New York Institute of Technology

NEW YORK INSTITUTE
OF TECHNOLOGY

School of Engineering and Computing Sciences

- Implementation of Declarative Sensor Network

Yu Du

Advisor: Wei Ding

Student name: Yu Du
Student NYIT ID: 0730048

Introduction

Despite years of development for sensor network research, programming of sensor network applications is still very complicated and hard to implement. That's because in most cases sensor network protocols and applications has to be coded in low-level languages, and it must explicitly contend with issues of wireless communication, limited resources, and asynchronous event processing. [1] Implementation of sensor network also required to balance the extensibility and extensibility of sensor communication protocols. Under such situation, even most skillful programmers can hardly implement the sensor network system with short, clear and informative expression if they are using regular languages.

During recent research, there comes a new language which called declarative language. It is a programming language, compiler and runtime system to support declarative specification of wireless sensor network applications.[1] It focus on program outcomes rather that implementation, it makes implementation of sensor network to a simple way: we just have to figure out the logical relation between the input, condition and output , and ignore how the program generate such relation. In such circumstance, implementation of sensor network system becomes much easier.

Prof. Boon Thau Loo, who leads the team from the University of Pennsylvania, developed several declarative language for sensor network[2], such as P2, RapidNet , which are very powerful and have great academic value. At first step of my project, we wonder to implement the project with P2 or Rapidnet as the declarative language. we tried P2 first, but P2 language package developed in 2004 and until now there are no technique support or maintains for it, it cannot be running on the platform even we tried seven different environment.(Ex, redhat 2.0, Ubuntu, Fedora, CentOS)

Then we tried RapidNet, which is a latest declarative language developed by UPENN group, this declarative language offers the API connect to NS-3, which is a well-known simulator for network. But after we installed the NS-3 and RapidNet under Ferdora OS, there is still no technique support for this project and finally the RapidNet cannot run with such environment.

At last we used MATLAB to implement the declarative sensor networks. In this paper, we used MATLAB 12.0 (2010RA).There are several advantages for using MATLAB.

1. Comparing to P2 and RapidNet, MATLAB is more familiar to most programmers. MATLAB is wildly used in academic and research institutions as well as in industry.
2. As a multi-paradigm numerical computing environment, MATLAB is easier to learn, even for new programmers.
3. MATLAB ships with Simulink as a simulator, which is powerful for most applications.

Analysis and Design

We first design the core algorithm of a sensor node communication to its neighbours. In declarative language, every predicate (or rule) is analogous to a table in a database, and tuples are similar to rows in a table. A tuple has many arguments to test whether the rules is true.

A rule instantiates tuples based on the truth of a logical expression. [1] Each rule in declarative language includes a head and a body. The body defines a set of preconditions. if the body is true, the head tuples will be created or executed. For example,

link (@AnyNode, root)
-shortestCost (@AnyNode, root ,COST) .

Above rule indicates if a node has shortest cost and can link to the root with the cost COST, it will link this node to the root. Following the above format of rules, we designed the simple rules to build a sensor network. The rules for tree routing are very simple:

1. Each node can only connected to the nodes in this radio range.
2. Because the range limit, not every node can directly connect to root. So we divided the whole sensor network to three range level- the nodes can directly connect to the root. A node can connect to another node that links to the root. Or a node is located at the edge of the network.
3. Every node is connected another node with higher level range and hence has the lowest cost.

Use declarative language, code should be:

path(@source,root,root,Cost):-dest(@Source,root)
link(@Source,root,Coost)

path(@source,root,parent,Cost):-dest(@Source,root)
shortestcost(@source,parent,parent,Cost)
link(@Source,parent,Cost1)
link(@parent,root,Cost2)

After the tree generation is implemented, we simulated how nodes create their link to different destination in declarative language to. For building links, the rules are:

1. If the two nodes are parent and child, they will directly create a link to each other.
2. If the two nodes can directly link to root, build a path between node1 to root and node2 to root.
3. If the nodes are not belong to situation 1 and situation2, find their shortest path to the nearest parent.

Using declarative language, code should be:

path (@Source , Dest , parent , Cost) :- dest (@Source , Dest),
link (@Source , parent , Cost1) ,
link (@parent , Dest, Cost2) ,
nextHop (@Source, parent, parent, Cost1) ,
nextHop (@parent , Dest , Dest , Cost2) ,
Cost=Cost1+Cost2

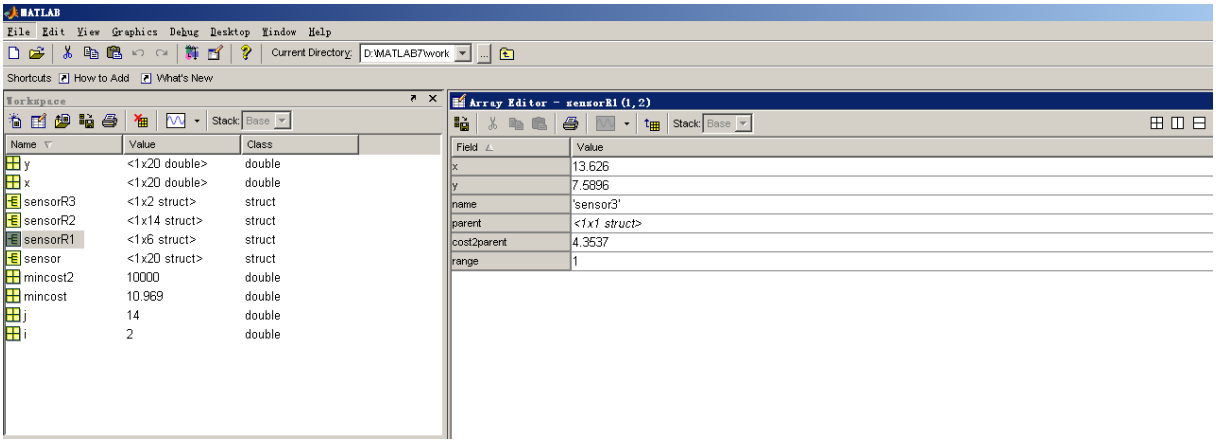
After designing the rules and algorithms for the sensor network, we can begin the coding implementation for the project.

Implementation

First we write a model for testing tree routing. For more concise expression of the program logic, we used struct in MATLAB. For initial a tree generation, core code is given below.

```
unction u= treegen()
%sensor= struct('x',{},'y',{},'name',{},'parent',{},'cost2parent',{},'range',{});
%sensorR1=struct('x',{},'y',{},'name',{},'parent',{},'cost2parent',{},'range',{});
%sensorR2=struct('x',{},'y',{},'name',{},'parent',{},'cost2parent',{},'range',{});
%sensorR3=struct('x',{},'y',{},'name',{},'parent',{},'cost2parent',{},'range',{});

sensor= struct('x',0,'y',0,'name',0,'parent',0,'cost2parent',0,'range',0);
sensorR1=struct('x',0,'y',0,'name',0,'parent',0,'cost2parent',0,'range',0);
sensorR2=struct('x',0,'y',0,'name',0,'parent',0,'cost2parent',0,'range',0);
sensorR3=struct('x',0,'y',0,'name',0,'parent',0,'cost2parent',0,'range',0);
for i=1:19

   x(i)=unifrnd(0,20)
   y(i)=unifrnd(0,20)
   sensor(i).x=x(i);
   %assignin('base','sensor(i).x',x(i));
   sensor(i).y=y(i);
   sensor(i).name=strcat('sensor',num2str(i));

end
x(20)=10;
y(20)=10;
 sensor(20).x=x(20);
 sensor(20).y=y(20);
 sensor(20).name=strcat('sensor',num2str(20));
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.

for i=1:19
   if dist(sensor(i),sensor(20))<5
      sensor(i).range=1;
      sensor(i).cost2parent=dist(sensor(i),sensor(20));
      sensor(i).parent=sensor(20);
      sensorR1(length(sensorR1)+1)=sensor(i);
   elseif dist(sensor(i),sensor(20))>5 & dist(sensor(i),sensor(20))<10
      sensor(i).range=2;
       sensorR2(length(sensorR2)+1)=sensor(i);
   else sensor(i).range=3;
      sensorR3(length(sensorR3)+1)=sensor(i);
   end

end
```

```
for i=1:length(sensorR2)
   mincost=10000;
   for j=1:length(sensorR1)
      if dist(sensorR2(i),sensorR1(j))<mincost
         sensorR2(i).cost2parent=dist(sensorR2(i),sensorR1(j));
         sensorR2(i).parent=sensorR1(j);
         mincost=dist(sensorR2(i),sensorR1(j));
      end
   end
end

for i=1:length(sensorR3)
   mincost2=10000;
   for j=1:length(sensorR2)
      if dist(sensorR3(i),sensorR2(j))<mincost2
         sensorR3(i).cost2parent=dist(sensorR3(i),sensorR2(j));
         sensorR3(i).parent=sensorR2(j);
         mincost=dist(sensorR3(i),sensorR2(j));
      end
   end
end
```



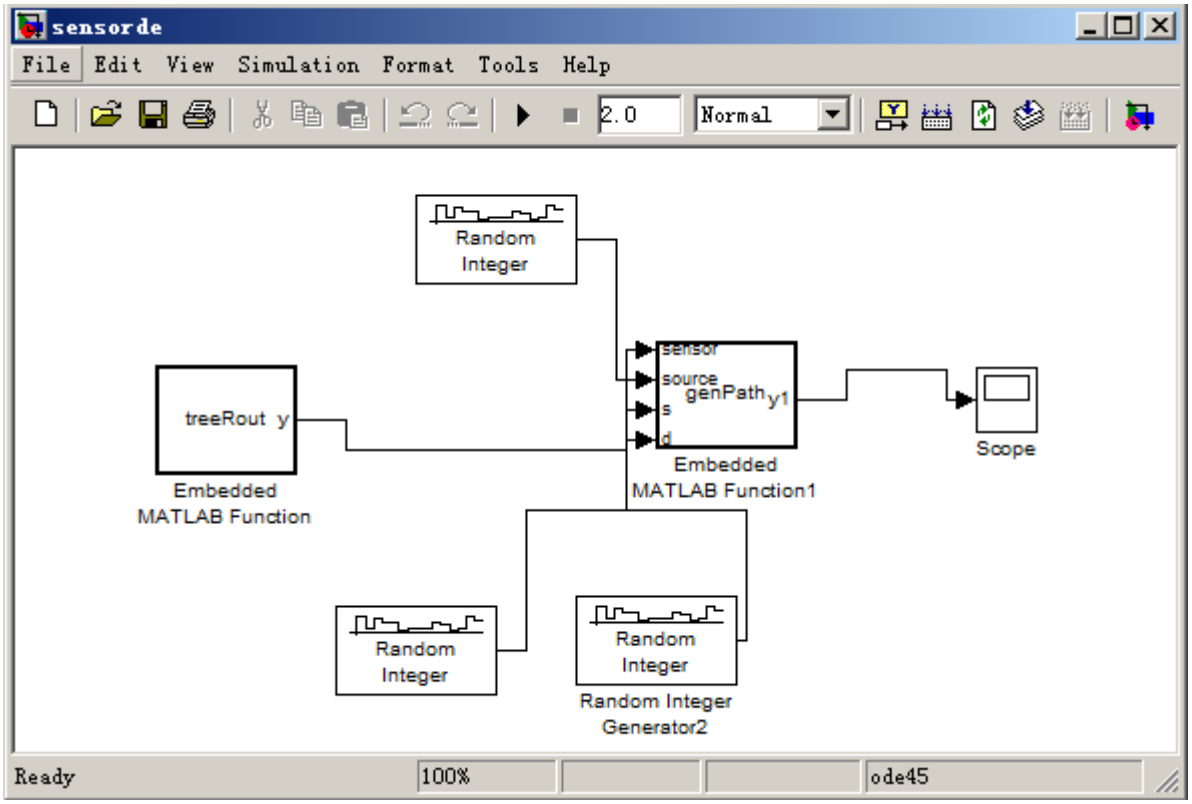
First, we created a structure for each sensor node, and then we created the parent-child relationship. We can check the result by executing the tree generation function. After verifying the correctness of results, we defined the algorithm for path generation. The core part for path generation is based on the cost to each neighbour, attached with some core part of code:

```
function y = treegen(u)
%clear all;
% This block supports an embeddable subset of the MATLAB language.
% See the help menu for details.
sensor=zeros(50,7);
for i=1:50
   sensor(i,1)=i;
```

```
end
for i=1:49

   sensor(i,2)=unifrnd(0,50)
   sensor(i,3)=unifrnd(0,50)

end

sensor(50,2)=25;
sensor(50,3)=25;

for i=1:49
   if sqrt((sensor(i,2)-sensor(50,2))^2+(sensor(i,3)-sensor(50,3))^2)<12
      sensor(i,4)=1;
      sensor(i,5)=50;
      sensor(i,6)=sqrt((sensor(i,2)-sensor(50,2))^2+(sensor(i,3)-sensor(50,3))^2);
   elseif sqrt((sensor(i,2)-sensor(50,2))^2+(sensor(i,3)-sensor(50,3))^2)>12 &&
sqrt((sensor(i,2)-sensor(50,2))^2+(sensor(i,3)-sensor(50,3))^2)<25
      sensor(i,4)=2;
   else
      sensor(i,4)=3;
   end
end

for i=1:49

   if sensor(i,4)==2
      min4r2=10000;
      for j=1:49
         if sensor(j,4)==1
            if sqrt((sensor(i,2)-sensor(j,2))^2+(sensor(i,3)-sensor(j,3))^2)<min4r2
               min4r2=sqrt((sensor(i,2)-sensor(j,2))^2+(sensor(i,3)-sensor(j,3))^2);
               sensor(i,6)=min4r2;
               sensor(i,5)=sensor(j,1);
            end
         end
      end
   end
end

for i=1:49

   if sensor(i,4)==3
      min4r3=10000;
      for j=1:49
         if sensor(j,4)==2
            if sqrt((sensor(i,2)-sensor(j,2))^2+(sensor(i,3)-sensor(j,3))^2)<min4r3
               min4r3=sqrt((sensor(i,2)-sensor(j,2))^2+(sensor(i,3)-sensor(j,3))^2);
```

```
                sensor(i,6)=sqrt((sensor(i,2)-sensor(j,2))^2+(sensor(i,3)-sensor(j,3))^2);
                sensor(i,5)=sensor(j,1);
            end
        end
    end
end

%model2
y = sensor;
x=sensor(:,2);
y=sensor(:,3);
plot(x,y,'o');
for i=1:49
    hold on;
    for j=1:50
        if sensor(j,1)==sensor(i,5)
            plot([sensor(j,2),sensor(i,2)],[sensor(j,3),sensor(i,3)],'b');
        end
    end
end

%model3
hold on;
source= randint(1,1,[0,49]);

for i=1:49
    if sensor(i,1)==source
        sensorS=sensor(i,:);
    end
end
plot(sensorS(2),sensorS(3),'*');
hold on;
while(sensorS(1)~=50)
    for j=1:50
        if sensorS(5)==sensor(j,1)
            plot([sensor(j,2),sensorS(2)],[sensor(j,3),sensorS(3)],'R');
            sensorS=sensor(j,:)
            hold on;
            break;
        end
    end
end
```

Upon the implementation of the algorithm, we designed the model in MATLAB Simulink to simulate the sensor network. The whole model is illustrated as below:

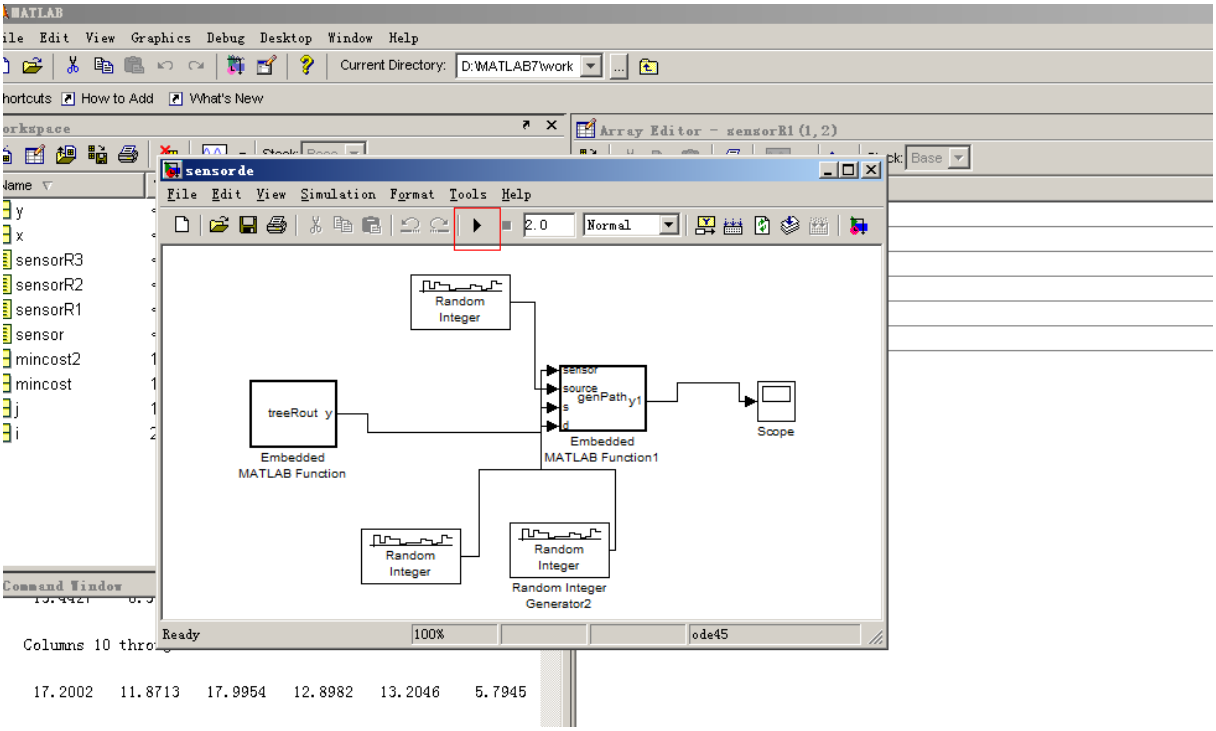


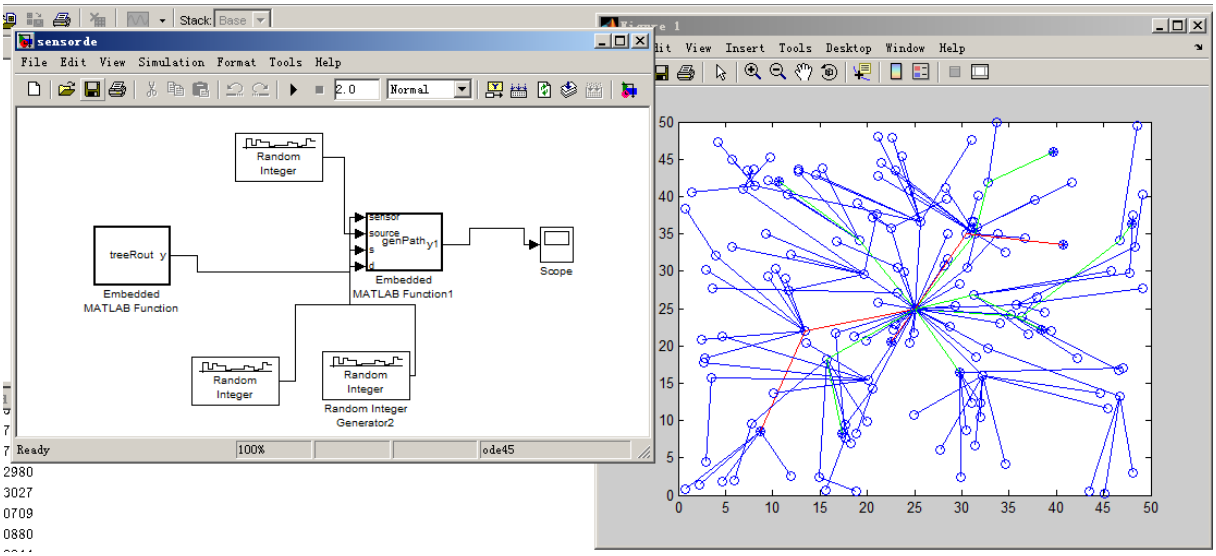
Here:

- treeRout: the generate the tree for the network.
- Random Integer: as the input source to generate data to simulate as a sensor node
- genPath: to create path between the nodes.
- scope: out put the result.

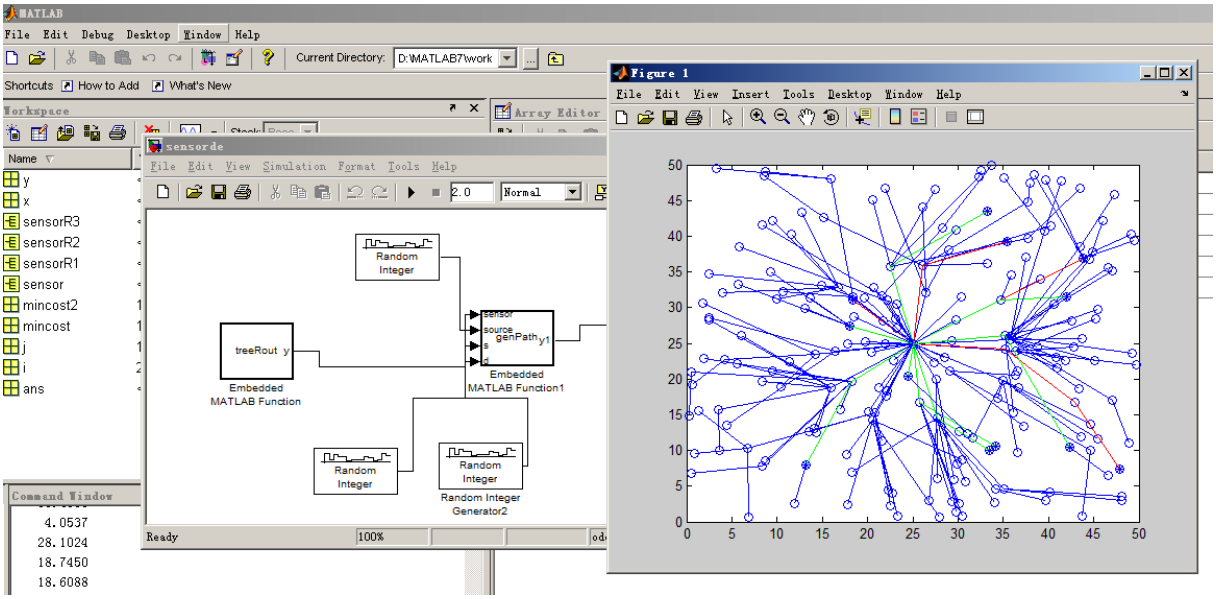
Testing and result

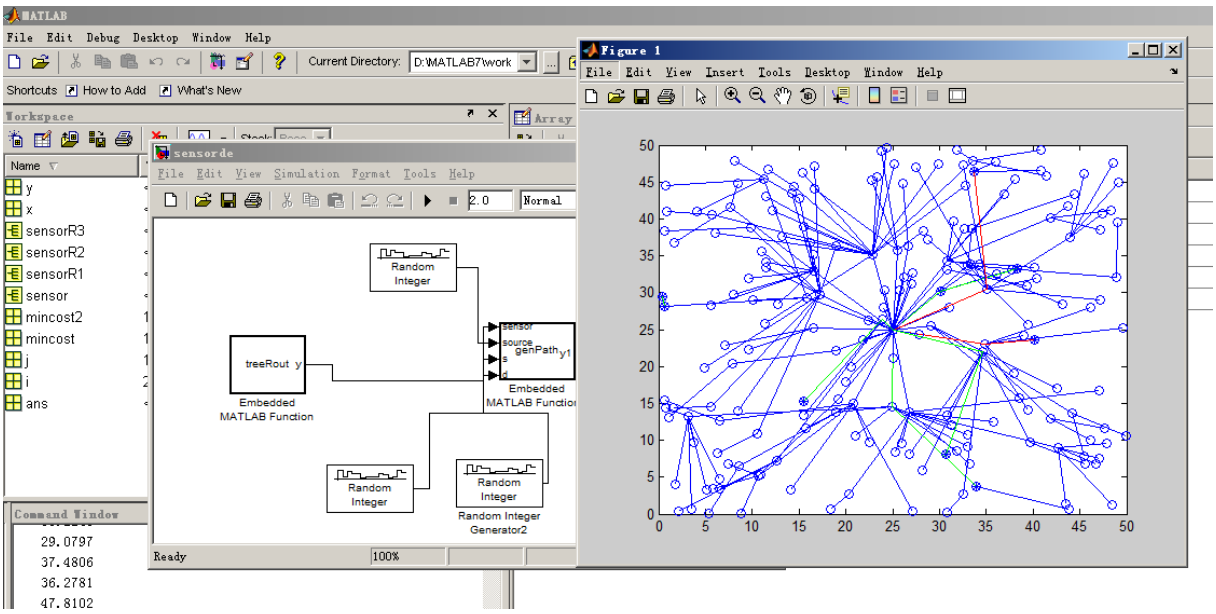
First the test model is executed, then the simulator is executed. The result shows below.





blue line -- for the tree routing
red line-- path from sensor node to root
Green line--path between different nodes





Conclusion, Evaluation and Further Work

This project can simulate the path and tree routing function with MATLAB/ Simulink. We also tried to use recursion to implement the algorithm, the basic logical for recursion is given below.

```
function findshortestpath(node1, node2)
if node1. neighbor!= node2
 findshortestpath(node1.neighbor,node2)
```

This algorithm can also implement the simple sensor network if we got few sensors(less than 20), when we test this algorithm with 50 sensor nodes, It often causes dead loop in recursive or memory overflow(MATLAB can contain 500 recursive function at mean time).

If possible, this program can extend by adding the 'real' sensors in the model because MATLAB 7 also support joint sensor and body sensor simulator in the Simulink, but that component requires the knowledge of signals and waves. We would like to go further once we have such knowledge.

Reference

[1] David Chu, Lucian Popa, Arsalan Tavakoli, Joseph M. Hellerstein, Philip Levis†, Scott Shenker, Ion Stoica. The Design and Implementation of a Declarative Sensor Network System, Proceedings of the 5th international conference on Embedded networked sensor systems, pages 175-188, 2007.

[2] Loo, B. T., Condie, T., Garofalakis, M., Gay, D. E., Hellerstein, J. M., Maniatis, P., Stoica, we (2009). Declarative networking. Association for Computing Machinery.Communications of the ACM, 52(11), 87. Retrieved from http://search.proquest.com/docview/237067019?accountid=12917

[3] Mao, Y., Loo, B. T., Ives, Z., & Smith, J. M. (2012). MOSAIC: Declarative platform for dynamic overlay composition. Computer Networks, 56(1), 64. Retrieved from http://search.proquest.com/docview/912942908?accountid=12917

[4] Yaman Sirin Fusun, Nau, Dana S . Declarative Reasoning about Moving Objects. Digital Repository at the University of Maryland University of Maryland (College Park, Md.) 28-Aug-2006