Probabilistic Robotics
Map Building in 2D

양정연

2020/12/10

Map Update Strategy with Inverse measurement(or Sensor) model

$$l(m_i \mid x_{1:t}, z_{1:t}) = l(m_i \mid x_t, z_t) + l\left(m_i \mid x_{1:t-1}, z_{1:t-1}\right) - l(m_i)$$

$$l_t = l(m_i \mid x_t, z_t) + \mathrm{l}_{t-1} - l_0\left(m_i\right)$$

$$= l(m_i \mid x_t, z_t) + \mathrm{l}_{t-1} - l_0$$

$$t > 0 : p(m_i \mid x_{1:t}, z_{1:t}) \rightarrow l_t$$

$$t = 0 : p(m_i) \rightarrow l_0$$

- If we calculate $l(m_i \mid x_t, z_t)$, then we update a map

- It is called

  $l(m_i \mid x_t, z_t) \rightarrow p(m_i \mid x_t, z_t) = \text{Inverse measurement model}$

  – From the current position, x and the current measurement, z, we estimate prob. Of whether a map mi is empty or occupied.

  → It is different with a classification probability.

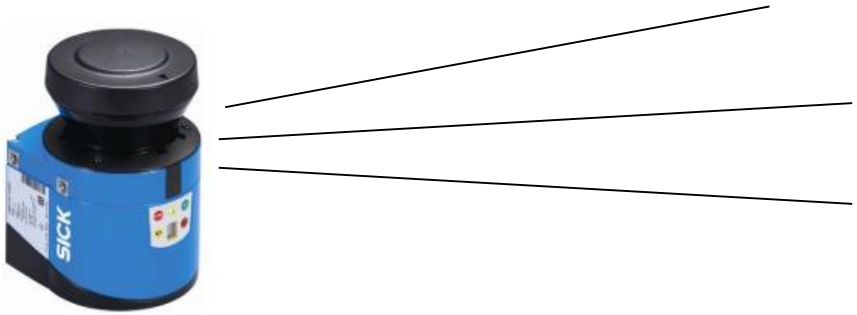
Extend Map Update into 2Dim

- Map update with Inverse Sensor Model

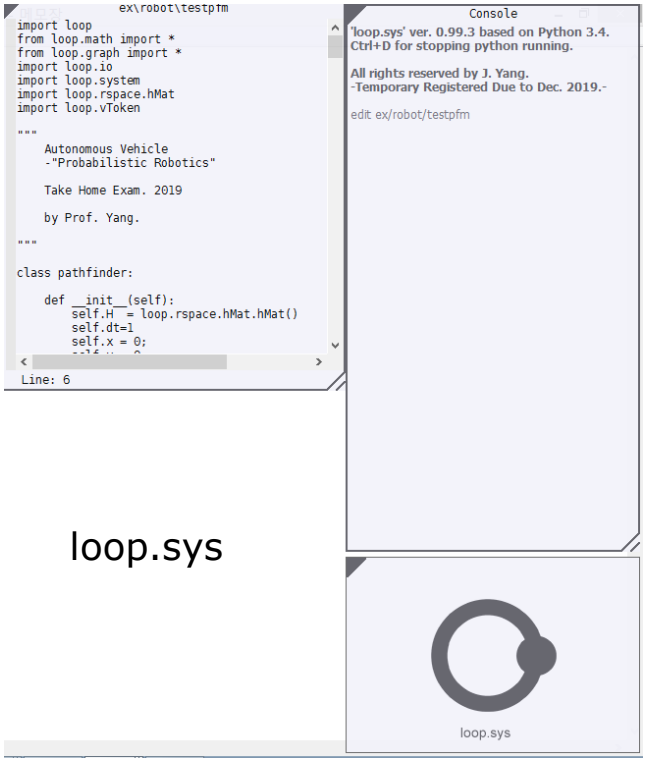
$$l_t = l(m_i \mid x_t, z_t) + l_{t-1} - l_0$$
$$= \text{InverseSensorModel}(\text{m}, \text{x}, \text{z}) + l_{t-1} - l_0$$

- SLAM uses Distance Metric Sensor like laser scanner

- Try to understand Distance Metric Sensor



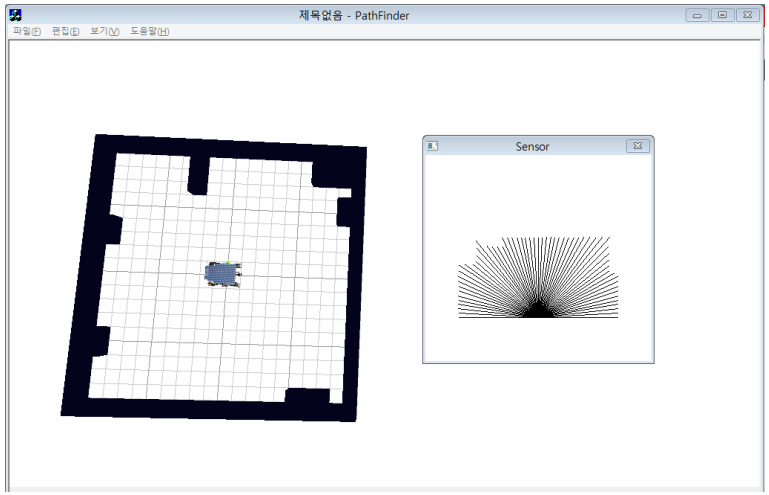
Two Programs are Used



loop.sys

Move(x,y,q)

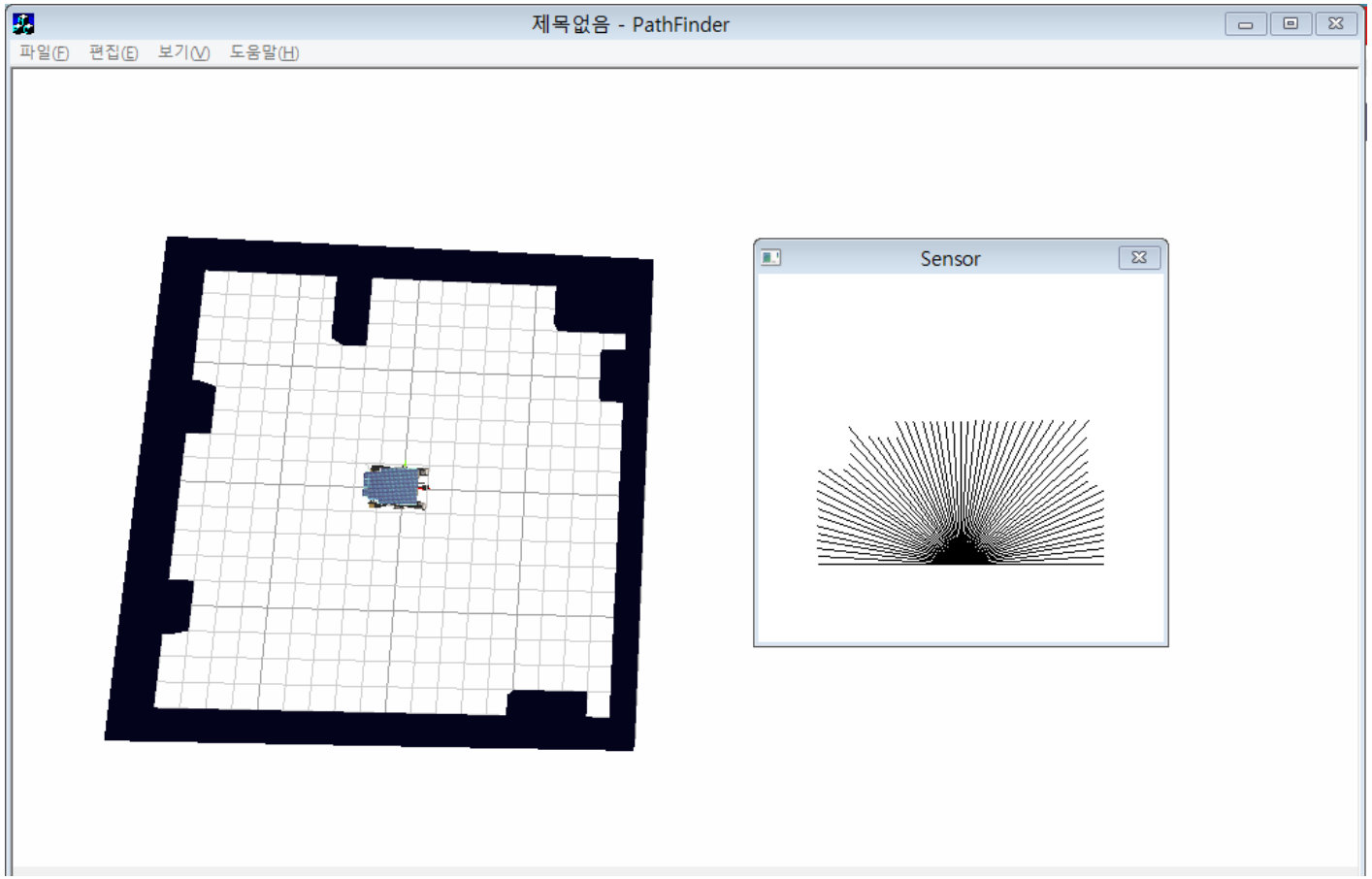
Sensor
Distance
information



Path finder 3D environment
Pathfinder.exe

- loop.sys is a Python-based program is communicated with Pathfinder.exe
  - Internal process communication (ICP)

Example: Path Finder with Laser Scanner



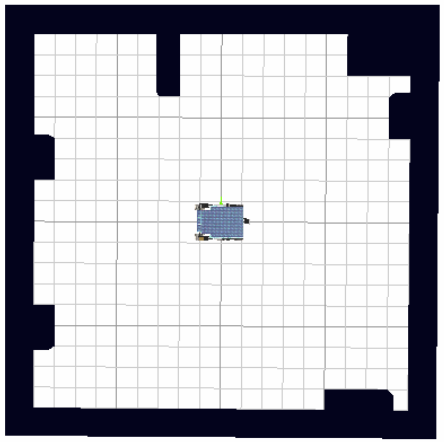
- Add walls and Distance Sensor

1. Wall and Sensor Model from XML

- Define wall, grid, and sensor model from “config.xml”

- Redefine Grid world, with 20x20

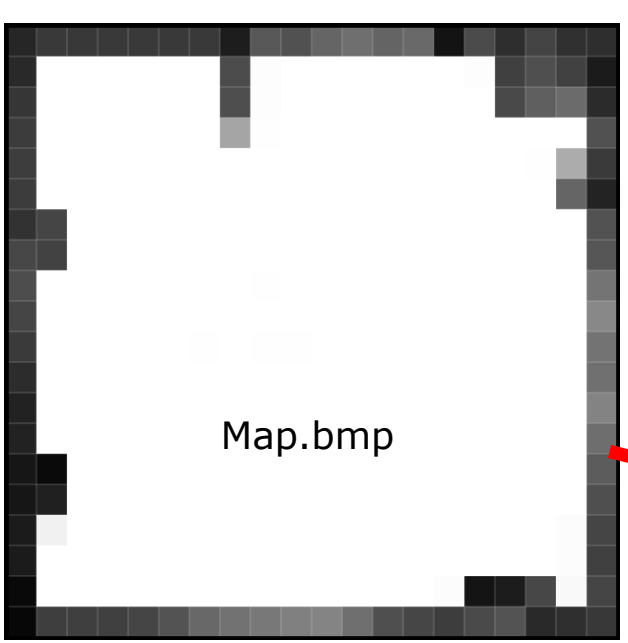
```
<config>
    <map path="map.bmp">
        <space w=20 h=20 z=20/>
        <grid w=21 h=21/>
    </map>

    <sensor start=-90 end=90 resolution=3>
    </sensor>

</config>
```



W=20
Grid =21

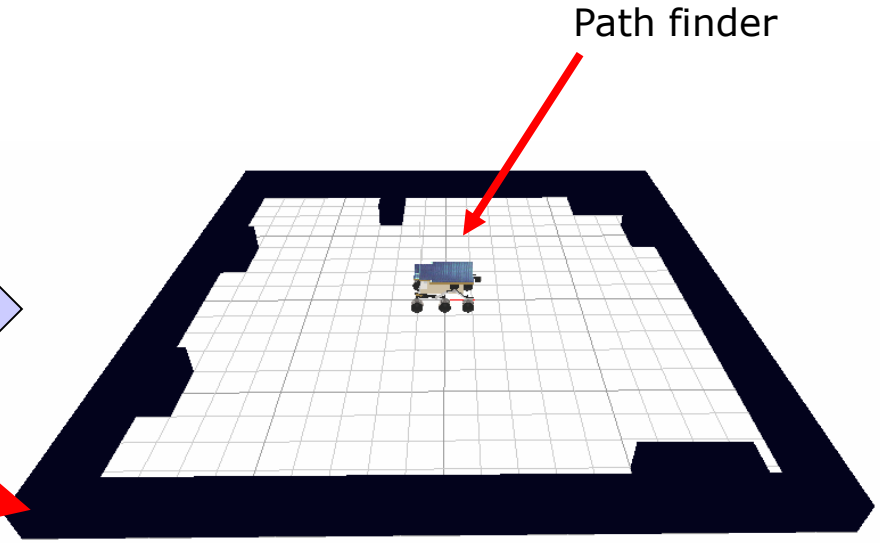
2. Add Walls

- Use a virtual map (“map.bmp”)



2D map with Photoshop

3D wall

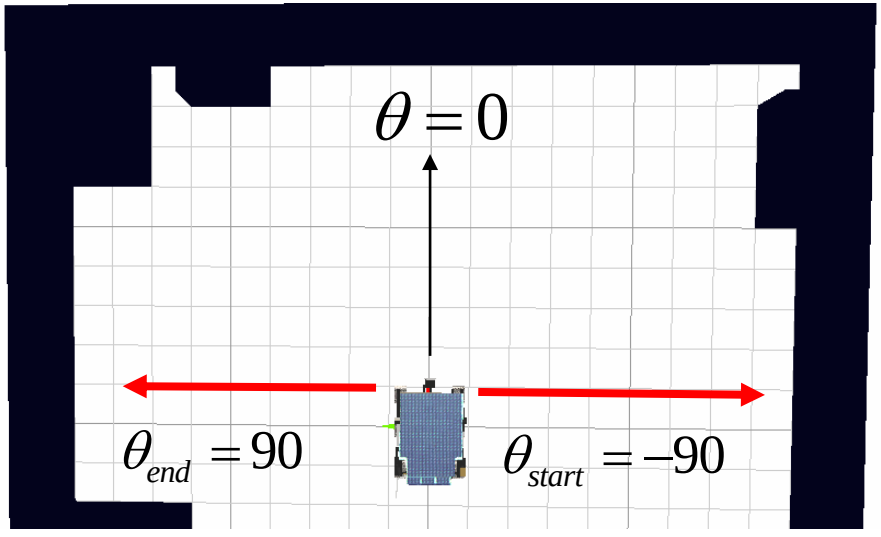


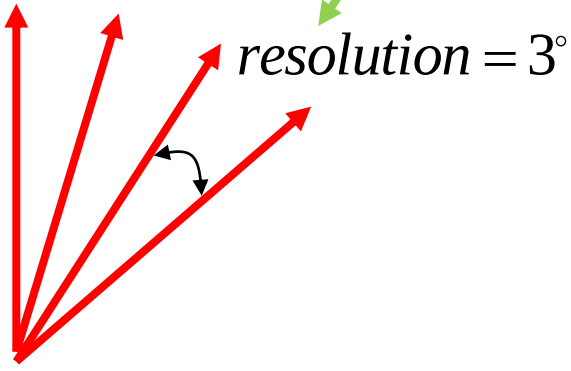
3D walls from 2D image

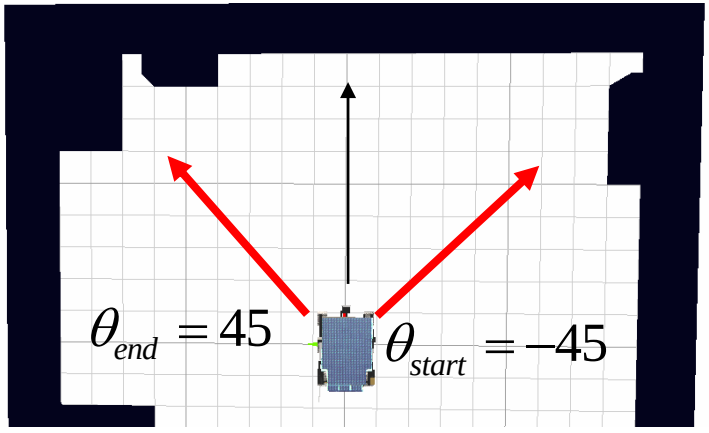
3. Virtual Sensor

- Define Sensor type from “config.xml”

```
<config>
    <map path="map.bmp">
        <space w=20 h=20 z=20/>
        <grid w=21 h=21/>
    </map>

    <sensor start=-90 end=90 resolution=3>
    </sensor>

</config>
```



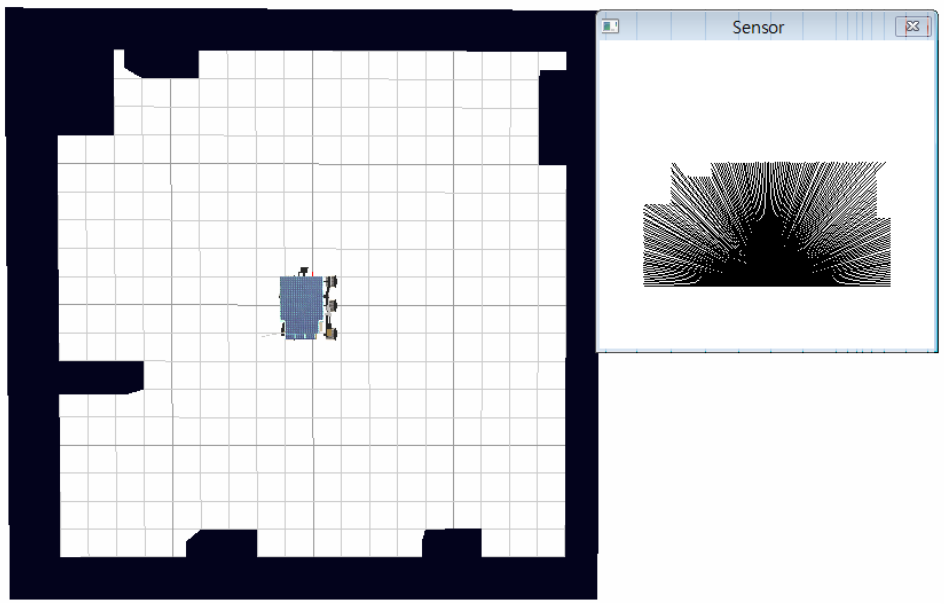


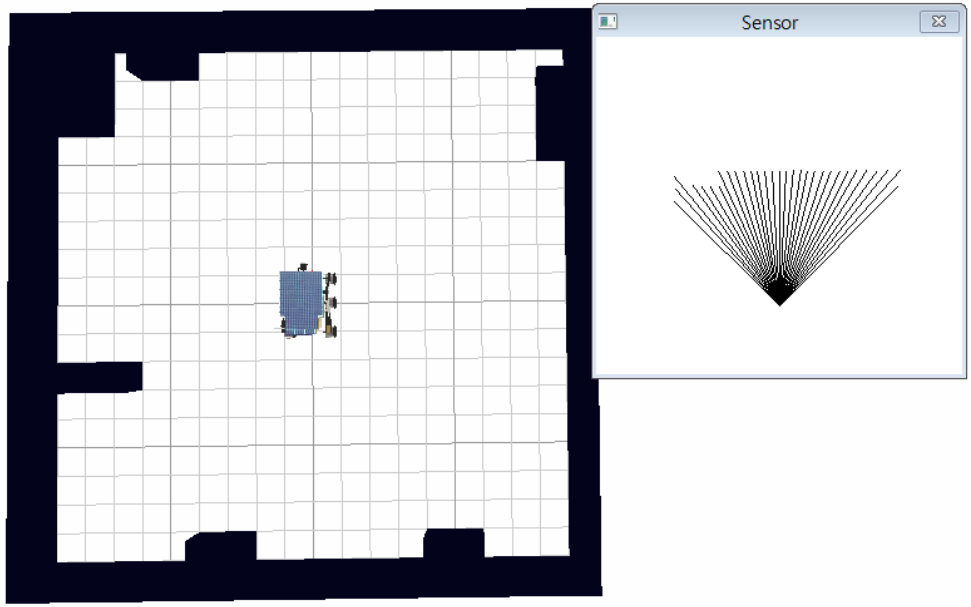


3. Virtual Sensor Data

```
<sensor start=-90 end=90 resolution=1>
</sensor>
```

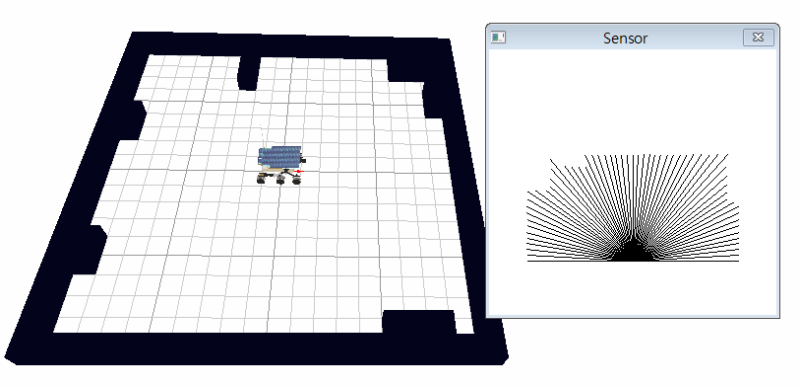
```
<sensor start=-45 end=45 resolution=3>
</sensor>
```

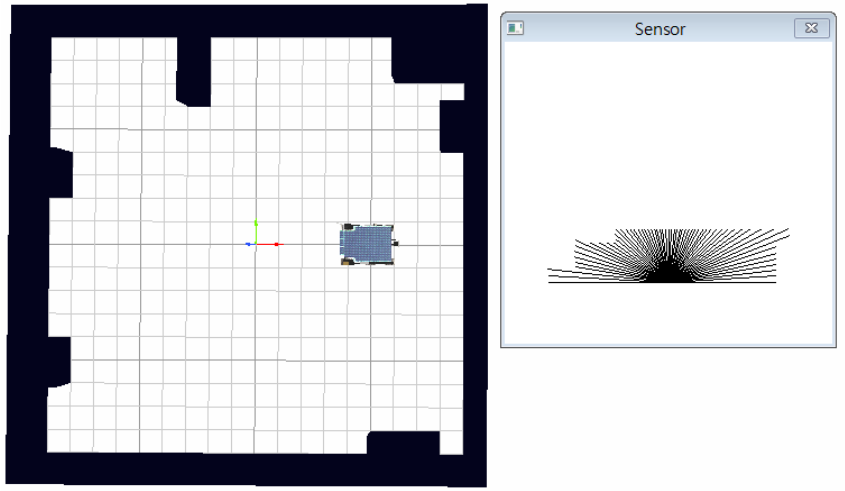




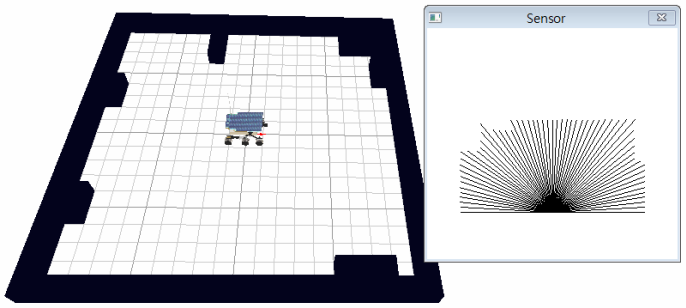
- High Quality Sensor has 0.5 degree resolution
  - Distance error is about 1mm.

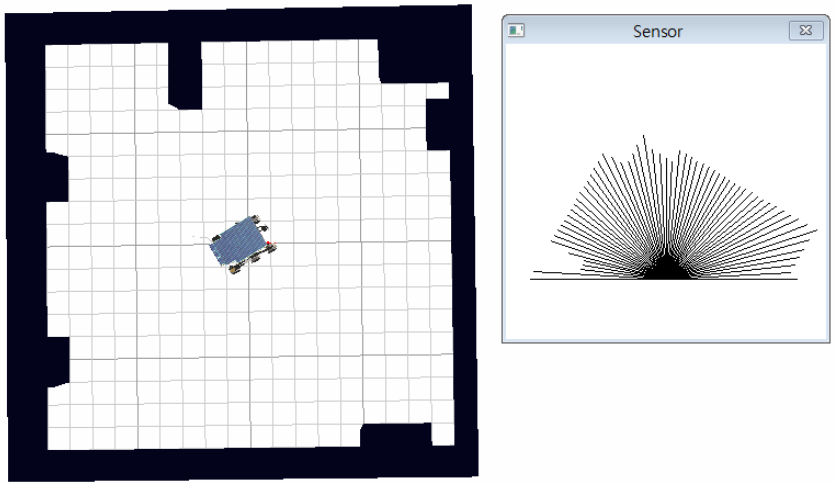
Ex1) Distance Metric with Robot Movements





- testpfm.pf.move(0,0,0) → testpfm.pf.move(5,0,0)





- testpf.pfm.move(0,0,0) → testpf.pfm.move(0,0,30)

2D Map Update

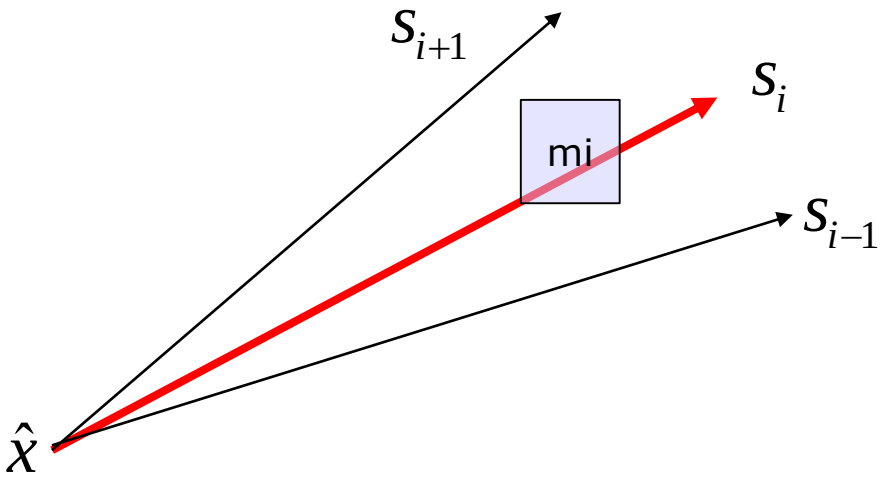
- Log Odds-based Map Update

$$l(m_i \mid x_{1:t}, z_{1:t}) = l(m_i \mid x_t, z_t) + l\left(m_i \mid x_{1:t-1}, z_{1:t-1}\right) - l(m_i)$$

$$\therefore l_t = l(m_i \mid x_t, z_t) + l_{t-1} - l_0$$

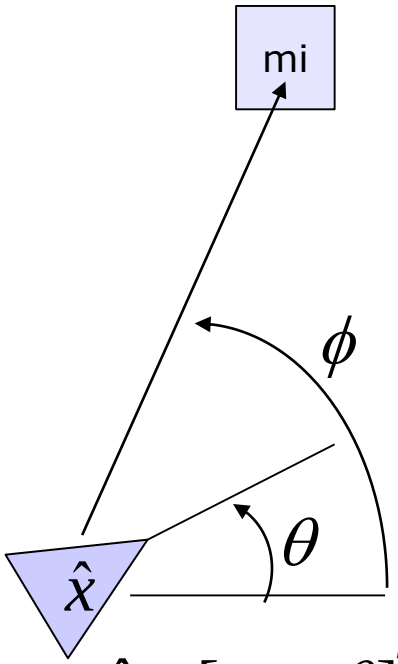
$$= InverseSensorModel + l_{t-1} - l_0$$

- Suppose that One Map, m_i with one Distance Sensor, s_i



Check Condition m_i and s_i

- 1. Find the angle, ϕ $\quad \phi = \text{atan2}(m_{ix} - x, m_{iy} - y)$



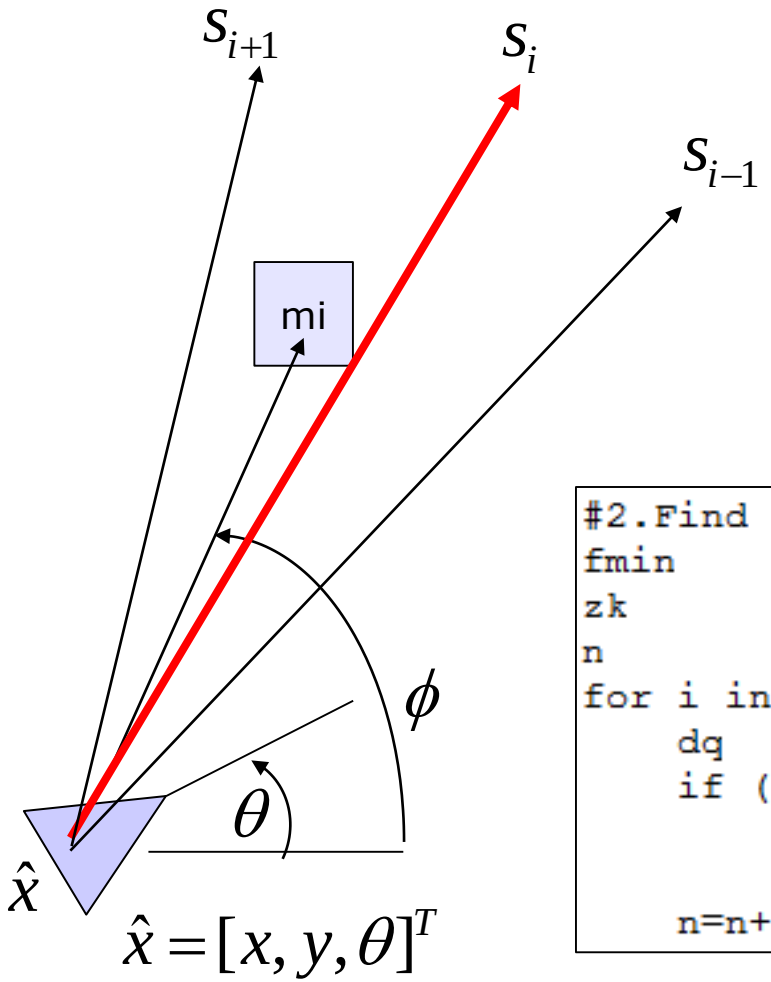

```
#1. Get angle from a robot to map,mi
dx   = mi.x-self.x
dy   = mi.y-self.y
r    = sqrt(dx*dx+dy*dy)
pi   = DEG(math.atan2(dy,dx))
pi   = DPI(pi)
```

$$r = \| m_i - \hat{x} \| = \sqrt{(m_{i,x} - x)^2 + (m_{i,y} - y)^2}$$

$\hat{x} = [x, y, \theta]^T$ $\quad \theta$: heading angle

Check Condition m_i and s_i

- 2. Find the best closest Sensor



$$i = \arg\min(|\theta + s_i - \phi|)$$

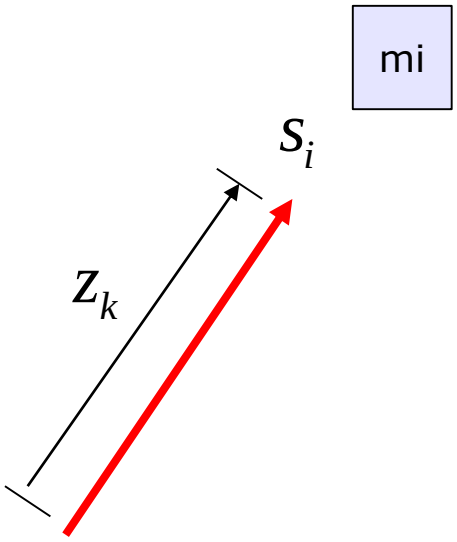
$$\rightarrow s_i$$

$$z_k = norm|$$  $$|$$

```
#2.Find the closest Distance
fmin     = 1e10
zk       = 0;
n        = 0;
for i in range(self.s,self.e+self.dq,self.dq):
    dq   = ABS(pi-i)
    if (fmin>dq):
        fmin     = dq
        zk       = dist[n]
    n=n+1
```

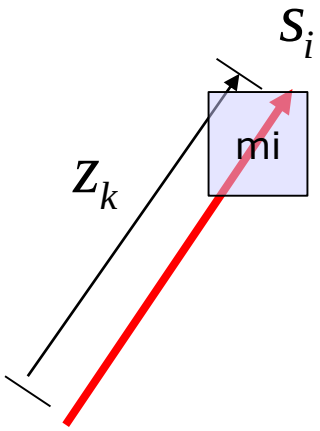
Use Sensor Model
3. Check Distance

Case 1) We Don't Know it



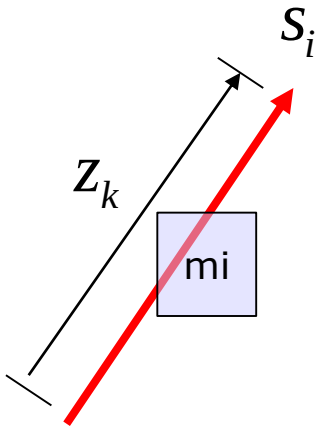
Map, mi is out of sensor area

Case 2) Occupancy Update



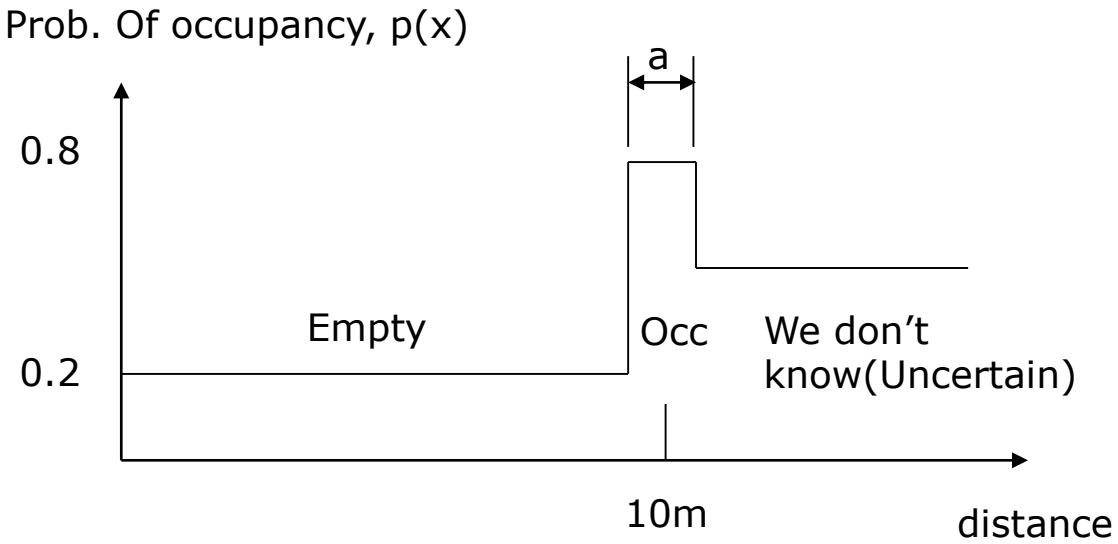
Map, mi is similar to sensor data

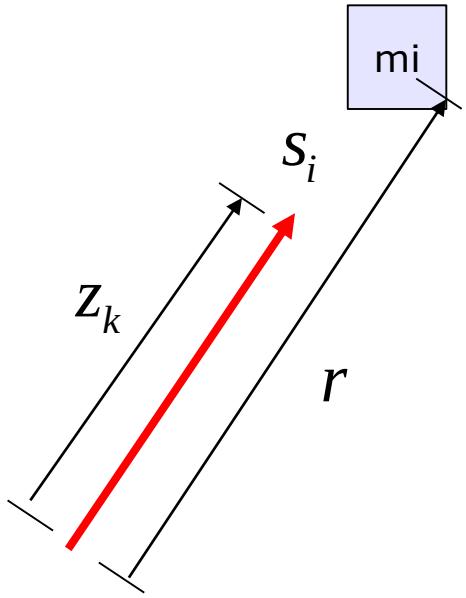
Case 3) Emptiness Update



Map, mi is inside of sensor area

Sensor Probability Model



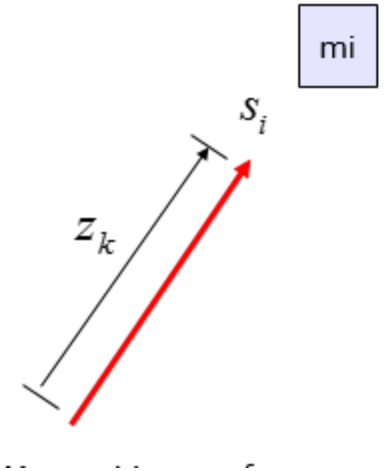



```
#3. check distance=(o-mi) < zk
if (r>zk+self.a/2):
    return self.l0          ——————→  Uncertain case

if (ABS(zk-r)<self.a/2):
    return self.lo          ——————→  Occupant case
if (r<zk):
    return self.le;         ——————→  Empty case
```

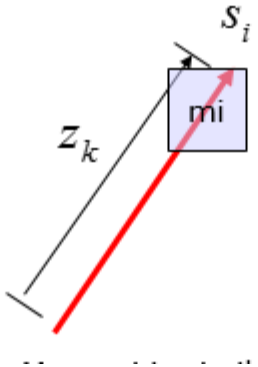

Use Sensor Model
3. Check Distance

$$\log\ \text{Odds} : l_0(\text{m}) = \log \frac{p(\text{m})}{1 - p(\text{m})}$$

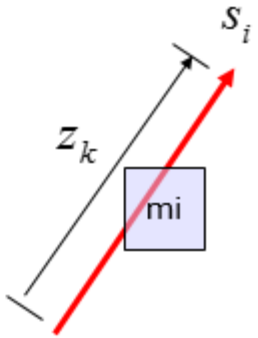
Case 1)
Uncertain



Case 2)
Occupancy Update



Case 3)
Emptiness Update

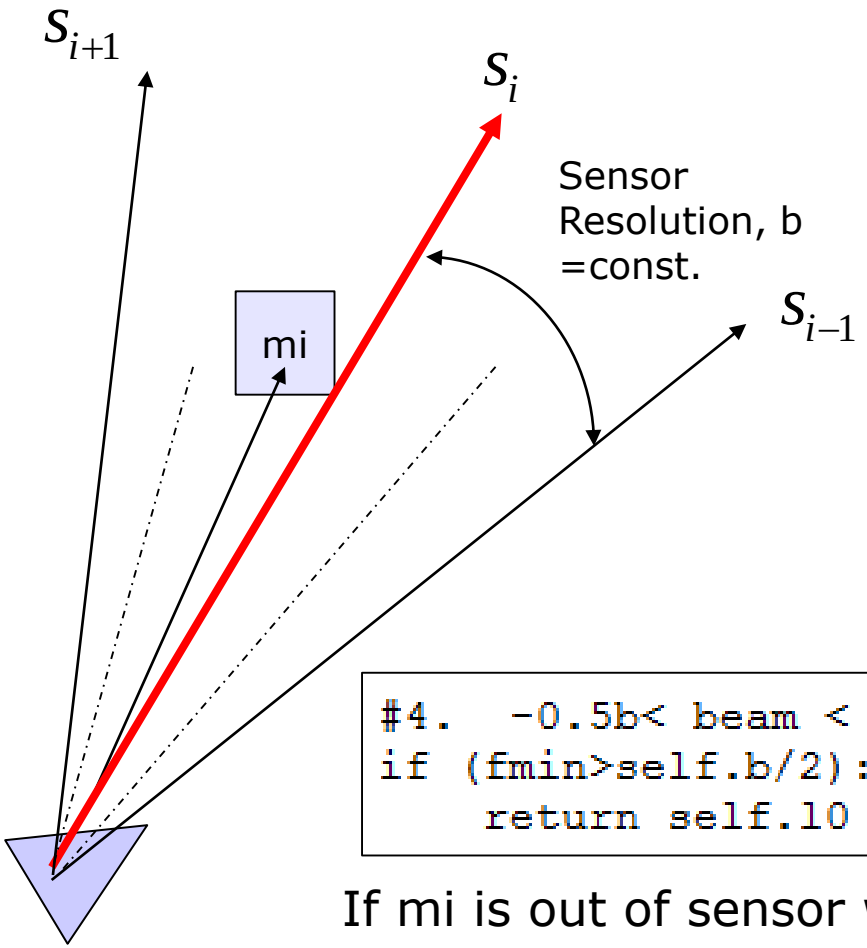


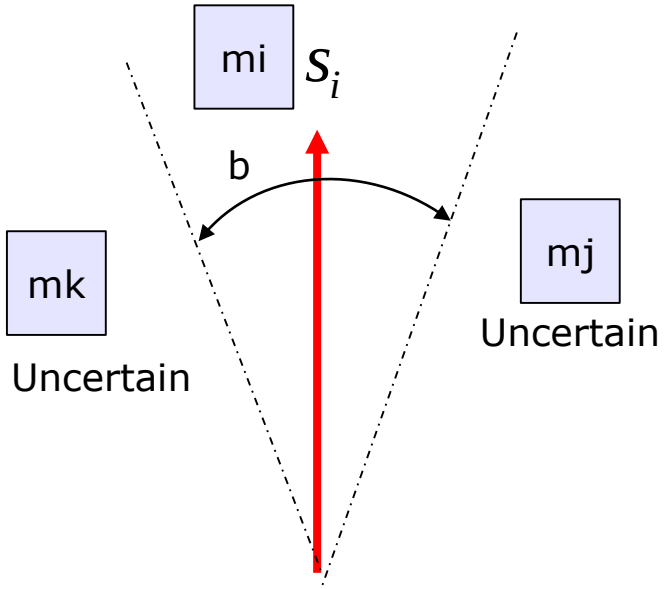
$$l0 = \log \frac{0.5}{1 - 0.5} = 0$$

$$l_{occ} = \log \frac{0.8}{1 - 0.8}$$

$$l_{empty} = \log \frac{0.2}{1 - 0.2}$$

Use Sensor Model
4. Check Sensor Width






```
#4.   -0.5b< beam < 0.5b
if (fmin>self.b/2):
    return self.l0
```

$$|\theta + s_i - \phi| < \frac{b}{2}$$

From Condition 1.

If mi is out of sensor width, then it is uncertain and use l0.

Ex2) Map Update with Linear Motion

- Map Update for every mi (0<i<20 and 0<j<20)

```
def update(self,x,y,q,dist):
    self.x = x
    self.y = y
    self.q = q

    for j in range(0,self.h):
        for i in range(0,self.w):
            mi  = self.m[j*self.w+i];
            mi.l= mi.l+self.invmm(mi,dist)-self.l0;
            mi.p= 1-1/(1+exp(mi.l));
            self.pm[j+1,i+1] = mi.p;
    surf(self.pm)
```

$$l_t = l(m_i \mid x_t, z_t) + l_{t-1} - l_0 = InverseSensorModel + l_{t-1} - l_0$$

Ex2) testpfm.py

```
"""
    Example
"""

def init():
    pf.reset();

def getdist():
    st = loop.sys.recv("loop.sys")
    tok= loop.vToken.vToken(st)
    tok.Separator("\n")
    ret = []
    for i in range(tok.GetSize()):
        noise   = 0.5*randn()
        ret.append(tok.f(i)+noise)
    return ret;

def updatemap():
    d = getdist()
    ms.update(pf.x,pf.y,pf.q, d)
```

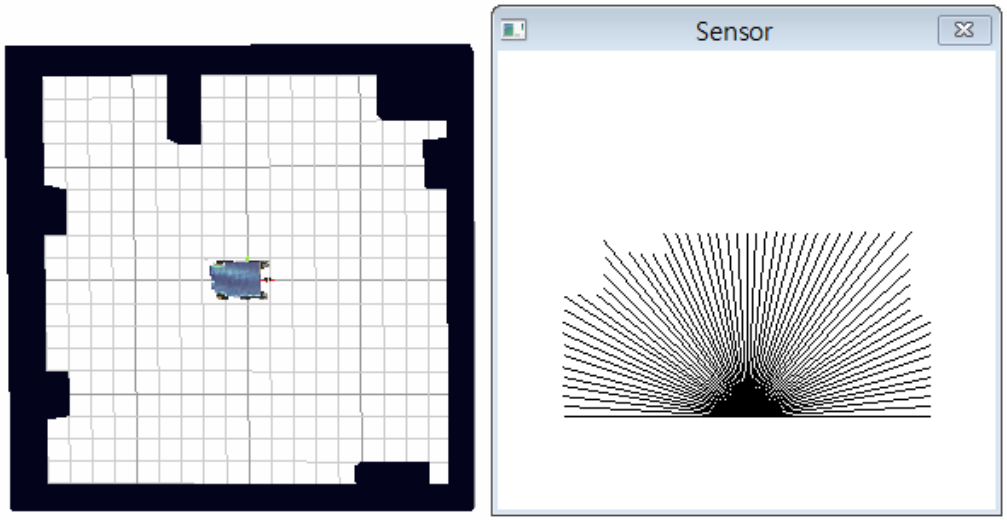
Get sensor data from pathfinder

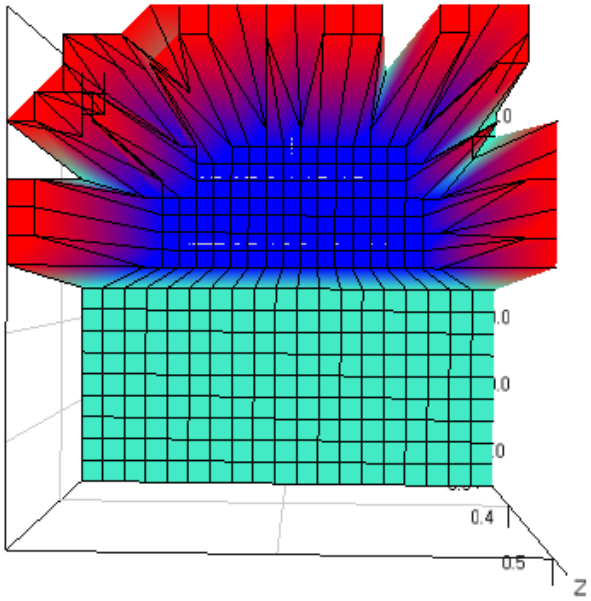
Sensor Noise with 0.5 Gaussian

Start angle=-90
End angle=90
Resolution=3 deg.
Thus, (90-(-90))/3
→60 + 1 (end angle)
ret=61 Distance Array

Whenever pf moves,
Call updatemap()

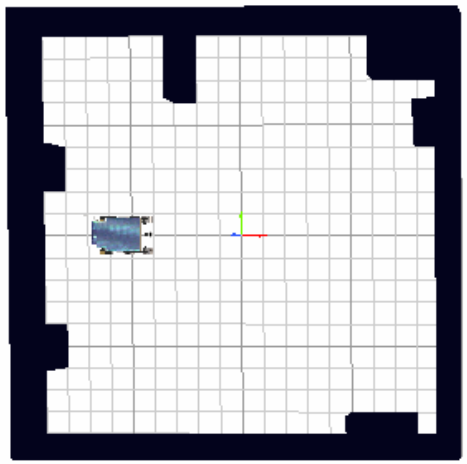
Ex2) testpfm.py

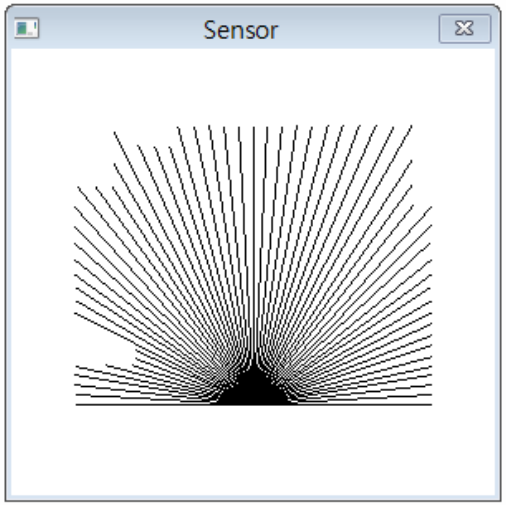


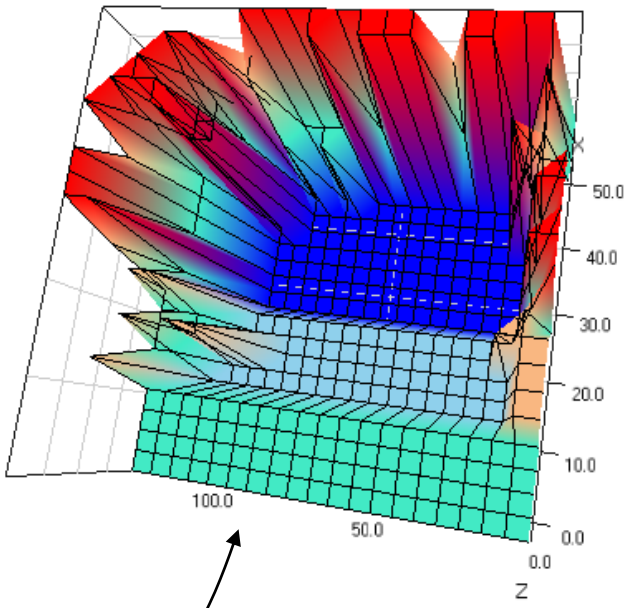



```
import testpfm as t
reload(t)
('Connected to PathFinder.',)
<module 'testpfm' from 'C:WWFLWWDRWW

t.pf.move(0,0,0)
t.updatemap()
```

Ex2) testpfm.py move(-5,0,0)








```
t.pf.move(-5,0,0)
t.updatemap()
```


HW) Complete Your Map Building

- Testpfm.py DOES NOT consider heading angle
  - Rotation is NOT considered.
- Complete your mapping with regarding to Rotation.
  - Think heading angle..

HW 2) Complete Your Map Building

- Testpfm.py DOES NOT consider heading angle
  - Rotation is NOT considered.
- Complete your mapping with regarding to Rotation.
  - Think heading angle..