Application Of Neural Networks To Robot Animals
Final Project CS547

Matthew A. Letter

&

Lin sun

University of New Mexico

mletter1@unm.edu

sun@unm.edu

Table of Contents

Abstract

The Problem of improving robot lifetime in the simulated world created by professor Caudell can be used as a great learning platform for the implementation, understanding, and improvement of neural networks. The LMS neuron can be used to classify objects according to their properties. In order to classify objects effectively and efficiently in a simulated world, a strong understanding of how to use these properties is more important than knowledge of the neuronal algorithm itself. Neurons, such as the LMS neuron, need to be implemented and tested in an environment in order to determine whether they are working and trustable.

In this project, we will use the animal simulation world, provided by Thomas Caudell, as a platform for testing various neural network architectures. The starting point of our overall case study involves a non-neuron robot study, which is used as a control case. After a base line was determined, a simple neuronal object-classifying architecture was introduced leading to the implementation of more neuronal architectures for the purpose of direction determination. These studies come to a crescendo involving the combination, implementation, and integration of all the neural architecture studies. These studies yielded results, which emphasize the LMS neuron's ability for classifying objects in the simulated world. Furthermore, it was found that different inputs affect the neuron classification abilities. The conclusions drawn at the end of this paper will examine and discuss what potential future works could be constructed from our final architecture and how the final architecture performed when compared with controls and with factoring in previous architectural improvements and failures.

Key words: LMS, Classification, Neuron, Neural networks, robot learning

1. Introduction

This paper rigorously establishes that neural network design models can be used to manipulate learning on simulated biological system. All of the research were done using professor Thomas Caudell's robot environmental model. The research, in total, involved 9 architectures. These architectures were used to establish the parameters of lifespan with respect to the simulated organism and its environment. Providing a basis for analyzing the implemented neural network algorithms.

The goal of building these neural networks is to create a "brain" for the robot entity to live as long as possible in its environment. A determination on what neuronal design structures degrade or enhance lifespan will be used to support architectural changes in the process of achieving this goal. There will be two major kinds of neurons implemented, classification neurons and direction choice neurons. Classification neurons will be used to classify objects within the robots environment and direction neurons could be activated subsequently to determine a direction of the nearest non-poisonous objects around it.

2. Approach

2.1 Architecture 0: No movement, measure lifetime

For the first architecture, the basic metabolic rate for the robot was tested. The robot's speed was set to 0 to hold the robots metabolism at a constant baseline. The robot was programmed to do nothing until it dies. The lifespan was measured to calculate the basic metabolic rate so we can gain a basic insight of the robot 's life consumption.

2.2 Architecture 1: Movement, measure lifetime as a function of speed without eating food

This test is designed to explore the effect, of the robots speed variation, on its run lifespan. The robots direction was also varied to see if other environmental factors were correlated to lifespan. The goal of this experiment is to determine a relation between the speed of the robot and its metabolism.

The robot will start at the same position at the beginning of every new life cycle. The direction and speed will be varied every thing else will be held constant. The robots head will be moved 5 degree counter-clockwise, upon reset. Lastly, the robot will be set to eat nothing in the world. This ensures that the only varied factor, which can affect the lifespan of the robot, is fixed as speed.

2.3 Architecture 2: Movement, measure lifetime as a function of speed with eating food

This test is designed to explore the robot's lifetime when eating everything it encounters. This is different from architecture 1 in that neuron will eat all of the food it encounters. When the robot is moving, it will always head to the brightest object in the simulation world, no matter what type of object it is. To get a well-distributed lifespan from the environmental makeup, the initial robot's head angle will be changed for each run. The initial position will be held constant at the point of origination. This architecture will be used as a control, for the future neuronal architectures. This control will be used for comparisons (like architectures 0 and 1) when deciding whether future architectures improve the lifespan of the robot.

2.4 Architecture 3: Neuronal Classification of food using Food RGB Values

Architecture 3 is designed for classifying food, in the world, as either good or bad based on whether environmental objects increase or decrease the energy of the robot. This architecture will rely on the use of the LMS neuron algorithm. The inputs are visual readings, as RGB values, and the output is 0 for bad or 1 for good. If the food increases the energy of the robot the food is classified as good; all other changes are considered bad.

In order to classify the objects, the LMS neuron's classification weights need to be determined. The objects encountered by the robot will be used as supervised training sets. The learning rate is set to 0.01. The stopping criteria for training is set as when the Root Mean Squared Error (RMSE) change between two consecutive life times is less than 0.00000001 or the life time is greater than 3000. This is done in order to avoid being caught in an infinite loop. Figure 1 shows a diagram of the neural design.

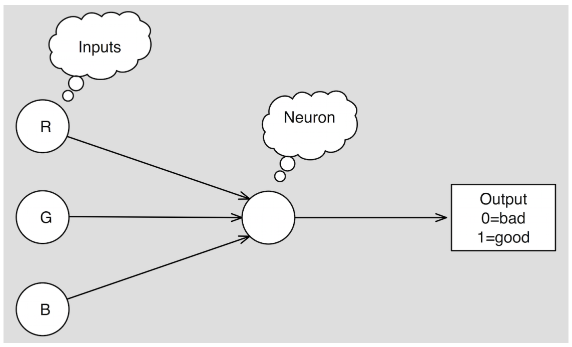

Fig 1: Network diagram of the LMS neuron for classifying food

2.5 Architecture 4: Neuronal Movement towards brightest object

The goal of this architecture is to let the robot move towards the brightest object in its environment using a winner-takes-all neuronal network implementation. In the architectural design process, movement is the next logical step after classification of food. The robot has to have a way of moving toward objects in its environment. In order to give the robot the ability to seek out objects a direction neuron will be added. This new neuron will make a direction choice.

The direction neuron will choose the direction based on the incoming light intensity. It is known that the light intensity is inversely related to the distance between the robot and an object. This means that objects having higher light intensity are closer to the robot than other objects with lower light intensity. In the simulation world, the robot goal is to survive for as long as possible. Therefore, it should try to move towards the objects in the world closest to it. In order to compare the intensity of different lights, a function to calculate the intensity of the light must be postulated.

Equation 1.1 shows how to calculate the intensity of lights based on its RGB values. A neuronal model, like the one in Figure 2, will be implemented. This is a two-layer neuron structure. The inputs for the neuron are RGB lights bands from 31 directions and the output is the angle of the brightest object with respect to the current horizontal axis. The first layer of neurons is used to calculate the intensity for each input, and the second layer of neurons will pick the highest input intensity. The direction that the robot will change towards is the direction of the winning light intensity.

$$\text{Intensity} = \text{sqrt}(R^2 + G^2 + B^2) \qquad \text{(eq 1.1)}$$

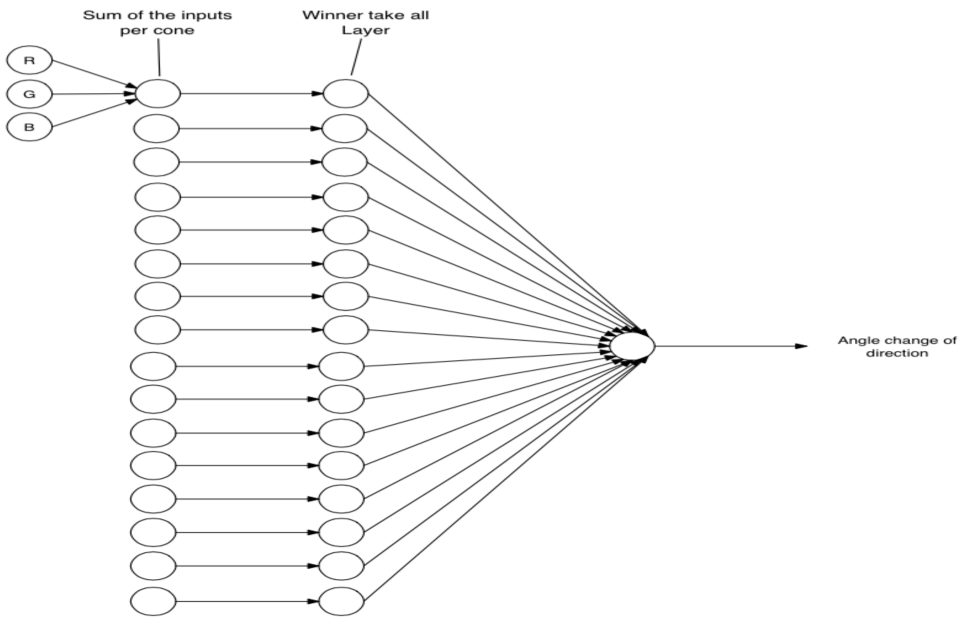

Fig 2: This is a three layer neural network where the RGB circles denote the intensity signal bands being received by the first layer. The second layer is a winner-take all network which selects the brightest cone. The third layer is a neuron, which computes the angle change for movement

2.6 Architecture 5: Move robot based on intensity and eat based on classification

Figure 3 shows the design for architecture 5. Building on architecture 4, the Integration of a classification neuron and direction neuron has been implemented in architecture 5. Each time the robot stops, the classification neuron is activated. A classification of the objects encountered by the robot is then neuronally computed from the learned weights in architecture 3. Therefore, if the objects are not poisonous, the robot should eat it. The robot will just skip the objects it considers poisonous. After an object classification, the direction neuron is activated to direct the robot in the direction with highest intensity and the robot proceeds in this direction. If the direction neuron can't find any object, the robot will rotate 45 degree clockwise and continue onward. Let it be noted that with the learned LMS classification neuron the robot should not waste energy on poisonous objects. Therefore, the robot in this architecture should have some sort of improved lifespan over our control (architecture 2).

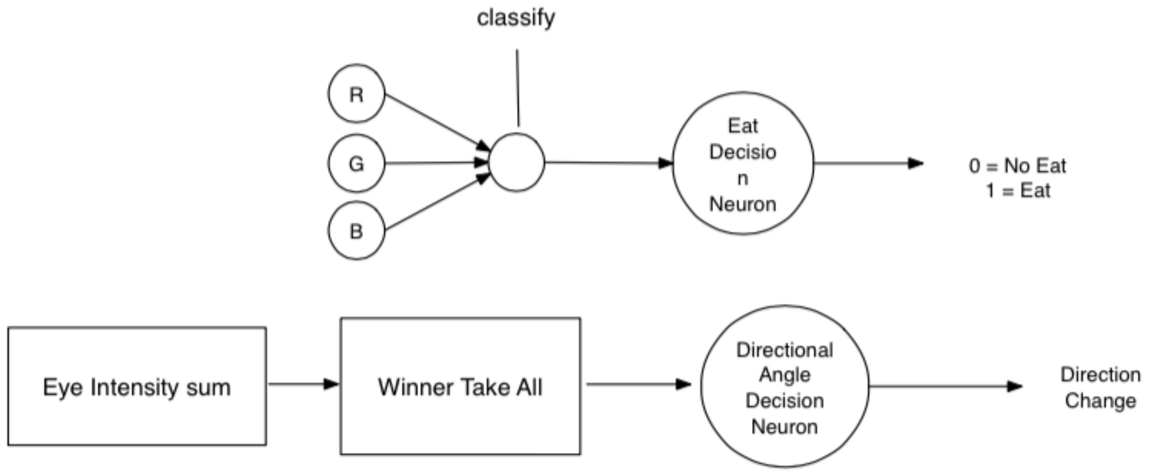

Fig 3: Network diagram of the LMS neuron for classifying food with the caveat of normalized inputs

2.7 Architecture 6: Neuronal Classification of food using Food RGB Percentage

In architecture 3, we use RGB values to classify objects. However, this does not address a certain problem where as robot moves farther away from an object, the light value for that object is decreased uniformly over all signal bands. The un-normalized input of RGB values cannot be trusted to classify objects, due to this fact. Moreover, when the robot moves it will pick up the direction with the highest intensity light value based on equation 1.1. It is possible that the direction picked up by direction neuron will lead the robot to poisonous objects; this is because direction is currently only based on the intensity of light not what type of object is producing the light. Therefore, we need to implement a “smart” direction neuron, which can orient in the direction with the closest nonpoisonous object.

To solve this problem we need implement a classification neuron that can classify objects regardless of the distance effect on light intensity. Leading to a new classification neuronal network architecture that takes normalized RGB values and classifies these values as either good or bad based on the design in Figure 4. Unlike architecture 3, the inputs are now normalized RGB values instead of the original RGB values. If the objects can increase the energy, the output is 1 (good); otherwise, it is 0 (bad). Equation 1.2 is used to normalize RGB values in this architecture, where V is individual band of the light R, G, or B. The new classification neuron will be trained using objects encountered in the robots environment. For randomization and testing purposes the head direction will change each time the robot re-spawns, varying the objects encountered. The learning rate for the classification neuron is still 0.01 while the stopping criteria is when RMS change, between two consecutive life times, is less than 0.00000001, or the life time is greater than 300, avoiding a potential infinite loop.

$$\text{Normalized } V = V/(R + G + B) \qquad \text{(Eq 1.2)}$$

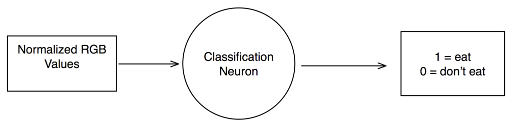

Fig 4: Representation of the normalized classification approach

2.8 Architecture 7: Eat Objects based on Neuron Classification using RGB Percentage and Choose Direction based on Light Intensity and Classification

In architecture 6, we implements a new classification neuron, which can classify incoming light based on normalized RGB making it so that the distance factor is not affecting neuronal outputs. Currently the robot wants to move towards the direction where the objects are closest to it and are non-poisonous. An upgraded direction neuron now needs to be implemented, which can select the direction with highest light density and where the light is classified as non-poisonous. Figure 5 shows how the new neurons will fit into the schema.

The new classification neuron will be incorporated into the direction neuron in order to improve upon direction choices. Now when the direction neuron checks some incoming light vector, the classification neuron will classify the incoming light before it hits the direction network. The output classification will then be multiplied by the intensity output, from first layer direction neuron. Therefore, if the light is classified as poisonous, the intensity is set to 0. Now, The light that the smart direction neuron picks up has the highest intensity and is also non-poisonous. Each time the robot stops, the classification neuron is activated to classify the objects encountered. If it is poisonous, the robot will just skip it. Otherwise, the robot will eat it. Then the smart direction neuron is activated to choose the direction to move in. If there is no suitable direction produced by the direction neuron, the robot will rotate 45 degrees clockwise, and move towards that direction.

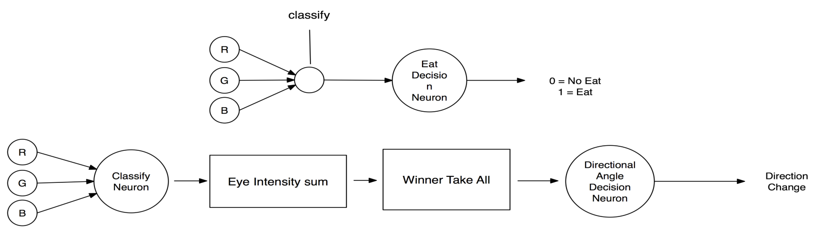

Fig 5: Network diagram of the LMS neuron for classifying food with pre-classification before winners take all network layers.

2.9 Architecture 8: Eat Objects with Small Mouse based on Neuron Classification using RGB Percentage and Choose Direction based on Light Intensity and Classification

Figure 6 shows the shape of the robot that the neurons are controlling. When the robot stops, it will use the classification neuron to classify objects it encounters. In this architecture only the middle light input cone will be used to classify objects (which is the beam in figure 6). However, when robot objects it will eat objects on the three forward facing sides. A new problem comes arises from this dichotomy.

Are we sure that the objects on the other two sides are non-poisonous? What If the objects on the mirrored two sides are poisonous but the object in the forward direction (side 0) is good? Then the robot will eat all objects around it even if they are poisonous. Upon this realization that the robot is wasting energy on poisonous objects the idea behind architecture 8 was born. In this architecture, the mouth of the robot will be made smaller, so that it can only eat the objects from the forward side (side 0). This is ideal because the eat classification neuron classifies objects in the forward direction.

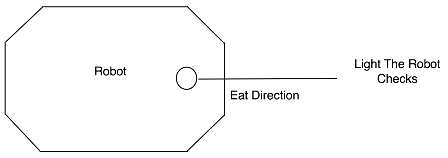

Fig 6: Diagram of updated eat direction, for the robot.

3. Results

3.1 Architecture 0: No movement, measure lifetime

Table 1: Over 200 runs, every run of a stationary robot had a lifespan of 5001 cycles		
Speed Rate	Lifetime Mean (Cycles)	Lifetime Standard Deviation
0	5001	0

From Table 1, we see that the robot life is 5001 cycles if it does not move and does not eat any food. It is know that the robot needs one cycle to jump out the loop initial start. Therefor, 5000 cycles of lifespan account for the robots life at birth. Upon calculation, the basic metabolic rate is 0.0002 unit/cycle.

3.2 Architecture 1: Movement, measure lifetime as a function of speed without eating food

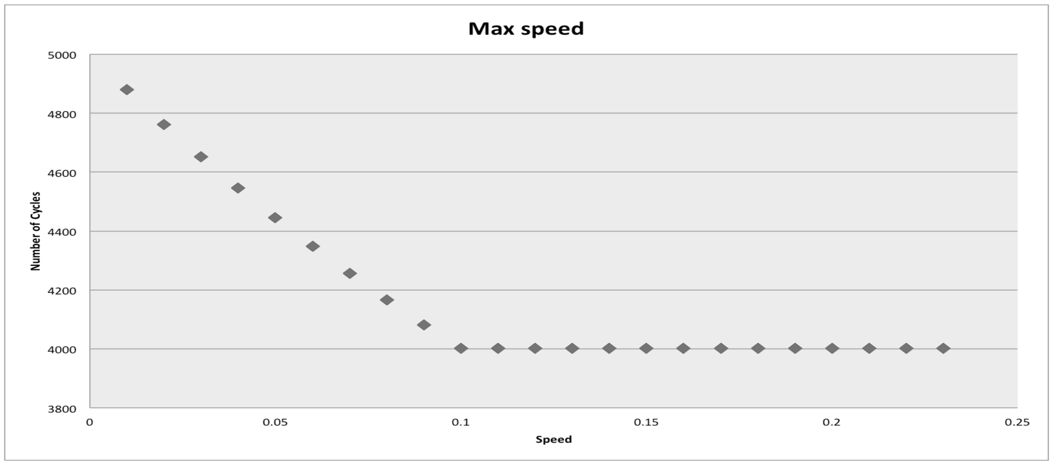

Fig 7: Shows the max speed of the robot to be 0.1

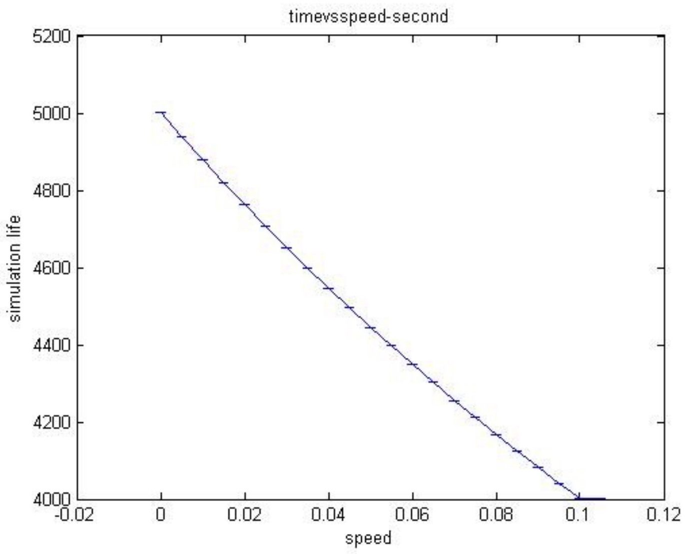

Fig 8: Average lifespan and standard deviation of the lifespan with respect to the robot's speed.

Figure 7 shows that the lifetime of the robot is directly related with its speed. When the robots speed is less than 0.1, and increasing iteratively, then the lifetime of the robot has a nearly linear functional correlation with speed. As the speed increases, the lifetime decreases. Note when the speed is over 0.1 the lifetime becomes constant. Changing the head angle, originating from the same origin, has limited effect on the life span of the robot. It is shown that the lifespan of the robot showed no change when a constant speed was set initially making the standard deviation 0.

From this experiment, we see that the movement metabolic rate is fixed when the movement rate is reached at 0.1. Therefore, in the following experiments for neuron trainings, we can fix the movement speed knowing that it is related to a constant life span.

3.3 Architecture 2: Movement, measure lifetime as a function of speed with eating food

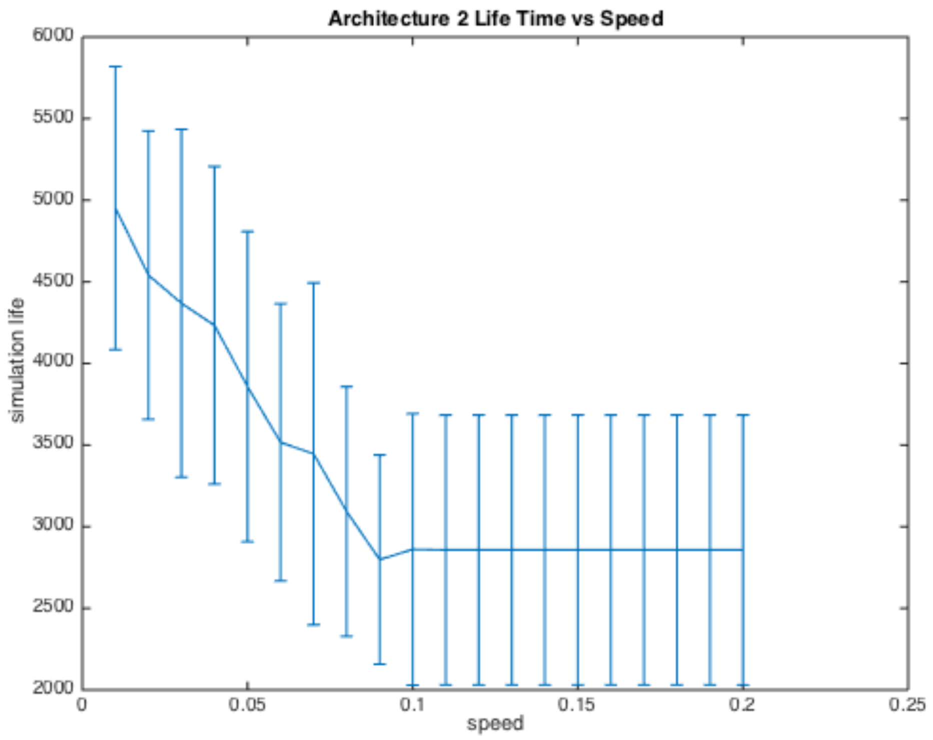

Fig 9: Error bar graph of varying speeds and directions where the standard deviation is represented as bars and the trend line represents the mean.

Table 2: Robot lifetime mean and standard deviation for different speed rate for comparison with future architectures.

Speed Rate	Lifetime Mean (Cycles)	Lifetime Standard Deviation
0.02	4541	884
0.04	4233	973
0.06	3517	849
0.08	3093	765
0.10	2861	832
0.12	2858	827
0.14	2858	827

Figure 9 shows the final result of lifetime vs. speed, with the robot eating all food it encounters. Like architecture 1, the lifetime is correlated with speed, further supporting our hypothesis from the previous architecture of their relation. As speed increases, the mean lifetime decreases. When the robots speed exceeds 0.1, the mean lifetime is stabilized. However unlike architecture 1, instead of getting a standard deviation of 0, we get a standard deviation that is greater than 0.The reason for this is that the robot can now eat everything it encounters in the world. We know this because of the observed affect on the robots life either positively or negatively based on the standard deviation bars around the lifespan mean trend line. Table 2 will serve as a base line for our future neural network implementation.

3.4 Architecture 3: Neuronal Classification of food using Food RGB Values

Figure 10 shows the training results for the classification neuron. Showing the RMS error dropping rapidly with large dramatic drops after each epoch. The RMS error values start to stabilized after a few runs, but an oscillatory change is noted that falls between 0.2 and 0.35 (see figure). So we could see that the neuron is trained well after iterations.

Table 3 shows the final weights for after training the neuron using RGB values. Weight 0 is used as the bias in the calculation. From observation of three weights, a preliminary conclusion can be draw such that the second bandwidth yields good food while the other two bandwidths are related to bad food. Using this information, we can now try to use these weights to classify objects that the robot encounters. Subsequently deciding if the robot should eat the objects or not. Verifying conclusion drawn from above.

Table 3: Root mean squared error (RMSE) vs. lifetime training result

Weight Index	0	1	2	3
Value	-0.0593627	-0.0574903	0.643285	-0.0570887

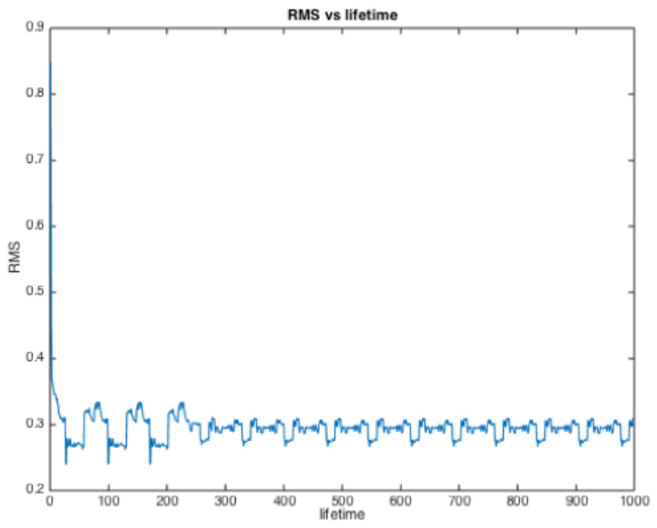

Fig 10: Direction decision neuron with readings, with a 5^{o} rotation change, after each life. This graph shows the RMSE dropping from 0.9 to asymptote around 0.3 suggesting that the LSM neuron has learned to classify the objects it has encountered.

3.5 Architecture 4: Neuronal Movement towards brightest object

In section 2.5, we postulated a theory about neuronal directionality decision-making. We now need to verify the neuronal algorithms correctness. There is an internally implemented function, called intensity_winner_takes_all, which returns the index of the receptor that has the highest intensity. We will compare the result from our implementation of a neuronal function with that of the internally coded function to see if they have the same result.

Figure 11 shows the results of this comparison. We can see that the neuron implemented here generates the same result as the internal function. Therefore, our direction neuron yields the same result as the internally coded function and we can now trust the direction picked by the direction neuron.

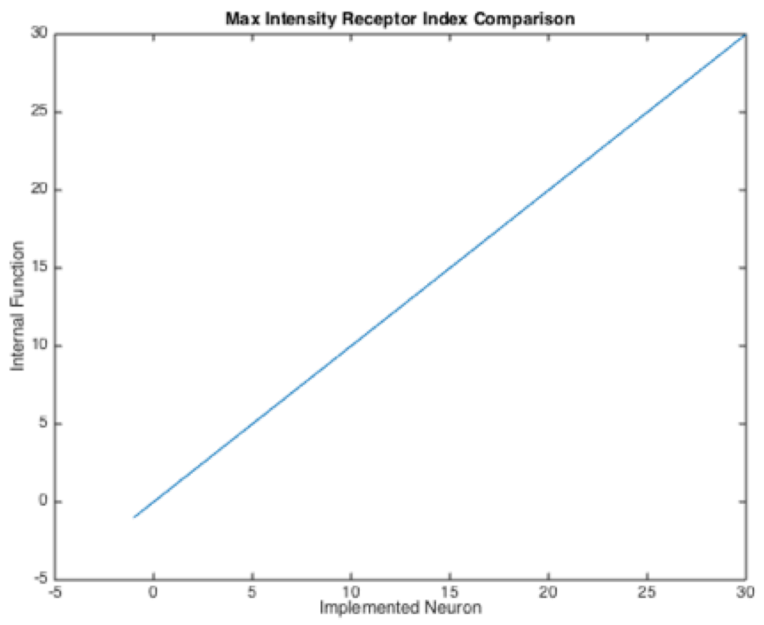

Fig 11: Intensity receptor index comparison between the internal function and the implemented neuronal function.

3.6 Architecture 5: Eat objects based on neuronal classification

In this architecture, the robot will have both a classification neuron and a direction neuron. This is in order to search for and eat good foods in the world. Comparing table 4 and table 2, we see that architecture 5 has a significant improvement in the mean lifespan over architecture 2. The improvement is attributed to the classification of food before eating. This is helping the robot selectively choose only food that is classified as good by the LMS neuron. Again we see a significant standard deviation with a constant speed. The standard deviation is stabilized when the speed reaches 0.1, as shown in Figure 12. The max speed of the robot in the simulation world is 0.1. This experiment directly shows that our new network architecture works well in classifying objects in the world.

Table 4: Robot lifetime mean and standard deviation for different speed rate for comparison with future architectures.

Speed Rate	Lifetime Mean (Cycles)	Lifetime Standard Deviation
0.02	4353	878.8
0.04	4323	965.2
0.06	3709	963.7
0.08	3661	1044.4
0.10	4884	1661.2
0.12	4862	1644.7
0.14	4862	1644.7

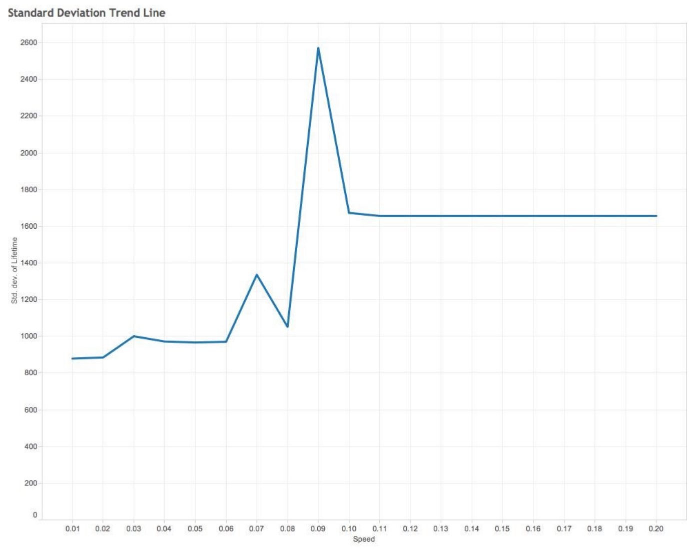

Fig 12: Standard deviation plotted as a line. We see the standard deviation levels out once the speed is constant.

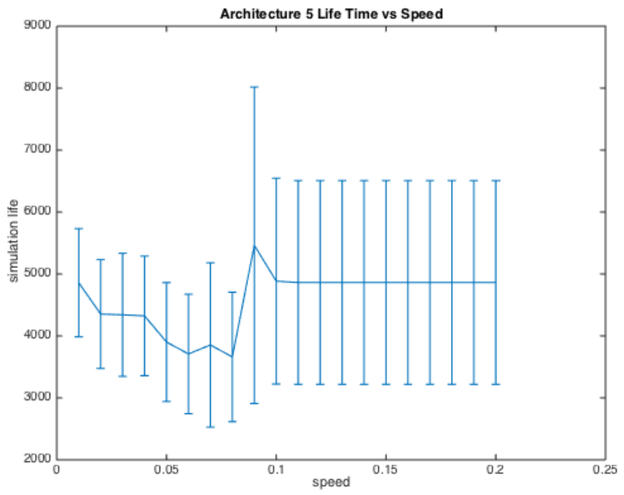

Fig 13: Intensity Receptor Index Comparison between the internal function and the implemented neuronal function. Note that the highest life span of the robot was 8000 and the minimum lifespan was 2500.

3.7 Architecture 6: Neuronal Classification of food using Food RGB Percentage

Normalization of the data appears to not be detrimental to the overall performance of the classification, which is reading incoming light based on figure 14. The large oscillation is hypothesized to be from the classification of good food in front of the robot and having bad food on the sides of the robot. This leads to a classification error where bad food on the sides of the robot is classified by the good food found in front of it. This will be addressed as a fine-tuning of architecture 7 by architecture 8. Take note that a biased input was included in the classification of the objects.

Through observation of the weights in table 5, a conclusion can be postulated that the second light bean contributes to the good food while the other two beams are related to bad food (also shown in a previous architecture). Furthermore, it seems that the third beam has higher contribution to bad food than the first beam through weight comparison. The next architecture will use a new classification neuron for exploring our hypothesis.

Table 5 Classification Neuron Weights after training with normalized inputs				
Weight Index	0	1	2	3
Value	0.0364013	-0.462995	1.26562	-0.833772

Table 6 Robot Lifetime Mean and Standard Deviation for Different Speed Rate with Direction selection for food		
Speed Rate	Lifetime Mean (Cycles)	Lifetime Standard Deviation
0.02	7337	1253
0.04	12763	4349
0.06	33165	15722
0.08	28193	11208
0.10	23935	9429
0.12	23911	9543
0.14	23911	9543
0.16	23911	9543
0.18	23911	9543
0.20	23911	9543

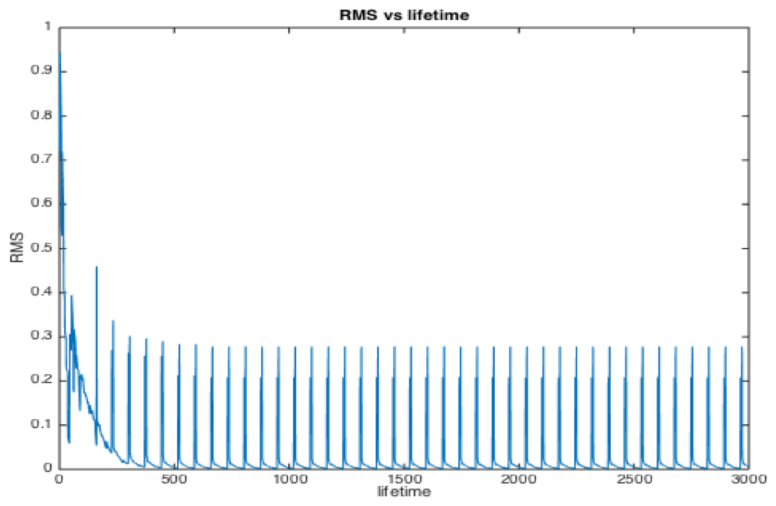

Fig 14: RMSE error drops as the LMS neuron adjusts its weights and learns how to classify objects found in its environment.

3. 8 Architecture 7: Eat objects based on neuron classification using RGB percentage and determine direction based on light intensity and classification

Comparing tables 4 and 6 shows that architecture 7 has gained large improvement in robot's lifespan. The lifetime has increases by nearly a factor of 4. This helps reinforce the conclusion that was introduced in architecture 6 where the second beam is related with good foods and that the other two beams are related with bad foods. Comparing figure 13 and figure 15 shows that the optimum speed, for robots survival, is consistently different for each architecture model. Exploring table 6 let it be noted that the best speed is 0.06. This is where the robot could survive for more than 30000 cycles on average.

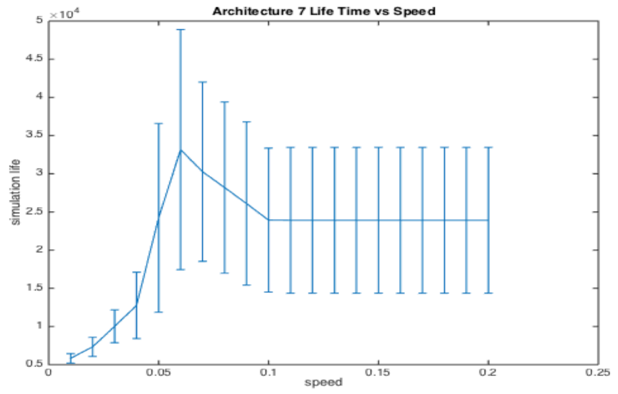

Fig 15: Mean lifespan trend line with standard deviation bars.

3. 9 Architecture 8: Eat Objects with Small Mouse based on Neuron Classification using RGB Percentage and Choose Direction based on Light Intensity and Classification

Comparing table 6 and table 7, we see that our new architecture has gained improvements. At the robots optimum speed, the robot can survive more than 50000 cycles on average per lifetime. Therefore, we can conclude that objects found on side 1 and 7 were not be classified directed by using a single beam. By classifying the objects correctly, the robot survives longer than all other previous architectures.

Table 7 Robot Lifetime Mean and Standard Deviation for Different Speed Rate with Direction selection for food

Speed Rate	Lifetime Mean (Cycles)	Lifetime Standard Deviation
0.02	8491	1200
0.04	24987	8612
0.06	51701	8355
0.08	28601	13893
0.10	32059	4848
0.12	30550	4426
0.14	30550	4426
0.16	30550	4426
0.18	30550	4426
0.20	30550	4426

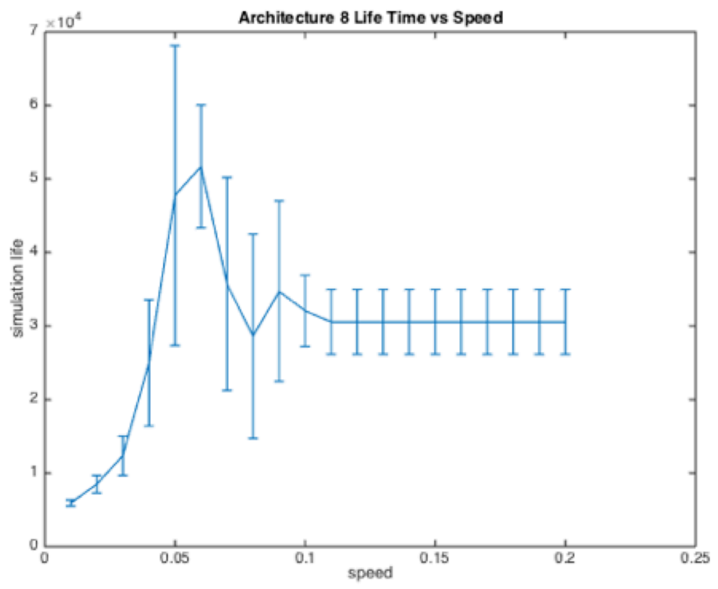

Fig 16: Mean lifespan trend line with standard deviation bars.

4. Discussion

The goal of this project was the Implementation of a neuronal network, which improves the robot's survivability in its environment. Two main types of neurons were used to build this network, classification neurons and direction neurons. Classification neurons are used to classify objects in the simulated world. The object fell into the category of either good for the robot or bad for the robot. Direction neurons were also used for making decision on which direction to move in after eating.

The implemented neuronal network was used to classify objects, based on their RGB values, and choose a path direction, based on the light intensity. Through our experiments (Comparing figure 9 and figure 13) we can see that our neurons are making improvements on the lifespan of the robot. Overall the average lifespan, for each speed, is being extended showing that the robot is gaining the ability to survive longer in its environment.

After this first dry run with our network we realized that there was a something that was not optimized. This optimization involved the classification of objects, regardless of the distance between the robot and the objects. This created a path for implementing a new neuronal architecture in which the robot now takes normalized RGB values instead of the original ones. Equation 1.2 was derived to transform the original RGB values to normalized values. This lead to the realization that when the robot picks a direction it doesn't take into consideration classification only light intensity. This lead to a needed upgraded to the direction neurons. It was determined that the neurons needed to pick a direction based on the nearest non-poisonous foods. A new architecture was used to check the lifespan of robot with this new brain. As shown in figure 15, the lifespan of the robot did improved. The lifespan of the robot was able to consistently reach 30000 cycles, which was a large jump when compared with the original lifespan.

Afterwards, we got a new insight, whether the objects on two adjacent sides are classified by our neurons correctly. We can see that our neuron will classify objects based on the forward light. This showed the need for improvement so that the robot only eats objects from one side, the side where objects are being classified correctly. As shown in figure 16, an unexpectedly large improvement was made through the implementation of this restriction. The lifespan average reached 50000 suggesting that our neuron architecture was receiving improvements that work well in the simulation world.

5. Summary and Conclusion

Through this project, we can see that neurons work well for the classification of objects. In our trials, neurons were used to classify objects in the world successfully. Also, direction neurons were used to help robot pick up directions successfully. Based on the results spread out over all the architectures, it can be clearly seen that the robot survival rate are improving with each new architectural addition. The algorithms used in the architectures are mainly LMS and winner-take-all algorithms. Visual light vectors were used as inputs for classifying objects. The light values are related with object food properties.

The robot only classifies objects on the forward facing side. Therefore, if there are some good objects lying on the other sides, the robot will miss it with a high probably. For future works, a good architecture would be one that implements a way to classify objects not found in the forward direction and try to eat these objects if they are good.

One idea that was hypothesized is the idea of rotating the body after eating from one side. A potential new issue was postulated with this hypothesis, whether the rotation energy consumption is larger than the potential energy that could be received from rotation of the robot. It has not been established that the rotation will consume energy or not in the simulation world. Therefore, there would need to be experiments made that explore whether or no this would be a potential issue.

There are other signals that can also be leveraged in the simulation world such as sound. An educated guess cannot be made on how the sound would work in neuronal classification, as no experiments have been done on sound. We do know that the light beam is being received by only one object, but what about sounds? Are they mixed signals from many of the objects? Can we distinguish the sound by just using the LMS neuronal archetype? These questions are very hard to answer and would need experiments to test each of them.

6. Acknowledgements

Great thanks should be given to Thomas Caudell for creating the simulated environment and for being one of the best professors at the University of New Mexico. Also, Thank Fengshu Xu for being of great help in ironing out the kinks of our architectures. Finally, Special thanks should be given to our team for making all architectures possible and for all the hard work they put in to the project.

7. References

1. Caudell, Thomas: Simulated world
2. Haykin, Simon S. *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ: Prentice Hall, 1999. Print

8. Appendix A

1. Perceptron.h

```
/**********************************************************/
/*  Perceptron header file                               */
/**********************************************************/

#ifndef PERCEPTRON
#define PERCEPTRON

typedef struct perceptron
{
	int input_num;
	double *weights;
	double v;
	double output;
	double error;
	int is_inner_neuron;		/*inner neuron has no error, but sigma instead*/
	void *param;		/*Reserved variable for future use*/
}perceptron;

/**********************************************************/
/*  Global variable                                      */
/**********************************************************/


/**********************************************************/
/*  Exportable functions                           */
/**********************************************************/
void perceptron_default(perceptron *neuron);

void perceptron_clear(perceptron *neuron);

#endif
```

2. Perceptron.c

```
#include "Perceptron.h"
#include <stdlib.h>

/**********************************************************/
/*  Exportable functions                           */
/**********************************************************/
void perceptron_default(perceptron *neuron)
{
	neuron->weights = (double *)malloc(sizeof(double) * (neuron->input_num + 1));
}

void perceptron_clear(perceptron *neuron)
{
	free(neuron->weights);
}
```

3. LMSAlgorithm.h

```
/*
 * LMSAlgorithm.h
 *
 *  Created on: Oct 24, 2014
 *      Author: lin
 */
#include "Perceptron.h"
#include <math.h>

#ifndef LMSALGORITHM_H_
#define LMSALGORITHM_H_

extern perceptron neuron_brain;
extern double accumulated_rms;
/***********************************************************/
/*  Exportable functions                                   */
/***********************************************************/
int LMScalculate(float *inputs, int input_num, int isCal, float expected);

#endif /* LMSALGORITHM_H_ */
```

4. LMSAlgorithm.c

```
/*
 * LMSAlgorithm.c
 *
 *  Created on: Oct 24, 2014
 *      Author: lin
 */
#define OUTPUT_NUM 1
#define LEARNING_RATE 0.01
#define NEURON_NUM 1

#include "LMSAlgorithm.h"
#include "Perceptron.h"

int initialized = 0;
perceptron neuron_brain;
double accumulated_rms = 0;
float forwardspeed;

void v_function(float* inputs, struct perceptron* p)
{
	int it = 0;
	p->v = 0;

	p->v += p->weights[0];/*Bias*/
	for(it = 0; it < p->input_num; it++)
	{
```

```
            p->v += (inputs[it] * p->weights[it+1]);
        }
}

void y_function(struct perceptron* p)
{
        p->output = p->v;
}

void adjust_function(float* inputs, struct perceptron* p, void *params)
{
        double target =*((double *) params);
        int idx = 0;

        p->error = target - p->output;

        p->weights[0] += (p->error * 1 * LEARNING_RATE);/*Bias*/
        for(idx = 0; idx < p->input_num; idx++)
        {
            p->weights[idx+1] += (p->error * inputs[idx] * LEARNING_RATE);
        }
}

void initialize(int input_num)
{
        int idx = 0;
        neuron_brain.input_num = input_num;
        perceptron_default(&neuron_brain);

    /*This part uses the old data*/
    neuron_brain.weights[0] = 0.493975;
    neuron_brain.weights[1] = -0.658991;
    neuron_brain.weights[2] = 0.720632;
    neuron_brain.weights[3] = -0.144448;

    /*This part is used to train the neuron*/
//      for(idx = 0; idx <= input_num; idx++)
//      {
//          neuron_brain.weights[idx] = 0.1 * (rand()%10);
//      }
}


/*********************************************************/
/*  Exportable functions                                 */
/*********************************************************/
int LMScalculate(float *original_inputs, int input_num, int isCal, float expected)
{
        int ret = 0;
        double e = (double)expected;
    float *inputs = (float*)malloc(input_num * sizeof(float));
    memcpy(inputs, original_inputs, sizeof(float) * ( input_num));

          if(!initialized)
          {
             initialize(input_num);
             initialized = 1;
          }

          v_function(inputs, &neuron_brain);
          y_function(&neuron_brain);

          if(neuron_brain.output > 0)
             ret = 1;

          if(isCal)
          {
             adjust_function(inputs, &neuron_brain, &e);

             /*Log rms*/
             accumulated_rms += (pow(neuron_brain.error, 2));
          }
          else
          {

          }

          return ret;
}

void reset()
{
          accumulated_rms = 0;
}
```

5. DirectionControlNeuron.c

```
//
//  DirectionControlNeuron.c
//  FlatWorldIIV1.0ClassVersion_dist2014
//
//  Created by lin sun on 11/22/14.
//  Copyright (c) 2014 lin sun. All rights reserved.
//

int set_direction(WORLD_TYPE *world, AGENT_TYPE *agent, int eye_idx)
{
    VISUAL_SENSOR_TYPE **eyes = agent->instate->eyes;
    int num_receptors;
    int num_bands;
    int receptor_idx = 0;
    int band_idx = 0;
    int max_receptor = -1 ;
    float intensity = 0;
    float maxintensity = 0;
    float bodyx = 0;
    float bodyy = 0;
    float bodyh = 0;
    int ret = 0;

    num_receptors = eyes[0]->nreceptors ;
    num_bands = eyes[0]->nbands ;

    read_visual_sensor(world, agent) ;
    extract_visual_receptor_values_pointer(agent, 0) ;

    for(receptor_idx = 0; receptor_idx < num_receptors; receptor_idx++)
    {
        intensity = 0 ;

        for(band_idx = 0; band_idx < num_bands; band_idx++)
            intensity += eyes[0]->values[receptor_idx][band_idx] ;
        ret = LMScalculate(eyes[0]->values[receptor_idx], agent->instate->eyes[0]->nbands, 0, 0);
        if(intensity > maxintensity && ret == 1)
        {
            max_receptor = receptor_idx;
            maxintensity = intensity ;
        }
    }

    return max_receptor;
}
```

6. Controller.c

```
//
//  Controller.c
//  FlatWorldIIV1.0ClassVersion_dist2014
//
//  Created by lin sun on 10/02/14.
//  Copyright (c) 2014 lin sun. All rights reserved.
//
#include "LMSAlgorithm.h"
#include "Perceptron.h"
typedef struct object
{
        float *inputs;
        int input_num;
        float output;
        struct object *next;
}object;

typedef struct object_list
{
        object *header;
        object *cur;
        int object_num;
}object_list;

static object_list list;
int epoch_num = 0;
double rmss[10000];
int object_num = 0;
float forwardspeed = 0.01;
float init_x;
float init_y;
float init_head_position = 0;

/*Unexported functions*/
object *addObject(float *input, int input_num, float output);
void clearObjects();

void agents_controller( WORLD_TYPE *w )
{ /* Adhoc function to test agents, to be replaced with NN controller. tpc */

        AGENT_TYPE *a ;
        int collision_flag=0 ;
        int i,k ;
        int maxvisualreceptor = -1 ;
        int nsomareceptors ;
        int nacousticfrequencies ;
        float delta_energy, old_delta_energy;
        float dfb , drl, dth, dh ;
        float headth ;
        //float forwardspeed ;
        float maxvisualreceptordirection ;
        float bodyx, bodyy, bodyth ;
        float **eyevalues, **ear0values, **ear1values, **skinvalues ;
        float ear0mag=0.0, ear1mag=0.0 ;
        time_t now ;
        struct tm *date ;
        char timestamp[30] ;
        int is_poisonous = 0;
        //temporary file
        char eye_data_file_name[20] = "./dat/eye_data_";
    char direction_data_file_name[20] = "./dat/direction_data";
        char eye_data_file_name_str[50] = "";
        int idx = 0;
    float stopping_criteria = 0.00000001;
        FILE *fp = 0x0;
    int ret = 0;
    float *input_data = 0x0;

        a = w->agents[0] ; /* get agent pointer */

        /* test if agent is alive. if so, process sensors and actuators.  if not, report death and
```

```
            reset agent & world */
          if(a->instate->metabolic_charge > 0.0)
          {
            /* get current motor rates and body/head angles */
            read_actuators_agent(a, &dfb, &drl, &dth, &dh ) ;
            read_agent_body_position( a, &bodyx, &bodyy, &bodyth ) ;
            read_agent_head_angle( a, &headth );

            /* read somatic(touch) sensor for collision */
            collision_flag = read_soma_sensor(w, a) ;
        skinvalues = extract_soma_receptor_values_pointer( a ) ;
        nsomareceptors = get_number_of_soma_receptors( a ) ;

            /* read visual sensor to get R, G, B intensity values to collect x values*/
            read_visual_sensor(w, a) ;
            eyevalues = extract_visual_receptor_values_pointer( a, 0 ) ;

            for(k = 0 ; k < nsomareceptors ; k++)
        {

           ret = LMScalculate(eyevalues[a->instate->eyes[0]->nreceptors/2], a->instate->eyes[0]->nbands,
0, 0);

           if((k == 0) && skinvalues[k][0] > 0.0 && ret == 1)
           {
              delta_energy = eat_colliding_object(w, a, k) ;

//             /*Train the neuron*/
//             if(delta_energy != 0)
//             {
//                object_num++;
//                LMScalculate(eyevalues[a->instate->eyes[0]->nreceptors/2], a->instate->eyes[0]->nbands,
1, (delta_energy > 0 ? 1.0 : 0.0));
//             }
                     }
        }

        /*Move robot*/
        ret = set_direction(w, a, 0);

        read_agent_body_position(a, &bodyx, &bodyy, &bodyth) ;

        if(ret == -1)
        {
           set_agent_body_angle(a, bodyth + 45);
        }
        else
        {
           set_agent_body_angle(a, bodyth + a->instate->eyes[0]->receptor_locations[ret] + a->instate-
>eyes[0]->receptor_directions[ret]);
        }
```

```
          /* move the agents body */
          set_forward_speed_agent(a, forwardspeed) ;
          move_body_agent(a) ;

          /* decrement metabolic charge by basil metabolism rate.  DO NOT REMOVE THIS CALL */
          basal_metabolism_agent(a) ;
          simtime++ ;
        } /* end agent alive condition */
        else
        {
          /* Example of agent is dead condition */
          printf("agent_controller- Agent has died, eating %d objects. simtime: %d\n",a->instate->itemp[0],
simtime ) ;
          now = time(NULL) ;
          date = localtime( &now ) ;

       sprintf(eye_data_file_name_str, "./dat/timevsspeed.csv");
          if((fp = fopen(eye_data_file_name_str, "a+")) != 0x0)
          {

                    fprintf(fp, "%f,%f,%d\n", forwardspeed, init_head_position, simtime);
                    fclose(fp);
          }

          /* Example as to how to restore the world and agent after it dies. */
          restore_objects_to_world(Flatworld) ;  /* restore all of the objects back into the world */
          reset_agent_charge( a ) ;              /* recharge the agent's battery to full */
          a->instate->itemp[0] = 0 ;             /* zero the number of object's eaten accumulator */

       /* keep starting position the same and change head angle */

       init_head_position += 20;

       nlifetimes++ ;
       if(nlifetimes%18 == 0)
       {
         init_head_position -= 360;
         forwardspeed += 0.01;
       }

       init_x = (Flatworld->xmax + Flatworld->xmin)/2;
       init_y = (Flatworld->ymax + Flatworld->ymin)/2;

          printf("\nagent_controller- new coordinates after restoration:  x: %f y: %f h: %f f: %f\n", init_x,
init_y, init_head_position, forwardspeed) ;
          set_agent_body_position(a, init_x, init_y, init_head_position) ;

          /* Accumulate lifetime statistices */
          simtime = 0 ;
        //maxnlifetimes
            if(nlifetimes >=  360) //|| (epoch_num > 10 && fabs(rmss[epoch_num - 1] - rmss[epoch_num -
2]) < stopping_criteria))   /*Add stopping condition for the neuron training to stop*/
            {
           /*plot data and clean up data*/

//           sprintf(eye_data_file_name_str, "%sdata.csv", eye_data_file_name);
//           printf("store the data and exit with epoch %d\n", epoch_num);
//
//           if((fp = fopen(eye_data_file_name_str, "w+")) != 0x0)
//           {
//              /*Log rms*/
//              for(idx = 0; idx < epoch_num; idx++)
//              {
//                 fprintf(fp, "%d,%lg\n", idx+1, rmss[idx]);
//              }
//
//              fprintf(fp, "\nThe final weights are:\n");
//
//              /*Log weights*/
//              for(idx = 0; idx <= neuron_brain.input_num; idx++)
//              {
//                 fprintf(fp, "%lg,", neuron_brain.weights[idx]);
//              }
//
//              fclose(fp);
//           }
//           epoch_num = 0;

                    exit(0) ;
            }
            else
            {
//                      rmss[epoch_num] = pow(accumulated_rms/object_num, 0.5);
//           printf("This is the %dth epoch with %f random speed with %f rms\n", epoch_num, forward-
speed, rmss[epoch_num]);
//                      epoch_num++;
//                      accumulated_rms = 0;
//                      object_num = 0;
            }
        } /* end agent dead condition */
}/*end agents_controller()*/

object *addObject(float *inputs, int input_num, float output)
{
        object *new_object = (object *)malloc(sizeof(object));
        new_object->inputs = inputs;
        new_object->input_num = input_num;
        new_object->output = output;
        list.object_num++;
        list.cur->next = new_object;
        list.cur = list.cur->next;
        list.cur->next = NULL;

        return new_object;
}

void clearObjects()
{
        object *o1;
        object *o2 = list.header->next;
        list.header->next = NULL;
        list.object_num = 0;

        while(o2)
        {
          o1 = o2;
          o2 = o2->next;
          free(o1);
        }
}
```