

## **Abstract**

FORREST JR., CHARLES EDWARD. A Neural Network Control System for the Segway Robotic Mobility Platform. (Under the direction of Edward Grant).

An Artificial Neural Network (ANN) is a network of simple processing elements that emulate neurons in the brain. The behavior of such a network is characterized by the synaptic connections between the input data and the processing elements. Here, an ANN was generated and used as part of a control system for a Segway Robotic Mobility Platform (RMP) being trained in obstacle avoidance behavior. The single sensor input to the control system is a SICK laser, a range-finding sensor; the control output is Pulse Width Modulation commands to the RMP's motors. The Segway RMP, neural network maps input sensor data directly to appropriate motor output commands for obstacle avoidance.

Obstacle avoidance training was accomplished in a simulated LabView world using supervised reinforcement learning and practices from evolutionary robotics. Synaptic connection strengths were stored in an array called the artificial "chromosome". The chromosome was randomly modified, and the response of the network was compared to a pre-defined desired output. The goal of the genetic algorithm training was to minimize the error between the desired and actual outputs, yet to ensure that local minima were avoided. Once the ANN was trained in simulation, it was transferred to an actual RMP for obstacle avoidance testing in the real world

. The benefits of training ANN's for obstacle avoidance tasks in simulation are demonstrated here. In the simulated world, training and testing can be done in virtual environments: offering greater control over environment complexity, testing the robustness of the controllers generated, and filtering the training data set. All of the foregoing reduces the cost of training and lead to the development of an optimized ANN controller for RMP obstacle avoidance. The ANN provided input pattern generalization for smooth motion, improved computational speeds, and added to the body of knowledge for RMP controller development.

**A Neural Network Control System for the Segway Robotic  
Mobility Platform**

**By**

**Charles E. Forrest Jr.**

A thesis submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the degree Master of Science

**Computer Science**

**Raleigh**

**July 14, 2006**

**APPROVED BY:**

---

**David J. Thunte**

**Co-chair of Advisory Committee**

**John Muth**

**Co-chair of Advisory Committee**

---

**Edward Grant**

**Co-chair of Advisory Committee**

## **Biography**

Charles Edward Forrest Jr. is a graduate student at North Carolina State University in Raleigh, NC. Born in Greensboro, NC and raised in Charlotte, North Carolina. Charles Jr. is the son of Charles and Marilyn Forrest. He is a graduate of Myers Park H.S. in Charlotte, NC and a graduate of NC Central University in Durham, NC.

Through Charles' seven-year academic matriculation, he has become a scientist, a developer, a technician, and an engineer. His ability to think, compose, analyze, and solve problems has become unquestionable. Charles has received the highest praises for his contribution to the websites at North Carolina State University NC Central University alike.

He has proven himself an effective leader and coordinator as project manager in the IBM I-mentor-u program. He has served as a tutor and mentor for the university and in the community through the Durham-based Project Excellence program and the Raleigh Boys and Girls Club. Above all, Charles is a spirit filled and spirit led person, attributing all of his achievements and successes as services to God and humanity.

# Table of Contents

List of Figures .....	v
List of Equations .....	vii
<b>I Introduction .....</b>	<b>1</b>
<b>II Literature Review .....</b>	<b>4</b>
<b>III Platform .....</b>	<b>8</b>
Balance Sensor Assembly .....	8
Controller Boards .....	8
Control Processor .....	9
Communication .....	9
SICK Laser .....	10
<b>1. ROBOTIC SYSTEMS MODELING .....</b>	<b>11</b>
1.1 Virtual World .....	11
1.2 Virtual Sensor .....	13
1.3 Adjustable Sensor Resolution .....	17
<b>2. CONTROLLER OPERATION .....</b>	<b>22</b>
2.1 Rule Based Controller .....	22
2.2 Neural Network .....	25
2.3 System Performance .....	27
2.4 Training Worlds .....	28
<b>3. NEURAL NETWORK TRAINING .....</b>	<b>31</b>
3.1 Organization .....	31
3.2 Training .....	33
Time-Step Independent Training .....	34
Time-Step Dependent Training .....	34
3.3 Genetic Algorithm .....	36
<b>4. TESTING AND RESULTS .....</b>	<b>38</b>
4.1 Practice Worlds .....	38
System/Network Performance .....	41
4.2 Performance of Neural Network Training .....	42

<b>4.3</b>	<b>REAL-WORLD TESTING .....</b>	<b>44</b>
<b>5.</b>	<b>CONCLUSION.....</b>	<b>47</b>
5.1	Closing Remarks .....	47
5.2	Future Research .....	48
<b>6.</b>	<b>BIBLIOGRAPHY .....</b>	<b>49</b>
	<b>APPENDICES .....</b>	<b>51</b>
A.1:	Actuator Assembly .....	54
A.2	Math .....	56
A.3	Neural Network.....	57
A.4	Rule-Based Control System.....	58
A.5	Miscellaneous Tools .....	59
A.6	Training .....	61
A.7	Virtual Sensor System.....	66
A.8	Available World Segments .....	69

## List of Figures

Figure 1: Seattle Robothon for a Balancing Robot Symposium.....	6
Figure 2: Two-Dimensional Virtual World - UCF Frame .....	12
Figure 3: The Robot Coordinate Frame .....	15
Figure 4: Obstacle Position Beam.....	16
Figure 5: Single Segment VLMS Range Fan .....	17
Figure 6: Adjustable Sensor Resolution (in meters).....	18
Figure 7: Virtual Actuator Geometry.....	19
Figure 8: Corner Avoidance Pseudo Code.....	23
Figure 9: Rule-Based Controller Geometry .....	24
Figure 10: Single-Layer Neural Network .....	25
Figure 11: Graph of Sigmoid Activation Function – Equation 12.....	26
Figure 12: Example Training Worlds .....	29
Figure 13: Chromosome Evolution: After 2 mutations, After 145 mutations .....	34
Figure 14: Local Minima .....	35
Figure 15: Genetic Algorithm - Chromosome Breeding Process .....	36
Figure 16: Genetic Algorithm – Resulting Gene Distribution.....	37
Figure 17: No Obstacle Cartoon Clip .....	38
Figure 18: Single Segment – Testing Deceleration .....	39
Figure 19: Single Segment - Concurrent Motor Speeds .....	40
Figure 20: Box Enclosure and Cone Left Open Practice Worlds .....	41
Figure 21: Training Performance over time.....	42
Figure 22: Cones Training .....	43

Figure 23: Segway Real-World Testing Environment..... 45

Figure 24: SICK LMS Range Fan ..... 46

Figure 25: RMP Reaction to obstacle within Forward Distance Threshold ..... 46

## **List of Equations**

Equation 1: Total Distance Travelled .....	20
Equation 2: Resulting Change in Orientation .....	20
Equation 3: Width of Robot Wheel Base.....	20
Equation 4: Direction of Travel .....	20
Equation 5: Radius of Circular Path Traveled .....	20
Equation 6: Chord – Absolute Distance Traveled .....	22
Equation 7 Left and Right Motor Velocities.....	22
Equation 8: LMS/VLMS Angular Distance and Angular Displacement.....	23
Equation 9: LMS/VLMS Current Input Angle .....	24
Equation 10: Advanced Rule-Based Controller: Forward and Side Distance .....	24
Equation 11: Motor Output from Neural Network .....	25
Equation 12: Neuron Activation Function .....	26
Equation 13: Chromosome Size.....	31
Equation 14: Chromosome Update .....	33
Equation 15: Performance Cost in Number of Transformations .....	42



## I Introduction

Autonomous robots, like the Segway Robotic Mobility Platform (RMP), are equipped with sensing devices that provide information about the world in which the RMP is operating. The control system is the software that processes that sensor information to make decisions concerning the robot's locomotion. The software developed for controlling the RMP is described further in *section III: Platform*. Our control system is reactive, meaning the output is a function of the sensor inputs only; it is open loop, because obstacle avoidance requires no information about past actions or the state of the robot. We describe sensor input as the smallest unit of independent information retrieved from a sensor – e.g., a single distance measurement retrieved from a radar system. Input data can be obtained from numerous sensing devices individually, or it can be the integration of the data sensed from many devices. Each input may have varying degrees of confidence, or importance, and may be independent of the sensing device from which the data was retrieved. The control system considers all inputs and provides a predictable response for the robot.

Two types of control system were designed and tested: a rule-based system, and a neural network. A well-written rule-based system performs effectively if it is given detailed information as its input. However, to derive a rule-based system that would be effective for all combinations of sensor inputs would be infinitely complex. At minimum, the resulting decision table would be computationally intensive. The objective was to introduce the neural network to a minimum set of input patterns and have it generalize an appropriate control action. Ideally, this behavior would replicate the response from the rule-based system under the same conditions. Then, we let the neural network replicate

this behavior in cases outside the training set. By doing so, we substitute the computationally intensive rule-based system for a neural network whose independent processing elements are inherently parallelizable. [15]

In 2002, a similar approach to developing a robot control system was researched at North Carolina State University (NC State) [1]. Our objective was to replicate the EvBot research approach at NC State for use on the Segway RMP. As both robots are two-wheeled and rely on skid steering, it was a logical starting point for the design of the RMP control system.

The neural network was trained using supervised learning [2]. In this process, a performance metric determines how well the training of the neural network imitates a path following task. The desired path could be retrieved from another algorithm written for the task, one that was entered manually into the system, or even retrieved from the joystick control actions of a human pilot.

The control system rules were derived to be:

1. IF the sensor detects no obstacle in view, THEN the robot must move forward at a user defined maximum speed.
2. IF in the presence of an obstacle, THEN the robot must slow down and turn to avoid the obstacle.
3. IF the robot must pass between objects, THEN the robot must remain centered between the objects.

Two rule-based control systems were developed to comply with the above requirements, and these were used to train the ANN. The two algorithms differ in their ability to avoid small obstacles and in their ability to train the ANN. These issues are

discussed further in *Chapter 2: Controller Operation*.

A variety of training methods were used to minimize the difference in performance between the rule-based algorithms and the ANN response. Generally, the training process involved gradually modifying the parameters that describe the ANN until a desired performance criterion was met. In doing so, the robots cognitive responses are manipulated incrementally until they match a desired behavioral pattern. Here, the ANN was being trained for obstacle avoidance. The training method is detailed in *Chapter 3: Neural Network Organization and Training*.

Once trained, the mature neural network was tested in simulation to determine if it performed similarly to the rule-based control system that was the basis of its design. These experiments confirmed that the neural network was generalizing unseen environments into familiar patterns from the training set. The control system was tested in a number of virtual practice worlds. The resulting motor speeds as a function of time were continuous, revealing smooth motor responses. More detail on the construction of these worlds and the performance of the neural network can be found in *Chapter 4: Testing and Results*.

## II Literature Review

The Segway Robotic Mobility Platform (RMP) is a balancing robot, successor to the Segway Human Transport vehicle. The first RMP's were developed and demonstrated by Segway at the MARS program review meeting in San Diego, CA in April of 2003. The Army Research Laboratory, along with other research institution across the country, acquired the Segway RMP on loan from the Defense Advanced Research Projects Agency (DARPA) to research feasible solutions for ground transport applications.

Before our research began, the Segway RMP at ARL was controlled via human-in-the-loop process involving input from a joystick, so the user was directly involved in the control loop. The system was not autonomous. Hence, the first objective was that of creating a simple rule-based system for autonomous control of the robot. Later, a decision was made to substitute any rule-based system developed with a trained neural network controller.

A performance evaluation at the University of Pennsylvania assessed the RMP's stability in motion. They identified a 0.4-second delay in appropriate motor activity due to the interference of the dynamic stabilization process. The longest delays, and the biggest risk to stabilization, occurred when the RMP was descending from an obstacle and operating on a low-friction surface [11]. The test platform used in this work was indoor-controlled; it minimizes the work of the dynamic stabilization process thereby increasing response time.

The artificial neural network, applied in this research as a control system for the RMP, is the brainchild of Warren Mculloch and Walter Pitts. They conceived of a network of weighted inputs whose sum would need to reach a particular threshold to

‘activate’ a switch [12]. The changing weight on the input connections, or synapses, emulates the reinforcement learning done in the brain. The switch mechanism is provided by an activation function – typically of sigmoid shape.

Research done by Minsky and Papert in 1969 revealed the limitation of a single-layer neural network (perceptron) to linearly separable patterns of input. Their research implies that a single-layer neural network should be capable of learning object avoidance behavior if and only if it can be described as a series of linearly separable arguments. By linearly separable, they meant that the input/output function must be one to one. Input patterns with similar outputs should not be separated.

A linear function can map the sum of distance measurement inputs to velocity to control acceleration in the presence of an obstacle. Two linear functions were used to control the acceleration of two motors independently; which gave a rudimentary obstacle avoidance behavior.

Corner avoidance behavior was achieved using a non-linear one-to-one function; this affects one motor only at a given input threshold. This makes it linearly separable. High input sums correspond to a negligible effect on the left motor output. Lower sums produce a negative output to reduce the speed of one motor.

In the course of our research, a more sophisticated rule-based system was developed to prevent collision with small objects. Through similar reasoning, behavior can be tokenized into a number of linearly separable patterns that are equal to the number of inputs plus one. Each input has a threshold where it influences the change of a single particular motor command. The extra token is used to move the Segway relative to the sum of all the inputs.

A neural network-based obstacle avoidance routine was designed by Andrew Nelson for the EV-bot at North Carolina State University in 2002. In this system, the input devices were tactile sensors evenly spaced like whiskers in front of the robot. Although it is not a balancing robot, the EV-Bot has a similar two-wheeled design. It was also trained from a rule-based system like the simple one described in *Section 2.1: Rule-Based Controller* [1].

The Segway RMP is a balancing robot that utilizes a proprietary balancing algorithm called dynamic stabilization. There are many types of balancing robots, for example the [Legway](#), [nBot](#), [Gyrobot](#), [Isis](#), and Bender shown in Figure 1.



**Figure 1: Seattle Robothon for a Balancing Robot Symposium**  
**From left to right: Legway, nBot, Gyrobot, Isis, and Bender.**

In 1989, Grant and Zhang created a balancing system that used ANN's for control. They designed a three-hidden layer perceptron capable of balancing a pole over a movable cart in two-dimensions. In this case, the training algorithm was developed using a simulator and manual joystick manipulation. This approach allowed a rule-based algorithm to be derived, because the simulator could be slowed down to compensate for slow human responses. A control law, or rule based control system was written by

observing and interpreting the user's interaction. This was compared to the behavior elicited from the user by the neural network. The two were shown to be similar [13].

Prior research on balancing robots using neural networks for obstacle avoidance tasks is described in [3]. The objective of this research is to apply similar concepts to develop an ANN-based control system for balancing a Segway RMP that is capable of obstacle avoidance. The research focuses on the necessary training interfaces and virtual environments.

### **III Platform**

The Segway Robotic Mobility Platform (RMP 200) is a two-wheeled robot designed to transport heavy materials in tight spaces. The RMP has two five horsepower motors powered by one to four 52-volt batteries. It has a payload capacity of 200lb, and utilizes skid steering to attain a zero turn radius [3].

#### **Balance Sensor Assembly**

The robot's Balance Sensor Assembly is a complex system of gyroscopes that provide data on the RMP's current pitch and pitch rate. When the RMP is tilted forward, the stabilizing algorithm drives the wheels forward with an appropriate acceleration to keep the RMP balanced. Segway designers coined the balancing algorithms "dynamic stabilization" [4]. The details of this process are beyond the scope of this research. It is important to remember that the fundamental movements: i.e., drive forward, drive reverse, and stop, all rely on this balancing system. To drive forward, the RMP tips forward at a specified angle for a specific duration, relying on the balancing algorithm to adjust the wheels speeds gradually accelerating so that balance is restored after traveling some specific distance. When the RMP reaches its maximum speed, it controls its angle of tilt by exerting a force in the opposite direction to its current motion.

#### **Controller Boards**

The RMP is equipped with dual redundant controller boards for safety and dependability. Both boards have digital signal processors responsible for monitoring the complete system for faults. The control board scans for an interrupt every ten milliseconds. At the same frequency, the controller polls the Balance Sensor Assembly to calculate the necessary motor speed adjustments to balance the robot. Motor adjustments



are made every millisecond to insure a smooth handling response [5].

### **Control Processor**

The Segway RMP is a powerful, highly maneuverable, self-balancing hand truck. It is successor to Segway's human transport model (HT). The Segway HT is a user controlled balancing scooter. The RMP model, however, can function as an autonomous robot. The point of difference between the HT and the RMP is the RMP's control processor. The object of the control processor is to issue speed and steering commands that can be interpreted by the RMP's onboard controller and translated into the motor commands. These commands move the robot while maintaining its upright balance. The control system of choice for the RMP was an IBM Think Pad, which was strapped to the RMP's top plate.

### **Communication**

The controller connects to the RMP via Controller Area Network (CAN) bus. Communication is configured to 100 Hz and messages are sent via two channels. Each channel can send only one message at a time, and that message is duplicated on the second channel for safety. If communication is not redundant, the RMP stops and balances. The message protocols break down the CAN messages into seven fields: Source, Destination, Header, Velocity, Turn-Rate, Status, and Status Parameters. The Status and Status Parameters were never used during the course of the research. Those fields deal with setting maximum velocity and other such default behaviors that were non-essential for direct motor control [6].

This is important because the neural controller issues speed commands for the individual motors. Later these instructions are translated to absolute velocity and turn-

rate. Essentially, skid or differential steering instructions are translated into a motion vector. Velocity and Turn-Rate calculations are discussed further in *Chapter 1: Robotics Systems Modeling*.

### **SICK Laser**

Along with the control system, a laser measurement tool called the SICK LMS 200 was mounted to the front of the RMP robot. The LMS is an optical measurement tool with a range of 80 meters for surfaces with high reflectivity. Our environment was indoor uncontrolled, so the LMS was expected to visualize a variety of surfaces, e.g., plastic cones, sheet metal from the garage, polished wood from the desks, even clothes as humans often moved objects in the RMP world during experiments. The documentation suggests the LMS range deteriorates to 10 meters for objects with low reflectivity. To accommodate for these surfaces, the control system only responds to a viewing radius of 8 meters. The maximum velocity of the Segway RMP is 8miles/hour or 3.6 meters/second. The LMS has a response time of 53ms, so theoretically the RMP will recognize inanimate objects from a distance of 7.81 meters. We did not have to account for dust, fog, or other outdoor conditions [6] [7].

The LMS operates by transmitting a pulsed laser beam. The beam is reflected with a .25 angular displacement over a 180-degree span. High resolution is difficult to process. The neural-network takes much longer to train with 720 inputs. Typically the train was cut down to 39 evenly displaced input measurements, which meant that at a maximum range of 8 meters the resolution would detect straight objects of 0.6 meters in length. At the forward distance threshold of one meter, the LMS will detect objects less than 8 centimeters in size [7].

## **1. Robotic Systems Modeling**

The Segway RMP has three main systems. The SICK Laser Measurement Sensor receives distance measurements from the environment. The control processor is the hardware that houses the control system and computes motor commands relevant to some appropriate behavior. Finally, the hardware controller boards (actuator) instruct the motors to spin the wheels at the appropriate rate.

It is not necessary that the sensor and actuator systems be modeled in simulation for the training of the controller to be successful. A reasonable alternative would be to retrieve actual input data from the LMS and drive the RMP itself using the rule-based control system. However, there are significant advantages in doing so because the sensor system can be positioned virtually, in fixed or random positions, with respect to obstacles. Consequently, specific patterns can be identified and used as input training patterns for the system. Also at this stage, duplicate patterns can be filtered out thereby to improving training performance. By rotating the virtual sensor perspective, specific input patterns can be obtained allowing the user more control over the training process.

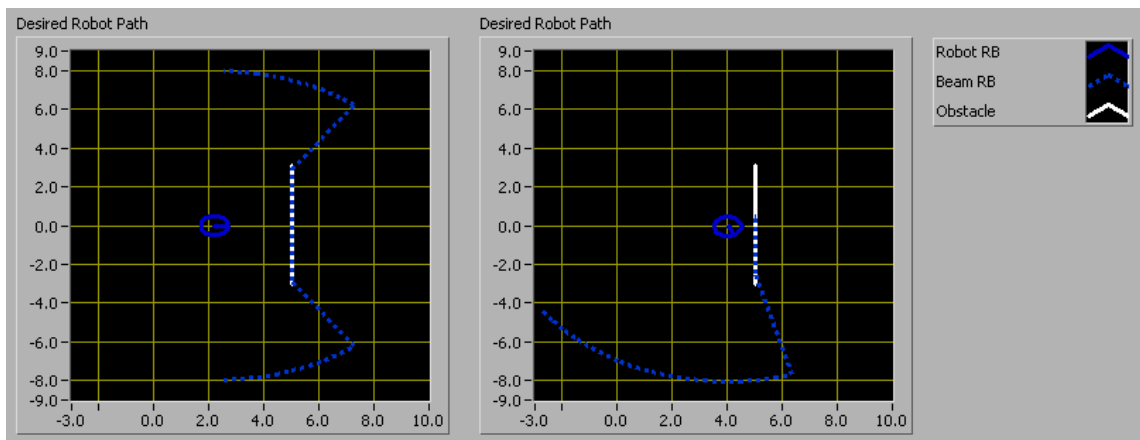
After motor commands are supplied by the control system, a virtual actuator calculates a new position in a fraction of the time required to move the actual robot. This system requires the translation of skid steering instructions to a motion vector. The motion vector is applied to the current position to calculate the new position. The process of determining the motion vector is shared. It is also required to communicate with the RMP's motor controllers. (*See Section III: Platform*).

### **1.1 Virtual World**

The SICK LMS input device works in two-dimensions to detect obstacles ranging

90 degrees in each direction from the center of the device. Naturally, our virtual world model is also in two-dimensions. The robot along with all other objects is represented in a single Cartesian plane.

The view is a two-dimensional aerial perspective of the world. A graphical user interface displays the world by drawing object coordinates. In Figure 2, the width and length of the RMP, the scan area of our input device, and other obstacles in the virtual world are shown. Motion and direction of the robot and its interaction with the obstacles can be seen. Through observation, it can be determined if the virtual robot is learning a desired behavior.



**Figure 2: Two-Dimensional Virtual World - UCF Frame**

By simulating the Robot in two-dimensions only, it should be noted that information about the height of objects in the world is missing. To make this an acceptable handicap, it was insured that the objects in the robot's world were visualized at the height of our LMS, which is approximately 1 meter from the ground. The robot was not tested to navigate slopes or uneven terrain. Generally, it was assumed that the control system should not need to know about changes in elevation, wobble, or tilt to

make decisions. Dynamic Stabilization makes this a difficult. Often the RMP will tilt slightly during operation causing the LMS to see over the cone testing obstacles. This causes the control system to provide widely disparate motor speeds from one interval to the next.

Obstacles in the virtual world were represented as concatenated line segments. Virtual worlds were described using a  $2M \times 2$  matrix of segment points - where M is the number of segments. Each point is stored sequentially in a text file and read into an array before being drawn. The segments themselves are not delimited, so adjacent segments cannot be represented by three points. Each segment has two entries a start and a destination.

The advantage of describing the world in this fashion is the ability to model virtual, high resolution, complex worlds. The disadvantage is that a significant amount of information is needed to describe simple objects. Circular objects could require an inexhaustible amount of computation to represent with precision. This disadvantage was overcome by creating simple training worlds with a few simple objects. Further detail on training worlds can be found in *Chapter 2: Controller Operation*.

## **1.2 Virtual Sensor**

The SICK LMS was replaced with a virtual range finding component. This virtual LMS (VLMS) software emulates the SICK by calculating distance inputs trigonometrically. The number of inputs is entered by the user, and the system calculates the distance from the sensor location to whichever obstacles are within range (8 meters) of the sensor. Otherwise, the maximum range is returned.

Obstacles in the environment are represented as line segments. Information about

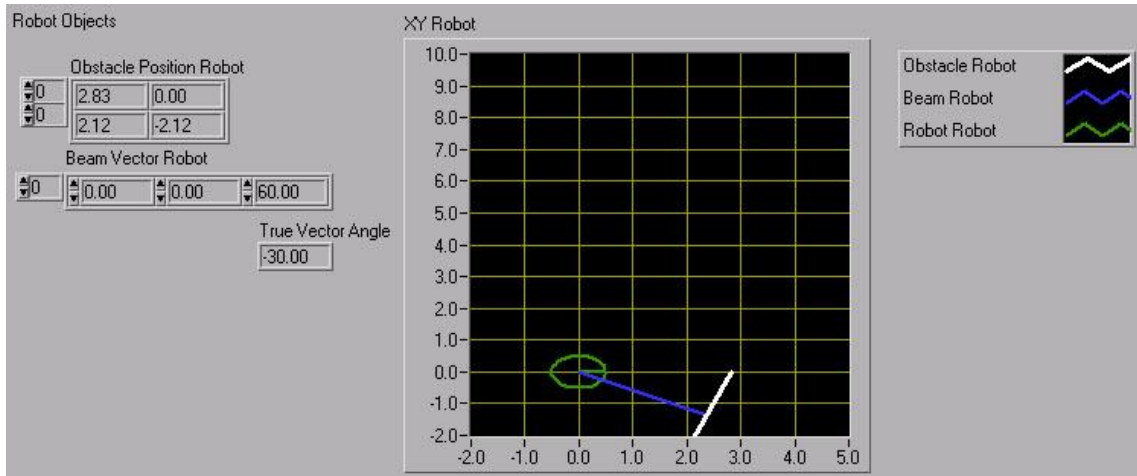
the slope, intersect, magnitude, and position of each line is calculated to determine if any objects are in view of the sensor. In our distance-measuring algorithm, every obstacle segment is translated twice – first with respect to the robot, then with respect to the input trajectory. This was done in three steps:

1. Translate each object from the Universal Coordinate Frame to the Beam Frame.
2. Determine the X-axis intercept by deducing the line from the line segment endpoints.
3. Determine the object with the shortest distance from the robot.

In the beam frame, the robot and obstacle segments are transposed and revolved with respect to the some directional input. The value of this input is the calculated length of the line-segment between the sensor and any obstacle in the input direction.

Consequently, if any obstacle segment has an X-intercept, which lies in the viewing range of the VLMS, then our object is within detectable distance. Furthermore, if the object is detectable, the X-intercept is the distance from the VLMS to the object.

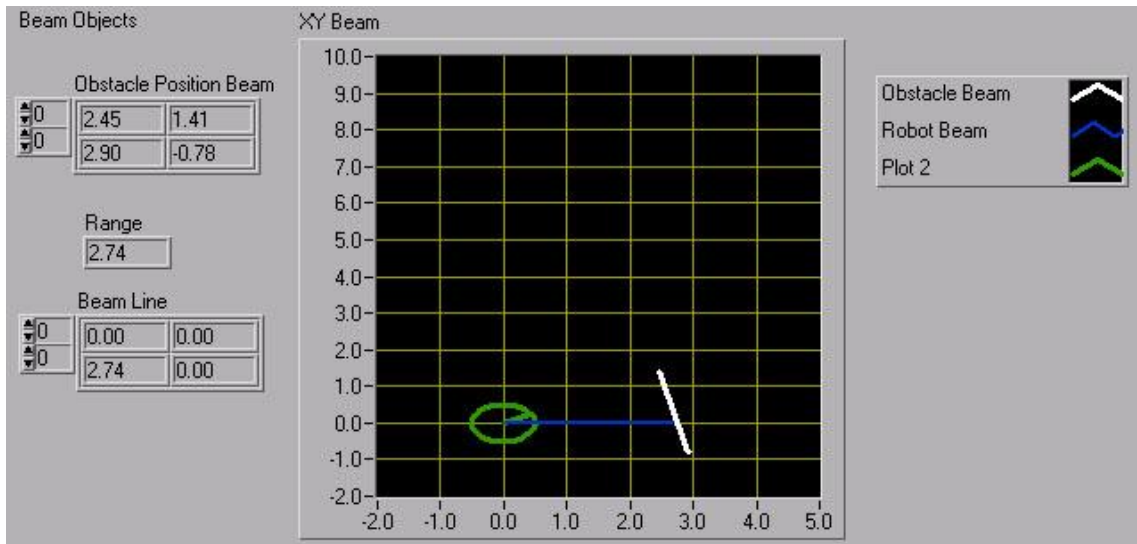
The sensor calculations and the rendering of the segments take place simultaneously for performance reasons. Since we need to manipulate the segment positions graphically, we save a step, go ahead and determine the sensor values.



**Figure 3: The Robot Coordinate Frame**

In the introduction to this section, objects are seen by the user in a single frame termed the Universal Coordinate Frame. In this frame, information about the location of objects and the robot is maintained. The viewing range of the VLMS can also be visualized by representing each beam as an object on the UCF coordinate plane. The sensor itself is not visible in the user interface. The general location can be perceived by viewing the beam origin. The orientation can be perceived by looking at the sensor range fan shown in Figure 2. In the real world, the sensor is attached to the body of the robot. Sensor location is adjustable, but it is static parameter recorded with respect to the robot. The VLMS is adjustable on the simulated robot just as in the real situation. Input from the LMS is then translated to UCF using robot position information.

Geometric transformations are used to translate objects from UCF to frame views with respect to the robot and vice versa. In the robot frame shown in Figure 3, the robot becomes the center of our world and each object is translated accordingly. The robot icon is a circle with a line drawn from the center to the perimeter. The line indicates the front of the robot. The LMS sensor sits there by default.



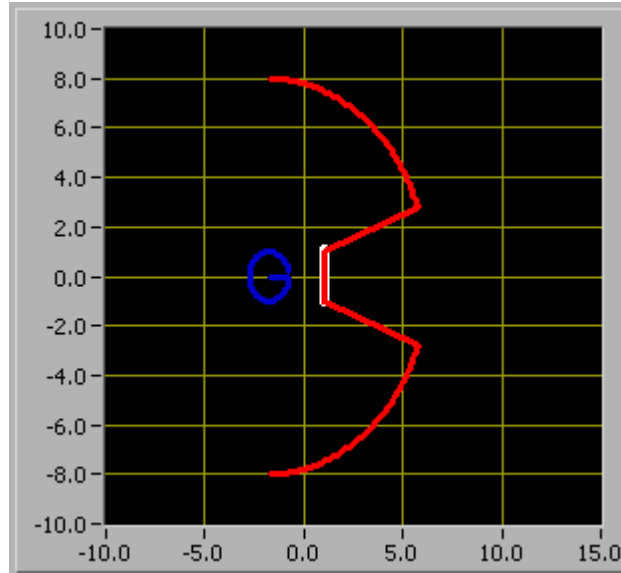
**Figure 4: Obstacle Position Beam**

To make the simulation more robust, it is not assumed that the virtual sensor will be mounted to the front of the robot. Another transformation is performed to image the robot and the obstacles with respect to the sensor inputs. The sensor has a range of input originating from the same location, but projecting at various angles. The world is rotated with respect to every input in the sensor system. In Figure 4, the robot and obstacles are transformed with respect to the single input beam. From this view, it is easy to determine the distance between the VLMS and the obstacle by finding the x-intercept of the obstacle segment. There are a few exceptions. Since we are dealing with line-segments rather than lines, we have to determine if the line-segment ever crosses the X-axis to begin with. As well, vertical line-segments that are not functions have to be approximated. As the slope approaches infinity, the value of X is the X intercept we are looking for.

The LMS can technically be mounted to any portion of the robot, and in any direction. In fact, the VLMS has built in flexibility to allow testing in that faculty. The



LMS was mounted on the front of the top-plate on our Segway RMP. This was a convenient location allowing room for the control processor to be mounted on the rear of the top-plate. Similarly, the VLMS was mounted to the front of the virtual robot.



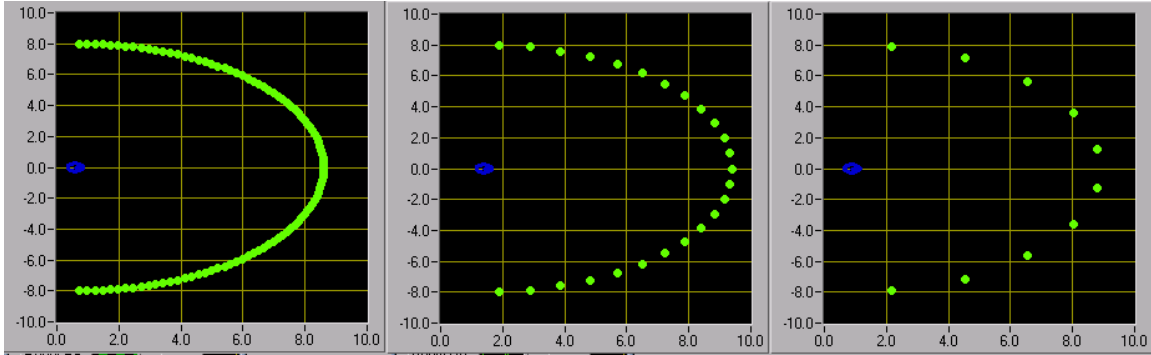
**Figure 5: Single Segment VLMS Range Fan**

The LMS has a 180-degree field of vision, with a half-degree resolution, and a maximum viewing range of 8 meters. Therefore, the LMS eye spans a two dimensional space with an area of  $32\pi$ . The virtual sensor implemented in this work emulates the LMS mounted to the Segway RMP

### **1.3 Adjustable Sensor Resolution**

The VLMS is capable of reducing the number of input measurements calculated. We may do this to reduce performance cost, or to improve the performance of the neural network. From the simulation interface, the user is asked to provide the number of inputs. The number of inputs to the VLMS is limitless; however having more than 360 would be a waste of processing performance. The SICK LMS that we are modeling has a maximum resolution of 360 inputs. When fewer inputs are requested, the input stream on the SICK is filtered to show only the requested quantity of input range values evenly

distributed over the view space. Consider Figure 6 below. It is quite conceivable that a lower resolution can be just as effective as the higher resolution depending on the size of the obstacles in our environment.



**Figure 6: Adjustable Sensor Resolution (in meters)**

**Input Resolution [100 inputs, 25 s, 10 Beams]**

#### 1.4 The Actuator Group

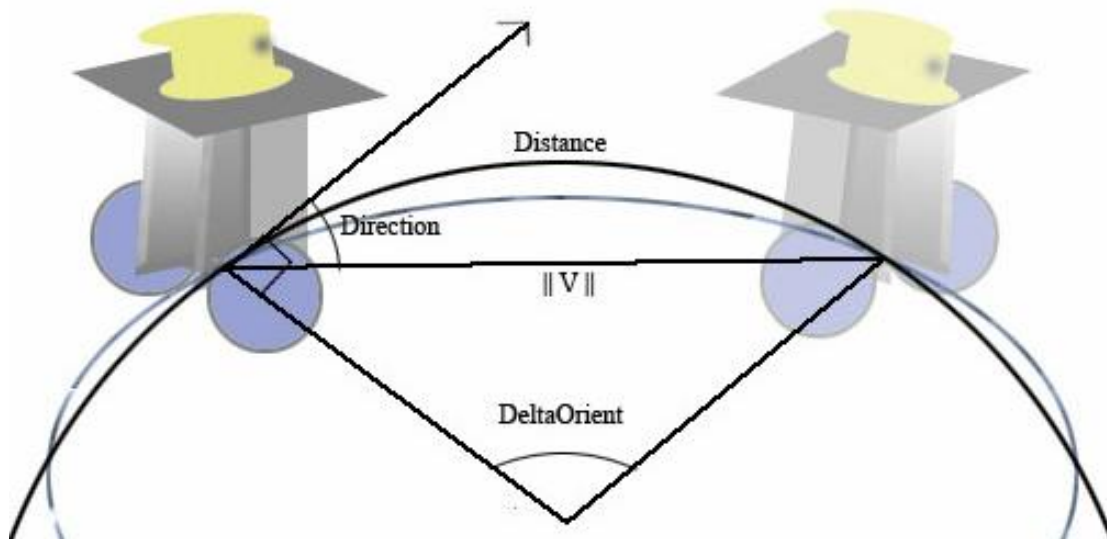
At each simulation time step, the following actions are performed:

1. The virtual sensor readings are updated.
2. The rule-based controller and/or neural network determine the appropriate wheel speed for each motor on the RMP.
3. The robot's next position is calculated as a function of its current position, orientation, the time step size, and the current wheel motor rates

This is nearly the same process used by Nelson [1], excluding a fourth step – validation. Validation prevents a collision using information the robot could not know. A crash incident is recorded and the network is retrained accordingly. In a real scenario, it is quite possible for a mal-functioning control system to drive the RMP into an invalid region. If that region is a wall or obstacle, the robot will crash. If validation is required, the neural network is producing erroneous values that cannot be validated in a real

scenario anyway. If the crash occurs during training, performance error should be high. Otherwise, the training algorithm is ineffective for the training set. The rule-based system is crashing too. If the crash occurs after training, the neural network is immature and needs to be retrained. Either way, let it crash. In simulation, it can only help to see the robot crash and burn. Insight may be acquired through how it crashes into the obstacle.

The 'Robot Vector' is a data structure that holds information concerning the current position and orientation of the robot at any point in our simulation. As the motors are powered in our virtual world, we calculate the distance and direction traveled by the robot geometrically. From this information, we can determine the distance traveled by our robot at a given time-step.



**Figure 7: Virtual Actuator Geometry**

The actuator group translates motor instructions into a translation vector. The translation vector contains information about the magnitude and direction of travel over the relevant time interval. We have to use that to relocate the virtual robot as described in Figure 7: Virtual Actuator Geometry. We calculate the total distance traveled by

averaging the two motor speeds and multiplying by the change in time.

$$\text{Distance} = (\text{RMV}/\text{LMV}) * dt \quad \text{Equation 1}$$

RMV stands for Right Motor Velocity and LMV stands for the Left Motor Velocity. If the RMV is not equal to the LMV, the robot will move in an arc with a constant turn rate. The derivative of turn rate would be the total change in orientation with respect to the time interval.

$$\text{Delta Orient} = \text{Abs} (\text{RMV} * dt - \text{LMV} * dt) / \text{WB} \quad \text{Equation 2}$$

With a linear turn-rate, the robot will either follow a circular path or spiral path. Generally, we determine that our path is circular if and only if the width of the wheelbase can be derived from the wheel travel distances and the change in robot orientation using the following formulae:

$$\text{Width} = (\text{Outer Arc Length} - \text{Inner Arc Length}) / \text{Delta Orient} \quad \text{Equation 3}$$

A spiral path is like a circular path whose radius changes over time. If the radius gets smaller, you have an inward spiral. If the radius gets larger, you have an outward spiral. The angle of orientation is small for a given time step, so the change in radius is very small. For simplicity, we can conclude that the radii are the same and the robot follows some circular path during the time-step. Making this assumption, the direction of our robot's movement is calculated as half the change in orientation.

$$\text{Direction} = 90 - (90 - (\text{Delta Orient}/2)) = \text{Delta Orient}/2 \quad \text{Equation 4}$$

$$\text{Path Radius} = \text{Arc Length} / \text{Delta Orient} \quad \text{Equation 5}$$

$$\text{Chord} = 2 \cos (\text{DeltaOrient}/2) * \text{Path Radius}$$

**Equation 6**

The magnitude of the translation vector is the distance between the start and destination position. We use the Arc Length and the Orientation Angle to find it. The vector V is the motion vector whose magnitude is the length of the Chord.

## 2. Controller Operation

### 2.1 Rule Based Controller

A rule-based control (RBC) system determines desired wheel speeds by applying a set of rules or predefined transformation functions to the inputs. The sensor system provides the RBC with data about the environment, and a desired maximum speed is built into the program. The maximum speed can be set by the user up to the maximum speed of the robot. By default, the maximum speed is set to 2 m/s. As the sum of the inputs approaches the maximum range of the virtual sensor (8 meters), the motor outputs proportionately reach the maximum speed.

Basic obstacle avoidance can be accomplished by adhering to two rules. Consider that the LMS provides symmetric distance information in a 180-degree span as shown in **image 4**. First, separate the inputs values into two equal segments. Then set the motor velocities proportional to the sum of the inputs on the opposite side [1].

$$\begin{aligned} \text{LMV} &= (\text{RSR} / \text{Max}(\text{RSR})) * 2 \\ \text{RMV} &= (\text{LSR} / \text{Max}(\text{LSR})) * 2 \end{aligned} \qquad \text{Equation 7}$$

LMV and RMV are the left and right motor velocities; LSR and RSR are the left and right sensor readings respectively. This is very basic obstacle avoidance. It works well to avoid walls and other obstacles.

A control system built on this rule alone is insufficient when the sum of the inputs on each side is equivalent and decreasing. This can happen if, for example, the robot is heading into a corner. In this situation, the controller is likely to continue straight forward and slow to a stop. The second rule was implemented to counter this problem. If the total of the sensor readings is less than some threshold, turn around [1].

```
If (LSR+RSR<Threshold)
  RMV = -LMV;
```

### Figure 8: Corner Avoidance Pseudo Code

A system based solely on these two rules was effective to train obstacle avoidance with many obstacles. Still, the training scenarios were insufficient to create a basis for the problem space. The above method does not account for the width of the robot. If the robot sees an opening, it will attempt to pass through it whether it is too big or not. Suppose for example, our RMP decides to navigate between cones. The rule-based system would not be able to determine if it could fit between the cones, or if they should be avoided. A third rule we created using a bit of geometry to replace the second. We simply determine if any obstacle is too close to the robot front or side as information is provided from the sensors.

First, we determine the angular distance between the sensor input beams. There can be a number of inputs from 2 - 360 representing 90 degrees to a half-degree resolution. Since we know that the input is evenly spaced, the angular difference can be determined using the number of elements in our input vector. The LMS viewing area is (-90 degrees, 90 degrees) with respect to the robot. The displacement is the angle of the very first input. It is equal to (-90) degrees plus half of the angular distance. Let an  $m \times 1$  vector  $S$  of angles represent filtered input readings from the LMS. Where  $2 < m < 360$

$$\text{AngleDist} = 180 / m$$

**Equation 8**

$$\text{AngleDisp} = (\text{AngleDist}) / 2 - 90$$

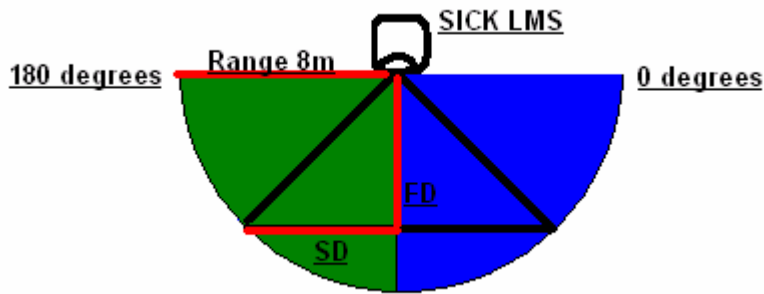
Once we have the displacement angle, we iterate through the inputs and determine the angular distance from the front of the robot to each input. From right to left the beam angles are a scalar multiple of the Angular Distance added to the angular displacement.

$$\text{InputAngle}[n] = S[n] * \text{AngleDist} + \text{AngleDisp} \quad || \quad n = \{0,1,2\dots m-1\} \quad \text{Equation 9}$$

After calculating the Input angle, the input value can be used to determine the distance of the object from the center of the LMS. Trigonometry tells us that the forward and side distance from the sensor to the robot can be determined by the following functions:

$$\text{FD}[n] = \cos(\text{InputAngle}[n]) * \text{Range} \quad \text{Equation 10}$$

$$\text{SD}[n] = \sin(\text{InputAngle}[n]) * \text{Range}$$



**Figure 9: Rule-Based Controller Geometry**

‘SD’ is the side distance from the sensor to the object in question. ‘FD’ is the forward distance from the sensor to the object in question. If the forward distance becomes less than our predetermined threshold, we are approaching an obstacle. We may or may not need to turn around. Only if the side distance is also less than half the robot's width can we conclude that our robot will not pass adjacent to the obstacle. If this is true, the robot applies the negative of the left speed command to the right motor. This turns the robot around without slowing down. The virtual system can comply simply enough, but without applying some principles of acceleration, the RMP may be unable to comply, or may respond erratically. We assume that such an action takes place at slow speeds; otherwise, inertia could be a problem.

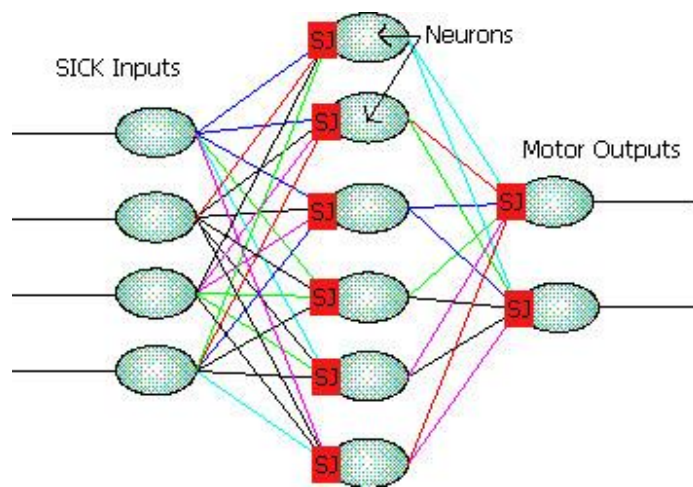


## 2.2 Neural Network

The focus of our research was the implementation of a single-layer neural network as the control system for our Segway RMP. A mature neural network should behave similarly to the rule-based control system when presented with similar input patterns.

The network structure has five components:

1. Sensor Inputs
2. Synaptic Connections
3. Summing Junctions
4. Neuron Activation Functions  
(Neurons)
5. Motor/Actuator Outputs



**Figure 10:** Single-Layer Neural Network

Sensor inputs are processed by a class we call the Neural Core. It provides the input-output mapping between the Sensor Inputs and the Motor Outputs. The network output can be described as a composite function of inputs.

$$\text{M.O.} = F(\text{Inputs}) = \text{Sum}(\text{NAF}(\text{SUM}(\text{Inputs})))$$

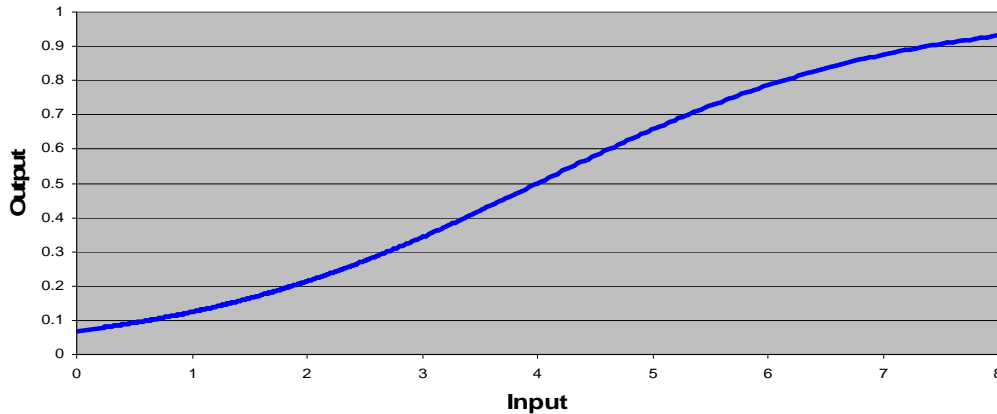
**Equation 11**

Motor output is the result of applying a linear activation function to the weighted output of the neuron activation function. The neuron activation function is a sigmoid that normalizes the weighted input from each of the sensors to a value between zero and one. The input values are combined with varying confidence by the summing junctions as they

enter the virtual neurons. The neuron activation function is a sigmoid described by the formula:

$$NAV(W,I) = 1/(1 + e^{-(\sum(W*I))})$$

**Equation 12**



**Figure 11: Graph of Sigmoid Activation Function – Equation 12**

'I' represents the Inputs and 'W' represents the synaptic weight values. It is not necessary that the activation function be of sigmoid shape, but it does offer unique flexibility. If the synaptic connection weights applied to inputs before entering the summing junctions are low, the sigmoid will have small points of reflection and behave similar to a linear function. However, if the output of the summing junction is high, the points of reflection will be distinct; the sigmoid will appear as a step function having only two outputs zero or one. When the summing junction has a sufficiently high output to produce a one value from the neuron activation function, we say that our neuron 'fired'. The combination of firing neurons then becomes the domain for our motor output function.

Synaptic connection weights are the weights applied to the inputs before being summed in the summing junctions. There are two sets of synaptic connections in a single-layer neural network. The first set connects the sensor inputs to the summing junctions whose output is the domain of the neuron activation functions. The second set of synaptic

connections describes the weight values applied to the neuron output. Refer to *equation 11*. All are real valued scalars, modified at random by the network.

### **2.3 System Performance**

The power in a neural network comes from its ability to use a small training set to learn a behavior that can also be applied outside the range of the data used for training.

Virtual worlds were described in a system of line segments. Processing range information is simple for a world with a single or a few objects. However, training with a world of objects becomes very costly very fast in terms of computing performance. The overall training time can vary greatly depending on the quality of the simulation.

The number of training cycles per training session depends primarily on how quickly the neural network can converge to a local minimum error or the error threshold. Other factors are the size of the network. How many inputs, neurons, and motors are in the system? The number of neurons can be dependant on the number of sessions if the user chooses dynamic growth. Multiple training sessions can help push the performance error out of a local minimum, allowing the neural network to continue its training. How many sessions is either user defined or determined by the error threshold.

The Virtual LMS Time complexity is third order. The number of operations is a function of the VLMS resolution, obstacles segments, and training examples. The inner loop contains one product operation and two trigonometry functions.

The Rule-Based Controller time complexity is second order. The number of operations is a function of the VLMS resolution, and the number of training examples. The inner loop contains five products, two sums, and two trigonometry functions.

The neural network has a fifth order training time complexity. The number of

operations is a function of the number of sessions, the average number of training cycles per session, the number of training examples, the resolution of the VLMS, and the sum of the number of neurons. The inner loop contains one sum and one product.

The simulation performance for a trained neural network is third order. The number of operations is a function of the number of training steps, the number of neurons, and the resolution of the inputs. The inner loop contains one sum and one product. An additional cost is involved in running the VLMS...

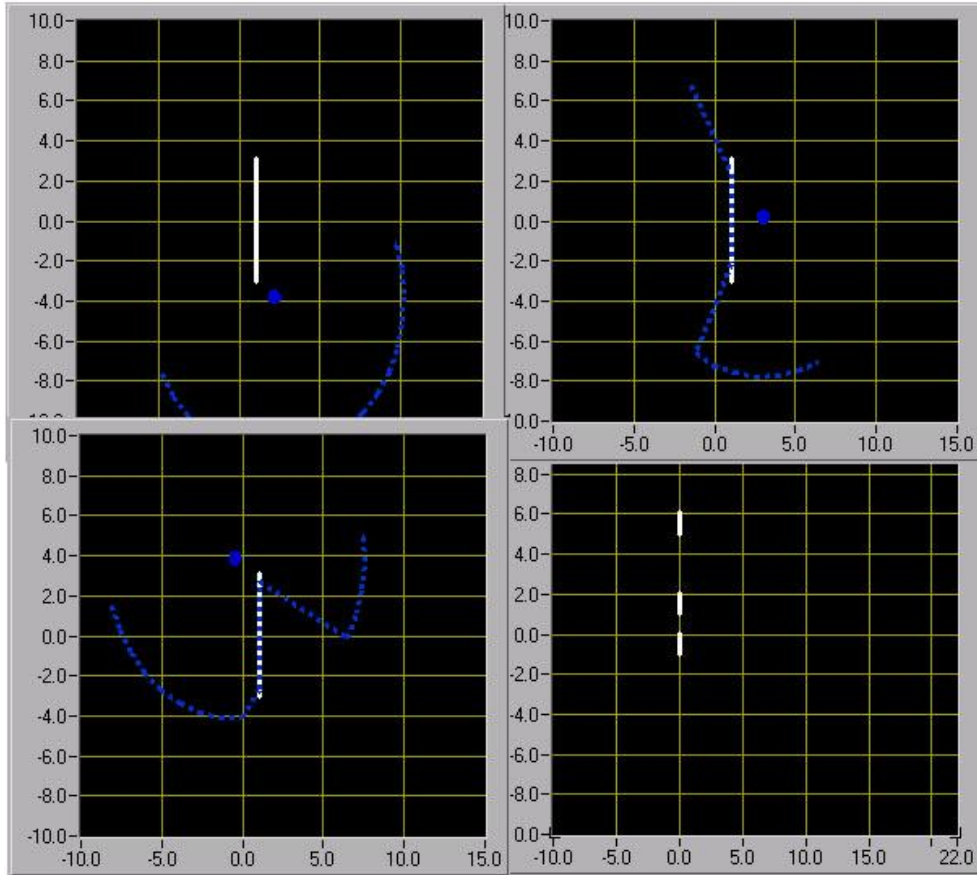
## **2.4 Training Worlds**

To curb some of the training performance costs, special training worlds were built starting with a single segment and moving towards a more complex multi-part segment.

A filter system was included to eliminate duplicate training samples.

The training worlds exist to give the neural network some obstacles to view sufficient to represent a basis for all possible inputs. Training worlds are composed of just a few obstacles revolved to create a sample set of input values for training the neural network.

The rules-based algorithm provides desired output speeds for this set of input values. The problem with using a single or too few obstacles is a learning deficiency in behaviors that relate multiple objects. The worlds made are as follows:



**Figure 12: Example Training Worlds**

***noObstacle***

This is an empty world with no obstacles. This is good for training the maximum speed. Training should be a very fast. There is little to learn.

***singleSegment***

This training world, shown in Figure 2 and Figure 12, contains a single 6-meter long segment. The segment is horizontal across the origin. By orienting the robot in fixed positions around the segment, various input patterns can be observed. That makes this training world effective at obstacle avoidance. It applies to worlds with long solid obstacles or walls.

### *conesTrainer*

Shown at the end of Figure 12, this world contains small objects spaced unevenly. This training world is useful for extracting behavioral responses to large and narrow passages. It is composed of three in-line segments. Two of the segments are .2 meters apart. Between the second and the third is a full meter gap. The width of our virtual robot is a half meter, so the robot should have no problem passing through the large opening, but should avoid the small one.

### 3. Neural Network Training

Training the neural network involves modifying the synaptic connection weights until the motor outputs of the neural network match closely to that of the rule-based system. We call the weight values genes, and store them in a single dimensional array we call the chromosome. The size of the chromosome is dependant on the number of inputs, the number of neurons, and the number of motors. This is calculated using the following formula:

$$\text{SizeOfChromosome} = I*N+N*M \qquad \text{Equation 13}$$

The variable 'I' is the number of inputs to the neural network. 'N' represents the number of neurons, or combinations of inputs. 'M' is the number of motors. There are (I\*N) synaptic link weight values that determine the confidence of a particular input stream as it enters the neurons. The neurons fire if the sum of the weighted inputs is high enough to reach the neuron activation energy.

The other N\*M entries describe the effect of each neuron activation on each of the motors. An input that has no effect on a neuron is one whose joining synaptic weight value is zero. Similarly, neuron activation may have no effect on a particular motor. In this case, the weight value of the synapse connecting the two should converge to zero.

#### 3.1 Organization

Training is accomplished through a process of supervised learning. Training data is gathered by applying the rule-based control system to various sensor input patterns, and recording the resulting motor response. The responses of a rule-based control system are our desired behaviors used to train the neural network.

Network training is highly dependant on the mature algorithm being used, and the training input data set. The algorithm determines how the neural network should behave, but without care, the network may detect behaviors outside of our objectives [2]. For example, the rule-based system may incorporate a series of forward, and then reverse motor commands in obstacle avoidance. This stutter step action we would like to be ignored by the neural network.

The training input data set determines the scope of the training. The training data serves as a basis for all possible input patterns. A training sample set with too few entries may train well, but not have the experience to resolve classes of input to the desired motor response. Ultimately, the training process should resolve a neural network that performs nearly identically to rule-based system for the training data provided. It must also perform reasonably for input patterns outside the training set. What is reasonable is qualitative in that the behavior must be observed.

Training is initialized by creating a chromosome array of the appropriate dimension given the current number of neurons, inputs, and motors. This chromosome is modified by an abstracted training process that produces: (1) an updated chromosome, and (2) a performance evaluation. In the training process, it was not expected that an initial chromosome would be transformed into a mature chromosome during a single session. This possibility existed, but the probability of occurrence was considered highly unlikely. The only expectation was that a local minimum of performance error would eventually be reached.

When the performance evaluation is poor and the user wishes to continue training, the mutated chromosome is passed back for re-training. Training the same chromosome



repeatedly is the same as extending the previous training session. This is helpful if the local minimum for performance was not reached in the previous training session. Typically, systems stuck in a local minimum must be somehow propelled from this condition. One solution is to train multiple initial chromosomes and accept the best of the set. Another is to pass hybrid chromosome of multiple optimized chromosomes. Another solution may be to increase the size of the chromosome and pass the larger chromosome for training.

Training organization can be described by the following composite function:

Updated Chromosome =S (Initial Chromosome, Inputs, Desired Speeds, Time steps)

Mature Chromosome =T (Updated Chromosome, Addition of Neurons, Breed Chromosomes)

#### **Equation 14**

The resulting chromosome we call the mature chromosome.

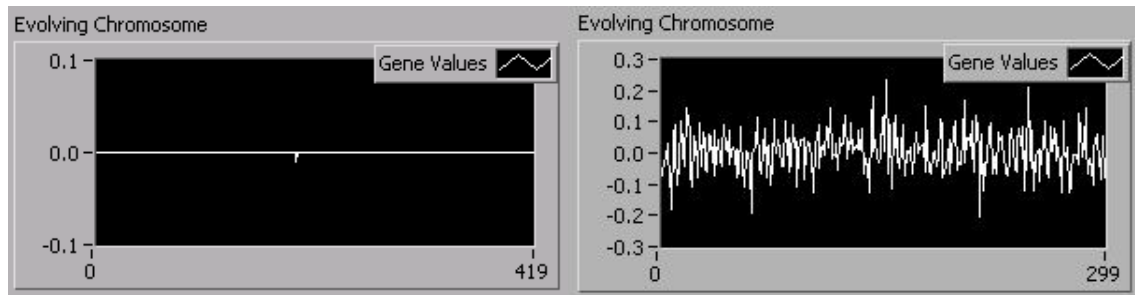
### **3.2 Training**

Motor output is the control system's response to input data presented by the sensor system at any given moment in time. For this reason, a single input/output pair provided by the rule-based system is regarded as taking one time-step. In the real world, adjacent time-steps are expected. In simulation, a time-step may be completely independent of its neighbors, or it can be acquired randomly. A typical training cycle consists of three tasks:

1. Apply an input pattern to the neural network described by the current chromosome.
2. Calculate the performance error in the motor output response of the neural network
3. Mutate the existing chromosome as necessary relative to the performance error.

We calculate error as the root-mean-square difference between the desired speeds as reported by the rule-based system and the motor output of our neural network. If the

error improves with the new chromosome, we keep it. Otherwise, we discard it and mutate the chromosome again. This systematic adjustment of the synaptic connection weights defines the current behavior of the chromosome in training.



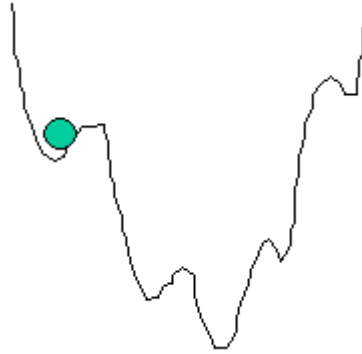
**Figure 13: Chromosome Evolution: After 2 mutations, After 145 mutations**

### **Time-Step Independent Training**

Three variations to this process were tested. In the first method, we solved for the chromosome that works well for the first time step and used it as the Initial chromosome for subsequent time steps. This is the training process described in [1]. Finding a chromosome that works for a single input output pair is similar to finding the line of best fit given only one point. Such a trivial situation always results in an exact solution. However, when you apply the so-called mature chromosome to subsequent time steps, the chromosome is highly ineffective and requires complete mutation. The error convergence is neither uniform nor predictable.

### **Time-Step Dependent Training**

The second method works the same as the first except we calculate the error for a given chromosome over all time steps. This method typically converges to a chromosome with a higher error than the allowed threshold. It finds the local minimum of error requiring a very loosely defined expectation.



**Figure 14: Local Minima**

Figure 14 is a visualization of how convergence to a local minimum may not produce the best results in the network. Small modifications to the chromosome produce greater errors; they are therefore discarded. Large changes are required at this point to displace the network so training can continue.

Determining when a local minimum has been reached is non-trivial. As a result, determining an appropriate training duration is equally difficult. An arbitrary number of training cycles tends often to be too many, or too few to bring the network into an appropriate error tolerance. The result was either an immature chromosome or an unneeded loss in performance.

Since we could not predict the number of training cycles, we used change-in-error to determine the best time to stop. After 50 training cycles that did not show improvement, the training session ended.

The chromosome is a composition of two sets of synaptic connection weights. The first set describes the weight values of inputs as they enter the first summing junctions. The second is the weight values of the activation functions before they entered the summing junctions. In yet another test, we tried to isolate parts of the chromosome to reduce our error. First, we allowed the chromosome to change only the first set of values for improvement. Then, after modifying that set, we turn to the second set and modify the

weights there. Mutation is toggled from first to second when the breakpoint has been reached and no further mutation in the current set has improved the performance of the network for arbitrary number of training cycles.

### 3.3 Genetic Algorithm

The third training method increased the efficacy of neural network training sessions with genetic algorithms (GA's). Genetic Algorithms are algorithms that exploit concepts of natural selection [8]. In nature, survival of the fittest suggests that the most fitting species will mate to produce even better offspring. The less fit species will pass away. The process continues repeatedly to refining the gene pool. Here, multiple initial chromosomes populate the gene pool. These chromosomes were refined and transformed into updated chromosome using training method two. The two best performing chromosomes were selected to breed and form new chromosomes. The gene pool was then repopulated with multiple instances of the four. The chromosome breeding process is shown in Figure 15 and Figure 16.

1. Decompose(Best Chromosome) - - > [Set1A Weights, Set2A Weights]
2. Decompose(Second Best Chromosome) - - > [ Set1B Weights, Set2B Weights]
3. Compose(Set1A, Set2B) - - > New Chromosome 1
4. Compose ( Set1B, Set2A ) - - > New Chromosome 2

**Figure 15: Genetic Algorithm - Chromosome Breeding Process**

Chromosome 1	Best
ai, ai, ai, ai, ai, ai, ai, ai, ai, ai, ai, ai, bj, bj, bj, bj, bj, bj, bj, bj	
Chromosome 2	Second
ci, ci, ci, ci, ci, ci, ci, ci, ci, ci, ci, ci, dj, dj, dj, dj, dj, dj, dj, dj	
Chromosome 3	Mix Breed
ai, ai, ai, ai, ai, ai, ai, ai, ai, ai, ai, ai, dj, dj, dj, dj, dj, dj, dj, dj	
Chromosome 4	Mix Breed
ci, ci, ci, ci, ci, ci, ci, ci, ci, ci, ci, ci, bj, bj, bj, bj, bj, bj, bj, bj	

**Figure 16: Genetic Algorithm – Resulting Gene Distribution**

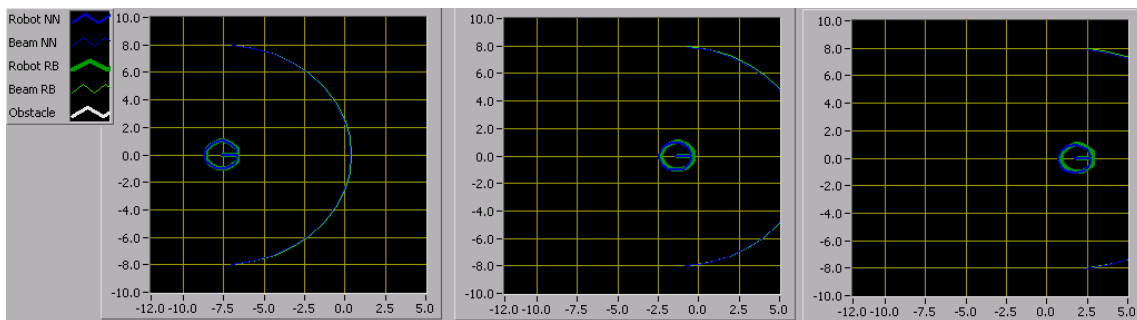
Eventually this process converges to a pool of chromosomes all having the same gene values. At this point of convergence, we say our pool is exhausted. If we have not yet achieved our desired error tolerance, we can replace the last three chromosomes with random chromosomes to refresh the pool. The best chromosome is kept for the next cycle of training.

## 4. Testing and Results

### 4.1 Practice Worlds

Recall from the introduction that the neural network controller was expected to exhibit three basic behaviors: forward motion at a constant velocity of two meters/sec, obstacle avoidance, and centering between multiple obstacles. The ANN training data was retrieved by positioning the robot in various positions about the training world and retrieving the desired response from the rule-based system. Practice worlds were designed to see how a mature neural network control system would respond in a complete virtual environment. Controller evaluation is qualitative. It was expected that the ANN might eventually behave similarly to the rule-based control system, but that the response of the two systems would not be identical. During testing, both control systems were run side-by-side to see if the appropriate behavioral response had been trained.

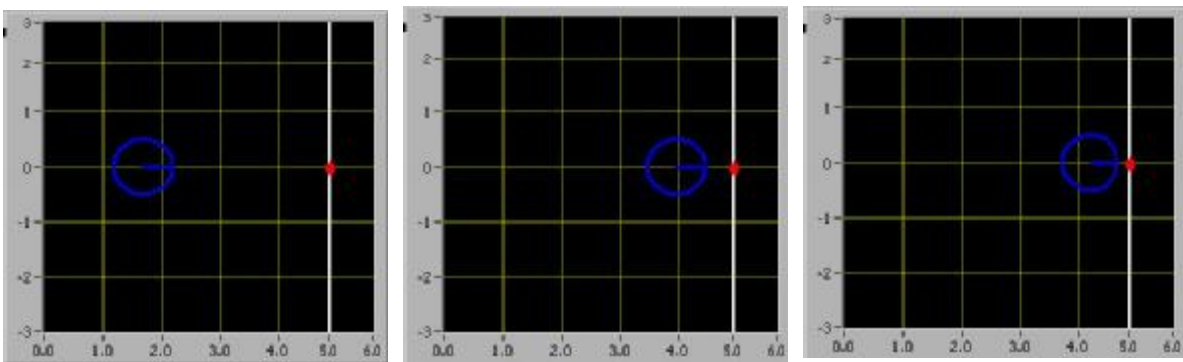
The default practice world was an open world with no obstacles. The rule-based system will drive the robot forward at a speed of 2 m/s in our virtual environment. In this case, the training environment and the testing environment is the same. Therefore, there is very little difference in the motor response of the two controllers. (See Figure 17)



**Figure 17: No Obstacle Cartoon Clip**

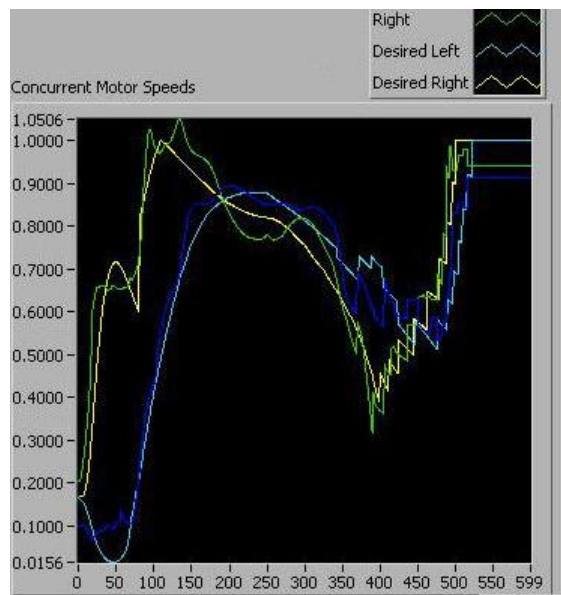
Our first test case involving obstacle interaction was with a single segment practice world. The objective was to show that obstacle interaction could be trained with a single-

layer neural network. The robot sensor information was filtered to include only one distance measurement directly in front of the robot. The network was given a single neuron for the control of both motors. With one neuron activation function, stereo control of the RMP motors is impossible. Both motor output responses are defined by the same function. The result is a neural network that performs well at decelerating and stopping on the approach of an obstacle.



**Figure 18: Single Segment – Testing Deceleration**

Then we tested for obstacle avoidance with a single input and two neurons. One neuron processes the deceleration; the other modifies the motor speeds once the forward distance threshold had been reached. With one input and two neurons, training was successful. The motor speeds of the neural network closely matched those of rule-based control system in the single segments practice world.



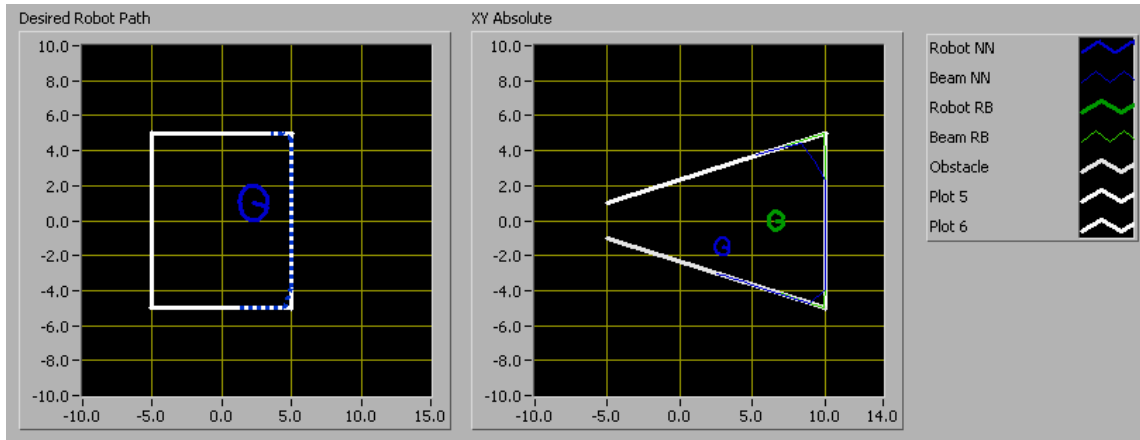
**Figure 19: Single Segment - Concurrent Motor Speeds**

An observation of the concurrent motor trends in

Figure 19 suggests that the ANN controller has generalized the elicited behavior from the neural network and smoothed its own motor responses. The jitter we see in the rule-based sensor is not nearly as obvious with the neural network.

Obstacle Avoidance and centering were tested using the test worlds depicted in Figure 20. The first image is a closed box shape. The robot is set in the center of the coordinate plane at some random angle. The edges of the box are line segments:  $(-5,5),(5,5)$  ;  $(5,5),(5,-5)$  ;  $(5,-5),(-5,-5)$  ;  $(-5,-5),(-5,5)$ . The door-less scenario is a straightforward method to burn-in test obstacle avoidance. This scenario was run over 1000 time-steps to verify that the robot would not escape the enclosure. It did not, suggesting that obstacle avoidance may have been trained successfully.





**Figure 20: Box Enclosure and Cone Left Open Practice Worlds**

Determining if the correct behavior has been adopted through training is challenging. Many times, the control system responds well for some sample scenarios and poorly in others. For example, consider using the box scenario in Figure 20, i.e., training for obstacle avoidance. An untrained controller that does not move, moves in a tight circular path, or turns in place might seem to operate correctly.

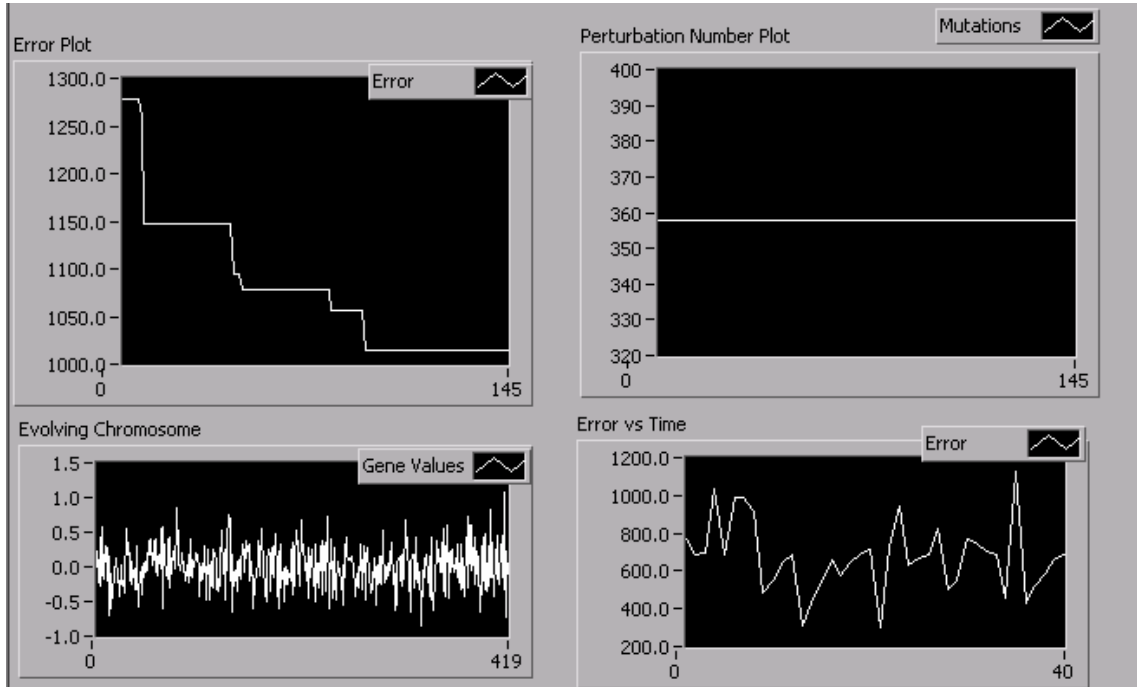
By offering an outlet, we can determine if the robot is interpreting the input and responding with the appropriate behavior. By changing the aperture size of the escape route, we can determine if the centering behavior has been trained as well.

### **System/Network Performance**

The performance cost of rendering the practice worlds can be very high, but acceptable without the high order training costs. It is also quite possible to load the trained chromosome directly on the Segway RMP to avoid world rendering costs. The largest world we created has 48 segments. Simulating the LMS with a high-resolution virtual sensor (360 inputs) and over a 1000 time-steps requires a significant number of point transformations.

Number of Transformations =  $360 \times 1000 \times 48 \times 2 = 34.5$  million **Equation 15**

## 4.2 Performance of Neural Network Training

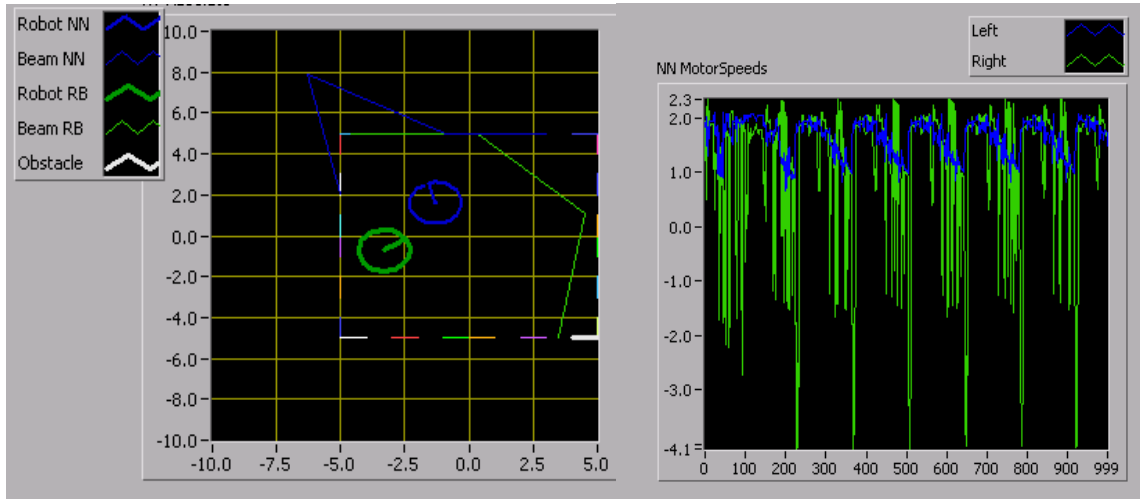


**Figure 21: Training Performance over time.**

The first graph, Figure 21, shows how training error is reduced as we converge to a better performing chromosome. It is a scaled visual of the current error over the 145 training cycles. The second graph shows how much the chromosome was modified over a complete session. The modification is directly proportionate to the current error.

However, if the error is higher than 0.2 meters/second (200 @ X1000), the perturbation rate is constant at .85% of the chromosome size. The third display allows us to watch the chromosome values as they mutate. There is no pattern to the mutation, nor was there any observable trend in its change in shape. The last display shows the error of the current chromosome over time. The errors were very high for some time steps and the errors

were very low for others. The neural network may follow the rule-based controller behavior for the most part, but there are a few times where the behavior of the two is largely different. In attempt to lower the error uniformly, we use the maximum error reported at any time-step as the error of the whole chromosome. Training was done using a weakest link mentality.



**Figure 22: Cones Training**

Tested in a world very similar to the box enclosure, the neural network was able to detect and avoid cone shaped obstacles in simulation. The key difference in this world is that openings allow the robot to see through the walls of the enclosure. To avoid obstacles the robot needs to detect and avoid openings that it cannot pass through. The rule-based trainer was given knowledge of the robot size and geometrically determined the distance of the object from the front and side of the robot. In simulation, the neural network was effective at learning this behavior. The results of the burn-in obstacle avoidance test are shown in Figure 22. Unfortunately, do to time constraints implementation of this controller on the RMP must be the subject of future research.

### 4.3 Real-World Testing

In Chapter 1: Robotics Systems Modeling, the Segway RMP is described having three main systems. The SICK Laser Measurement Sensor receives distance measurements from the environment. The control processor is the hardware that houses the control system and computes motor commands relevant to some appropriate behavior. Finally, the hardware controller boards instruct the motors to spin the wheels at the appropriate rate.

Until now, the sensor and actuator systems have been substituted by virtual systems. The sensor provides direction inputs to the control processor, which in turn passes motor commands to be executed by the actuator system. The RMP control processor manages each of these systems.

First, we initialize of the SICK LMS. We turn on the Segway RMP and put it into balance mode. In balance mode, the dynamic stabilization process maintains the Segway's upright position while the motor controllers accept commands for speed and direction of desired travel.

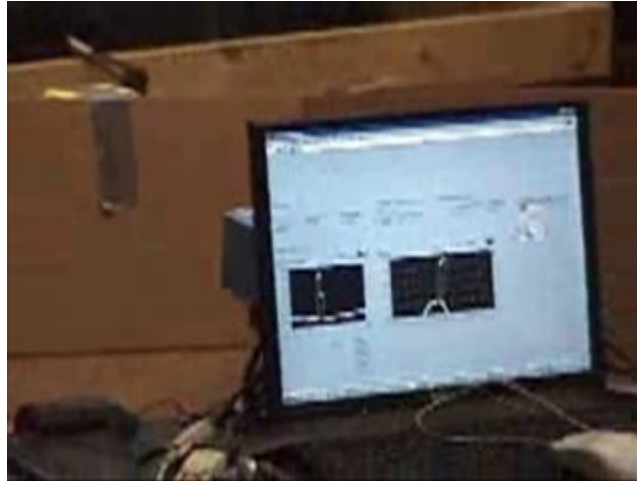
In the control loop, we retrieve the current input data from the LMS, which tells us about our environment. Then we pass that input to the neural core of the control processor mounted to the RMP. The behavior of an artificial neural network is stored in the synaptic connection weights. The array of weights we call the chromosome is transferred via text file from the controller used in simulation to the control processor mounted to the RMP. Just as before, the neural network produces the appropriate motor responses for the input data provided by the sensor. A scale factor on the output of the controller was used to reduce the motor speeds to a safe testing speed.

The real testing environment built was only about 150 sq feet in area. The speed of the RMP does deteriorate in the presence of an obstacle, but the turn radius in the skid steering would cause sporadic motion except that we scaled down the motor commands of the network. As stated previously, the RMP motor controllers do not accept skid motor instructions, so a routine was designed to translate the individual motor commands to a speed and motion vector. Then, the information was communicated across both channels to the RMP as described in introductory *section IV: Platform*. This process was repeated closed loop for a predefined time unless the halt exception was thrown from the user-controlled mouse.



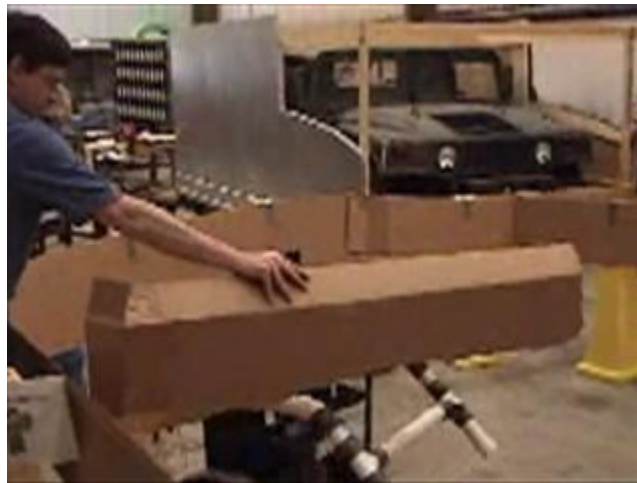
**Figure 23: Segway Real-World Testing Environment**

The inputs retrieved from the LMS are directional range values. Three hundred and sixty real-valued ranges are associated with each angle starting from the right most to the left most sensor reading. These range values were connected graphically as point vectors. The sensor input fan is shown in Figure 24.



**Figure 24: SICK LMS Range Fan**

We found during testing that the RMP would avoid obstacle in front of it, travel in the center of the testing world, and react to obstacles within its forward distance threshold by turning in place. Figure 25, shows holding an obstacle being held in front of the LMS well within the FDT to observe this action.



**Figure 25: RMP Reaction to obstacle within Forward Distance Threshold**

## **5. Conclusion**

### **5.1 Closing Remarks**

The design and development of a control system for the Segway RMP was successful. The RMP was tested using high and low resolution input data from the SICK LMS. The ANN performed comparably to the rule-based system used for training it. The ANN proved to be a highly adaptive alternative to rule-based controller development.

The RMP is a highly maneuverable transport vehicle, but the usefulness of this research is not confined to this particular robot. This research serves as a testimony of the effectiveness of the single-layer neural network to observe input patterns from a distance-measuring device.

A virtual world is a well-designed learning tool for the production of a mature chromosome that can be exported to the robot. The adaptable virtual sensor design allows for some flexibility in the type of sensor information retrieved. It would be easy enough to shorten input range of the sensor and decrease the number of inputs in order to simulate the tactile sensors on the Ev-Bot used in [1]. In building a useful virtual world for an RMP robot, the following had to be addressed: the RMP actuator system, the VLMS, and the object representation. The actuator system is responsible for updating the robot's position vector once the desired instantaneous wheel velocities have been received from the control system. The VLMS used a series of object transformations to determine the distance of objects in multiple directions from the sensor location. These objects, including the robot itself were recorded as a series of line segments drawn with respect to the current robot position.

Testing proved the neural network did successfully elicit the behavior of the rule-based controller algorithms. Obstacle avoidance behaviors were observed from the mature neural network in simulation and on the RMP. Multiple practice worlds were designed for the virtual robot to explore. Only two were designed for the RMP. When the rule-based and ANN controllers were ported onto the real RMP, and tested, the real robot performed as it had been trained to do in the virtual world. Figure 23 shows a real training world similar to the box test enclosure in *section 4.1: Practice Words*.

## **5.2 Future Research**

As stated at the end of *Chapter 4*, the RMP was never tested with small cone-shaped obstacles as was done in simulation. This was primarily due to time constraints at toward the end of the testing process. A controller effectively trained and tested in simulation should be effective at small obstacle avoidance on the RMP as well. This statement should soon be verified.

Another progressive step in this research would be to train behaviors that cannot be easily written into a rule-based control system. The supervised learning method we used during this research can be duplicated with training data retrieved from a joystick. In this manner, we may be able to elicit behaviors such like wall hugging or other behaviors not so easily defined by rules.

This can be accomplished by taking samples of input and associated motor response retrieved from a joystick or other human-in-the-loop input device. This training data can then be fed into the training scenarios in place of the current rule-based methods. If the user's behavior can be elicited successfully, it may be very useful for industrial or military applications.



## 6. Bibliography

1. Nelson A.L, Grant E., Lee G., ““paper\_archive\_nelson/nelson-caine-2002.pdf”,” in Proceedings of the ISCA 15th International Conference: Computer Applications in Industry and Engineering (CAINE-2002), San Diego CA, Nov. 7-9, 2002, pp. 92-97, ISBN: 1-880843-45-5.
2. Haykin, Simon. Neural Networks: a Comprehensive Foundation. Prentice Hall, New Jersey, 2nd edition, 1999.
3. The Science Behind the Technology. Segway Inc 2006  
URL:[http://www.segway.com/segway/how\\_it\\_works.html](http://www.segway.com/segway/how_it_works.html)
4. Segway Robotic Mobility Platform. 2006. Corporate Website. Segway Inc. URL:  
<http://www.segway.com/products/rmp>
5. Component Details of the Segway HT. Segway Inc. 2006. URL:  
[http://www.segway.com/segway/component\\_details.html](http://www.segway.com/segway/component_details.html).
6. Segway Robotic Mobility Platform User Guide. Version 1.3. 2006. Segway Inc. Manchester,NH
7. Laser Measurement System, Technical Description, SICK AG, Germany, June 2003.
8. Institute of Research in Information Science in Toulouse. URL:  
[http://www.irit.fr/COSI/glossary/fulllist.php?letter=Genetic\\_Algorithms](http://www.irit.fr/COSI/glossary/fulllist.php?letter=Genetic_Algorithms)
9. Nelson, Andrew L., “[Competitive Relative Performance and Fitness Selection for Evolutionary Robotics](#),” Doctoral Dissertation, North Carolina State University, Raleigh, North Carolina, May 2000.
10. Metric Conversion Tool, <http://www.metric-conversions.org/length/meters-to-miles.htm>
11. University of Pennsylvania, *Segway RMP Research at the GRASP Lab at the University of Pennsylvania*, URL:  
<http://www.cis.upenn.edu/marsteams/Segway/Report.pdf>
12. McCulloch, W. S. and Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115-133.
13. Grant, E.; Zhang, B. A neural-net approach to supervised learning of pole balancing  
Intelligent Control, 1989. Proceedings., IEEE International Symposium on Volume ,

Issue , 25-26 Sep 1989 Page(s):123 - 129

14. Larson, Ted. Balancing Robot Project. 2004 URL:

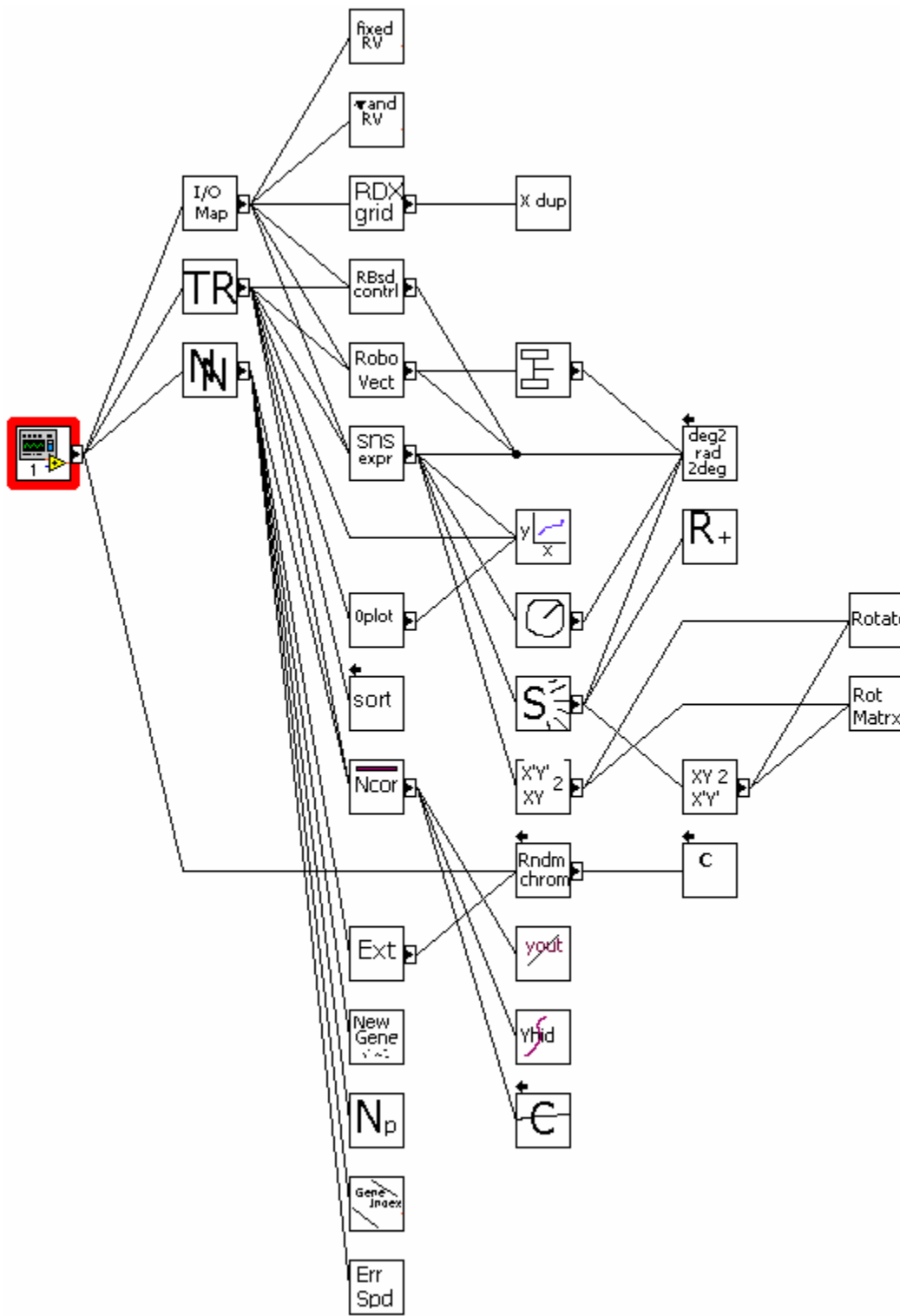
<http://www.tedlaron.com/robots/balancingbot.htm>

15. Morchen, Fabien. Analysis of Speedup as Function of Block Size and Cluster Size for Parallel Feed-Forward Neural Networks on a Beowulf Cluster. IEEE Transactions on Neural Networks, VOL. 15, NO. 2, March 2004

## Appendices

The following is an organized list of program data designed to help the reader understand the development of the virtual training world. The program modules are broken up into eight categories. Module information includes the class title for code reference, a connector pane for component abstraction, a list of dependant program modules, and a short description of components function. The following diagram can be used to reference each module's relative position in the program hierarchy.

## Virtual Training Environment - Program Component Hierarchy



## A.1: Actuator Assembly

### robotVector.vi

The Robot Vector holds information concerning the position and orientation of the Robot at any given time. This class is responsible for updating the robot vector after each instance of motion in the virtual world. The input motor speeds are used to calculate the change in position required to update the Robot Vector.

#### Connector Pane



#### List of SubVIs



**deltaPosition2.vi**

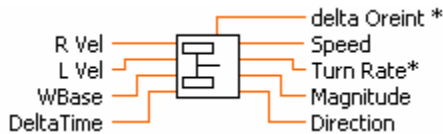


**deg2rad2deg.vi**

### deltaPosition2.vi

This class is responsible for translating skid steering instruction to a motion vector. It calculates the robot's change in position, so the Robot Vector can be updated. This is useful to the RMP controller boards and the virtual actuator system.

#### Connector Pane



#### List of SubVIs



**deg2rad2deg.vi**

### fixdRV2.vi

This class is responsible for positioning the robot in specific locations without regard to the motor commands or current position of the robot. This is relevant to training in the virtual world only.

#### Connector Pane



### **randRV.vi**

This class is responsible for positioning the robot in random locations without regard to the motor commands or current position of the robot. This is relevant to training in the virtual world only.

#### **Connector Pane**

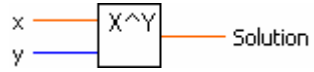


## A.2 Math

### pow(x,y).vi

This is an exponential operator: x to the y power.

#### Connector Pane



### deg2rad2deg.vi

This translates radian angles to degrees and vice-versa.

#### Connector Pane





### A.3 Neural Network

#### neuralCore.vi

This is the neural network as described in section 2.2. The neural network program model has 3 abstracted components. The first system splits the chromosome into two arrays of synaptic connection weights. The first group is processed at the summing junction in “neuralnetsig” sub function. The second group is processed at the summing junction in “neuralnetlin” sub function.

#### Connector Pane



#### List of SubVIs



**neuranetsig.vi**



**neuranetlin.vi**



**splitChromo.vi**

#### neuranetsig.vi

This calculates the output of the primary summing junction and sigmoid neuron activation function in the neural network.

#### Connector Pane



#### neuranetlin.vi

This calculates the output of the secondary summing junctions and in turn, the resulting motor speeds.

#### Connector Pane

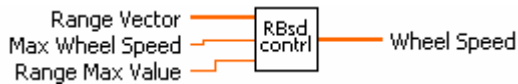


## A.4 Rule-Based Control System

### rbcontrol.vi (simple)

This class houses the rule-based training algorithm for basic obstacle avoidance. It suffices for large monolithic obstacles. System details are described in chapter 2.1. The sensor inputs are retrieved in an array whose size implies the sensor resolution. The maximum range and wheel speed values are necessary to create the proportional relationship between the input values and the desired motor speed.

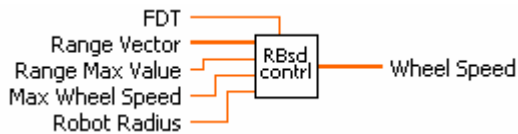
#### Connector Pane



### rbcontrol2.vi

This class houses the rule-based training algorithm for small obstacle avoidance. It is the improved Rule-based system described in chapter 2.1. The Forward and Side Distance Thresholds in conjunction with the size of the robot are used to determine whether the robot can pass between obstacles.

#### Connector Pane



#### List of SubVIs

 **deg2rad2deg.v**  
**i**

## A.5 Miscellaneous Tools

### chromosome.vi

Class combines two arrays of synaptic weights into single array called the “chromosome”.

#### Connector Pane



### elimdup.vi

For performance reduction, this class eliminates duplicate patterns from the training data. Duplicate entries are erased.

#### Connector Pane



### xygraph.vi

This class is used to draw the world of obstacle segments. The UCF segment points are entered and drawn to a graph in the training and testing user interface.

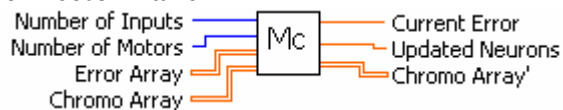
#### Connector Pane



### MergeChromo.vi

This class is responsible for the breeding of chromosomes in the genetic algorithm described in chapter 3.3.

#### Connector Pane



#### List of SubVIs

 `splitChromo.vi`

 `chromosome.vi`

sort

**sortarray.vi**

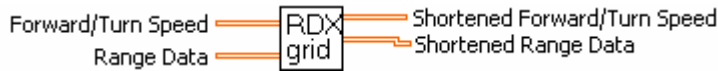
Rndm  
chrom

**RandomChromosome.  
vi**

### **Reducegrid.vi**

This class eliminates duplicate patterns from the training data. It assembles the input output pairs, removes the duplicates, then passes

#### **Connector Pane**



### **List of SubVIs**

X dup

**elimidup.vi**

### **splitChromo.vi**

This class separates chromosome into 2 synaptic weight arrays.

#### **Connector Pane**



## A.6 Training

### startTraining1-1.vi

This class is responsible for training the neural network as described in chapter 3.2: time-step independent training.

#### Connector Pane



#### List of SubVIs



**RandomChromosome.vi**



**TrainedSession.vi**



**trainingneuralnet1.vi**

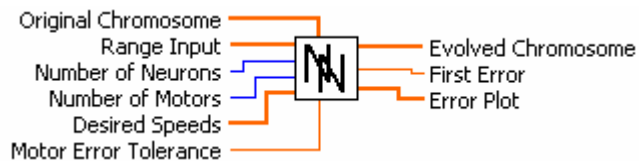


**mapping.vi**

### trainingneuralnet1.vi

This class is responsible for training the neural network as described in chapter 3.2: time-step independent training.

#### Connector Pane



#### List of SubVIs



**speedError.vi**



**nueralCore.vi**



**randGeneIndex.vi**



**Pnumber.vi**



**chromoexpansion.vi**

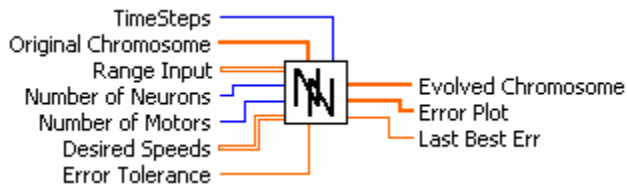


**geneEvolution.vi**

### trainingneuralnet2.vi

This class is responsible for training the neural network as described in chapter 3.2: time-step dependent training.

#### Connector Pane



#### List of SubVIs



**speedError.vi**



**nueralCore.vi**



**randGeneIndex.vi**



**Pnumber.vi**



**chromoexpansion.vi**



**geneEvolution.vi**

### Pnumber.vi

This class is responsible for calculating the number of chromosome mutations necessary in the training cycle. The number of mutations is relative to the current error.

#### Connector Pane



### randGeneIndex.vi

This class is responsible for selecting a gene at random from the chromosome. Once isolated, the gene may be altered slightly during training.

#### Connector Pane



### startTraining3-2.vi

This class is responsible for training the neural network as described in chapter 3.2: time-step independent training.

#### Connector Pane



#### List of SubVIs



**RandomChromosome.vi**



**TrainedSession.vi**



**Read From Spreadsheet File.vi**



**mapping.vi**



**trainingneuralnet2.vi**



**sythObstacle.vi**



**MergeChromo.vi**

### chromoexpansion.vi

This enables the size of the chromosome to be changed during training. Rebuilding the chromosome maintains the existing neurons with existing synaptic connections. However, there is no evidence that chromo expansion is more useful than initializing a new chromosome to be trained.

#### Connector Pane



#### List of SubVIs



### geneEvolution.vi

This class modifies the value of a gene slightly.

#### Connector Pane



### GrowNeurons.vi

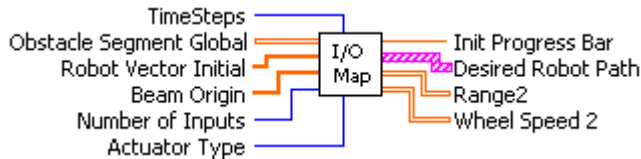
This class is an adder. It adds neurons to the neural network when a switch is thrown.

#### Connector Pane



### mapping.vi

#### Connector Pane



#### List of SubVIs







**Sensor.vi**



**Reducegrid.vi**



**fixdRV2.vi**



**rbcontrol2.vi**

### **RandomChromosome.vi**

This class initializes a chromosome of the designated size with small random weight values. These value are to be modified through training.

#### **Connector Pane**



#### **List of SubVIs**



**chromosome.vi**

### **speedError.vi**

This class determines the mean-squared difference in the motor response of the neural network and that of the rule-based control system. The resulting error calculation is used as the only quantitative performance metric to our system.

#### **Connector Pane**

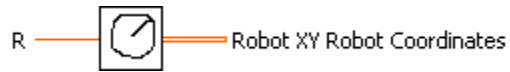


## A.7 Virtual Sensor System

### drawRobot.vi

This class is responsible for determining the robot image coordinates for the virtual world.

#### Connector Pane



#### List of SubVIs



### Sensor1.vi

Provided the position of the robot and all obstacles segments, this class produces the range data that would be retrieved from a range-finding device like the LMS

#### Connector Pane



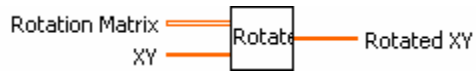
#### List of SubVIs



### rotate.vi

This class rotates the image coordinates in our virtual world.

#### Connector Pane



### rotationMatrix.vi

This class rotates the image coordinates in our virtual world. It supports the rotate class.

#### Connector Pane

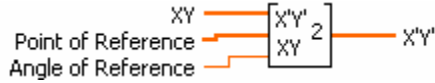


#### List of SubVIs


### XYReverseTx.vi

This class rotates and translates a pair of coordinates with respect to a provided position and direction.

#### Connector Pane



#### List of SubVIs

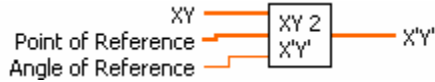
 **rotate.vi**

 **rotationMatrix.vi**


### XYTransformation.vi

Object segments were translated from UCF to determine their location with respect to other objects in the virtual environment. By providing this system with a point and orientation of an obstacle as it exists currently, we can rotate and translate each object in the world with respect to the provided object such that the provided object is at the origin facing the X-axis. All other objects are oriented around it as they were in the original view.

#### Connector Pane



#### List of SubVIs

 **rotate.vi**

 **rotationMatrix.vi**

### RangeFinderAxis.vi

This class finds the range of an obstacle from the sensor by finding the X-intercept of the obstacle. (See Chapter 1.2)

#### Connector Pane



### SimMini.vi

This program determines the distance (within 8 meters) from the VLMS sensor to the any obstacle that lies in some specific direction.

#### Connector Pane



#### List of SubVIs



**deg2rad2deg.vi**



**rangeFinderAxis.vi**



**XYTransformation.vi**

### synthObstacle.vi

This class draws the static virtual world from the obstacle segments retrieved from the world files listed in Appendix 8.8.

#### Connector Pane



#### List of SubVIs



**GraphXY.vi**

C:\Documents and Settings\Charles\Desktop\Segway\Master\SharedTools\GraphXY.vi

## A.8 Available World Segments

The following segments points comprise the test and training worlds used in the simulations. Every two points represent the endpoints to line-segments in the virtual world.

<p><b>Big Doorless</b></p> <p>-15 15  15 15  15 15  15 -15  15 -15  -15 -15  -15 -15  -15 15</p>	<p><b>Cone Left Open</b></p> <p>-5 1  10 5  10 5  10 -5  10 -5  -5 -1</p>	<p><b>Cones Trainer</b></p> <p>0 -1  0 0  0 1  0 2  0 5  0 6</p>	<p><b>Doorless</b></p> <p>-5 5  5 5  5 5  5 -5  5 -5  -5 -5  -5 -5  -5 5</p>
<p><b>Doorway Left Open</b></p> <p>-5 5  10 5  10 5  10 -5  10 -5  -5 -5  -5 -5  -5 2</p>	<p><b>Single Segment Trainer</b></p> <p>0 -3  0 3</p>	<p><b>Door Left Open</b></p> <p>-5 5  10 5  10 5  10 -5  10 -5  -5 -5</p>	<p><b>Concave</b></p> <p>-5 5  5 5  5 5  5 -5  5 -5  -5 -5  -5 -5  0 0  0 0  -5 5</p>

Long Tunnel		No Obstacle		Cones			
-5	3	100000	100000	-5	-5	-0	5
20	3	100001	100001	-5	-4	-1	5
20	3			-5	-3	-2	5
20	-3			-5	-2	-3	5
20	-3			-5	-1	-4	5
-5	-3			-5	0	-5	5
				5	-5	-0	-5
				5	-4	-1	-5
				5	-3	-2	-5
				5	-2	-3	-5
				5	-1	-4	-5
				5	0	-5	-5
				-5	5	0	5
				-5	4	1	5
				-5	3	2	5
				-5	2	3	5
				-5	1	4	5
				-5	0	5	5
				5	5	0	-5
				5	4	1	-5
				5	3	2	-5
				5	2	3	-5
				5	1	4	-5
				5	0	5	-5