**Application Of Neural Networks To Robot Animals**

**Final Project CS547**

Matthew A. Letter

&

Lin sun

University of New Mexico

Neural Networks 547

mletter1@unm.edu

sun@unm.edu

Table of Contents

**TOC \o 3-3 \t "Heading, 4"**

**Abstract(Todo) PAGEREF \_Toc \h 3**

**1. Introduction PAGEREF \_Toc1 \h 4**

**2. Approach PAGEREF \_Toc2 \h 4**

*2.1 Architecture 0: No movement, measure lifetime PAGEREF \_Toc3 \h 4*

*2.2 Architecture 1: Movement, measure lifetime as a function of speed without eating food PAGEREF \_Toc4 \h 4*

*2.3 Architecture 2: Movement, measure lifetime as a function of speed with eating food PAGEREF \_Toc5 \h 5*

*2.4 Architecture 3: Neuronal Classification of food using Food RGB Values PAGEREF \_Toc6 \h 5*

*2.5 Architecture 4: Neuronal Movement towards brightest object PAGEREF \_Toc7 \h 6*

*2.6 Architecture 5: Move robot based on intensity and eat based on classification PAGEREF \_Toc8 \h 7*

*2.7 Architecture 6: Neuronal Classification of food using Food RGB Percentage PAGEREF \_Toc9 \h 8*

*2.8 Architecture 7: Eat Objects based on Neuron Classification using RGB Percentage and Choose Direction based on Light Intensity and Classification PAGEREF \_Toc10 \h 9*

*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 PAGEREF \_Toc11 \h 10*

**3. Results PAGEREF \_Toc12 \h 11**

*3.1 Architecture 0: No movement, measure lifetime PAGEREF \_Toc13 \h 11*

*3.2 Architecture 1: Movement, measure lifetime as a function of speed without eating food PAGEREF \_Toc14 \h 11*

*3.3 Architecture 2: Movement, measure lifetime as a function of speed with eating food PAGEREF \_Toc15 \h 13*

*3.4 Architecture 3: Neuronal Classification of food using Food RGB Values PAGEREF \_Toc16 \h 14*

*3.5 Architecture 4: Neuronal Movement towards brightest object PAGEREF \_Toc17 \h 15*

*3.6 Architecture 5: Eat objects based on neuronal classification PAGEREF \_Toc18 \h 16*

*3.7 Architecture 6: Neuronal Classification of food using Food RGB Percentage PAGEREF \_Toc19 \h 18*

*3. 8 Architecture 7: Eat objects based on neuron classification using RGB percentage and choose direction based on light intensity and classification PAGEREF \_Toc20 \h 20*

*3. 9 Architecture 8: Eat Few Objects based on Neuron Classification using RGB Percentage PAGEREF \_Toc21 \h 21*

**4. Discussion PAGEREF \_Toc22 \h 22**

**5. Summaries and Conclusion PAGEREF \_Toc23 \h 22**

**6. Acknowledgements PAGEREF \_Toc24 \h 22**

**7. References PAGEREF \_Toc25 \h 22**

**8. Appendix A PAGEREF \_Toc26 \h 23**

# Abstract(Todo)

Add abstract here, single space, 12pt intentionally blank

# 1. Introduction

This paper rigorously establishes that neural network design models can be used to manipulate learning on simulated biological system. All the research was done using professor Thomas Caudell’s robot environmental model. ADDIN EN.CITE <EndNote><Cite><Author>Thomas Caudell</Author><DisplayText> (Caudell)</DisplayText><record><ref-type name="Generic">13</ref-type><contributors><authors><author>Thomas Caudell</author></authors></contributors><titles/><title>Flatworld</title><periodical/><dates><year/><pub-dates/></dates></record></Cite></EndNote> (Caudell) The research totally involved 9 architectures, used to establish the parameters of lifespan with respect to the simulated organism and its environment. These provide a basis for analyzing the implemented neural network algorithms, the goal of which is to create a “neuron” based brain for the robot organism to live as long as possible in its environment, and determine what neuronal design structures degrade or enhance this attribute. There will be two major kinds of neurons implemented, classification neuron and direction neuron. Classification neuron will be activated by the robot to classify the food in the simulation world, while direction neuron could be activated to pick up a direction for robot to search for nearest non-poisonous foods around it in the simulation world.

# 2. Approach

### 2.1 Architecture 0: No movement, measure lifetime

For the first architecture, we will test the basic metabolic rate for the robot. The robot’s speed was set to 0 to get rid of the effect of speed on the result; therefore the robot was programmed to do nothing until it dies. Finally the lifespan was measured to calculate the basic metabolic rate.

### 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 and varying the robots direction correlated to lifespan. The goal of this experiment is to determine a correlation between the speed of the robot and its metabolism. Furthermore, the direction change should yield insight into the world around the robot. The robot will start at the same position where the direction and speed will be varied. To try test different directions, we will turn the neuron head to 5 degree counter-clockwise each time. When the neuron is moving, it will always head to the brightest object in the space, no matter if the food is poisonous or not. The robot will be set to eat nothing in the world.

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

This test is designed to explore the robot’s average lifetime if the robot has no “brain”. 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 space, no matter what type of object it is. To get a well-distributed random lifespan for individual speed, the initial robot head angle will be changed for each run, but the initial position will be held constant at the point of origination. Actually, this architecture will generate the control experiment for the future architectures. Therefore, for a new architecture, we can use this control experiment for comparison to decide if our new implemented architecture improves the lifespan for the robot or not.

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

This test will do classification of food in the world as either good or bad based on whether the food could increase or decrease energy of the robot using LMS neuron architecture. Fig 1 shows the frame for the classification neuron, where the inputs are visual readings as RGB values and output is 0 or 1. If the food increases the energy of the robot, the food is classified as good and the neuron output is 1; otherwise, we say that the food is bad and the output is 0 now. For a simple starting neural architecture an LMS neuron is used to classify the food. It is known that, in order to classify the objects, we should know the weights for classification neuron. Therefore, in this architecture, we will use the objects encountered by the robot to train classification neuron. To pick up objects randomly, we will change the head direction each time, so that the objects encountered by the robot is randomized. The learning rate of classification neuron is 0.01, while the stopping criteria for training is when RMS change between two consecutive life times is less than 0.00000001 or the life time is greater than 3000 to avoid infinite loop.



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 using a winner-takes-all neuronal network implementation. In our design, movement is the next logical step after classification of food since robot has to have a way of moving toward objects. In order to give the robot the ability to seek out objects a direction neuron will be added that will make a direction choice.

In this architecture, The 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, which 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 wants to survive for as long as possible. Therefore, it should try to move towards the objects in the world closest to it. First, to compare the intensity of different lights, we need have a function to calculate the intensity of the light. Equation 1.1 shows how to calculate the intensity of lights based on its RGB values. Based on the thought process above, a neuron with frame in Fig 2 will be implemented. Actually it is a two-layers neuron structure, where the inputs for the neuron are RGB lights bands from 31 directions, while the output is the angle of the lightest object with respect to the horizontal line. The first layer neuron is used to calculate the intensity for each input, and the second layer neuron will pick up the input which generate the highest intensity. So the direction that the robot will change towards is the direction chose by direction neuron based on light intensity.

Intensity = sqrt(R^2 + G^2 + B^2) (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

Fig 3 shows the frame for architecture 5, where we will integrate classification neuron and direction neuron. Each time the robot stops, classification neuron is activated to classify the objects encountered by the robot. If the objects are not poisonous, the robot will eat it. Otherwise, the robot will just skip the objects. After that, direction neuron is activated to pick up the direction with highest intensity, so that the robot will move towards that direction. If direction neuron can’t find any object, the robot will rotate 45 degree clockwise and then move towards that direction. Because the neuron could classify the objects, the robot will not eat poisonous objects. Therefore, the robot will not waste energy on poisonous objects, which means that robot in this architecture should have larger average lifespan than the robot in 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 subset, as robot moves farther away from an object, the light value for that object is decreased uniformly over each signal band. 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. However, 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 move towards the direction with the closest nonpoisonous object. To solve this problem we need implement a classification neuron which can classify objects regardless of the distance. Thus, a new classification neuronal network will be created that takes normalized RGB values and classifies these values as either good or bad based on the frame in Fig 4. Similar to classification in architecture 3, the inputs are normalized RGB values instead of original RGB values. The output is 0 or 1. If the objects can increase the energy, the output is 1; otherwise, it is 0. Equation 1.2 is used to normalized RGB values in this architecture, where V is individual band of the light, R, G or B. In this architecture, we will train the new classification neuron using objects encountered by the robot. To randomize the objects encountered by the robot, we will change the head direction each time. The learning rate for the classification neuron is still 0.01 while the stopping criteria is when RMS change between two consecutive life time is less than 0.00000001 or the life time is greater than 300 to avoid infinite loop.

Normalized V = V/(R + G + B) (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 the light based on normalized RGB values. Therefore, for the object to eat, we can ignore the distance factor. It is known that the robot wants to move towards the direction where the objects are closest to it and are non-poisonous. Therefore, we need to implement a upgraded direction neuron, which can select the direction with highest light density and the light is classified as non-poisonous. Fig 5 shows two kinds of neurons we will implement for the robot. To make it easier, we will integrate new classification neuron with direction neuron to make a smarter direction neuron. Each time when the direction checks a light, the classification neuron will classify the neuron first. Then we will multiply the output by classification by the intensity output from first layer direction neuron. Therefore, if the light is classified as poisonous, the intensity is set as 0. So the light which smart direction neuron picks up has the highest intensity and also non-poisonous. Therefore, like architecture 5, each time the robot stops, classification neuron is activated to classify the objects encountered. If it is poisonous, the robot just skip it. Otherwise, the robot will eat it. Then smart direction neuron is activated to choose a direction to move. If there is no suitable direction made from 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

Fig 6 shows the shape of the robot we are controlling in the simulation world. When the robot stops, it will first use classification neuron to classify the objects it encounters. In the architecture, we use the middle light to classify objects, which is the beam in fig 6. However, when robot eat objects, it will eat objects on three sides. Therefore, a new problem comes. Are we sure that the objects on the other two sides are non-poisonous? If the objects on the other two sides are poisonous but the objects on side 0 are good. Then the robots will eat all objects, therefore, the robot is wasting energy on poisonous objects. So in this architecture, we will narrow down the mouse of the robot, so that it can only eat the objects on side 0, which can be classified directly by classification neuron.



Fig 6: Diagram of the new 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. So totally there are 5000 cycles of lifespan given initially by the world to the robot at birth. Therefore, we could see that the basic metabolic rate should be 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 and standard deviation of the lifespan with respect to the robot’s speed.

Figure 7 shows that the lifetime of robot is directly correlated with its speed. When robot speed is less than 0.1 and increasing, the lifetime of the robot has a nearly linear functional correlation with speed, where as speed increases, the lifetime is decreased. However, when the speed is over 0.1, the lifetime is constant. Therefore, the lifespan does not change anymore, even though the speed is increasing. What’s more, changing the head angle originating from the same origin has limited effect on the life span of the robot, as the life span, with constant speed, showed no change. Therefore, the standard deviation for the lifetime for the same speed with different head orientations is 0.

From this experiment, we could see that the movement metabolic rate is fixed when the movement rate is reached at 0.1. Therefore, in the following experiment for neuron trainings, we could fix the movement speed in order to get rid of the effect of speed on lifespan.

### 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 goes over 0.1, the mean lifetime is stabilized again like architecture 1. However, instead of getting standard deviation as 0, there is a standard deviation greater than 0 for this architecture because the robot can eat everything it encounters in the world, which is effecting its life either positively or negatively based on the standard deviations range around the trend line lifespan mean. Moreover, table 2 provides the basic line for our future neural network implementation. If neuron network could classify the objects correctly, theoretically the robot should survive longer than current robot for each speed.

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

| **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 |

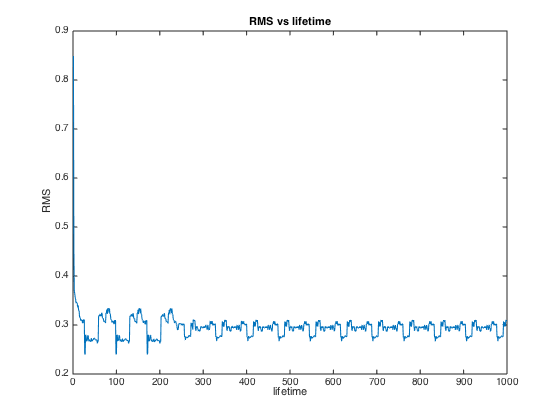
Fig 10 shows the training results for the classification neuron. We could see that at the beginning the RMS drops very quickly and dramatically. However, after a few cycles, the RMS values is stabilized but still tremble within 0.2 and 0.35. So we could see that the neuron is trained well after iterations. Table 3 shows the final weight for the trained neuron using RGB values. Weight 0 is used for bias in the calculation. Upon the observation of three weights, we could get a preliminary conclusion that the second bandwidth yields good food while the other two bandwidths are related with bad food. Then in the following experience, we will try to use these weights to classify the objects robot meets and decide if robot should eat the objects or not. Then we can verify our primary conclusion.

Fig 10: Direction decision neuron checking with a 5o 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 objects it has encountered.

### 3.5 Architecture 4: Neuronal Movement towards brightest object

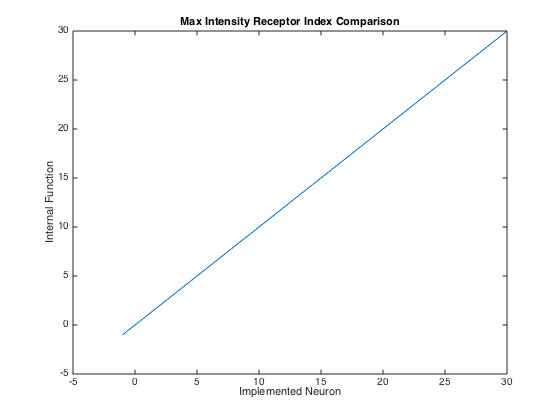
In section 2.5, we postulated a theory about neuronal directionality decision-making. We now need to verify the neuronal algorithms correctness. There is an internally implemented function, called intensity\_winner\_takes\_all, which returns the index of the receptor that has the highest intensity. We 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 result of this comparison. We can see that the neuron implemented here generates the same result as the internal functions. Therefore, our direction neuron yields the same result as the internally coded function and we can trust the direction picked up by 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 could activate both classification neuron and direction neuron to search and eat good foods in the world. After implementation, we could see how the lifetime changes.

| **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 |

By comparing table 4 and table 2, we could see that architecture 5 gains a significant improvement over the mean lifespan found in architecture 2. The improvement is attributed to the classification of food before eating, thereby helping the robot selectively choose only food that is classified as good by the LMS neuron. Like architecture, there is a significant standard deviation for the same speed. More interesting, the standard deviation is stabilized when the speed reaches 0.1, as shown in Fig12. Probably the max speed of the robot in the simulation world is 0.1. But this experiment proves directly that our new network architecture works well in classifying objects in the world.



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



Figure 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

| 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 |



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

Normalization of the data appears to not be detrimental to the overall performance of the classification of incoming light based on figure 14. The large oscillation is hypothesized to be from classification of good food in front of the robot while there is bad food to the sides of the robot, since the robot classified the food as good all the food around it gets eaten. This will be addressed as a fine-tuning to architecture 7 with architecture 8. Take note that a biased input was included in the classification of the objects. Besides, upon observation of the weights in table 5, we can see get the same conclusion that the second light bean contributes to the good food while the other two beams are related to bad food. What is more, it seems that the third beam has higher contribution to bad food than the first beam by comparing their weights. And the following architecture will use the new classification neuron and can prove our hypothesis.

| 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 |

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



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

By comparing table 6 and table 4, we could see that our architecture gains a big improvement in robot’s lifespan. The lifetime is almost 4 times as big as before. Therefore, it solids conclusion in architecture 6 that the second beam is related with good foods, while the other two beams are related with bad foods. By comparing fig 13 and fig 15, we could see that the best speed for robot to survive is changing if we change the architecture. From table 6, we can see that the best speed is 0.06, where the robot could survive for more than 30000 cycles in average for one life time.

### 3. 9 Architecture 8: Eat Few Objects based on Neuron Classification using RGB Percentage

| 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 |

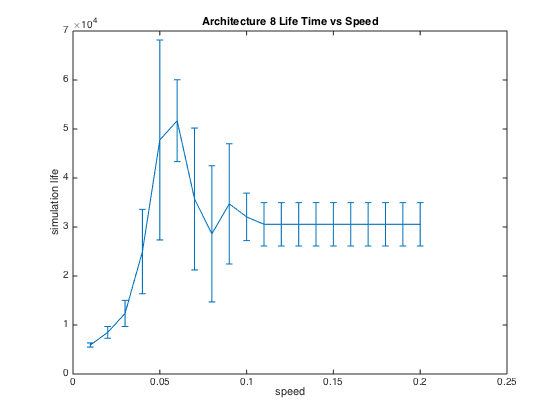


Figure 16: Mean lifespan trend line with standard deviation bars.

By comparing table 6 and table 7, we could see that our new architecture works well. At the best speed, our robot could survive more than 50000 cycles per life time. Therefore, we could conclude that objects on side 1 and 7 could not be classified directed by single beam. However, by classifying the correct objects, the robot could survive longer than other architectures.

# 4. Discussion

# 5. Summaries and Conclusion

# 6. Acknowledgements

Thank Thomas Caudell for the animation robot model and explaining how to implement LMS neuron to classify objects. Thank Fengshu Xu for sharing ideas with our group to implement each architecture.

# 7. References

ADDIN EN.REFLIST

1. Caudell, Thomas. "Flatworld."
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