Intelligent Systems Simulation Project
(236754)

Two-Robot Source Seeking with Point Measurements Algorithm

Submitted by:
Omri David
Eran Goldemberg

Under the guidance of: Yotam Elor
Under the supervision of: Prof. Alfred M. Bruckstein

November, 2011

Table of Contents

Problem definition

In this project we create a simulator of two robots reactive gradient following algorithm using point measurements was presented.

Using indirect motion based communication.

The robots compare point measurements and drift with the gradient.

The simulator have two modes of spreads: a peak mode and a gradient follow by the x axis.

In addition the user can define some manipulations in the robot's world.

He can define the parameters of a linear function and a level of noise that define the distance between the robots.

The User can control on the speed and the zoom of the robots and see their locations in a graph.

The simulator provides a Solution based on the paper "Two-Robot Source Seeking with Point Measurements" by Yotam Elor and Alfred M. Bruckstein.

The Algorithm

The system comprises two robots r_1, r_2. Each robot has a global location. The distance between the robots is denoted by D and given by $D = \|X_2 - X_1\|$.

The center of mass point is given by $\frac{1}{2}(X_1 + X_2)$. We define $X_i(t)$ as the location of robot r_i at time t.

Let $z_i(X)$ be the value of the scalar function at point X $(0 < Z_i(X) < \infty)$.

Let $D_i = f(z_i)$ be the desired distance for robot r_i.

While executing the algorithm, robot r_i attempts to keep a distance D_i from robot r_j.

The function f is user designed and affects the ultimate system behavior.

When executing the proposed algorithm, each of the robots move according to a sum of two velocities:

$$\dot{X}_1 = v_0(\vec{V}_{2\to1} + \vec{V}_1)$$

$$\dot{X}_2 = v_0(\vec{V}_{1\to2} + \vec{V}_2)$$

Where v_0 is normalization constant, By changing the velocity $\vec{V}_{j\to i}$, robot r_i seek to maintain the desired distance from r_j via:

$$\vec{V}_{j\to i} = A_{ji} \cdot \hat{u}_{ij}$$

$$A_{ji} = sign(D - D_i) \in \{0,-1,1\}$$

Let $\hat{u}_{ij}$ be the following unit vector $\hat{u}_{ij} = (X_j - X_i)/D$.

Where the function $sign(x)$ equals 1 if $x > 0$. 0 If $x = 0$ and -1 if $x < 0$. Hence A_{ji} is the sign of the velocity:

If $D > D_i$ then $A_{ji} = 1$ so r_i is attracted toward r_j.

If $D < D_i$ then $A_{ji} = -1$ so r_i is being repelled from r_j.

If the robots would move only according to the velocities $\vec{V}_{2\to1}$ and $\vec{V}_{1\to2}$, they did forever occupy a line. To avoid that, the velocities $\vec{V}_1$ and $\vec{V}_2$ which cause the robots to orbit each other are introduced. $\vec{V}_i = \hat{u}_{ij} \times \hat{u}_{up}$ ($\hat{u}_{up}$ is the unit vector which point up).

The user can define a noise sensors for $z(\cdot)$: $\hat{z}_i = z_i + N_z$.

N_z is a random variable represent the measurement error.

Main Classes

- Actor
This is an abstract class, represents an object that has action and drawing methods.

- Chart
Holds the variables and lines that create a chart that presents the pheromone value of each robot and the pheromone of the CM.

- Constant
Holds the constants required for the simulator.

- GradientFollow
Responsible for the robot's movement according to the algorithm. It holds the robot's current position, and is responsible for the move.
While executing the algorithm, robot r_i attempts to keep a distance D_i from robot r_j.

- Graph
Holds a list of pheromone values for each robot and for the CM.

- Ground
Responsible for creating the array named 'Markings' of the pheromone value for each location in world. Values are determined by the chosen spreads.

- Menu
Responsible for all the buttons, text boxes and scroll bars for the user. The user can select the spread he want, the manipulations of the world, the F function, the noise, the zoom and the speed of the robots.

- My list
A class that extends a list of actors, and implements the "Actor" methods for a list of actors.

- My robot
My robot is an Actor (by extending it). This class represents a robot in the world, that has coordinates, radius, and his own drawing method.

- Obstacles
Represents the obstacles that delimit the world. It extends Actor, and implements its own draw method.

- SimulatorApplet
This is the main class. It implements runnable. In initialization the world and menu objects are created and initialized. Then a new thread is created and that thread invokes the run method.

- View
This class is responsible for the functionalities of dragging objects, and zooming on the world.

- World
This class represents the area that the robots are located at. The class Implements the MouseMotionListener, MouseListener and MouseWheelListener Interfaces. Thus it has methods to catch events from mouse and keyboard, for Handing input from user or mouse actions such as dragging or manipulating.

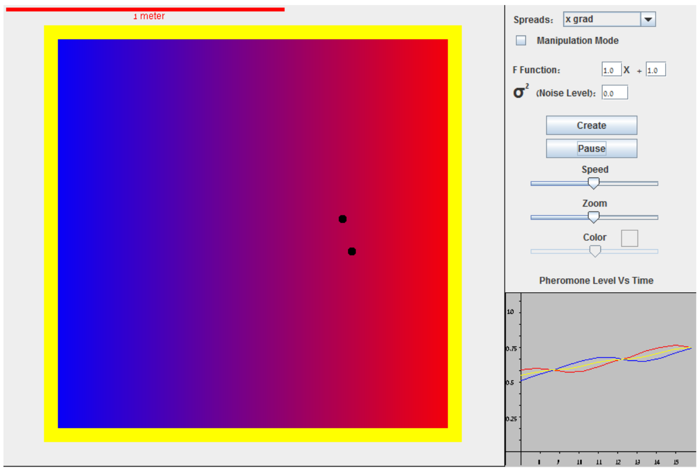
- WriteValue
This class extends TimerTask and has a Graph object.
Each time unit the pheromone values of the two robots and center of mass are saved in the lists in the graph object. In other words, this class maintains the history of pheromone values, by which the graphs will be drawn later.

Special Features and Difficulties

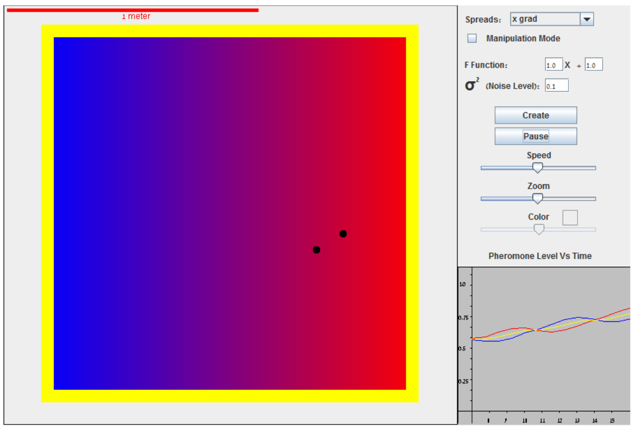
Noise Level

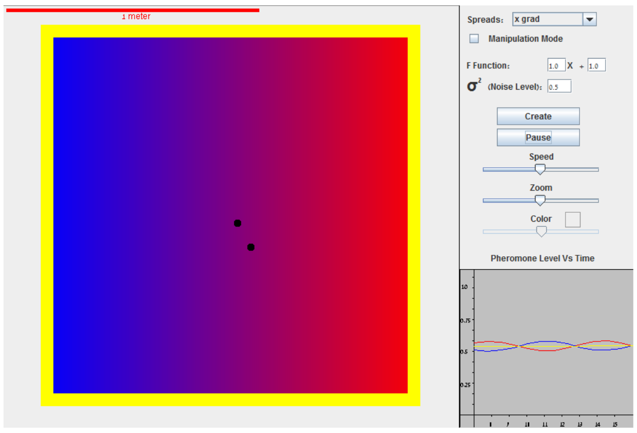
We wanted the user to have option for adding a Gaussian noise to the motion of the robots. The goal of this parameter is to add a random value to the motion of the robots that represents the measurement error.
We used the 'nextGaussian' method in the Random Class, which we weren't really familiar with, and had to look for it.
The function Returns the next pseudorandom, Gaussian ("normally") distributed double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.
The user defines the noise value he wants. This value represents the expected variance he wants for the noise. The calculation is to take the standard deviation and multiply it by the random number. We add the result to the sample of the pheromone level according to the robot's location.

```
Random rand = new Random();
double next = rand.nextGaussian();
value = Ground.markings[index]+Math.sqrt(world.ground.mNoiseLevel)*next;
```



$\sigma^2 = 0.0$





$\sigma^2 = 0.1$

$\sigma^2 = 0.5$

<u>Creating a graph for sampling the pheromone level of the robots</u>

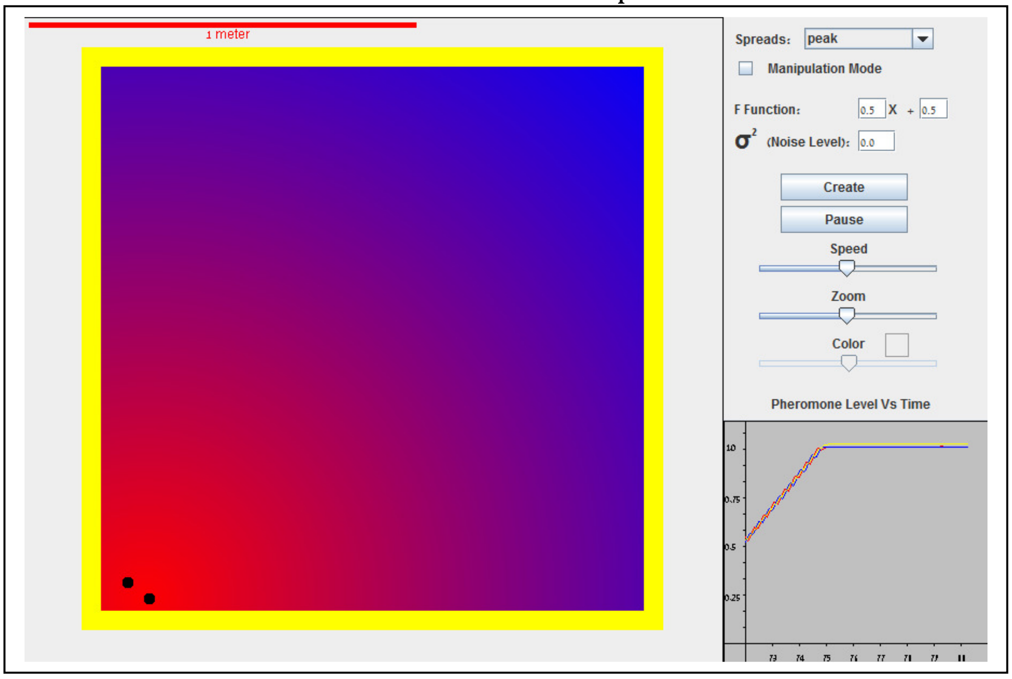
An important feature in the simulator is the ability to see the pheromone level of the robots according to the time.
Therefore we decided to add a mechanism of graph that will present the pheromone level of each robot for each time unit.
We were thinking how to save the pheromone values of each robot for every time unit. We then decided to create a database that will save these values so we'll be able to draw the desired graphs according to the stored values.
First, we created a class called 'Graph' which has three double Linked List: Two lists that save the pheromone value for each robot, and one list that saves the pheromone value of the CM.
Second, we created class called 'WriteValue'.
In each time unit, the pheromone values of the two robots and center of mass are saved in the lists in the graph object.
Finally, in the 'Chart' class we eventually create the graph. It creates the axes and the text for the axes. It uses the saved values for the robots and the CM and adding line between each two adjacent points.
It was needed to do the graph by default spaces between the points. Therefore it takes the expected width and height and calculates the expected spaces in variable named 'delta'.

<u>One difficulty</u> we had in creating the graph is that we didn't know that the axis grow down and not towards up. (At first we got graphs going down instead going up so we saw the graph is symmetric with the function y=0.5 and then we understood what the problem was).

<u>Another difficulty</u> we had is that we had to design a Y axis that fits the scale of 0.0 to 1.0. So it demanded great mathematic skills in order to find the exact location of the 0.25, 0.5, 0.75 and 1.0 numbers on the axis.
We also had to find a constant to add to the drawing function so that the graphs will start to be drawn from the correct height in the graph.

Graph for example

F function 0.5x+0.5 in peak mode



Spreads: peak

Manipulation Mode

F Function: 0.5 X + 0.5

σ² (Noise Level): 0.0

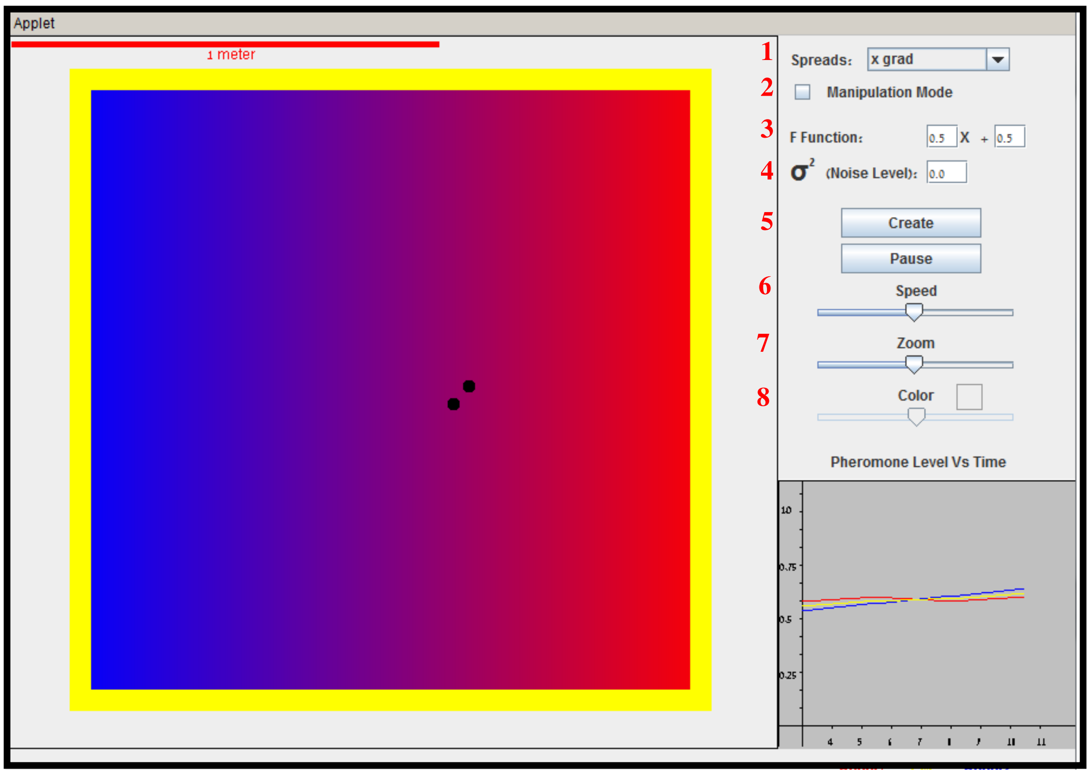
Manipulation Mode

An important feature in the simulator is the ability to add painting of the world by the user, which can change the pheromone level in the painted area.

One difficulty we had was how to determine the scroll's colors.
It was necessary to make sure that the chosen colors would not be confusing the user. For example, instead of purple we had shades of green in previous version. That wasn't so good because when we painted green areas with blue color, we got lighter green in the edges- which was quite confusing.
Therefore we work only with shades of blue and red.

Another difficulty we had was to get smooth painting. In a previous version we used a squared pattern; each pixel was painted in the same color, which was the new chosen color to paint. This wasn't a good solution, so we had to change it and come up with a better idea. We chose a Gaussian pattern in the size 21x21, where the center of the pattern is the area selected with the mouse, and his 120 neighbors cells are painted according to a formula: we take into account the current color and the new color that is chosen, and each one of the two colors is multiplied by a weight. The weight is heavier for the new color for cells closer to the selected area. The weight is heavier for the old color for cells that are farther than the selected area. In that way, we got painting much smoother and neater.

The Simulator



Spreads: x grad

F Function: 0.5 X + 0.5

σ^2 (Noise Level): 0.0

1 Spreads – The user can choose between “X grad” mode and “Peak” mode for the values in the world’s locations.

2 Manipulation Mode- By select the manipulation mode, the user can select a color in the color scroll bar and paint in the selected color on the world pheromone’s values.

3 F function – The user can define what will be the linear function that control the robot’s algorithm. He can choose what will be the derivation and intersection with the y axis.

4 Noise Level- The user can define a noise for the value which the robot sample. The recommended noise is between 0.001 to 1.

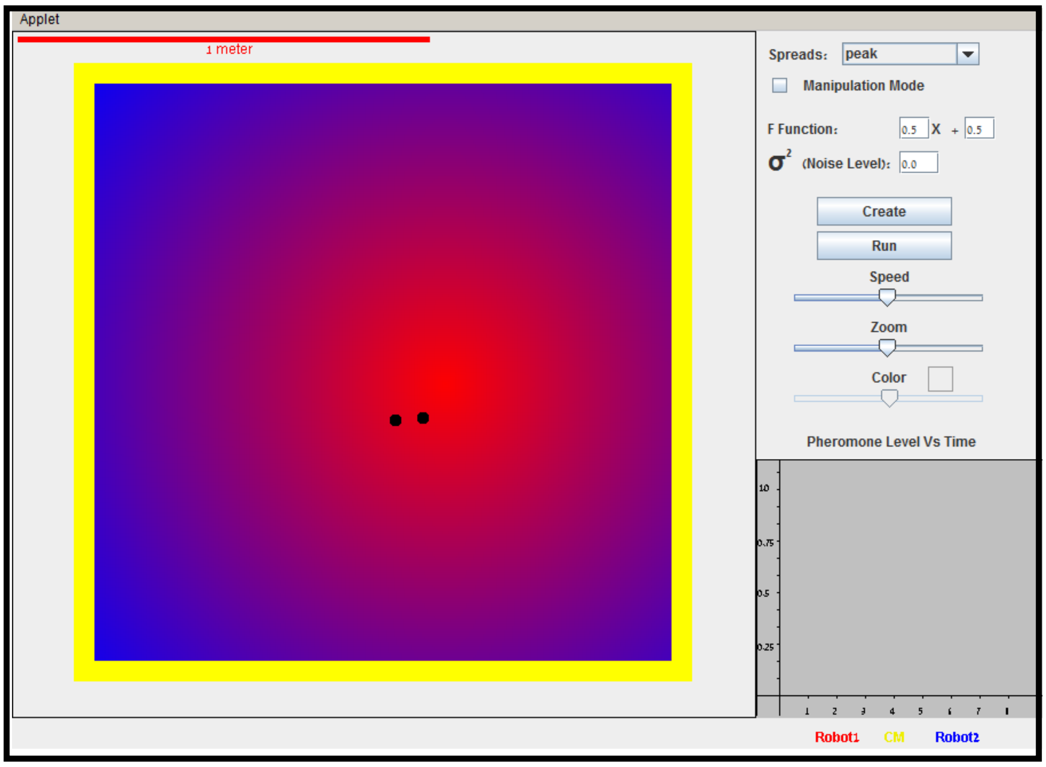
5 Create- The Create button initiate the world according the parameters of the spreads, F function and noise level and randomize a location for the robots .

Run/Pause – The Run button cause the robots starting their move according the algorithm. By Pressing the Pause button the robots will freeze their motion.

6 Speed- Control the speed of the motion.

7 Zoom- Control the zoom in/out of the world.

8 Color – Control the color for the manipulation mode.



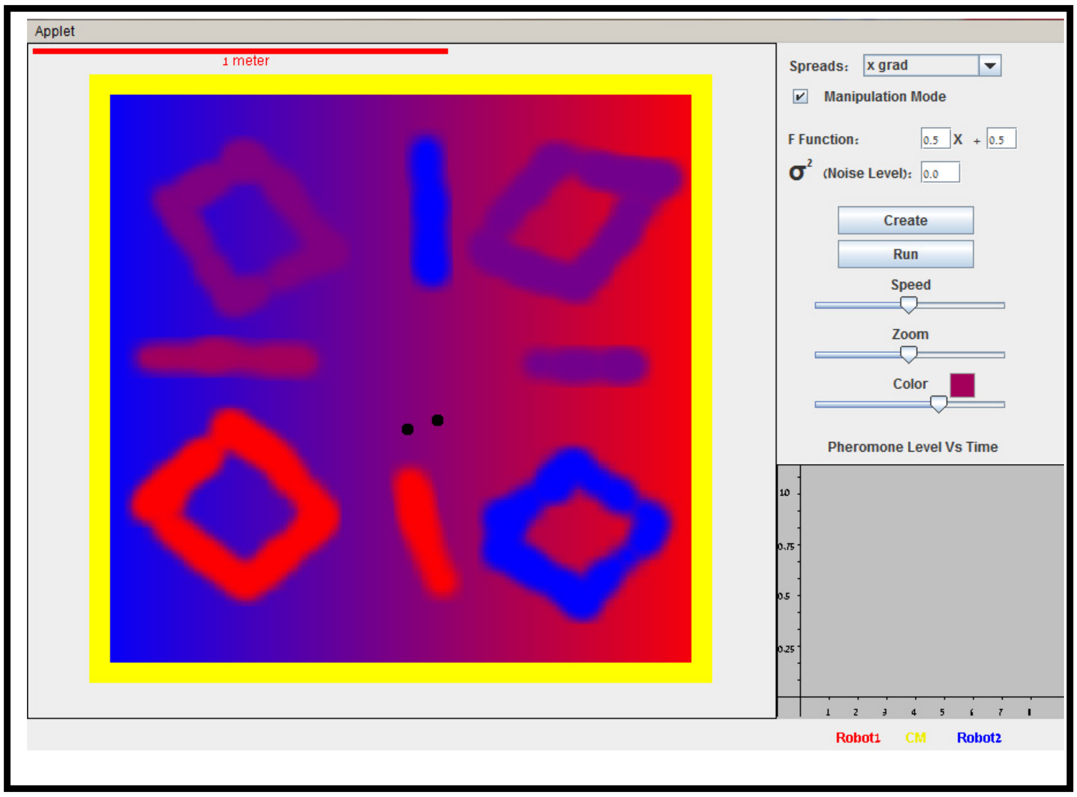
Spreads: peak

Manipulation Mode

F Function: 0.5 X + 0.5

σ^2 (Noise Level): 0.0

Peak Mode



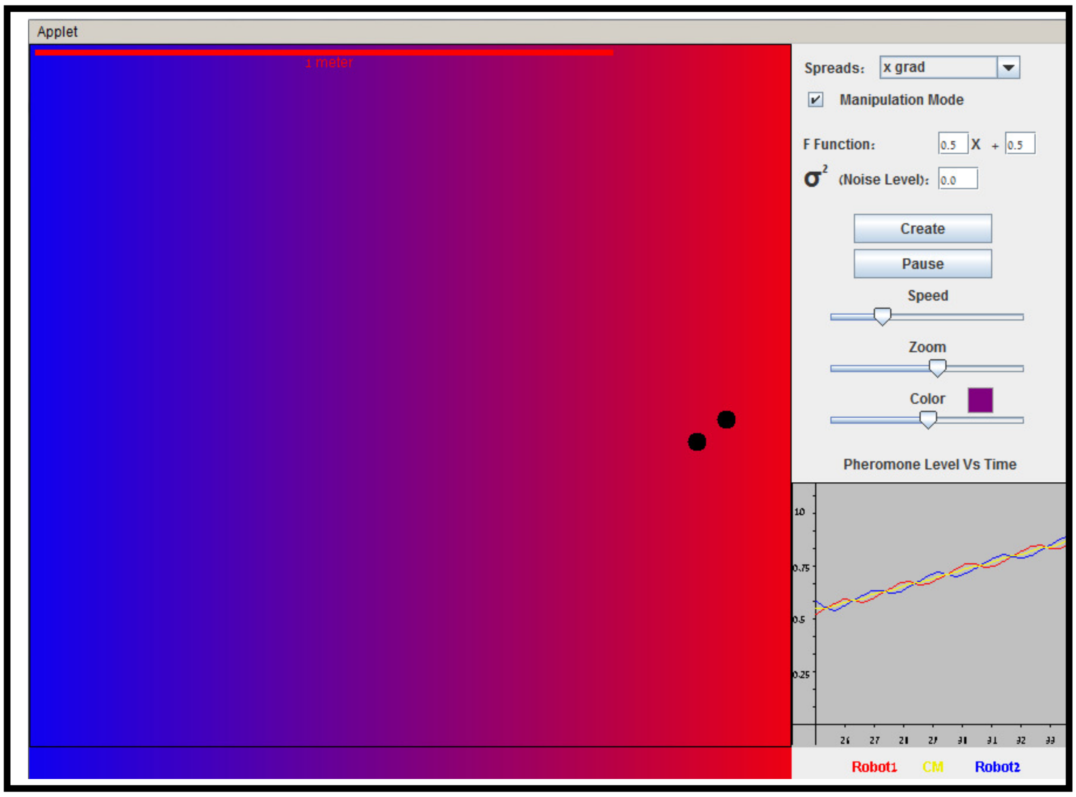
Spreads: x grad

✓ Manipulation Mode

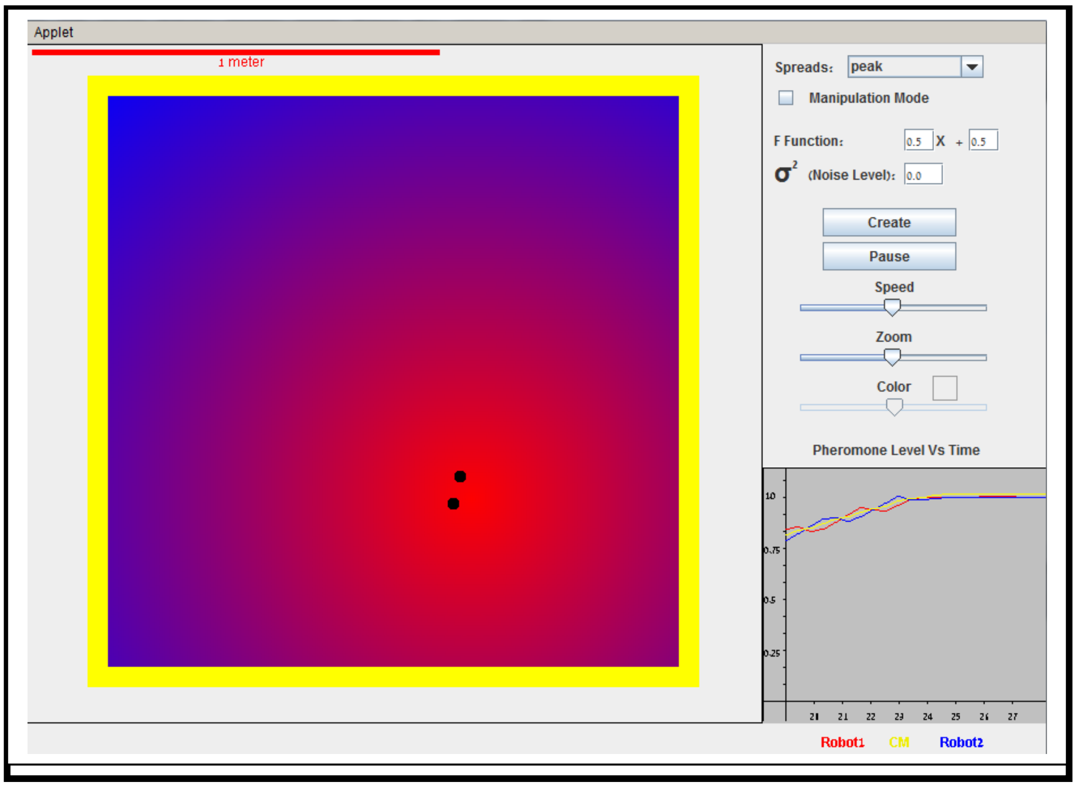
F Function: 0.5 X + 0.5

σ^2 (Noise Level): 0.0

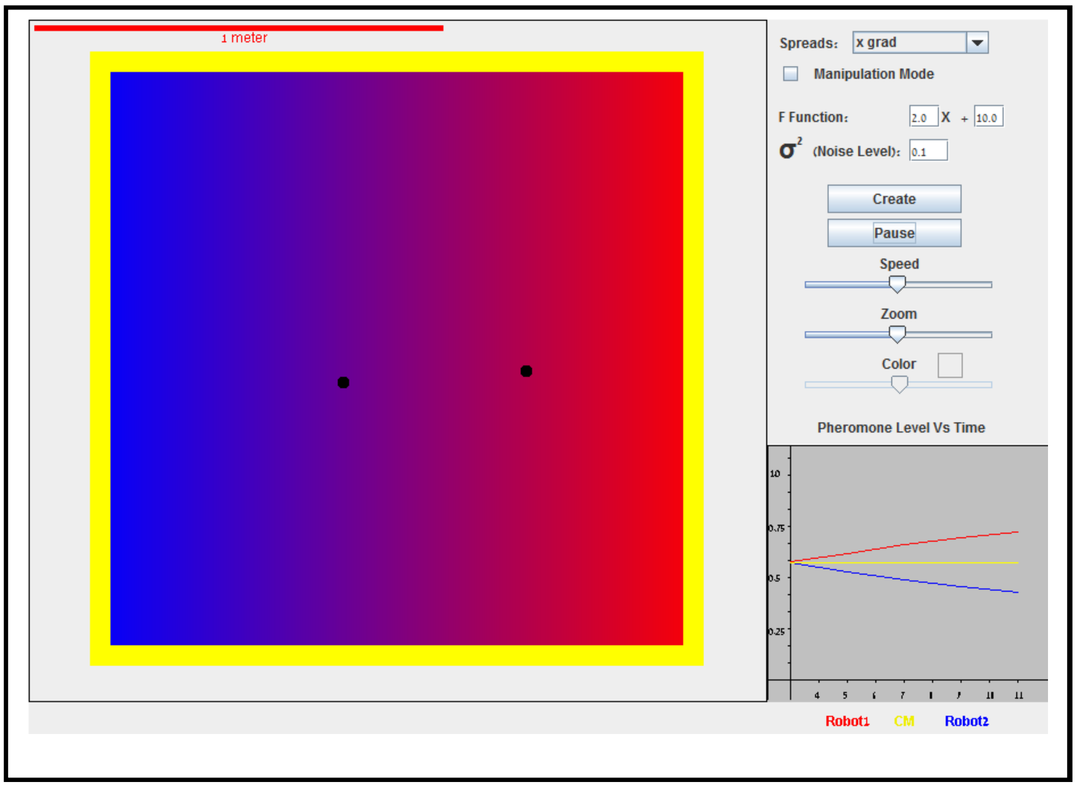
World with manipulations



Playing with Zoom and speed



Graph in spread of peak


Spreads: x grad

F Function: 2.0 X + 10.0

σ^2 (Noise Level): 0.1

With function 2x+10 and noise of 0.1