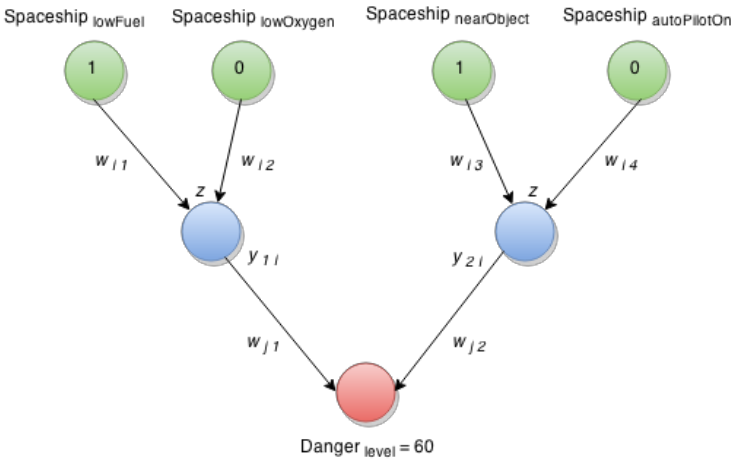
$$b = a\ bias\ term,\ if\ applicable$$

z = *total output* x = *input*

w = *weight of y of decision unit* y = *output of decision unit*


Spaceship lowFuel
Spaceship lowOxygen
Spaceship nearObject
Spaceship autoPilotOn
1
1
1
1
w i 1
w i 2
w i 3
w i 4
z
z
y 1 i
y 2 i
w j 1
w j 2
Danger level = 100


Spaceship lowFuel
Spaceship lowOxygen
Spaceship nearObject
Spaceship autoPilotOn
1
1
0
0
w i 1
w i 2
w i 3
w i 4
z
z
y 1 i
y 2 i
w j 1
w j 2
Danger level = 40


Spaceship lowFuel
Spaceship lowOxygen
Spaceship nearObject
Spaceship autoPilotOn
1
0
1
0
w i 1
w i 2
w i 3
w i 4
z
z
y 1 i
y 2 i
w j 1
w j 2
Danger level = 60

Once this model has been trained, it can then make a guess when a new input vector is given, in the service of generating a correctly valued output. In more general terms, the network tries to minimize the error that it makes in predicting different outputs, for each input vector. Systematically, the network learns by increasing weights for correctly valued cases, and decreasing weights for incorrectly valued cases.

Using advanced techniques that are built on top of a Neural Network, a system can begin to learn what features it should be paying attention to -- opposed to the engineer hardcoding the rule set into the network directly. In order to do this, the network can utilize a methodology known as back-propagation. The back-propagation algorithm is a constant step size, gradient descent procedure to iteratively adjust the weights of the neurons in a multi-layer feedforward neural network (Antsaklis 4). Using the power of derivatives, and rate measurements, back-propagation allows for a network to take the Error Derivative, with respect to the total output of the network. If one calculates this error derivative on the output layer/node, using the chain rule, one can calculate the error derivative, with respect to the total input of the layer preceding the output layer. Here is how you find the error E, and the derivative $\frac{\delta E}{\delta y_j}$ of the output layer/node:

$$E = \frac{1}{2} \sum_{j \in output} \left(t_j - y_j \right)^2$$

$$\frac{\delta E}{\delta y_j} = -\left(t_j - y_j \right)$$

This approach enables the network to learn important features, beyond those of which it was initially programmed. It learns by observing how the error E changes, with respect to the output y_j. There are several other advanced methodologies using Neural Networks, and they

add to capabilities that can be demonstrated through a Deep Neural Network. The back-propagation algorithm is just an example of many expansions of this approach, that allows for very deep neural network problem solving. Space Mission Systems that utilize Deep Neural Networks for task performance have an advantage over Space Mission Systems that don't. Like I stated before, this is a complete paper in and of itself.

The final approach that lends support to an Artificially Controlled space mission is called Evolutionary Computation, and this is defined as computational architecture that is closely modeled after the understood nature of evolution. Evolutionary Algorithms are used to solve computationally heavy problems by allowing the program to evaluate solutions based on how various candidate solutions behave within specific landscapes. This allows these algorithms to evaluate large amounts of different solutions to decide which would be the best solutions to the given problem.

Evolutionary Computing allows computer scientists to take advantage of the creative, random, and iterative properties that evolution & nature have to offer. This enables programs to advance the behaviors of the organisms within their local environments, and actually produce offspring that have combinations of attributes from several different local organisms. The process begins with an initial selection of a population of models/solutions, and this can be arbitrary.

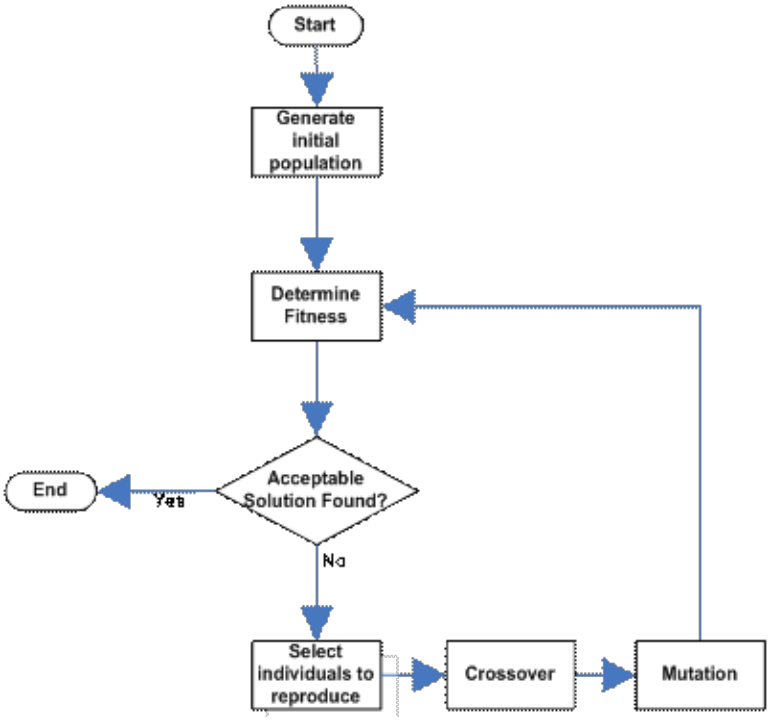
In evolutionary computational techniques, a population of these models is created and input parameters are varied. The results are evaluated using fitness functions and a percentage of the highest fitness individuals from one generation is promoted to the next generation, while new models are created through variation -- by mutation or cross-over (Terrile 2). In evolutionary computation, a population is defined as a set of possible solutions to the given problem. The fitness function yields a $fitness_{score}$ per each solution within the

population. This $fitness_{score}$ function can take on many different forms, as some methods work better with specific problems, and not as good with others. To judge $fitness_{score}$, we can use a threshold. This is what the threshold function θ would look like:

$$if\ fitness_{score} \geq \theta \qquad then\ population_{score} = 2\,fitness_{score}$$

$$if\ fitness_{score} < \theta \qquad then\ population_{score} = fitness_{score}$$

After all of the solutions within the population have been scored, the evolutionary system will then analyze for whether or not an acceptable solution has been reached within the population. If an acceptable solution has not been met, the system will then use the scores to decide which solutions should be reproduced to generate the next population of solutions. Consequently, if an acceptable solution has been met, then the generation stops evolving, and the solutions stay active within the environment. Here is an illustration of this process.



Let's observe a twisted scenario where the $spaceShuttle_{cabin}$ has been breached by a $gas_{mysterious}$. Let's assume that there is no device on board that can detect all types of gases, and their respective elemental composition. If one were to design an evolutionary system that is purposed with guessing what gas is inside the cabin, the system could look like this:

Initial Population	Fitness Score / test	Acceptable Solution
$gas_{hydrogen}$ $gas_{nitrogen}$ gas_{helium}	*is determined by how different the given solution's gas composition, gas_x, is to the gas composition inside the cabin, $gas_{mysterious}$*	*is achieved if a given solution's gas composition, gas_x, is indifferent from the gas composition inside the cabin, $gas_{mysterious}$*

Lets assume that the gas that has leaked inside of the cabin, is N_2O; if we begin with this $population_{initial}$, none of the proposed solutions from this generation will reach $solution_{acceptable}$ that is similar to $gas_{mysterious}$ in the cabin. When that happens, the current generation undergoes a crossover, and a mutation that causes the next generation of solutions to become crossbreeds of the generation that precedes. This allows the system to try many different combinations of solutions. Soon enough after a few generations, this Evolutionary System will generate $population_{new}$ that contains N_2O; thus, matching $gas_{mysterious}$ in the cabin -- thus, reaching $solution_{acceptable}$. This will cause the evolutionary system to cease reproducing more generations -- at least until a new gas composition leaks into the cabin, and system begins again.

Limitations of Neural Networks, Fuzzy Logic Controllers, and Evolutionary Computational Systems exist in a myriad of facets. Ultimately, these systems are, by nature, bounded by the rationality behind their design. Just as in scientific history, models existed, that were hailed as the best approach for that time. Like with many scientific methodologies,

there may exist a better approach that is not yet in use. However, with just these three methodologies in the race to create an Artificially Intelligent Agent, we can now design an artificially controlled space mission, that can perform a task better than its human counterpart. In fact, there has been a framework made by engineers at NASA that encompasses all three of these methodologies into a single framework specific for space exploration by artificial systems; ironically it is called PERSON. This framework is just one of many that encompass the benefits received when using Fuzzy Logic Controllers, Deep Neural Networks, and Evolutionary Computational Systems together.

Sources:

Jang, Jyh-shing. "4." *Neuro-Fuzzy and Soft Computing*. Prentice Hall, 1997. Print.

Antsaklis, P.j. "Neural Networks for Control Systems." *IEEE Transactions on Neural Networks* 1.2 (1990): 242-44. Web.

Scharnow, Jens, Karsten Tinnefeld, and Ingo Wegener. "The Analysis of Evolutionary Algorithms on Sorting and Shortest Paths Problems."*Journal of Mathematical Modelling and Algorithms* 3.4 (2004): 349-66. Web.

Johnston, M.D, and Adorf, H.M. "Scheduling with Neural Networks The Case of Hubble Space Telescope." *Computers and Operations Research, Special Issue on Neural Networks* (n.d.): n. pag. Print.

Terrile, Richard J., and Christopher Adami. "Evolutionary Computation Technologies for Space Systems." (n.d.): n. pag. Print.