International Conference on Innovative Applications of Computational Intelligence on Power, Energy and Controls with their Impact on Humanity (CIPECH14) 28 & 29 November 2014

Self Learning Robot Using AtMega32

H. Srikanth Kamath
Electronics and Communication Department
Manipal Institute of Technology
Manipal, Karnataka, India
srikanth.kamath@manipal.edu

Sneha Das
Electronics and Communication Department
Manipal Institute of Technology
Manipal, Karnataka, India
dsneha91@gmail.com

***Abstract*— The field of neurorobotics is still in its infancy however, its intersecting motivations are not. On one hand, theories of neuroscience that require immersion in the real-world can be embedded in mobile agents creating complex patterns of activity believed to be a requirement for understanding higher-order neural function. On the other, the cognitive capabilities of humans remain unparalleled by artificial agents. Emulating biology is one strategy for creating more capable artificial intelligence. Despite these strong motivations for creating neurorobotic entities technological hurdles still remain at all levels. This paper explains the design of a neural network that would model some real world phenomena. A moth's behaviour was modelled via the movements of a robot, through the Hebbian learning process. A SOINN (Self Organized Incremental Neural Network) that equips these robots with basic intelligence was built.**

Keywords- Neurorobotics; Artificial Intelligence; Hebbian Learning; SOINN

I. INTRODUCTION

Neurorobotics, a combined study of neuroscience, robotics, and artificial intelligence is the science and technology of embodied autonomous neural systems. Neural systems include brain-inspired algorithms, computational models of biological neural networks and actual biological systems. Such neural systems can be embodied in machines with mechanic or any other forms of physical actuation.

Building a neural network would be the software basis of the self-learning robot. The robot built would use a neural network to learn to react to certain stimuli. Heat or ultrasound was initially considered as an input to the system, but after doing further research it proved to be non-feasible. Finally, a light-guided robot was used as a real world interface.

A moth is always attracted to light, but when it gets too close to light, it will quickly fly away due to the intense heat. The attraction to light shows excitation while the aversion at close proximity shows an inhibitory effect. This type of system would have multiple layers of Hebbian learning. Secondly, the locomotive property of the robot is used to acquire data pertaining to the surrounding weather conditions.

An elementary eight neuron network uses Hebbian Learning to train a robot to respond intelligently to input light stimuli. This in effect would fulfill the self-learning ability of the robot.

One of the most common examples of conditional learning such as Hebbian learning is seen in Pavlov's experiment with his dog. In this experiment, when food was offered to the dog, it caused the dog to salivate. At first, the sound of a doorbell elicited no such response. However, Pavlov decided to sound the bell when he offered food to the dog. After a few repetitions of this experiment, the dog began to salivate at the sound of the bell even when no food was present. Here food was the unconditioned stimulus, and the doorbell was the conditioned stimulus.

The objective of this paper is to propose a basic design example of artificial intelligence which gives a review of the various applications of neurorobotics today.

A. Abbreviations and Acronyms

SOINN: Self Organized Incremental Neural Network

II. BACKGROUND THEORY

A. Biological Basis

A neuron consists of a cell body with dendrites and an axon, which terminates onto a muscle fiber or the synapse of another neuron. It receives signals in the form of charges (sodium, potassium, calcium ions) from other neurons that have their axon terminals sharing synapses with its dendrites. These charges are integrated spatially (across number of neurons that synapse onto it) and temporally (charges received over time) and change the membrane potential of the neuron.

The membrane potential usually increases with the influx of sodium ions and the efflux of potassium and calcium ions. An increase in membrane potential is also referred to as a depolarization.

©2014 CIPECH14.

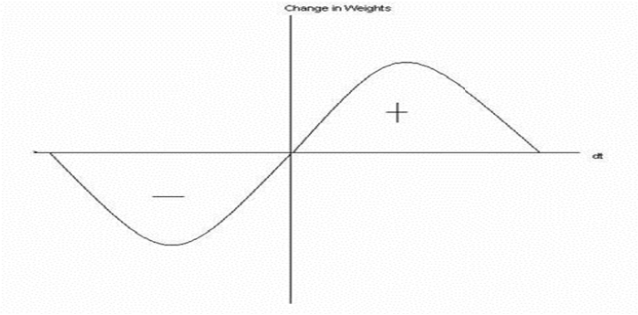

Fig.1 Biological Basis of the changing weights

Once the membrane potential increases beyond a certain threshold potential, specific for the particular neuron, an action potential is fired.

Action potentials are characterized by a strong depolarization, followed by a hyperpolarization (decrease in membrane potential). The action potential is then transmitted down the axon to the axon terminals where it is passed onto the next neuron through a synapse.

Action potentials are all or none events, and their method of generation can be characterized by an 'integrate and fire approach'. That is, a cell integrates charge signals it receives from other neurons and only after its threshold is reached it fires an action potential, thereby transmitting a signal to its postsynaptic neuron. After firing this action potential, the firing neuron goes through a refractory period. During this time, it cannot fire another action potential even if the signals it receives push its membrane potential beyond its threshold value. At the end of the refractory period the membrane potential is at its resting state. All of the charge entering a neuron from other cells does not contribute to the rise in membrane potential. Some of the charge constantly leaks out of the neuron. This is called leakage current. This leakage factor is taken into consideration while coding the neural network.

B. Hebbian Learning

The neurons connected in a network can be used to display Hebbian Learning. This is used as the learning approach as it is a supervised form of learning wherein the desired output is used as a supervisory signal.

Donald O. Hebb a Canadian Neuropsychologist proposed the following postulate:
When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

The principles underlying his postulate came to be known as Hebbian Learning. Therefore, if two neurons in a network (a post-synaptic and pre-synaptic) neuron fire repeatedly in the correct order, the connection between them is strengthened. The order of the neurons firing is also of significance as it can lead to a decrease in the weights between the neurons as well. If a presynaptic neuron fires shortly before the post-synaptic neuron their connection is strengthened, however if the post-synaptic neuron fires shortly before a presynaptic neuron their connection is weakened. If the time interval between the firings of these two neurons is very large, they cannot be correlated. Fig.1 shows how the timing of action potential affects the weights between two neurons.

C. Background Math

Integrate and Fire Equations:
Temporal & Spatial Integration of voltage:

$$\tau dV / dt = V_{rest} + V_{exc} + V_{inh} \quad (1)$$

Where τ is our decay constant. V_{rest} is the resting membrane potential, which we set to 0; V_{exc} is the voltage inputs due to excitatory connection and V_{inh} is the voltage inputs due to inhibitory connections.

The weights were updated according to the following equation:

$$W_{ij} = W_{ij} + T_{ij} \quad (2)$$

Where i corresponds to the presynaptic neuron, j corresponds to the postsynaptic neuron, W_{ij} is weight of the connection, T_{ij} is the weight of the connection * Learning / Unlearning rate.

In the spatial integration of the charges in and around a neuron, some amount of charge is lost in the form of leakage. This has been set to 0.999 in this case.

III. METHODOLOGY

The operation of the robot that was built is very similar to the Pavlov's experiment discussed earlier.The light shining in front of the robot and behind the robot elicits no response but pressing the push button causes the robot to move forward or backward.

Pressing the button while shining the light on the robot, causes the neural network programmed into the robot to associate the light input with the push button input. Hence the robot moves forward or backward depending on whether the light is shined behind or in front of it in the absence of the push button input.

There are other neurons in this network that play an inhibitory role and prevent the robot from going too close to the light. They too in effect display learning. Initially, the robot goes very close to a light source before it decides to move in the opposite direction. As time passes by the robot gets more

responsive to the light and does not get too close to either light source.

The design process was done in numerous stages.

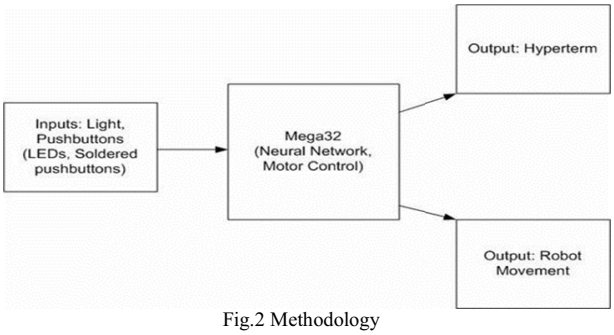

Fig.2 Methodology

The first stage was to simply get the neural network working. A simple 3-neuron network was made initially and was made to work. A fourth inhibitory neuron was used for inhibitory effects. The neurons were operated by simulation on AVR Studio, the results for which are discussed in the result analysis. Four more neurons were added to give bi-directionality to the neural network.

The changing weights of the neural network were shown through HyperTerminal via serial communication. The neural network has 4 inputs (forward and backward light sensors and the pushbuttons) and two outputs (forward and backward locomotion). The following diagram shows the network, where solid lines indicate excitatory connections and dashed lines indicating inhibitory connections.

An explanation for each of the eight neurons is as follows:

Forward Pushbutton: The input to this neuron is the pushbutton. It has an excitatory connection to the forward motor neuron with a very large weight since it represents the 'unconditioned stimulus' to the network.

Forward Light: This takes the voltage from the front light sensor as an input. This has an excitatory connection to the forward motor neuron and an inhibitory connection to the backward motor neuron. This is because we decided our neurons couldn't subtract the input directly, so instead we coded it so that the front sensor will excite the front motor neuron and inhibit the back motor neuron. This was simple to implement.

Forward Shock Light: This takes the voltage from the front light sensor as an input similar to the forward light. The difference between this and the forward light neuron is that this has a much higher threshold. Thus, if the input voltage from the front sensor is low, it means that the robot is pretty

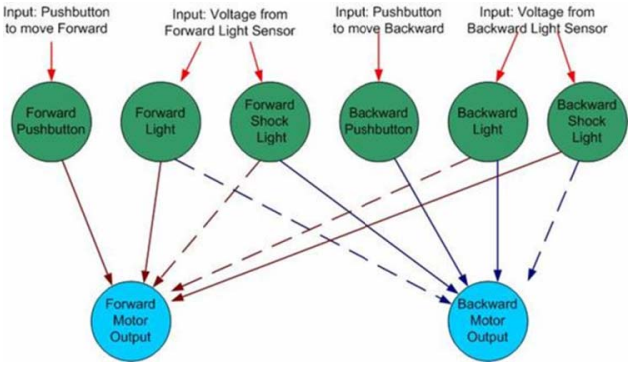

Fig.3 Relationship of the eight neurons in the robot

far from the front light source and is in no danger of being burned. But, if the robot is very close to a front neuron, then this neuron will spike very frequently. This is to warn the robot that it is too close and should not move quickly in the front direction. Thus if this neuron fires, it actually inhibits the forward motor output neuron, and in essence, competes with the forward light neuron. Furthermore, if this neuron fires, it is also telling the robot that it should move in the opposite direction to be safer, so it has an excitatory connection to the backward motor output. This adds to the intelligence of the robot and in effect its similarity to a moth's behavior.

The backward pushbutton neuron, backward light neuron, and the backward shock light neurons all behave very similarly to the descriptions above but in the opposite directions. Finally, the forward motor output and the backward motor output neurons basically fire depending on which input neurons are fired and what the strengths of those connections were.

A. Software Design

The programming of the neural network is being done in a series of stages. Each neuron on the network had the following variables (or array elements) associated with it:

outputV: element of an integer array variable indicating if a neuron had spiked.

weights: row of weight matrix (matrix of float variables) specific to the particular neuron.

V: float variable indicating the membrane potential

Spiketime: element of a char array denoting time elapsed since neuron had spiked.

Vthresh_(motor/shock/light/push): integer variable denoting the threshold of a specific kind of neuron.

Leakage: constant factor less than 1.

Lrate/ULrate/ULrate_inhib: float variables indicating how fast a particular neuron in the network learns or unlearns.

The entire network was programmed in four stages described below.

1) Stage 1:

Structure of Network: This network consists of three neurons, a pushbutton neuron (Neuron 0), a light neuron (Neuron 1) and a motor neuron (Neuron 2). Each neuron has been coded separately because their membrane potentials are updated differently and thus couldn't be effectively generalized in a loop. In the case of the pushbutton neuron, when the button was pressed, it adds a factor of its threshold to its existing membrane potential. This factor was kept as '0.8' so that the pushbutton wouldn't fire every time it was pressed. The pushbutton had to be pressed twice for it to fire. The light input neuron added the input from pin A.0 of the ADC on the microcontroller chip. The motor neuron updated its membrane potential by adding the weights of its presynaptic neurons that had fired in the present cycle to its current membrane potential. After the membrane potentials were updated in a specific neuron of the network, the code checked to see if the membrane potential was greater than the threshold voltage for that neuron. If it was the 'outputV' variable for that neuron was set to '1', indicating a spike, and 'spiketime' was set to 'refrac', which indicated that the neuron had entered a refractory period of length 'refrac' (in ms). In the case of the motor neuron, the timer for the motor 'mtimer' was set to '0' so that the motor control code could begin its execution.

Changing of Weights: This section is the most complicated part of the code because one had to be very careful of timing while updating neurons. The weights between neurons are updated in order to denote Hebbian learning. The model used to update the weights is an approximation of the biological model shown below.

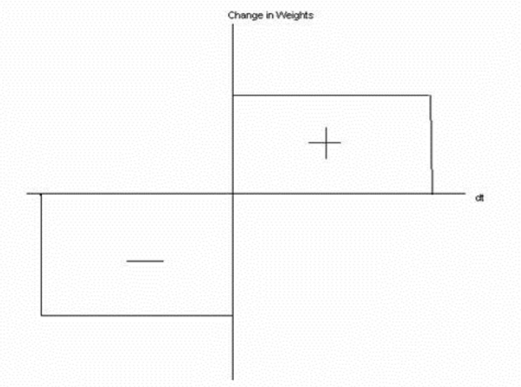

Fig.4 Approximation of the changing weights

Weights between two neurons need to be updated from the viewpoint of the presynaptic neuron and the postsynaptic neuron. There were three cases that need to be considered which are diagrammed and explained below:

a) Case 1:

Suppose the spike of the presynaptic neuron occurs at the start of every other cycle. This is denoted as 'timing reference' in the diagram. The timing in the program is setup so that the refractory period is twice the length of the cycle. The small vertical slashes between the two spikes of the presynaptic neuron indicate a new cycle.

So if the post synaptic neuron fires as seen in the figure below then it has fired after the presynaptic neuron has fired and the weights must be increased. In the code, this case is accounted for by checking if the spiketime of the postsynaptic is greater than the presynaptic neuron. Since spiketime counts down, a greater value for postsynaptic spiketime means that it fired after the presynaptic neuron.

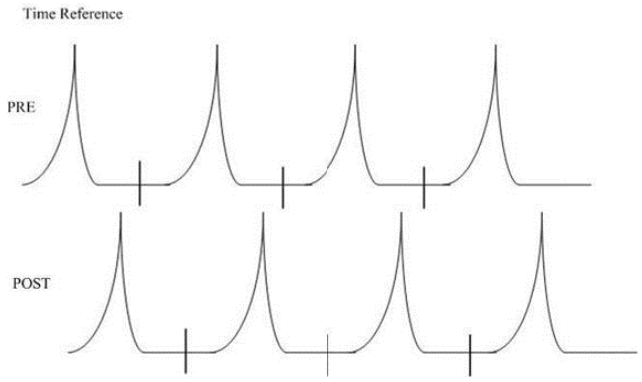

Fig: 5 Spiking in Case 1

b) Case 2:

If the postsynaptic neurons fire as seen in the figure below then it has fired before the presynaptic neuron for that cycle and so the weights between the two neurons must be reduced. This is done by checking if the postsynaptic spiketime is non-zero and less than presynaptic spiketime.

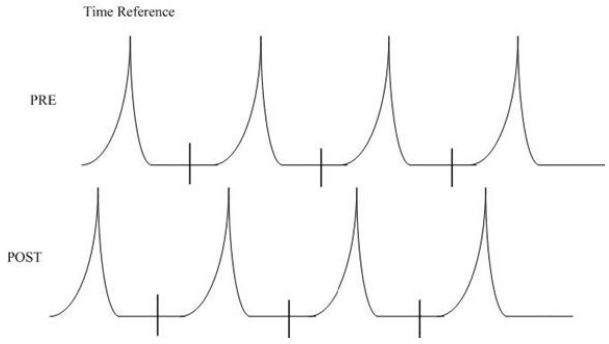

Fig. 6 Spiking in Case 2

c) Case 3:

In this case, the timing reference on the postsynaptic neuron

was set, that is, the spiking of the postsynaptic neuron occurs at the start of every other cycle. Here the only time the weights needed to be adjusted were, if the presynaptic neuron has fired before the postsynaptic neuron. This can be checked by seeing if spiketime for presynaptic neuron is not 0. If our check is true then the weights are increased between presynaptic and postsynaptic neurons. In a given cycle the presynaptic neuron can never fire after the postsynaptic neuron due to the manner in which the code executes in a given cycle.

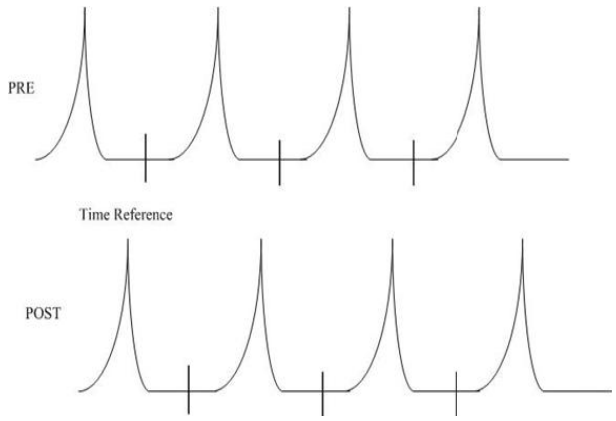

Fig.7 Spiking in Case 3

2) Stage 2:

Structure of Network: In this stage an inhibitory neuron (Neuron 3) is added to the network. This neuron also received its input from the same ADC pin as Neuron 1 but its threshold is set much higher and its 'outputV' variable assumes a value of '-1' when it spikes. The role of this neuron is to stop the motor from going too close to the light. It achieves its purpose by reducing the membrane potential of the motor neuron (Neuron 2) to which it is connected. Since its output spike is assigned a negative value the membrane potential of the motor neuron is updated by subtracting the weight between this neuron and the motor neuron from the membrane potential.

B. Timing Section

Neural network time: All the neurons in the network and their weights were updated every 40 ms. The refractory period was set to 80 ms. The refractory period had to be at least twice as long as it takes to update every neuron in the network. The reason for this can be seen from the model that was used to change weights of the neuron.

Also, if we observe the diagram below, we see that if the refractory period is shorter than the neural network time, the area for learning and unlearning begins to overlap.

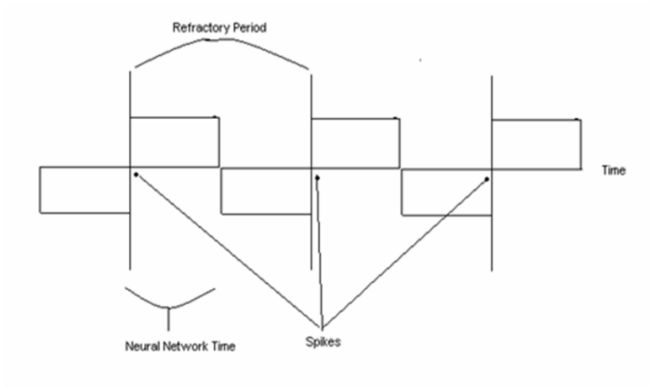

Fig. 8 Timing of Neural Network

ADC time*:* This is the amount of time the ADC is polled to get voltage values from the LEDs. This time has to be the same as the amount of time allotted to update the network, because voltage inputs from LEDs are inputs into the network.

Motor Time*:* In order to move the stepper motor through one step, we need to send it four consecutive pulses of equal width at equal time intervals. In order to overcome inertia and show suitable amount of movement for a single output spike the number of motor steps needed to be adjusted. The best results were obtained with three motor steps per spike of the output motor neurons. Besides this, the amount of time needed between pulses to the motor needed to be adjusted for optimum motor movement. The motor timing would significantly affect the timing of other parts of our neural network if this was made too long, however the time between pulses were long enough so that the motor could move through an entire step without shivering.

User Interface (HyperTerminal): In order to check the code, LEDs and HyperTerminal were used to debug. HyperTerminal was a great tool to help debug because it showed how the weights changed and if the program was functioning successfully. The HyperTerminal was used to train the robot to associate light input with the push button input and also to show learning and unlearning during this training process. The weights that are displayed on the screen are multiplied by a 1000 so that there is no need to display decimal points.

Changing of Weights: The weight adjustments for this additional neuron are similar to those described in the previous stage.

The learning rate for this neuron is set higher than the unlearning rate because one does not want the robot to forget quickly its proximity from the light source and crash into it.
For the robot to learn quickly, when it is supposed to stop while nearing a light source the starting weights between this neuron and the motor neuron are higher than the starting

weights between the regular light neurons (Neuron 1) and the motor neuron. The threshold voltage for this neuron is also set higher because this neuron inhibits only if the robot is too close to the light source.

Stage 3:

Structure of Network: Another set of four neurons are used to control backward motion of the robot in response to another LED that sensed light coming from the rear of the robot. This network of four neurons is coded very similar to the network described above in stages 1 and 2 of the design. The light neurons in this case received their input from pin A.1 of the ADC.

Changing of Weights: The weights in this network were also adjusted similar to how they were adjusted in stages 1 and 2.

C. Componenet Specifications and Hardware Design

One of the primary components of the robot are the LEDs used as inputs. The LEDs have been used as light sensors. These LEDs are cheap, directional, sensitive, and easy to use, making them great light sensors to serve the purpose. It was established that LED were a good choice as sensors. The LED voltage output went into PORTA of the MCU, which is the ADC. In order to make the signal going into the ADC less noisy, a simple low pass RC filter was built to take out the high pass noise.

The brain of the robot lies in the MCU, namely the ATMega32 in this case. AVR family has a GCC based IDE that is free for the whole range of their processors. From smallest to biggest, one code to rule them all. This without doubt is beneficent to use and implement. AVR is an 8-bit CPU and on the same clock it is 4 times faster than the 8-bit PIC and 12 times faster than the 8051.An ATMega32 development board was used, as it simplified the task of interfacing our components .The onboard RS232, LCD, LEDs and pushbuttons, without doubt enables us to test the code with great simplicity which were all reasons as to why AtMega32 was considered as the brain of the robot.

For the purpose of data acquisition, the Arduino Uno was used. The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins, 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

The various parameters of weather were acquired through the interfacing of the BMP085 sensor to the Arduino Uno. The specifications of the BMP085 are as follows: -

- BMP085 is a temperature, pressure, altitude sensor
- The sensor not only talks to the Arduino, but it also outputs to the serial port back to the PC.
- It features a measuring range of anywhere between 30,000 and 110,000 Pa for pressure.
- It can measure temperatures from 0 to 65 °C.
- The BMP085 has a digital interface, I2C to be specific.
- I2C is a synchronous two-wire interface, the first wire, SDA, transmits data, while a second wire, SCL, transmits a clock, which is used to keep track of the data.

XCLR acts as a master reset. It's active-low, so if it's pulled to GND it will reset the BMP085 and set its registers to their default state.

Fig.9 Backend of the BMP085 breakout board

Atmel® Studio 4 is the integrated development platform (IDP) for developing and debugging Atmel ARM® Cortex®-M processor-based and Atmel AVR® microcontroller (MCU) applications. The Atmel Studio 4 IDP gives a seamless and easy-to-use environment to write, build and debug our applications written in C/C++ or assembly code. Atmel Studio 4 supports all 8 and 32-bit AVR MCUs, the new SoC wireless family. It also connects seamlessly to Atmel debuggers and development kits. Additionally, the IDP now includes two new features designed to further enhance your productivity. This is the primary tool that is used for this robot.

Hardware/Software Tradeoffs:

1. The connections between the neurons were coded in software rather than via hardware so that the overall wiring of the robot would be less at the expense of a little more complex code.
2. The entire program was written in C rather than using both C and Assembly because we had enough time to run it in C. Using C made the code much more readable at the expense of less efficiency.
3. Floats were used for few of the variables in the code instead of fixed point binary because there was no huge timing problem and floats seemed to be doing a proper job in changing all the important parameters. Floats gave more accuracy at the expense of some speed.
4. The input neurons thresholding were coded instead of using voltage to frequency converters. This way, we can change the thresholds (and hence the frequency of spikes) as required both for testing and data purposes.

IV. RESULT ANALYSIS

The mechanical design involved producing a robot that would move appropriately to the spiking of the motor neurons. The LED circuitry cut off most of the high frequency noise and provided very smooth signals for the MCU to determine the intensity of the light at both the front and the back of the robot. The wiring for the motors worked well because we were able to get enough torque and pulse the robot correctly. The crux of the robot, the neural network, worked as hoped. Since the network inherently had some randomness involved, reaction to the inputs was not predictable. Every trial that was run, the program would output results that would vary. This was good because this Hebbian learning network had to figure out its own level of performance. Fixed targets that needed to be reached could not judge this level. This added difficulty to our task because it was hard to know if there was a problem of where to start debugging. Most debugging was based on ensuring that the correct neurons fired by using the LEDs. HyperTerminal was used a lot to examine the changing weights as the amount of light entering the LEDs were varied. Based on debugging, reasonable values were determined for the different thresholds of the different neurons and appropriate connections between the neurons that would show correct learning.

Outside factors such as the type of light source and the amount of ambient light also added more randomness to the network. The effect of ambient light was eliminated as much as possible, but the system would react a little differently when tested in daylight versus at night. Furthermore, brighter or dimmer light sources would also add variability. To deal with this, the voltage output of the LED was measured whenever a different light source was used to determine if the variability was significant enough to change some parameters in the code. Usually though, this was not the case and the network could handle many different situations and light sources within reasonable limits.

By splitting the neural network into stages, complexity could be added to the robot. Thus, smaller networks were perfected first, and then only after this worked, the entire eight-neuron network was built and tested.

A. Speed Of Execution

Speed overall was not a huge problem in the program. By outputting the timer onto HyperTerminal, it was concluded that it took an average 3-4 ms to get through the entire neural network every time. This large time was a consequence of how some of the variables involved floats, such as the weights. Multiplication of floats takes many cycles, and we had to do many multiplications, so this slowed the overall speed down. This was not a factor though because each motor pulse needed to be about 20 ms for the motors to behave nicely. As explained above, this resulted in executing our neural network code every 40 ms. This time was needed because it took about 240 ms for one motor movement, and we didn't want to interface many neuron firings together in order to show each individual firing through the robot. Thus, because it ran through the code every 40 ms, there were no speed problems and the ADC, motors, and the neural network showed no time problems.

One speed issue that had to be taken into consideration was when the weights were outputted to the HyperTerminal. It takes about 1 ms to transmit one character. Since it had to be made sure that the motor code could execute every 20 ms, the maximum allowed time through our neural network and Hyper Terminal code together was 19 ms. If the neural network would take 4 ms that meant that there were 15 ms extra time in which it could output to HyperTerminal. It was made sure that we would only output at most 10 characters (2 weight values) to be safe. After doing this, all parts of the code worked correctly and within appropriate time limits.

B. Accuracy

Complete accuracy is hard to determine for the neural network because it will always behave a little differently in every trial. Overall though, we had a sense of what should happen over time. At first, the robot should only move when a pushbutton is pushed. After the robot is trained, it should be able to move only with light. If a light source is far away, the robot will slowly move towards it and is it gets closer; the excitatory neurons will fire more causing the robot to move faster.

As the robot gets even closer though, the inhibitory networks will kick in stronger and the robot should start moving slower again. When the light is really close, the robot should stop moving and actually start slowly moving in the other direction. Thus there comes to a point where the robot will just move back and forth (jittering) but won't approach the light any more.

The above scenario would happen in many trials, but there were always trials where one or two weights would be much weaker than others even after learning. Thus, in some trials, it would take longer to inhibit than in other trials. The robot would keep trying to move closer to the light even if it was already so close. In biology terms, this is kind of like how some animals survive and some don't. Those animals that adapt and learn quicker have a better chance to survive than those that don't.

In other trials, inhibition would be very strong and the robot would stop moving when it got close to the light right away. Thus it learned very quickly an appropriate safe distance from the light. By looking at the weights, we could

see how all the excitatory and inhibitory weights evolved throughout a trial.

The speed of learning itself also varied greatly from trial to trial. While animals can learn, they can also unlearn, and in fact, the unlearning rate is greater than the learning rate for stability purposes. During the initial learning phase when we trained the robot to respond excitatory to light while holding the pushbutton, we noticed the great variations in learning times. In some trials, the network would learn very quickly and would keep learning and never unlearn. The excitatory weights would increase relatively quickly. In other trials, the network would keep learning and unlearning, but the weights would slowly increase. Thus, the robot would learn but it would take much longer than if the robot only continuously kept learning. In even other trials, the robot would only unlearn and would only learn very slowly or never learn.

In these trials, the excitatory weights were barely trained and the robot still would not respond to light. We were able to track all these weight changes during the learning phase via HyperTerminal. Therefore, this also shows real biological situations in how some animals learn much quicker than others, and how even others may never learn at all. We were able to see mathematically through our weights just how learning would occur in our robot, but many of these same situations could also be seen in the real-world.

Therefore, accuracy was hard to determine, but in every trial, we could always see evidence of learning. In almost all trials though, the robot would behave, as it should after a certain amount of time. It would usually move slowly when the light was far away and faster as it approached the light source. If inhibition were initially strong enough, it would once again slow down, as it got too close to the light. If inhibition was not strong enough, then the robot would still move quickly even if near the light, but then the inhibitory neurons would be firing quickly which would result in the inhibitory weights to increase. Thus, the robot would start to slow down or stop, but this would just take longer than if inhibition was strong to begin with.

After this training though, if the robot is put back in the middle of the course, it would slow down sooner as the light got too close, showing that it did learn from the previous time. Thus, not only were we able to see the robot learn real-time, it was also able to remember what it learned over time through the strengths of certain weights over others. While the weights may be different, overall behavior did follow what we hoped it would. Therefore, our neural network did model the situation we set out for it to model accurately and we did meet the specifications we did set up for ourselves.

V. CONCLUSION

In conclusion, the robot has reached the specifications intended to achieve at the start. The designing and modelling of the eight neurons, the development of the ADC code, decision on using the LED as a light input and choosing the Hebbian learning rule for self learning purposes have all been a good decision. The integration of the software design with the hardware is something was fully achieved. This involves the observation of the changing weights on HyperTerminal, and the integration of the code with the locomotion of the robot. Upon completion of the task at hand, data logging features were added to the robot. These could include remotely acquiring data and analysing it. Adding these extra functionalities could take place using the Uno.

There are also possibilities of building a larger neural network up to 16 neurons to enhance the functionality of the robot and improve on its directionality. Moreover, there also exists a possibility of making the robot react to various other types of stimuli along with the light input.

REFERENCES

[1]Zurada, Jacek M., "*Introduction to Artificial Neural Systems"*, January 1992.

[2]Blum Adam, '*Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems", John Wiley & Sons Inc.,* June 1992.

[3]Barrett, Steven F, *"Atmel AVR Microcontroller Primer: Programming and Interfacing"*, May 2008.

[4]Prabhu, S. M., & Garg, D. P. (April 1996). Artificial neural network based robot control: An overview. *Journal of Intelligent and Robotic Systems*, 333-365.

[5]Stojanovic, R., & Karadaglic, D. (2007). Sensors and their Applications XIV (SENSORS07). *Journal of Physics: Conference Series 76 (2007) 012054*.

[6] Brooks R and Mims M 2001 Development of an inexpensive handheld LED-based Sun photometer for the GLOBE program Journal of geophysical research 106, 4733–4740.

[7] Miyazaki E, Itami S and Araki T 1998 Using a light-emitting diode as a high-speed, wavelength selective Review of scientific instruments 69 11 547-588.

[8] Lau K T, Baldwin S, Shepherd R L, Dietz P H, Yerazunis W S and Diamond D 2004 Novel Fused-LEDs Devices as Optical Sensors for Colorimetric Analysis Talanta 63 1 167-173.

[9]I., Engedy, & Horvath, G. (2009). Artificial neural network based mobile robot navigation. *Intelligent Signal Processing,IEEE International Symposium* (pp. 241-246). Budapest: IEEE.