6.894 PSET # 5 ,Pervasive Computing

Fang Hui g0404400@nus.edu.sg

Nov.13,2004

Files:

Fh.py: Main program
Fhgraph.py: Graph data structure and searching
Fhcricket.py: Cricket handler interface and beacon data structure
Gui.glade: Graphical interface in XML format
./maps & ./graphs: Map and graph raw data

Configuration:

1. Cricket listener is connected to i-PAQ by serial port. The two Beacons are laid on the two up corners of the map.
2. The cricketd host ip address and port are defined in the source file "fhcricket.py" Line 73:
   class CricketHandler:

   def __init__(self,host="127.0.0.1",port=2947):
   ……
3. The map file: Here we take 2.png and 2.graph as test case. Since we have to know the beacon's location in advance to compute the exact position of listener in one plane, we assume the first beacon is at (0,0) , and the second is at (width,0). The coordination's of two beacons are defined in the file "fhcricricket.py" Line197:
   b1.set_xy(0,0)
   b2.set_xy(2178,0)
   Moreover, to correctly display in the zoomed map on our i-PAQ screen, the scale ratio is also define in the Line191:
   proportional = 2

Running Steps:

1. Run */usr/bin/start-cricketd* on i-PAQ. You can verify if the cricket is working by *telnet cricketd-server-ip-address 2947*.
2. Run *python2.3 fh.py*
3. Put the listener on the map and move to different sites. And view the path result shown on the i-PAQ screen. The correct path should be from last position to new position, in the closest distance meaning. (Consider the inaccuracy of the RF-ultrasound working mechanism).

Note: if you pull the mouse on the map, the (x, y) coordination will be displayed automatically in the text entry above in the menu, which can facilitate your view on the location accuracy.

Implementation:

Main program entrance:

```
if __name__=="__main__":

    app = gui()

    if  CRICKET_ON :
        # Setup socket to recv data from cricketd host which contacts listener by serial
line
        cricket_handler = fhcricket.CricketHandler()

        # register the callback function “set_current_location”
        cricket_handler.register_callback(app.set_current_location)

        # whenever there’s msg received from socket, call the “handle_input” function,
which will call again “set_current_location” later.
        gtk.input_add(cricket_handler.socket, gtk.gdk.INPUT_READ,
cricket_handler.handle_input)

        # idle operations
        gtk.idle_add(cricket_handler.idle)

    gtk.main()
```

So the “*cricket_handler.handle_input( )*” will be called each time *cricketd* sends distance message to program.

At first each line message like

```
$Cricket2,ver=3.0,space=MIT19,id=20,dist=57,duration=05
$Cricket2,ver=3.0,space=MIT19,id=20,dist=5C,duration=37
$Cricket2,ver=3.0,space=MIT19,id=20,dist=5C,duration=49
                  … …
```

Will be parsed in “ *handle_line( )*” function. And store the key-value information into a dictionary. If the beacon name (space) and distance (dist) from beacon < - > listener/ipaq are obtained, it will update the beacon readings:

self.new_beacon_reading(space, distance)

, and do

update_beacons( )

Which actually computes listener’s position by triangulating the two beacons and listener itself, if the beacons and listener information is complete.

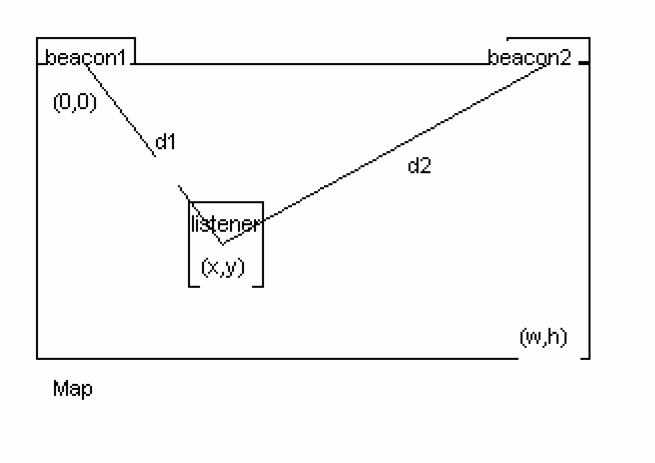
new_location = self.triangulate(b1,b2)

Then decide if it is in moving state or standstill. If the listener is in new position and standstill, i.e. stable, the callback function

set_current_location( )

Will do the left path finding work and display the new path dynamically.

Description on triangulation
As shown in the figure below, we have obtain the distance from beacon1 and beacon2, respectively (Here I adjust the distance by *mathematical "mean"* method).



Thus we have two equations:

$$x^2 + y^2 = d1^2$$
$$(x-w)^2 + y^2 = d2^2$$

The solution is simple:

$$x = w^2+d1^2-d2^2 / 2w$$
$$y = sqrt(d1^2 - x^2)$$

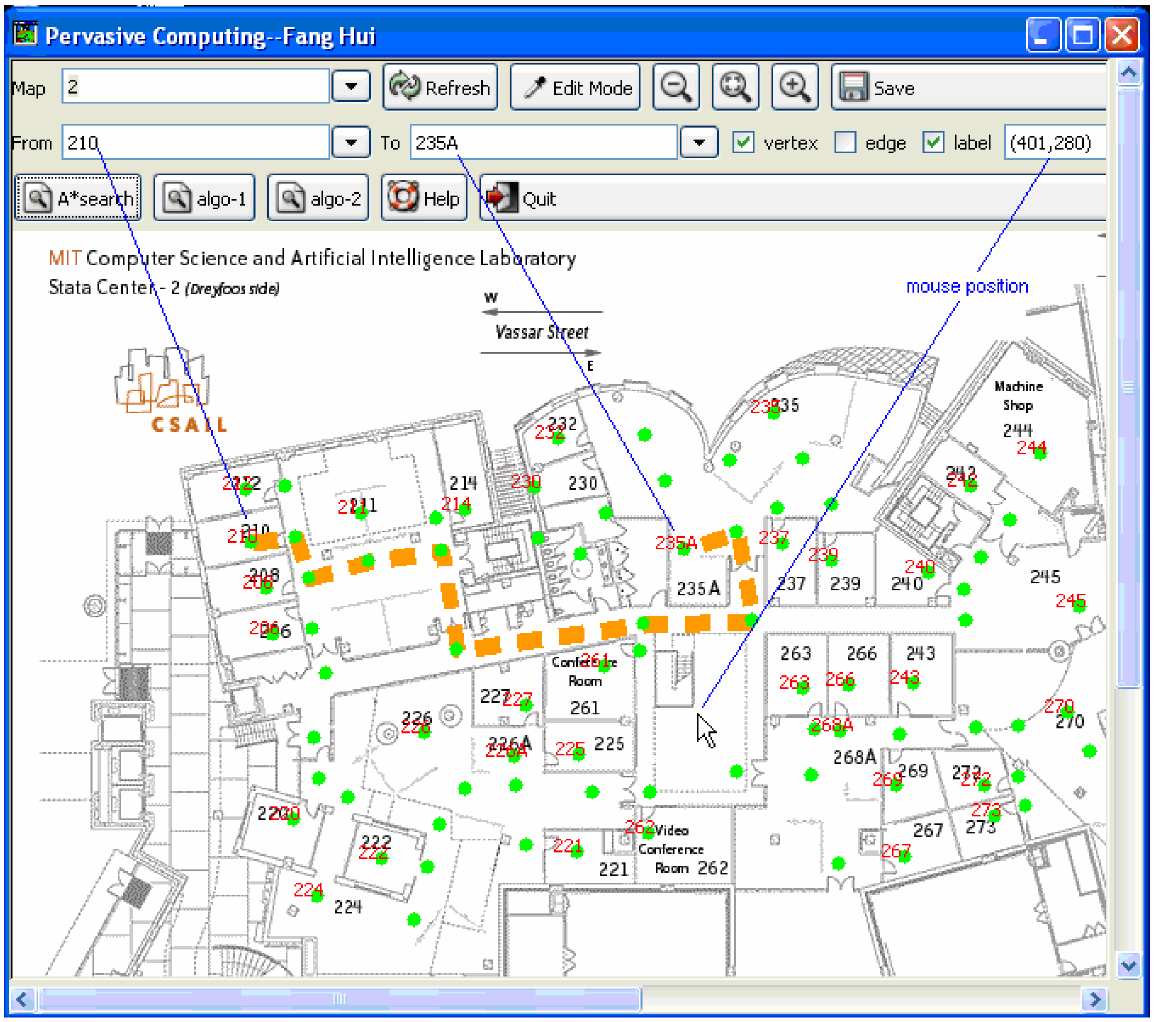
The position will be scaled into the size of map displayed in i-PAQ.

Description of set_current_location:
Actually this operation is very similar with the mouse operation to fulfill the selection of starting point and ending point, which is already implemented by me in the previous problem set. You can also now select the point by just left-clicking the vertex existing in the map, under "search-mode". (The "edit-mode" is when you press down the "edit-mode" button. By default, it is search-mode.)

So the mouse selecting start-end points and cricket handler use the same back function, when we consider each new current location information delivered from the cricket was regarded as one mouse click on the map.

The rule is (refer to fh.py Line 395):

1) When you point to one empty area without vertex specified, I assume you de-select both points.
2) When you do point to one vertex, and neither the starting point nor ending point selected, you set the starting point.
3) When you do point to one vertex, and only ending point not yet selected, you set the ending point.
4) When you do point to one vertex, and the ending point already selected, I assume you try to find the path from previous ending point to new position.


Pervasive Computing--Fang Hui
Map 2
Refresh
Edit Mode
Save
From 210
To 235A
vertex
edge
label
(401,280)
A*search
algo-1
algo-2
Help
Quit
MIT Computer Science and Artificial Intelligence Laboratory
Stata Center - 2 (Dreyfoos side)
mouse position
Vassar Street
CSAIL
Machine Shop
Conference Room
Video Conference Room