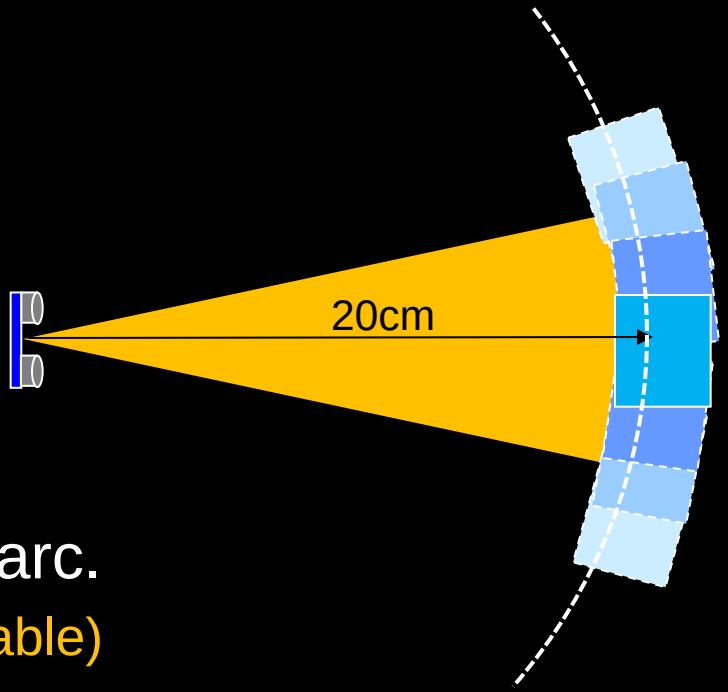
Improved Sensor Model Mapping

Error Distribution

- When object is detected at, say 20_{cm}, it can actually be anywhere within the beam arc defined by the 20_{cm} radius.
- The likelihood (or probability) that the object is *centered* across the arc is greater than if the object was off to the side of the arc.
  (if location is considered to be a random variable)
- We can thus express the sensor reading itself as a set of probabilities across the grid.



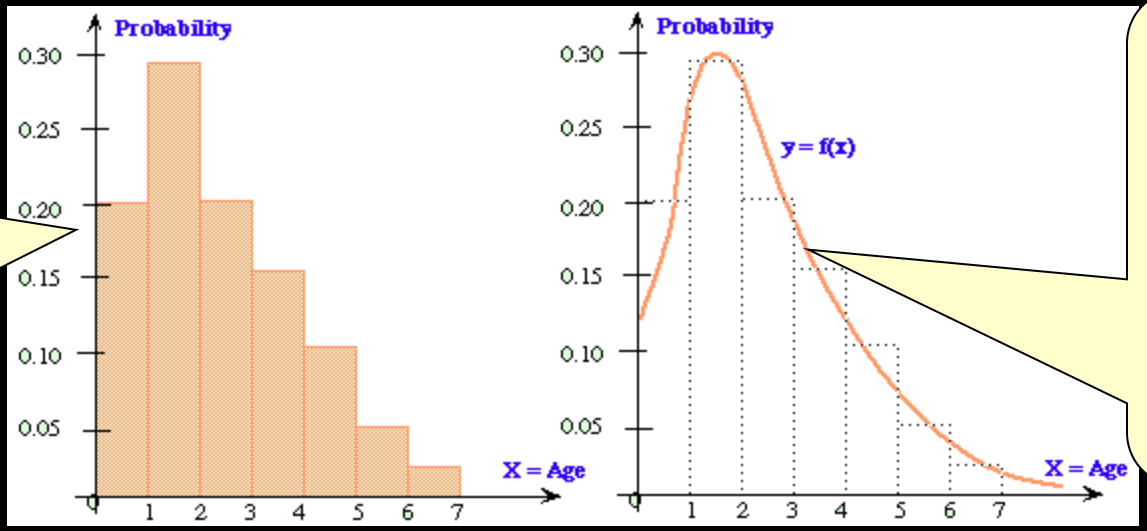
Probability Density Functions

- Random variables operating in continuous spaces are called *continuous random variables*.

- Assume that all continuous random variables posses a ***Probability Density Function*** (PDF).

E.g.,

Probability distribution of a car on the road being a certain age.



Probability Density Function for this distribution.

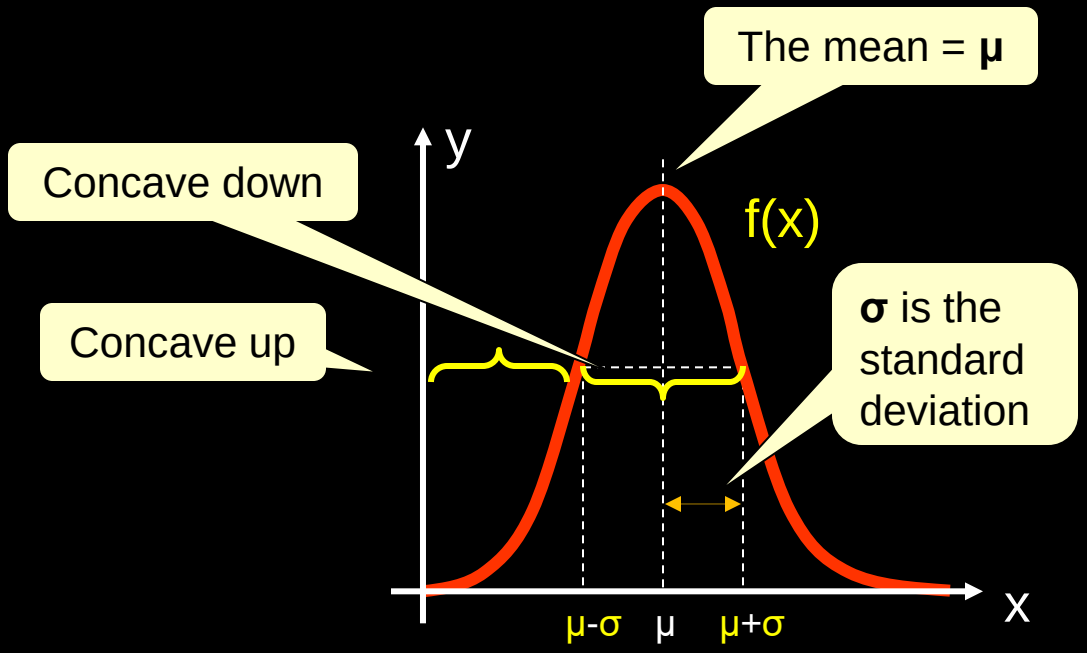
(Also known as the ***Probability Distribution Function***).

Probability Density Functions

- Common PDF is the ***normal distribution***:
  - given *mean* μ and *variance* σ^2 the normal distribution is …

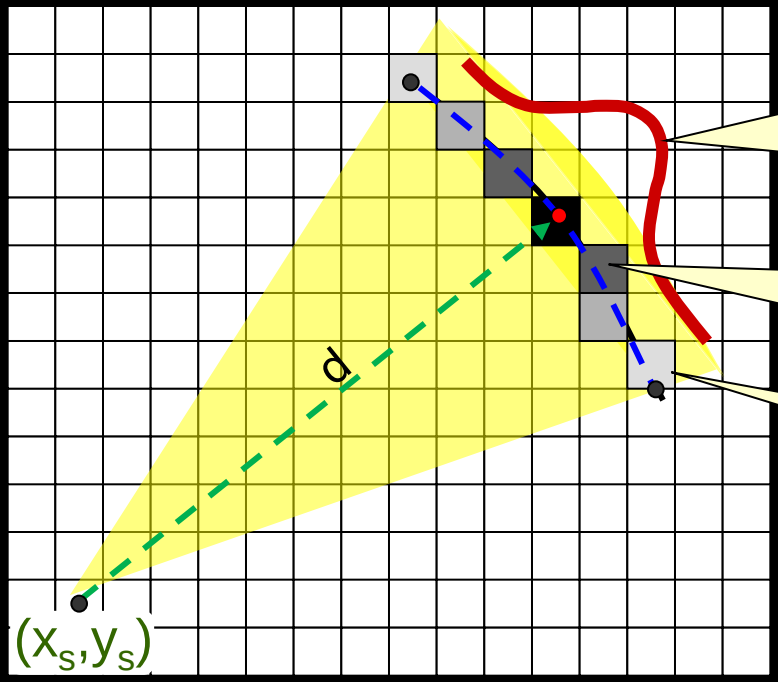
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Don't worry, you do not need to understand this.



Gaussian Distribution

- A more realistic sensor model assigns probabilities to the cells according to some error distribution such as this ***Gaussian*** (or ***Normal***) distribution.



We will now apply this distribution across our arc to assign a "***probability that it is occupied***" value to each cell.

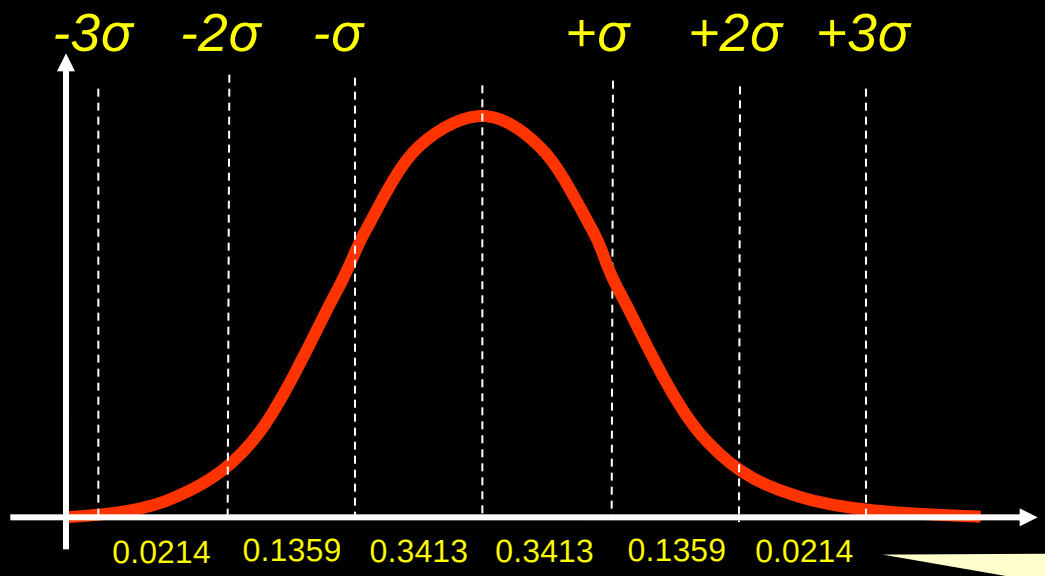
Darker means "**more** likely" that object was at this cell.

Lighter means "**less** likely" that object was at this cell.

Gaussian Distribution

- How do we implement this on our occupancy grid ?

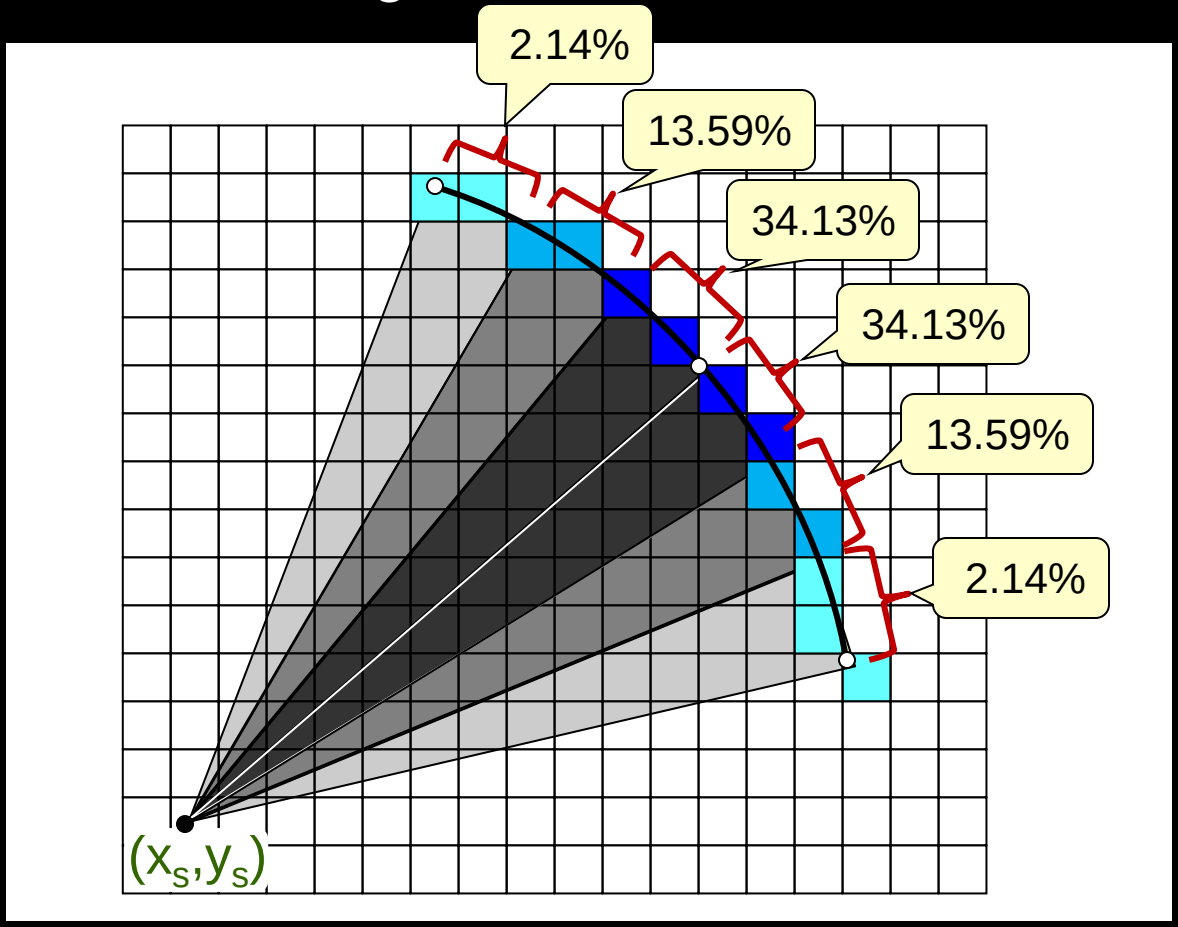
- Often the probabilities are approximated using what is known as the *six-sigma* rule. Which essentially divides the probabilities into 6 probability regions.



Total of probabilities add to 0.9972 ≈ 1.0

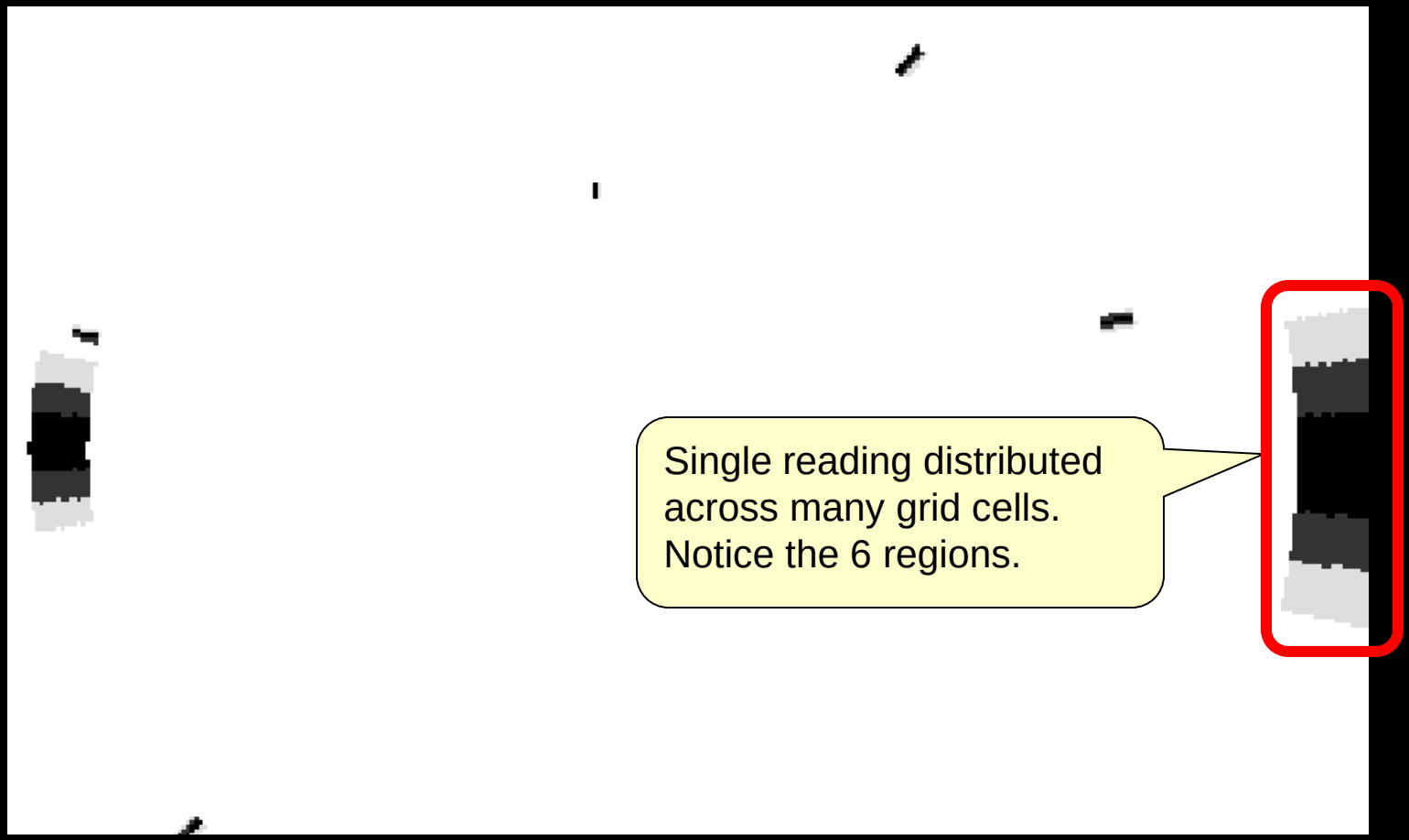
Applying Gaussian Distribution

- Divide arc into 6 “wedges” and apply the specific probabilities to the cells in each wedge.



Applying Gaussian Distribution

- Here is the result of applying the Gaussian distribution across the angle:

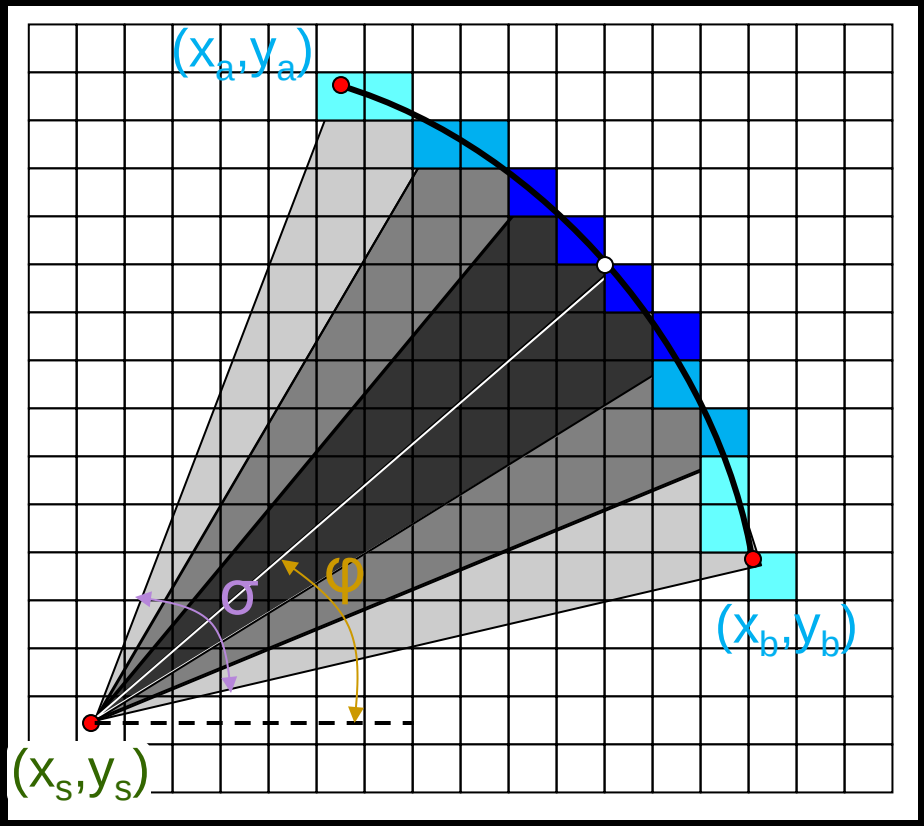


The code

- Recall the code for filling in grid cells along the arc:

$\omega = \sigma / \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$

```
FOR a = -σ/2 TO σ/2 BY ω DO {
  objX = xs + (d * cos(φ + a))
  objY = ys + (d * sin(φ + a))
  grid[objX][obyY] = 1
}
```



- We need to set the probability now instead of setting to 1.
- Can grab the probability from a hard-coded array:

```
static final float[] SIGMA_PROB =
    {0.0214f, 0.1359f, 0.3413f, 0.3413f, 0.1359f, 0.0214f};
```

The Code

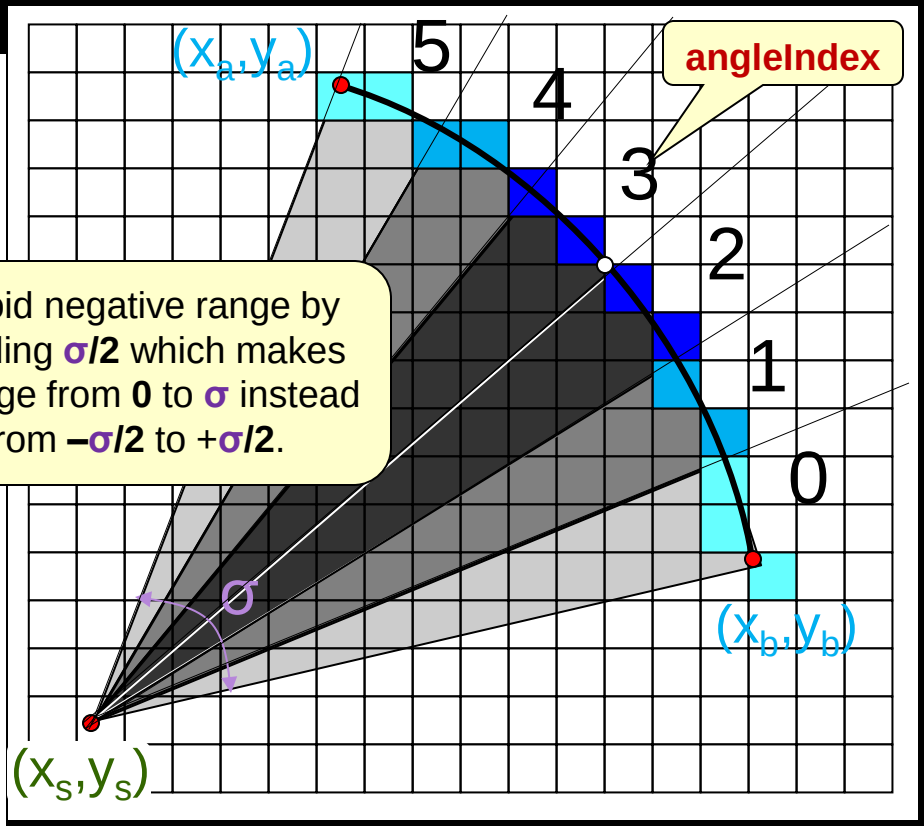
- Just need to find the index i to look up into array:

```
SIGMA_PROB = {0.0214, 0.1359, 0.3413,
              0.3413, 0.1359, 0.0214}

ω = σ / √((x_a - x_b)² + (y_a - y_b)²)

FOR a = -σ/2 TO σ/2 BY ω DO {
    objX = x_s + (d * cos(φ + a))
    objY = y_s + (d * sin(φ + a))

    percentArc = (a + σ/2)/σ

    angleIndex =(int)(percentArc*5.99)

    grid[objX][obyY] =
            SIGMA_PROB[angleIndex]
  }
```

Avoid negative range by adding **σ/2** which makes range from **0** to **σ** instead of from **–σ/2** to +**σ/2**.

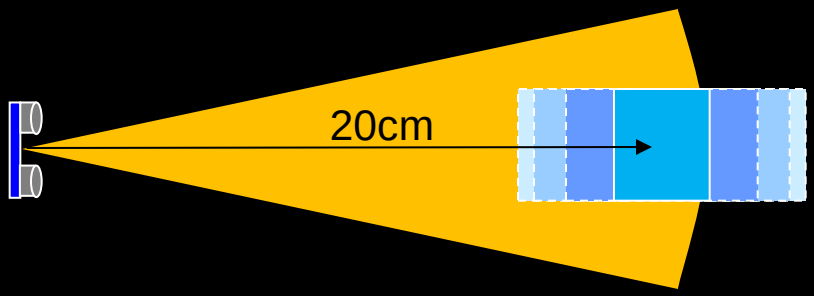
This is the amount of processing so far that we reached during the FOR loop (i.e., 0% to 100%)

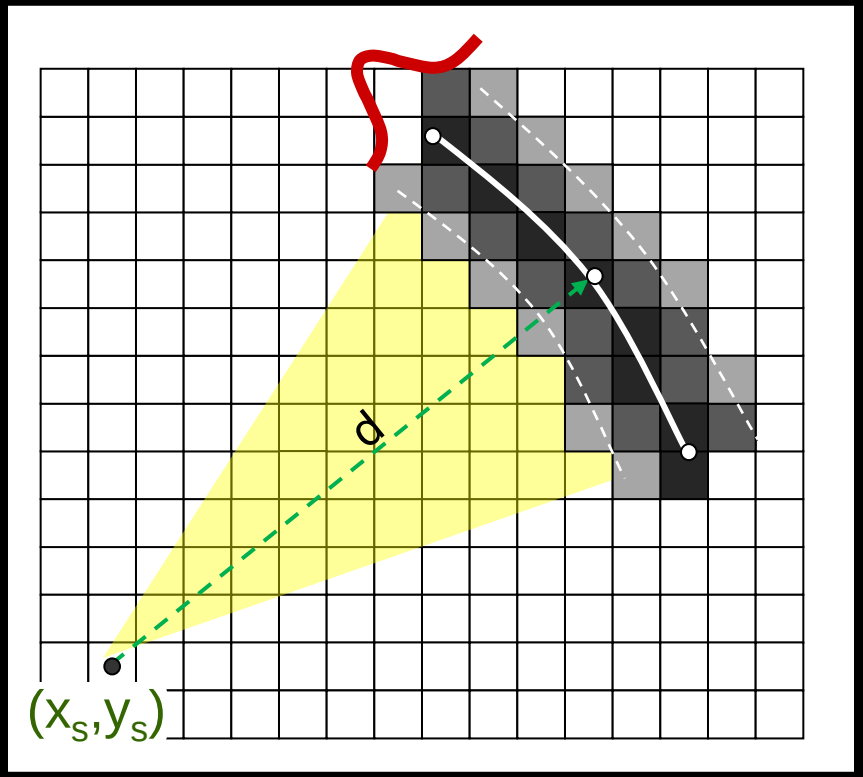
Multiplying by 5.99 and then truncating to an integer, will ensure indices in the 0 to 5 range).



Applying Gaussian Distribution

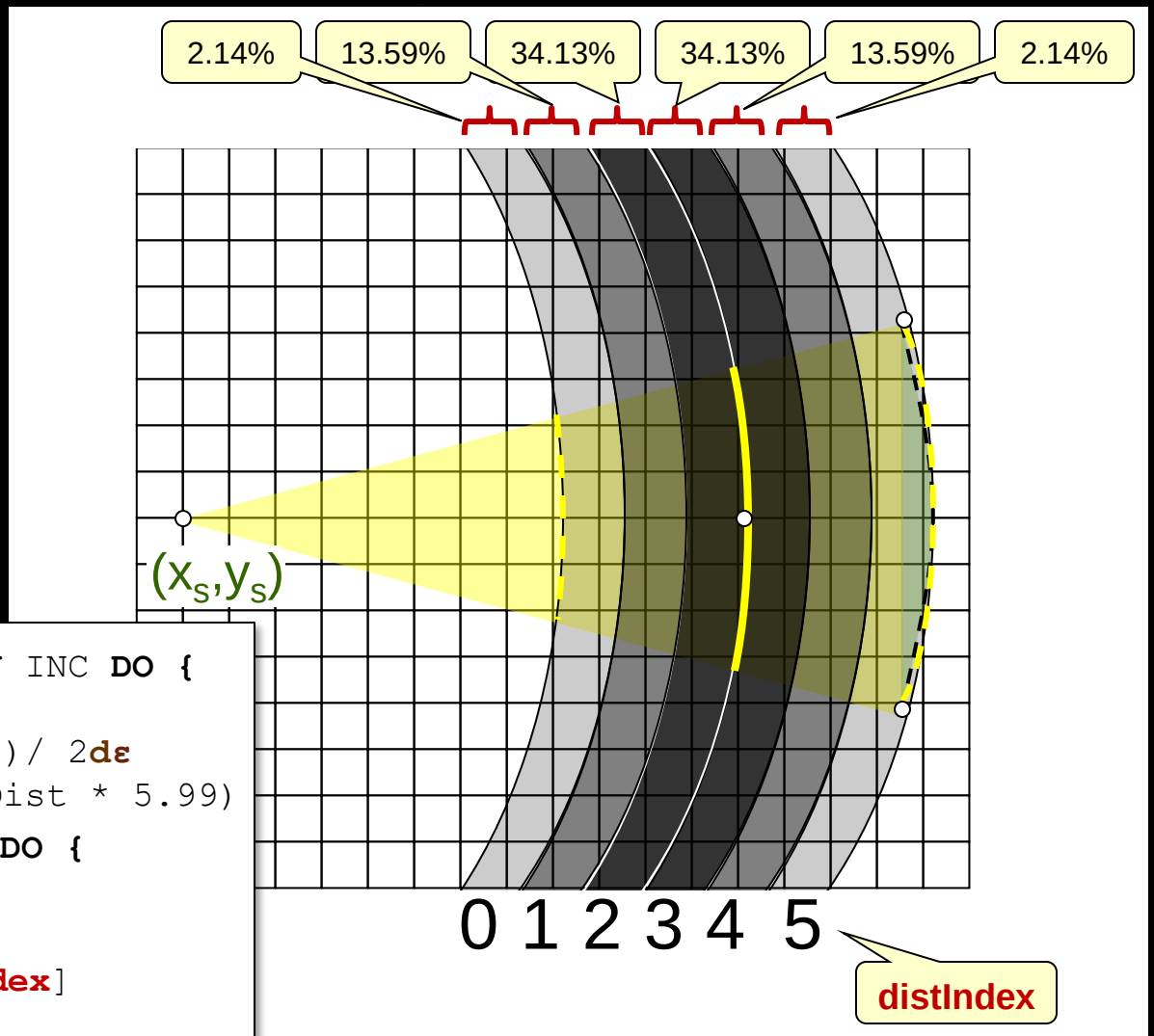
- Should also apply the distribution to distance since object is more likely at the distance range measured than closer or further.





Applying Gaussian Distribution

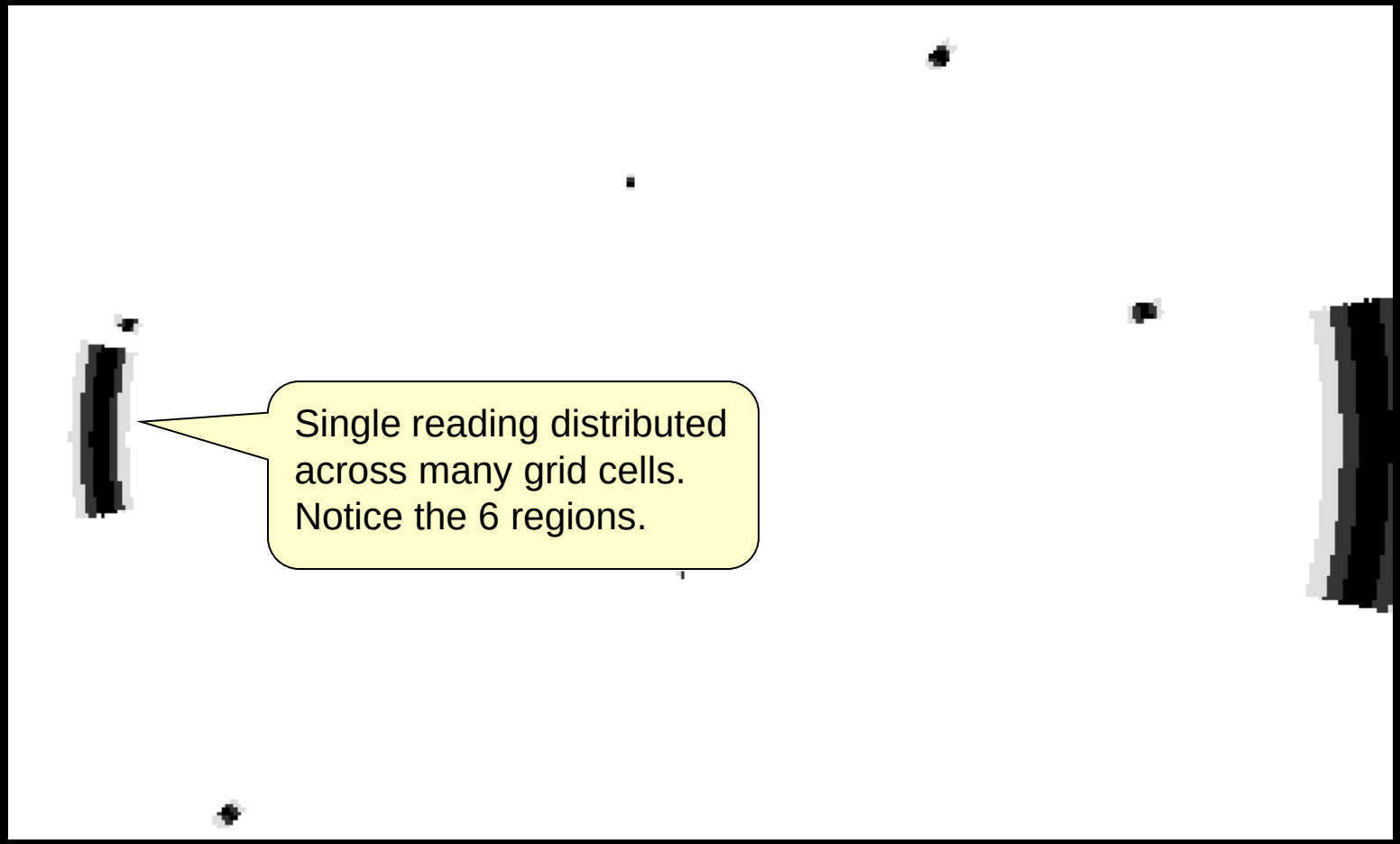
- Divide distance range into 6 “bands” and apply the specific probabilities to the cells in each band.




```
FOR r = d*(1-ε) TO d*(1+ε) BY INC DO {
    ...
    percentDist = (r - d*(1-ε))/ 2dε
    distIndex = (int)(percentDist * 5.99)
    FOR a = -σ/2 TO σ/2 BY ω DO {
        ...
        grid[objX][obyY] =
              SIGMA_PROB[distIndex]
    }
}
```

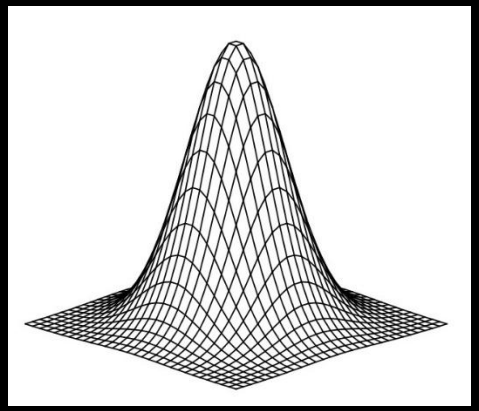
Applying Gaussian Distribution

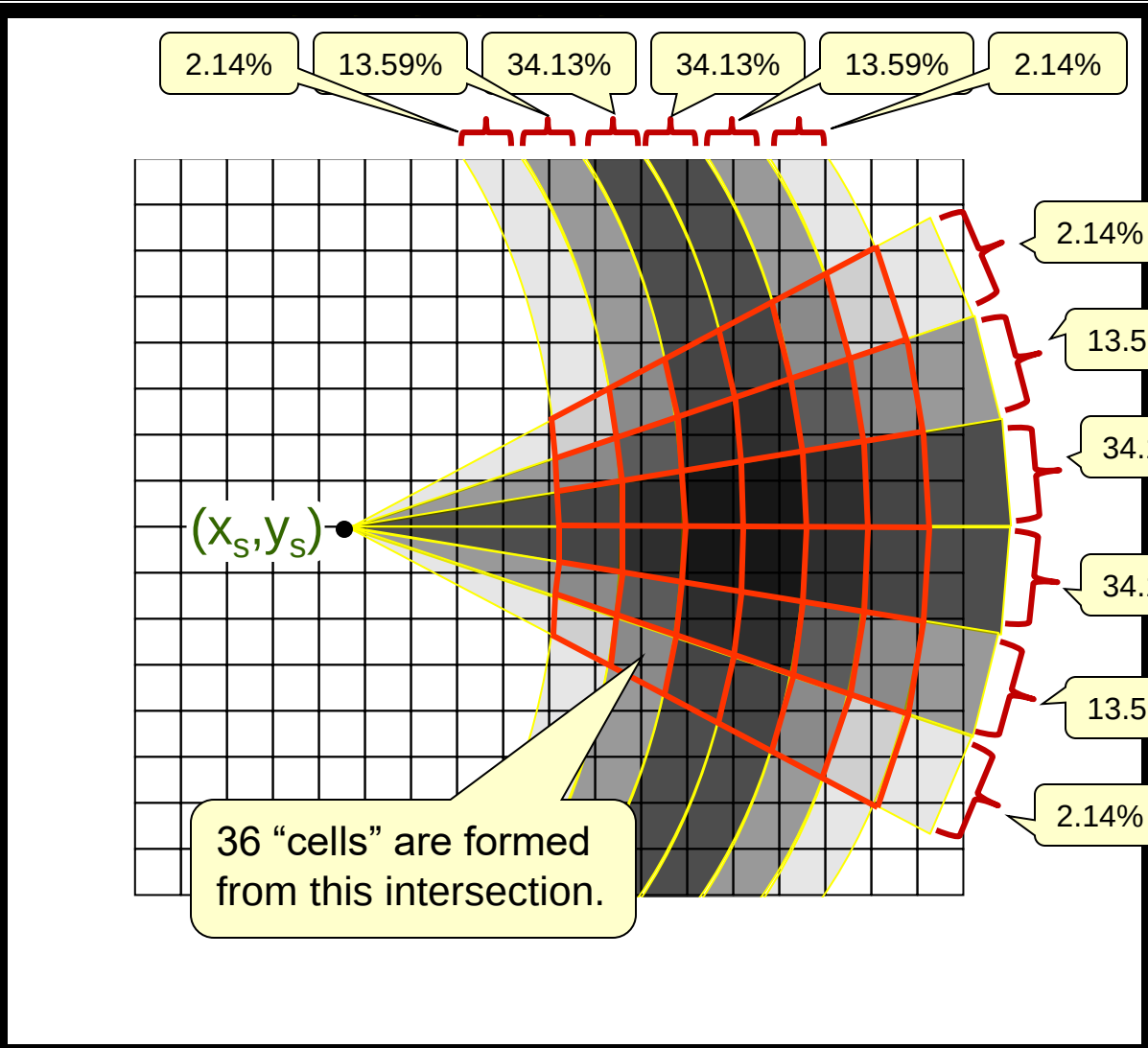
- Here is the result of applying the Gaussian distribution to the distance:



Applying Gaussian Distribution

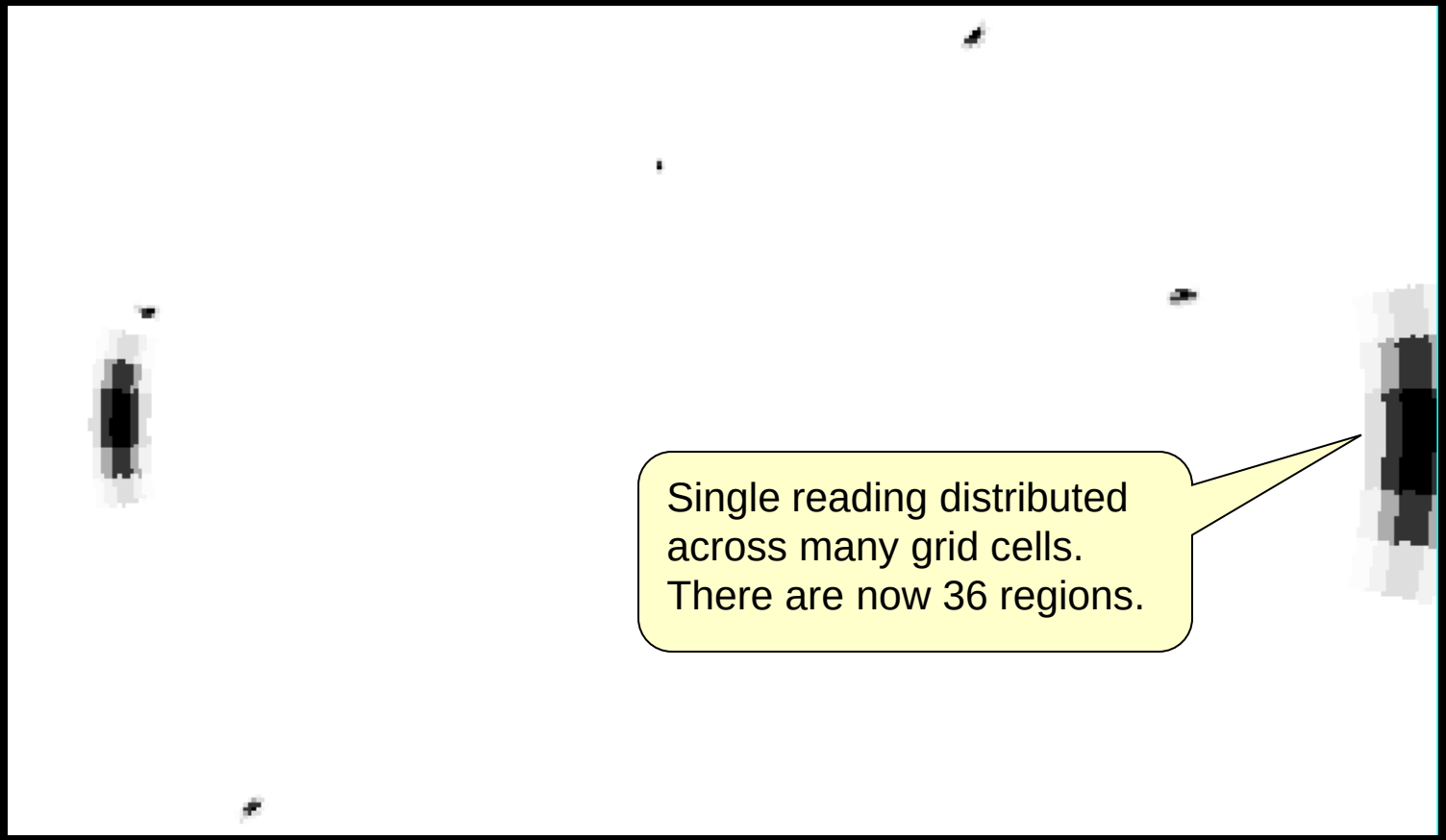
- Finally, apply probabilities along both angle as well as distance:





Applying Gaussian Distribution

- Here is the result of applying the Gaussian distribution to both angle and distance.



Applying Gaussian Distribution

- Here are the probabilities that are to be assigned to each of the 36 cells:

0.05%	0.29%	0.73%	0.73%	0.29%	0.05%
0.29%	1.85%	4.64%	4.64%	1.85%	0.29%
0.73%	4.64%	11.65%	11.65%	4.64%	0.73%
0.73%	4.64%	11.65%	11.65%	4.64%	0.73%
0.29%	1.85%	4.64%	4.64%	1.85%	0.29%
0.05%	0.29%	0.73%	0.73%	0.29%	0.05%

- Can just store the probabilities in a 1D array:

```
static final float[] SIGMA_PROB =
    {0.0214f, 0.1359f, 0.3413f, 0.3413f, 0.1359f, 0.0214f};
```

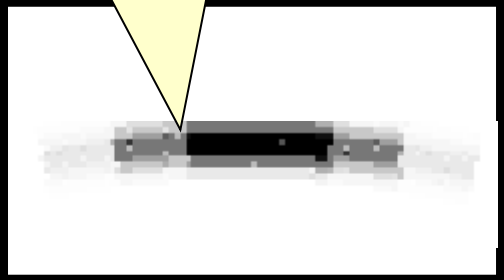
- Then combine both directions through multiplication:

```
probability = SIGMA_PROB[angIndex] * SIGMA_PROB[distIndex];
```

Ensuring Consistency

- The technique just shown, if not careful, does not properly assign probabilities across the wedge for a single sensor reading.

Due to round-off inaccuracies, there will likely be some grid cells counted twice and some not counted during a single update. This may lead to a **speckled** pattern.



This can also occur if the increment on the **FOR** loop for the distance is not small enough. It should be smaller than the grid's precision to ensure that no grid cells are missed:

```
for (double r=0; r<limit; r+=INC) {
    ....
}
```

e.g., **INC = 1**

INC = 0.75

INC = 0.5

INC = 0.25

0.25 avoids speckled pattern.

Ensuring Consistency

- To avoid speckled pattern, create a temporary grid

1. Create to be same size as entire grid

```
temp = new float[width][height];
```

2. Initialize all values to 0

```
temp[i][j] = 0;
```

3. Apply all readings to the temporary grid by *setting* the cell values (i.e., not adding them)

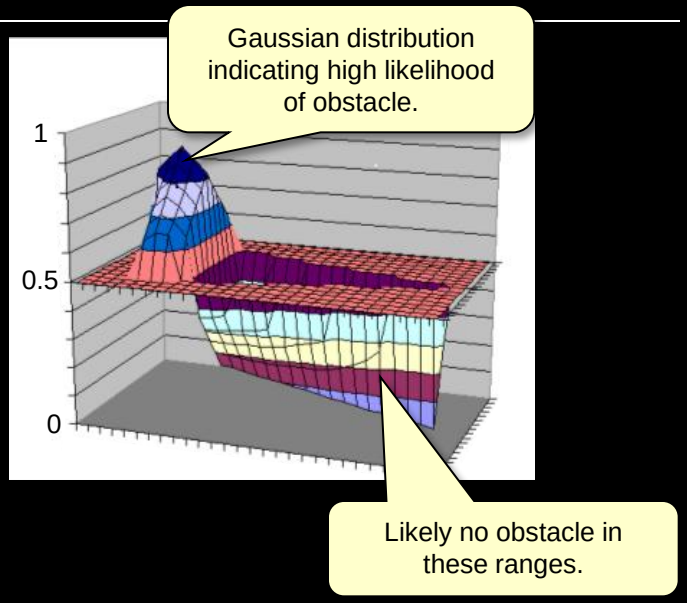
```
temp[i][j] = SIGMA_PROB[angIndex] * SIGMA_PROB[distIndex];
```

4. Merge temporary grid with complete map once reading probabilities have been completed

```
grid[i][j] += temp[i][j]
```

Non-Obstacle Certainty

- Another way to refine the grid is to say something about the certainty that an obstacle is NOT there.



Obstacle CANNOT lie in here otherwise distance reading would have been smaller.

We won't apply any distance distribution.

We can **decrease** occupancy grid values here according to the angular distribution.

Obstacle lies in here somewhere according to a Gaussian distribution.

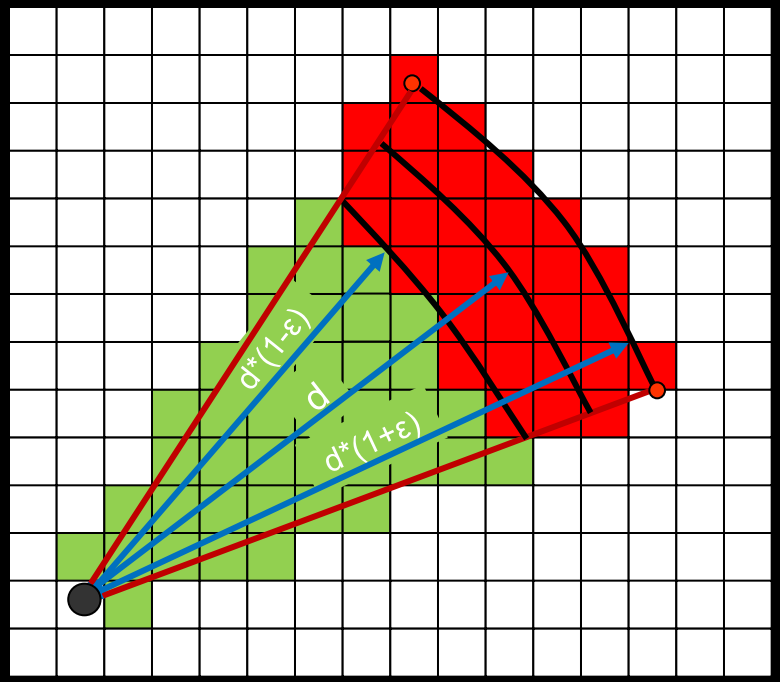
```
{0.0214f, 0.1359f, 0.3413f, 0.3413f, 0.1359f, 0.0214f};
```

Non-Obstacle Certainty Range

- Currently, the current FOR loop code only updates radius values that are between d*(1-ε) and d*(1+ε) … red cells.
- But now we need to update cells with radius values from 0 up to d*(1- ε) as well … green cells:

Start at **0** now instead of `d*(1-ε)`

```
FOR r = 0 TO d*(1+ε) BY INC DO {
    ...
    FOR a = -σ/2 TO σ/2 BY ω DO {
        ...
        IF (r < d*(1-ε)) THEN
            lighten the cell using angular
            distribution only (not distance)
        ELSE
            darken the cell as before using
            both angular and distance
    }
}
```



Start the Lab …