Sensor Models

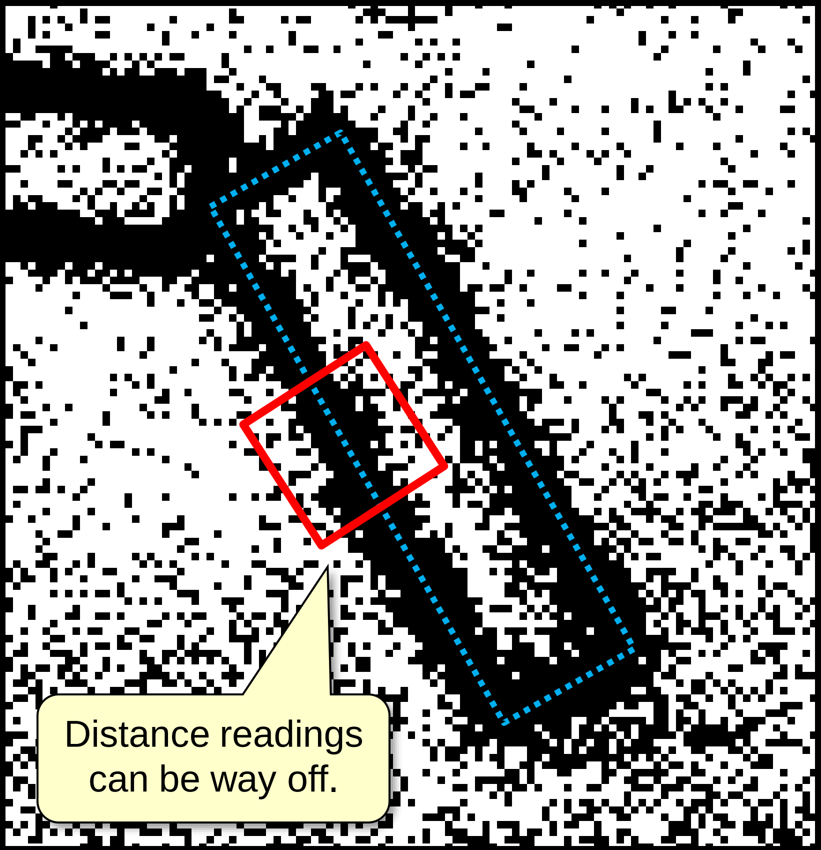
Problems With Sensor Data

- Raw maps we made assumed that sensor data was accurate
  - Each reading indicated a part of the obstacle

- Result is that objects appear on map as a fuzzy set of points with a lot of noise

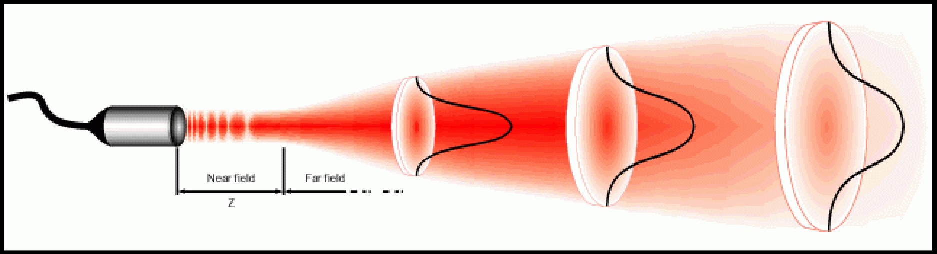
- Problem is due to inaccurate sensors and inability to be precise due to:

  1. Beam-width Error
  2. Distance Error
  3. Reflection Error

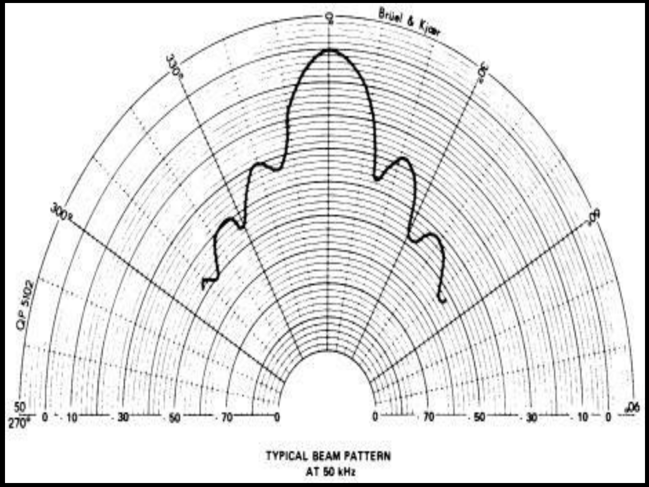


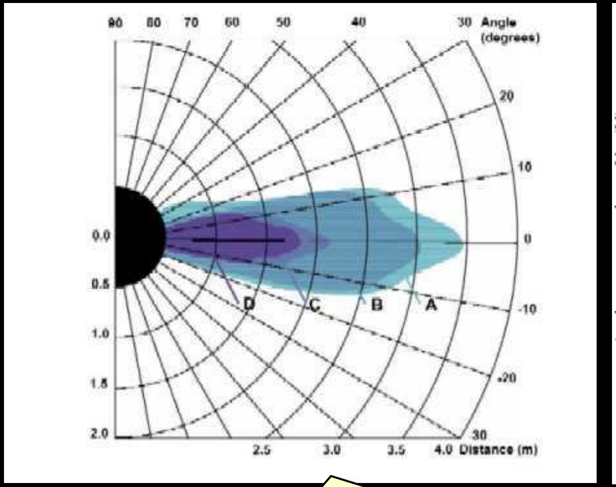
1. Beam-width Error

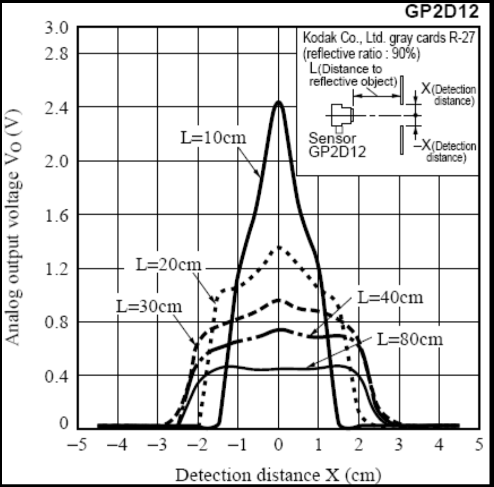
- Shape of beam is not a simple wedge:



  - wider objects near center of beam result in better accuracy



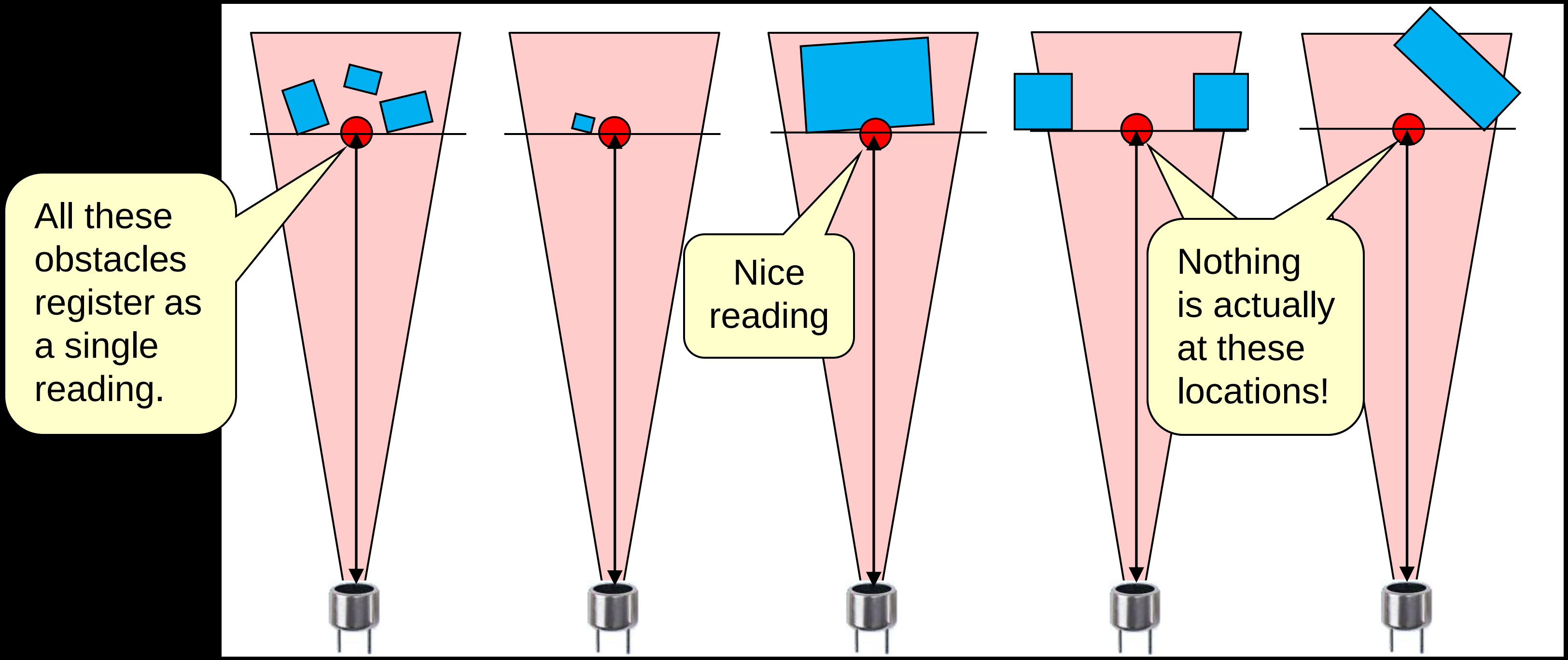




These are sensor specifications showing the sensor beam's shape.

1. Beam-width Error

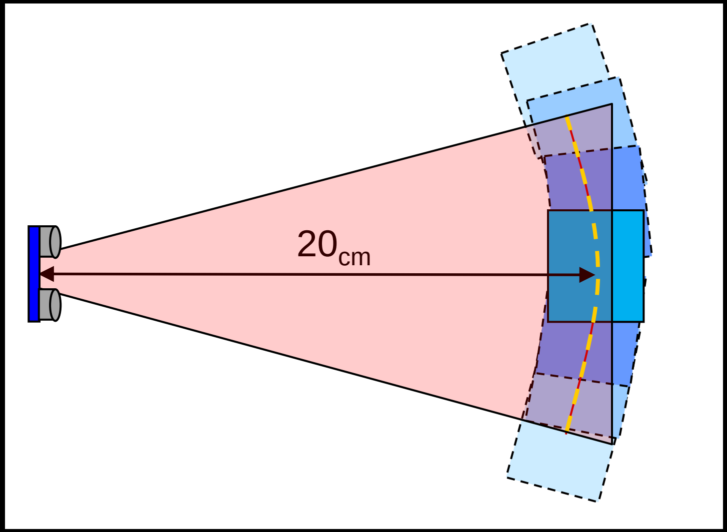
- So, when we receive a range reading, we are actually unsure as to whether or not the reading accurately represents what is directly ahead:



1. Beam-width Error

- When a single reading is obtained, object may actually be anywhere along the width of the beam, not necessarily at the center of the beam.

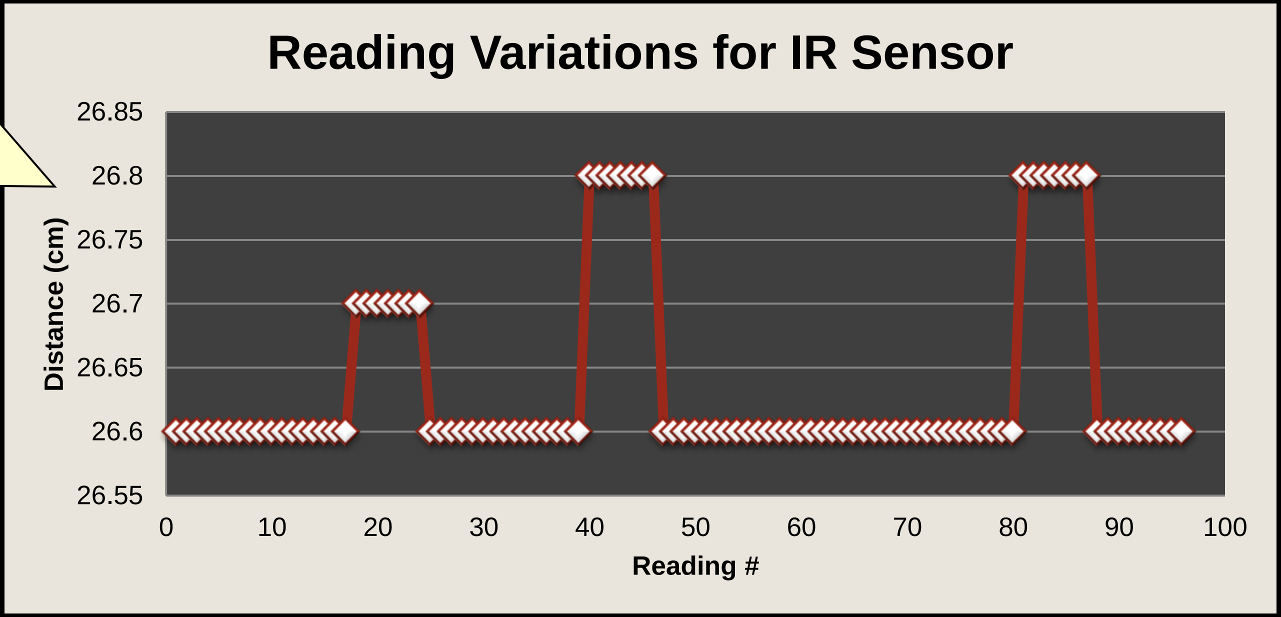
- When an object is detected at, say 20_{cm}, it can actually be anywhere within the beam arc defined by the 20_{cm} radius:



2. Distance Error

- In addition to the location of the obstacle on the beam, sensors themselves give inaccurate distances.

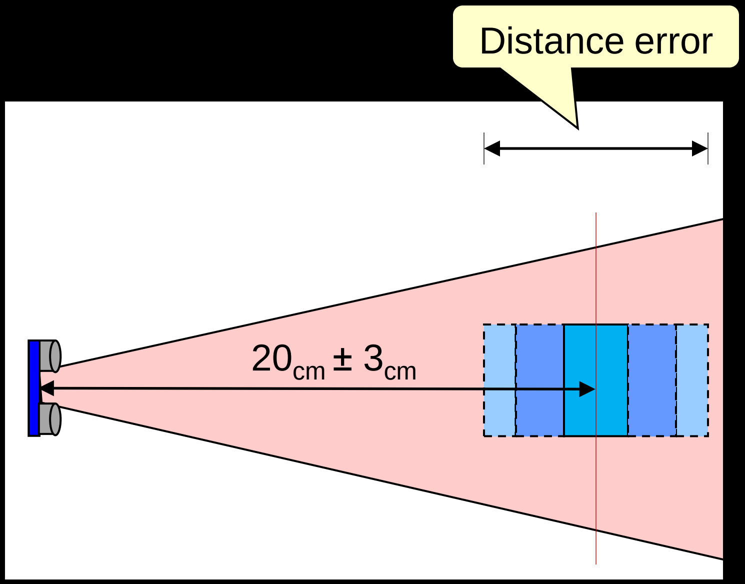
Readings vary even though obstacle is stationary



2. Distance Error

- When a single reading is obtained, object may actually be closer or further than what is expected.

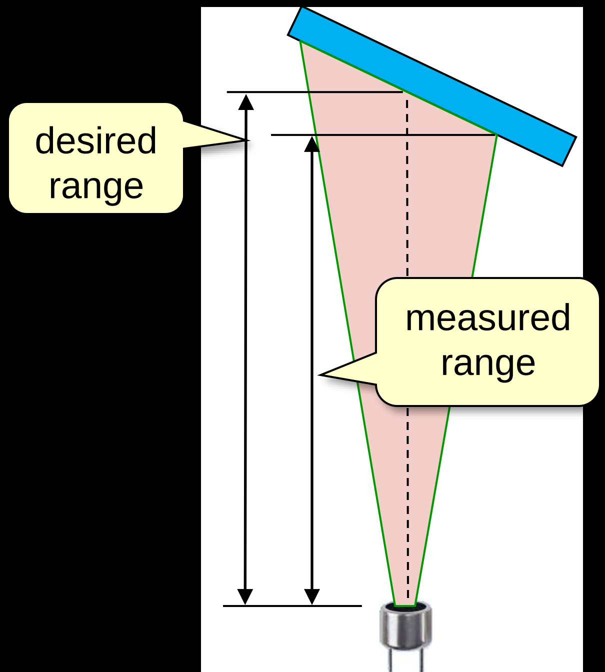
- When object is detected at, say 20_{cm}, it can actually be closer or further within some tolerance:

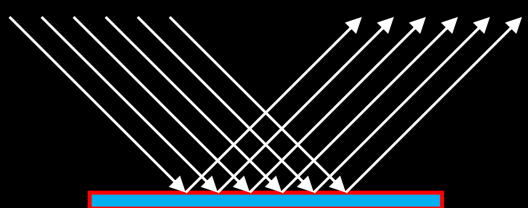


3. Reflection Errors

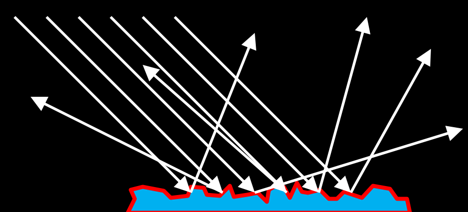
- Sensitivity to obstacle angle can result in improper range readings.

- When beam's angle of incidence falls below a certain critical angle ***specular reflection*** errors occur.





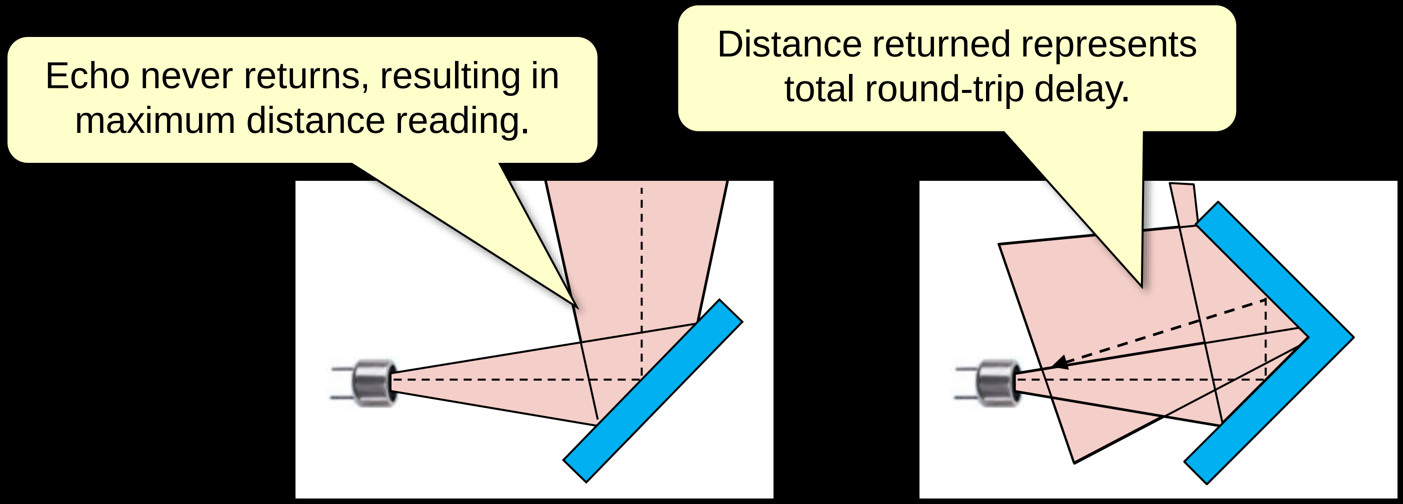
Specular Reflection
(smooth surfaces)



Diffuse Reflection
(rough surfaces)

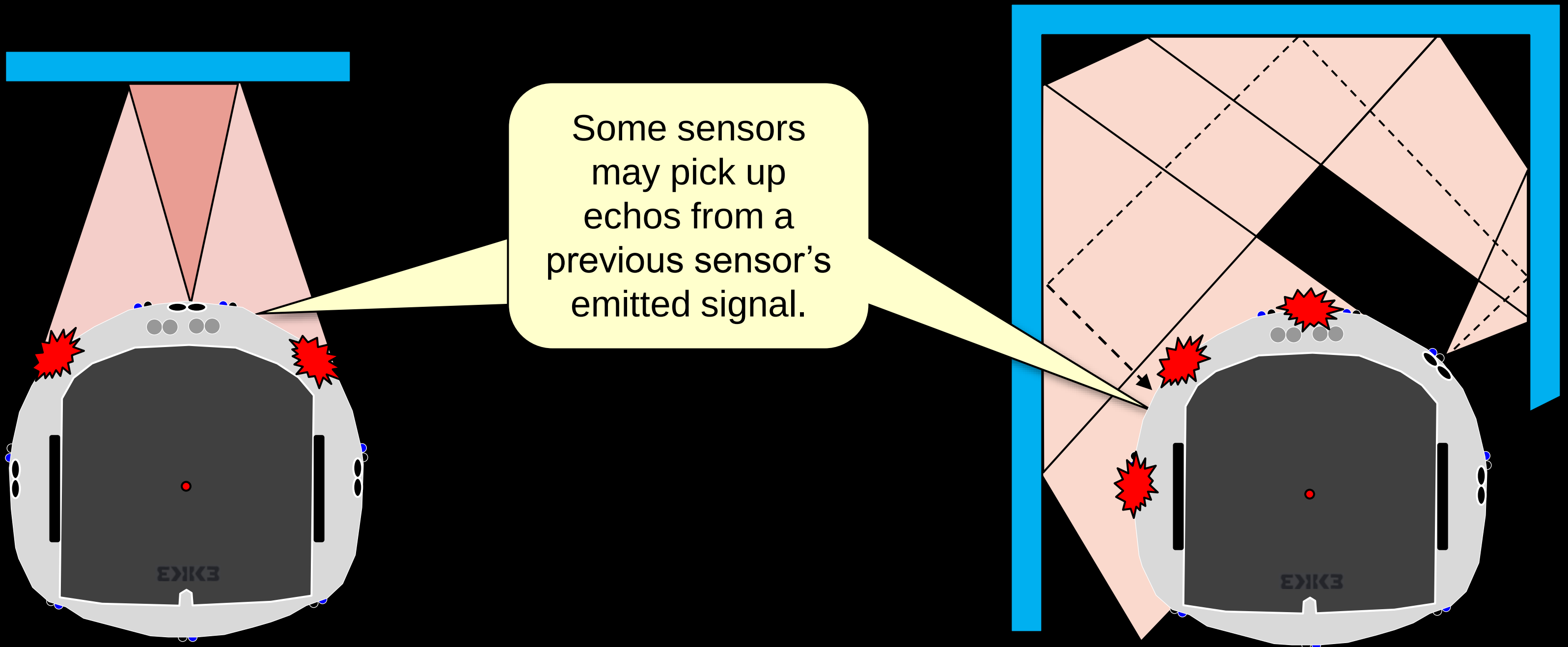
3. Reflection Errors

- Specular reflection can cause reflected ultrasound to:
  - never return to the sensor
  - return to the sensor too late
- In either case, the result is that the distance measurement is too large and inaccurate



3. Reflection Errors

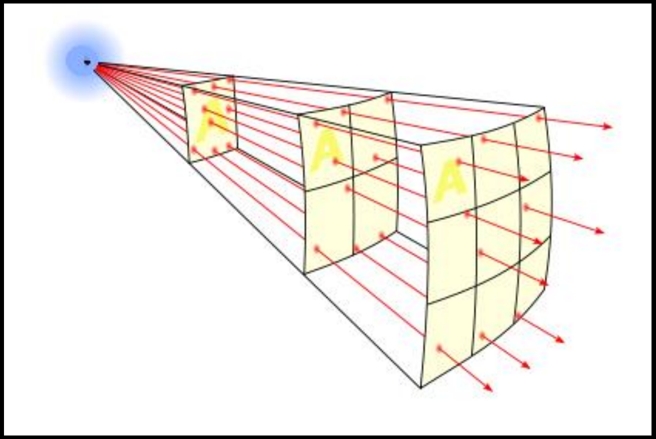
- Using multiple fixed position sensors can lead to another problem called ***crosstalk***:
  - A form of interference in which echoes emitted from one sensor are detected by others



Sensor Models

- Before using a sensor for mapping, a ***sensor model*** should be developed:
  - specifies how the sensor readings are to be interpreted
  - depends on physical parameters of sensor
    (e.g., beam width, accuracy, precision etc…)
  - must be able to deal reasonably with noisy data

- For range sensors, they all have similar common characteristics that must be dealt with:
  - distance errors (distance accuracy)
  - angular resolution (beam width)
  - noise (invalid data)



Sensor Models

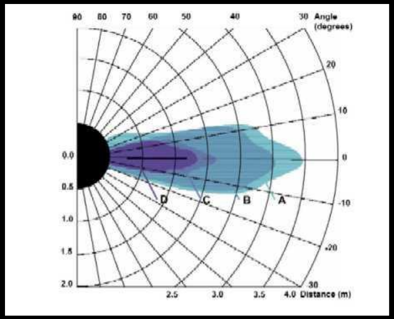
- Various ways to come up with a sensor model:
  - Empirical: Through testing
  - Subjective: Through Experience
  - Analytical: Through analysis of physical properties

- Once sensor model is determined, it is applied to each sensor reading so as to determine how it affects the map being built.
- We will consider our sensor models in terms of how they are used in generating occupancy grid maps.

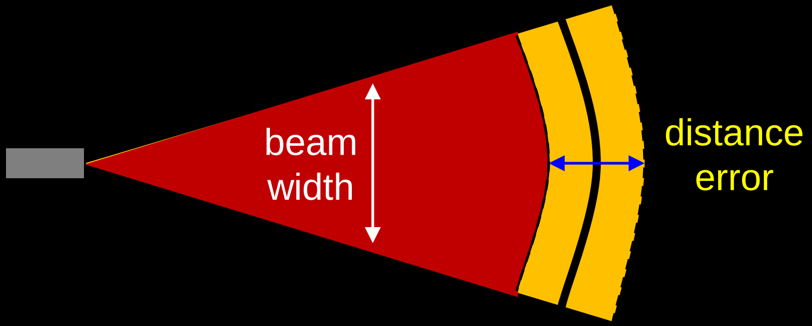
Sensor Models

- Our sensor model will consider distance accuracy and beam width.

- Sensor beam width is not easy to model
  - has different width at different distances
  - different obstacles have different reflective effects

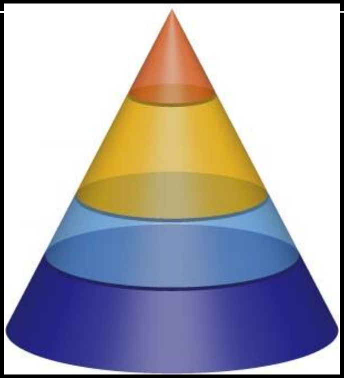


- Most models keep things simple by assuming that the beam is a cone-shaped wedge:



Sensor Models

- We will assume that our sensors have beams with this simple cone shape:
  - actually an approximation of the true shape
  - simplifies calculations
  - will vary beam width and distance error with each sensor



- How do we pick beam width and distance error ?
  - beam width and distance error may vary between individual sensors of the same type
  - beam width and distance error usually obtained through experimentation
  - take average of many readings at certain distances

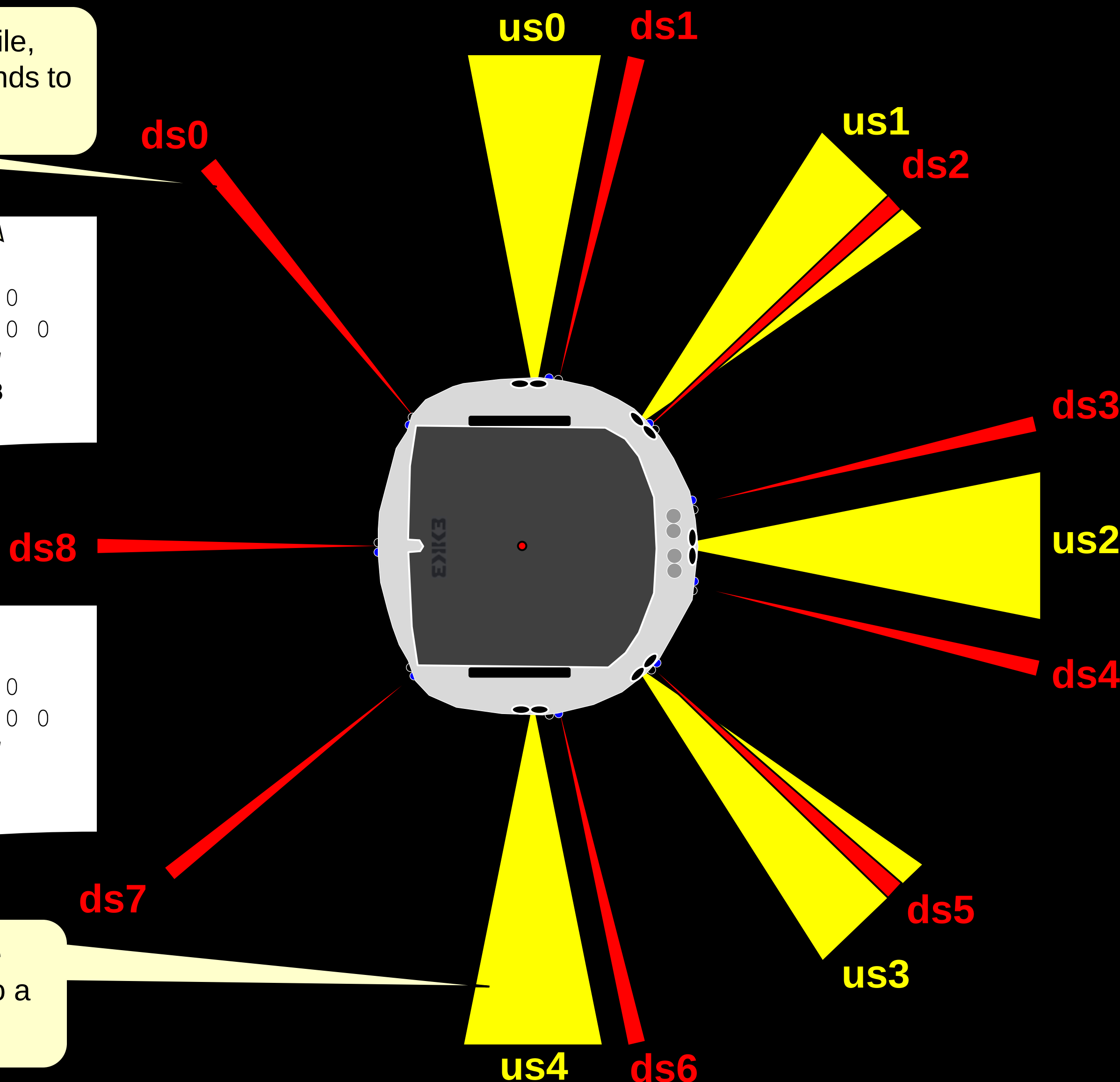
Khepera III Sensor Beam Width

In the **Khepera3_DistanceSensor.proto** file, the **aperture** is set to **0.03** which corresponds to a **1.72°** beam width.

```
PROTO Khepera3_DistanceSensor [
  field SFVec3f    translation   0 0 0
  field SFRotation rotation      0 1 0 0
  field SFString   name          "ds"
  field SFFloat    aperture      0.03
  field SFInt32    numberOfRays  3
]
```

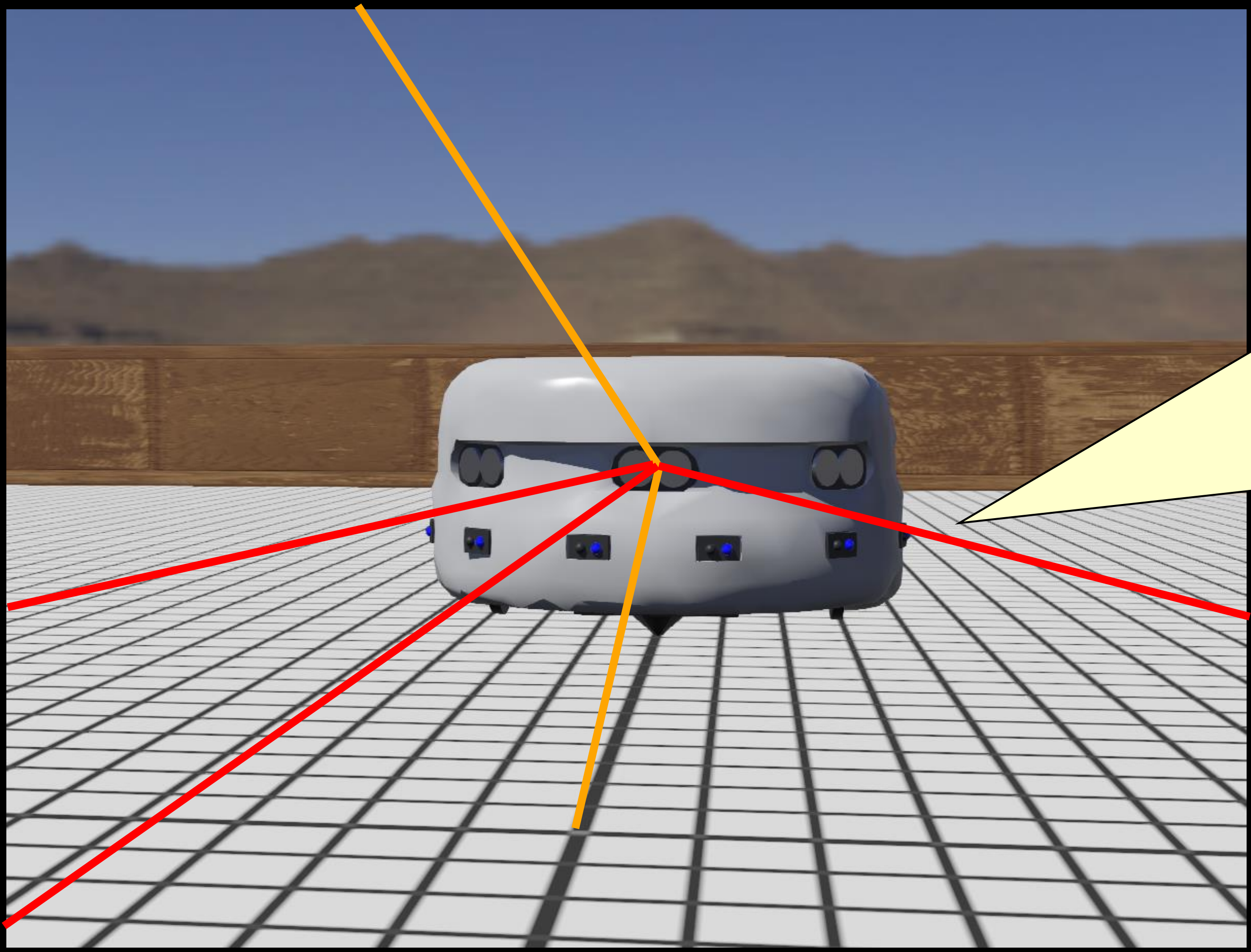
```
PROTO Khepera3_USSensor [
  field SFVec3f    translation   0 0 0
  field SFRotation rotation      0 1 0 0
  field SFString   name          "ds"
  field SFFloat    aperture      0.4
  field SFInt32    numberOfRays  5
]
```

In the **Khepera3_USSensor.proto** file, the **aperture** is set to **0.4** which corresponds to a **22.9°** beam width.

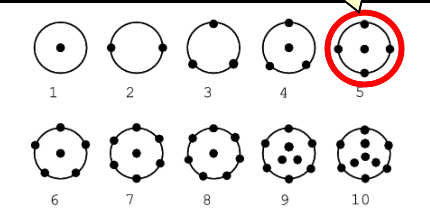


Ultrasonic Sensor Simulation

- Webots simulates the ultrasonic sensor by using up to 10 individual rays:

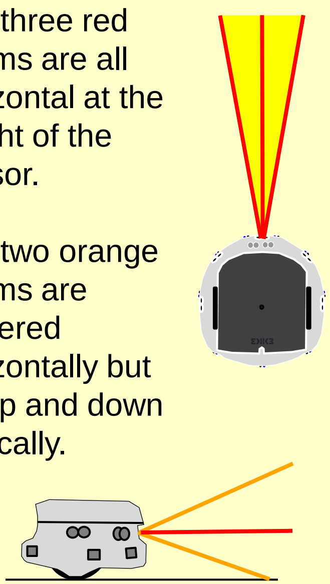


We will use this one.

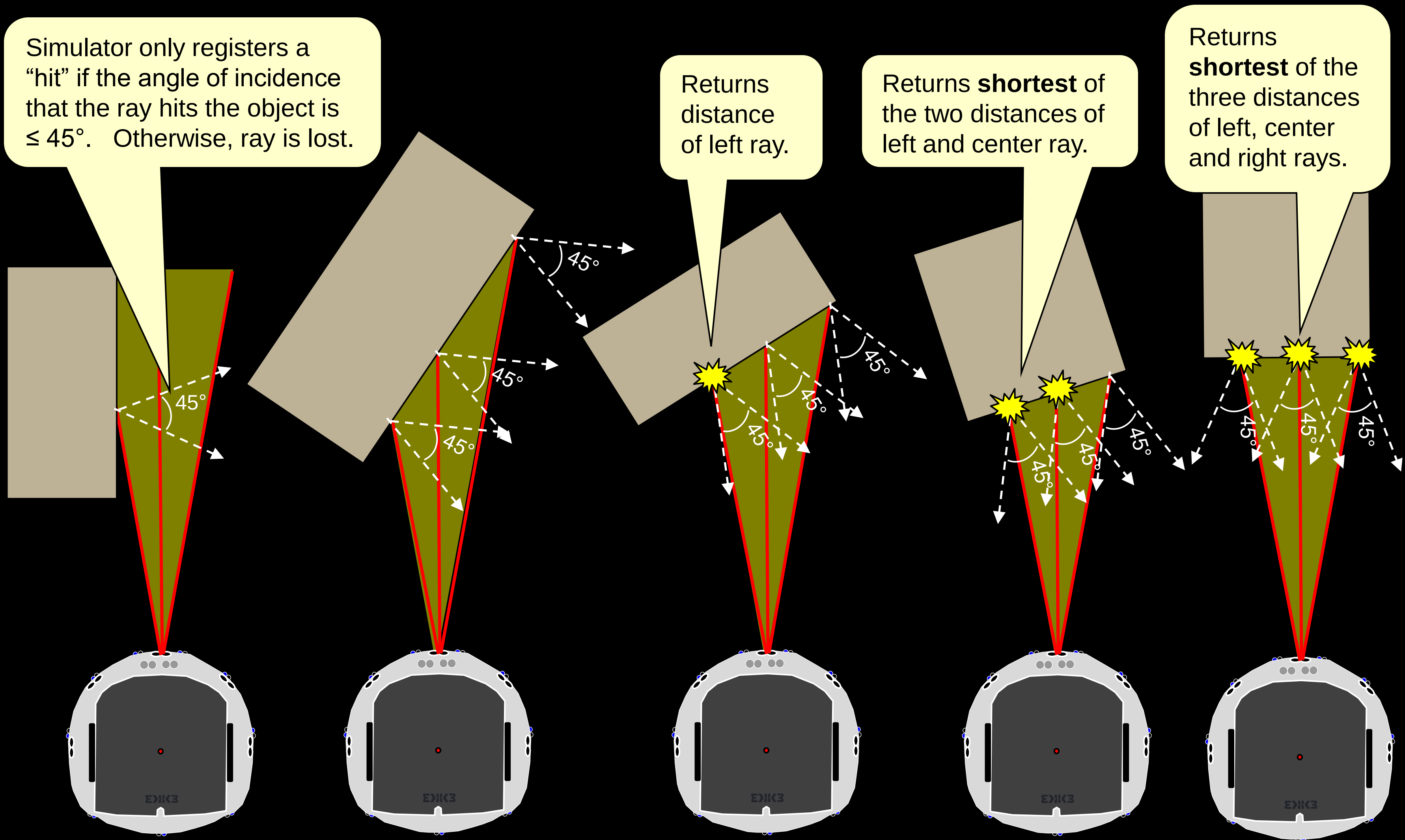


The three red beams are all horizontal at the height of the sensor.

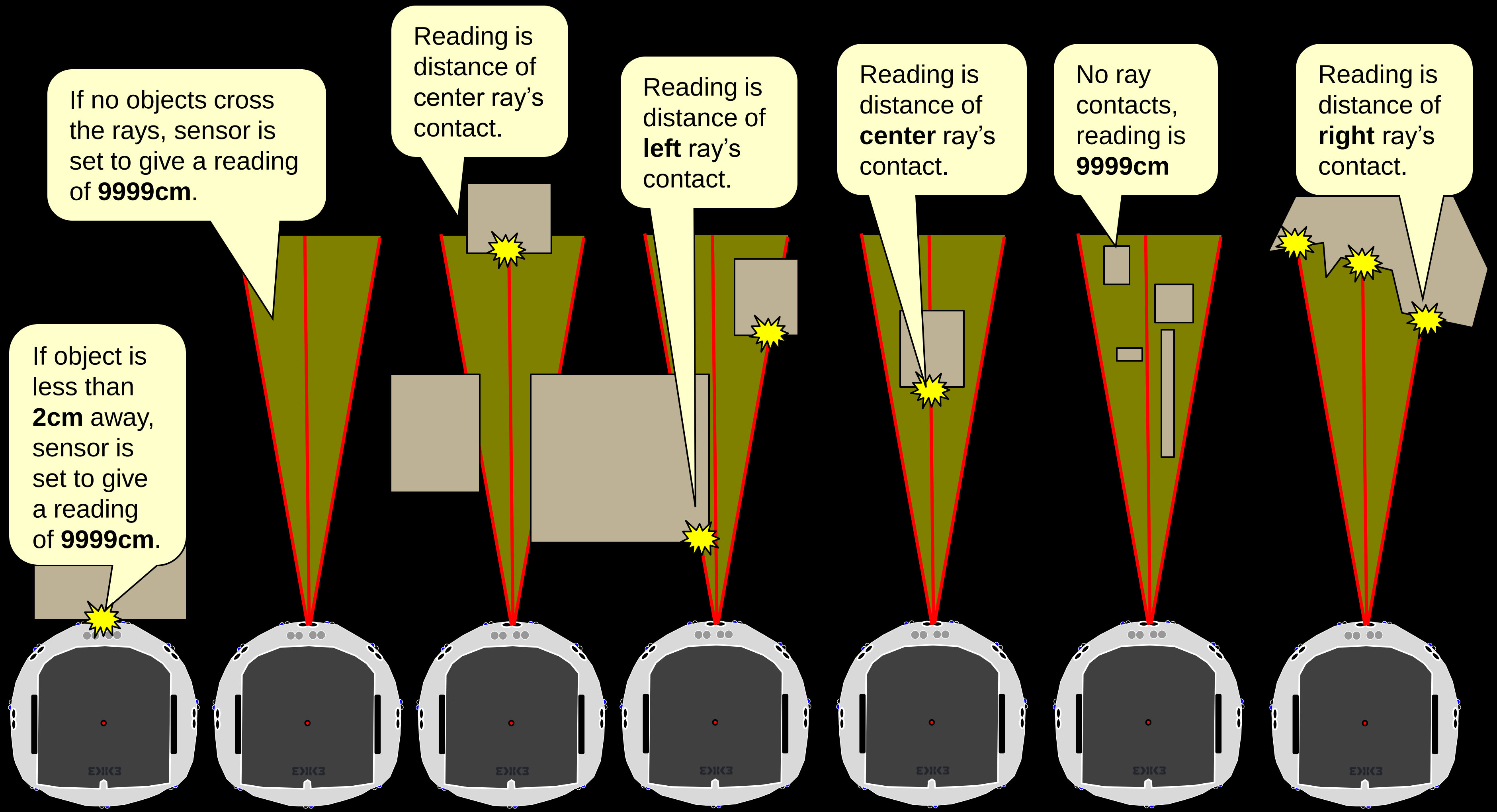
The two orange beams are centered horizontally but go up and down vertically.



Webots Sonar Sensor Simulation

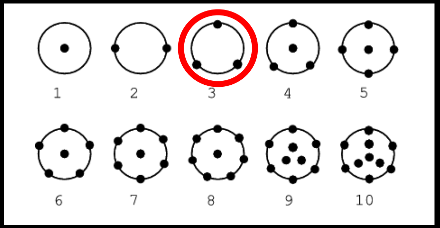


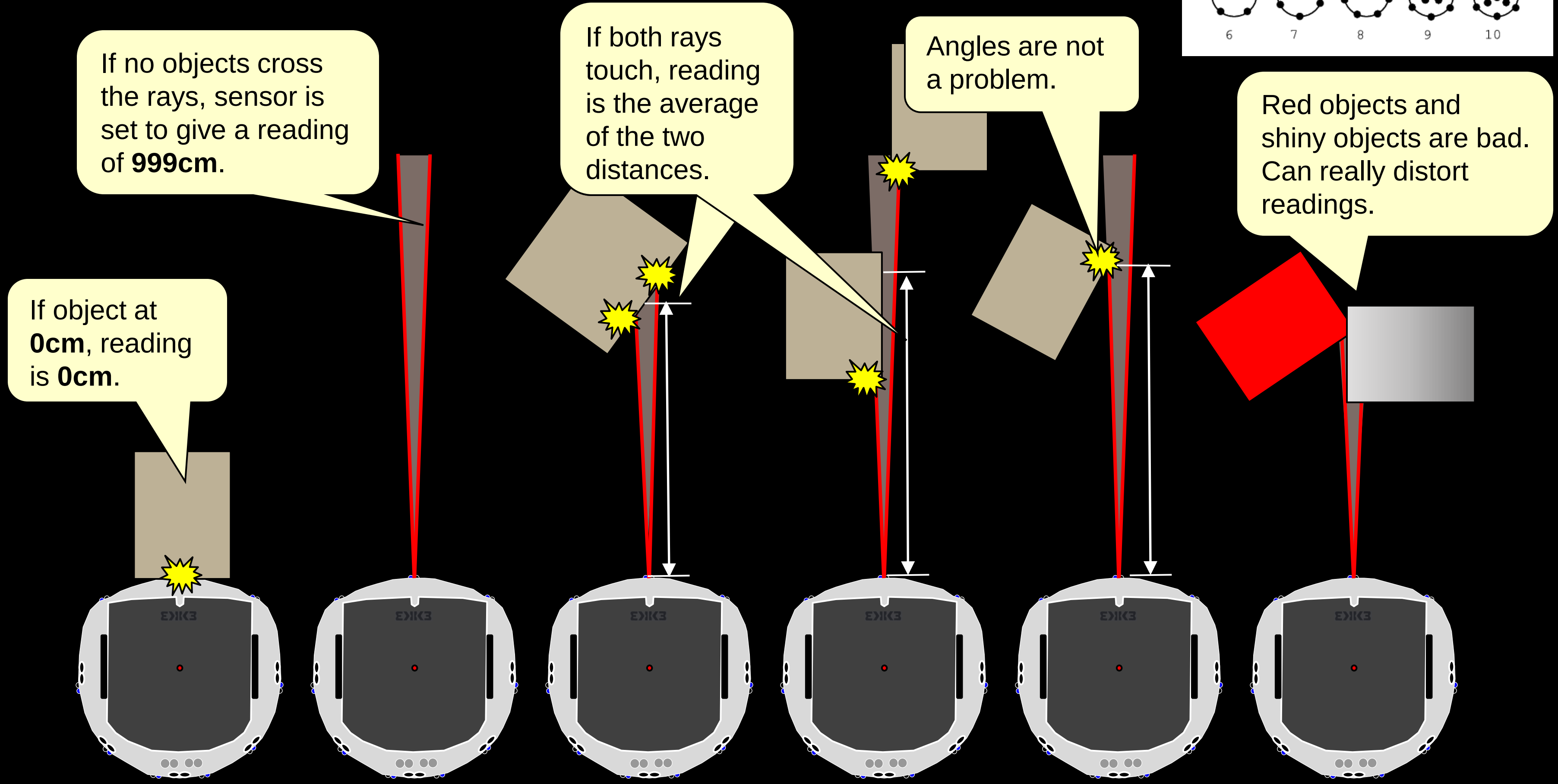
Webots Sonar Sensor Simulation



Webots IR Sensor Simulation

- Our IR sensors are set to use just 3 rays:





Webots Sensor Distance Error

- Our sensors are set to give an accurate reading ± some error each time.
  - These are defined in the look-up tables of the **Khepera3_DistanceSensor.proto** and **Khepera3_USSensor.proto** files.

IR Sensors have a distance error of **±3.0%**

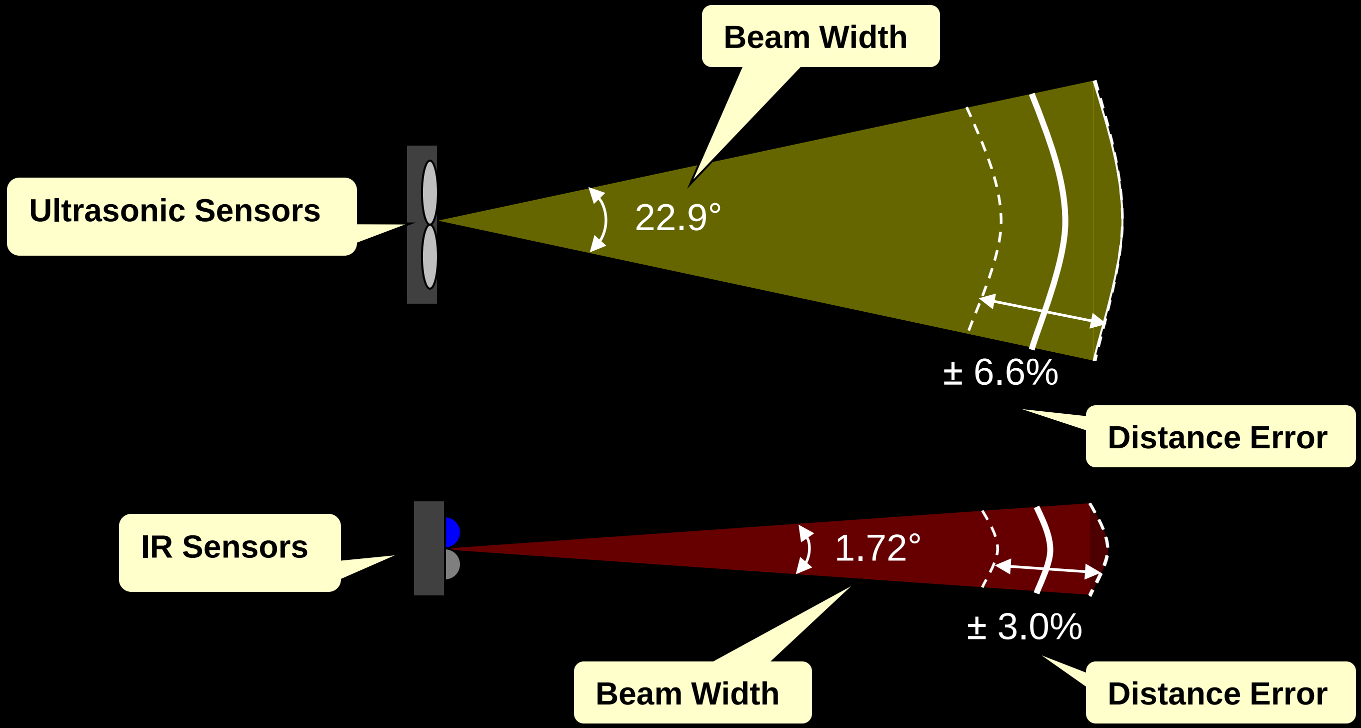
Range	Error
0cm – 5cm	±0.0cm
5cm	±0.01cm
10cm	±0.04cm
30cm	±0.3cm
60cm	±1.25cm
100cm	±3cm

Ultrasonic Sensors have a distance error of **±6.6%**

Range	Error
0cm – 5cm	±0.0cm
5cm	±0.03cm
10cm	±0.1cm
30cm	±0.7cm
60cm	±2.5cm
100cm	±6.6cm

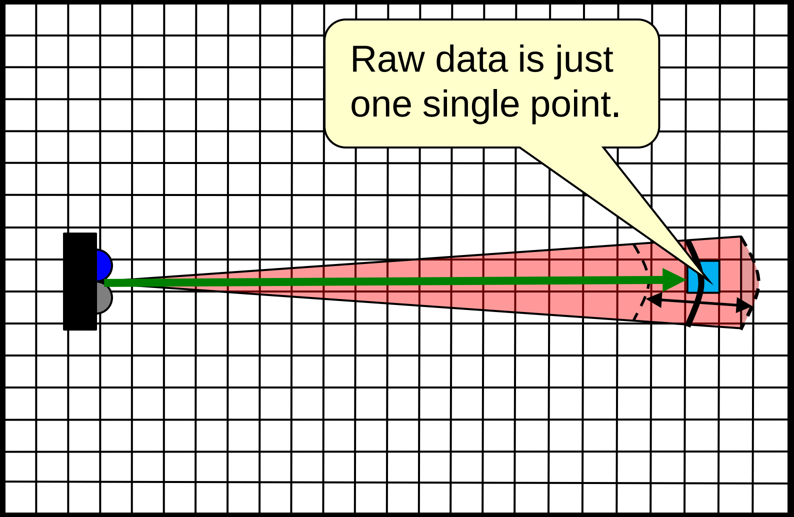
Our Sensor Models

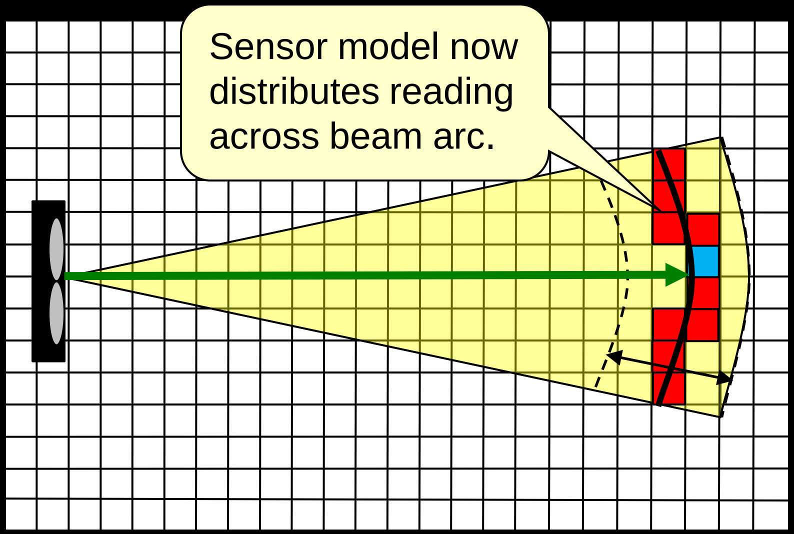
- Here are the two sensor models that we will use:



Applying A Sensor Model

- When we detect an obstacle, we will project the sensor model onto the grid by spreading the reading across the beam arc, since the obstacle is not necessarily at the center.





Sensor Model Implementation

- How do we compute which cells are affected by our sensor model ?
  - Need to determine the cells covered by each arc.

- First, compute arc endpoints:

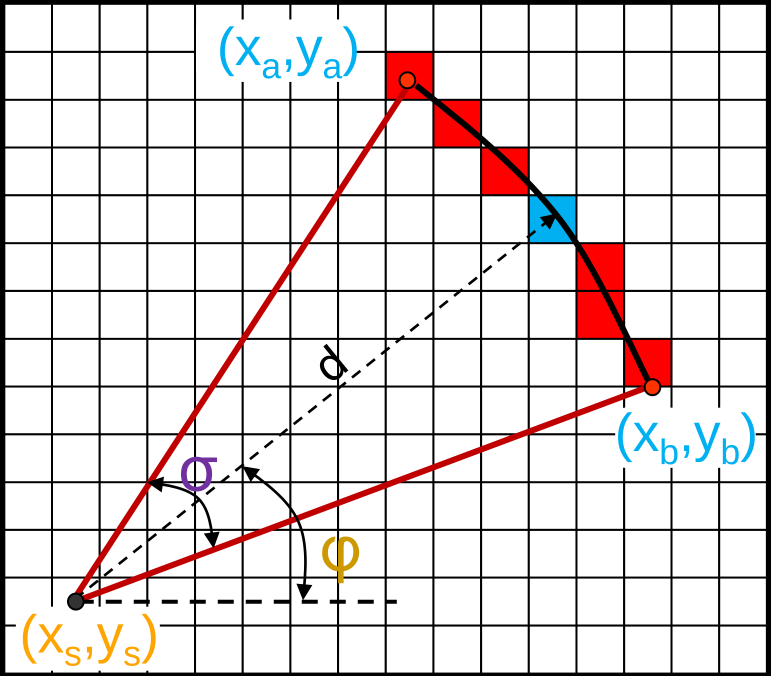
(x_s,y_s) is the sensor's location, not robot's

φ is the sensor's direction, not robot's

$$x_a = x_s + d * COS(\varphi + \sigma/2)$$
$$y_a = y_s + d * SIN(\varphi + \sigma/2)$$
$$x_b = x_s + d * COS(\varphi - \sigma/2)$$
$$y_b = y_s + d * SIN(\varphi - \sigma/2)$$

d is the sensor's range reading

σ is the sensor's beam width



Sensor Model Implementation

- Then compute the angular interval so that we cover each cell along the arc once (roughly):

This will be a float.

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

- Now go to each grid cell along the arc and increment it accordingly:

If you find that some cells are being skipped, then you can set **ω** = **ω** / 2 to double-up on the number of cells to fill in.

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

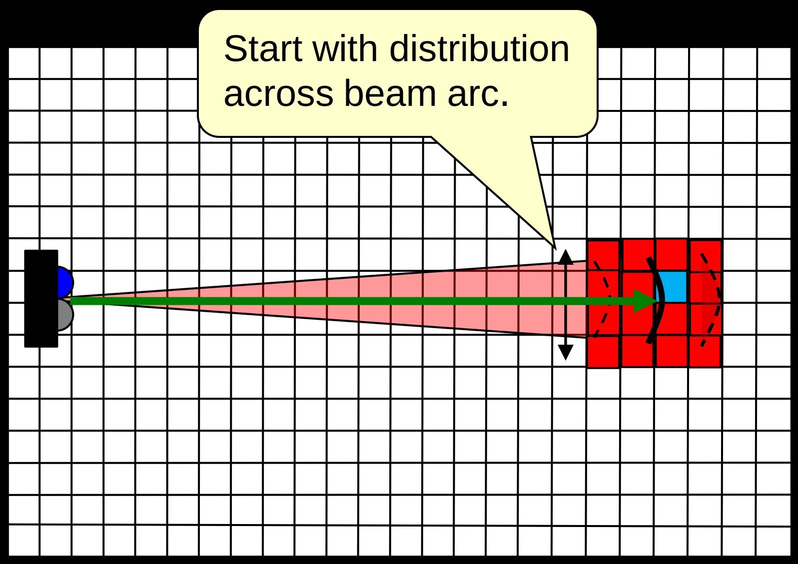
This will be a **double**

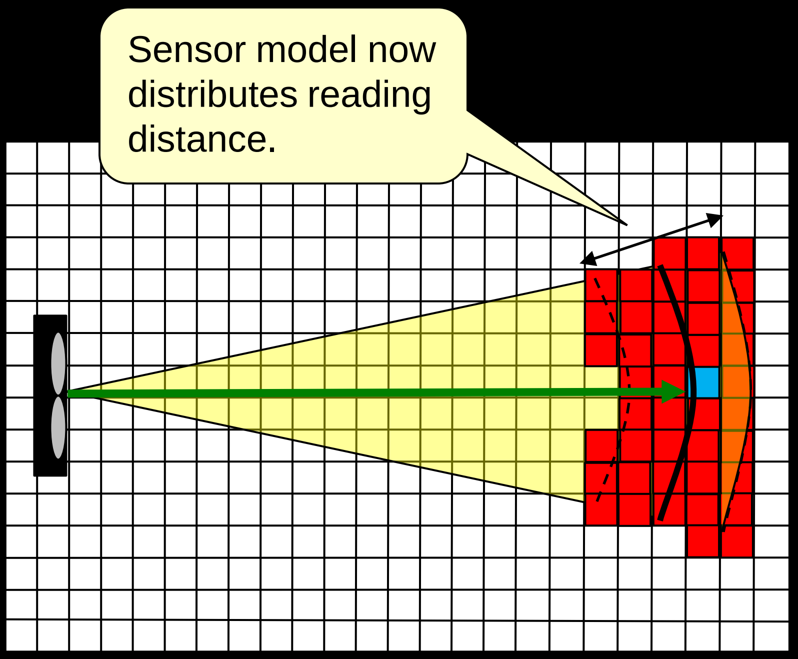
φ is the <u>sensor</u>'s direction, which is the <u>robot's angle</u> plus the sensor's <u>angle offset</u>.

But first make sure that (**objX**, **objY**) is within the grid range!!

Applying A Sensor Model

- We will also distribute the reading according to the distance error to accommodate inaccurate range readings from the sensor.





Sensor Model Implementation

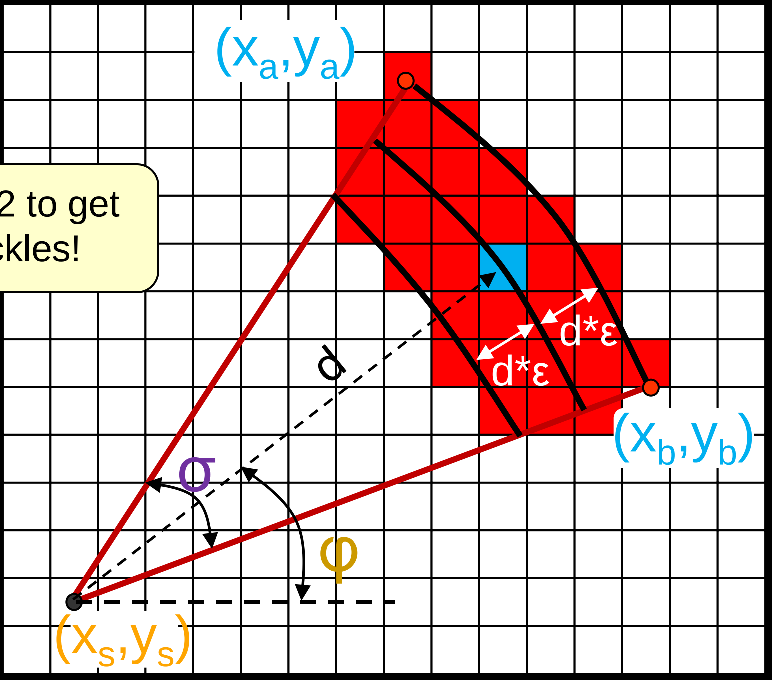
- How do we do this ?
  - we iterate through ranges corresponding to the range ε defined by the % distance error (e.g., 0.03 for 3%).

Choose an INC which is less than 1 to make sure that no cells are skipped. Perhaps INC = 0.25.

```
FOR r = d*(1-ε) TO d*(1+ε) BY INC DO {
    xa = xs + (r * COS(φ + σ/2))
    ya = ys + (r * SIN(φ + σ/2))
    xb = xs + (r * COS(φ - σ/2))
    yb = ys + (r * SIN(φ - σ/2))
   ω = (σ / √((xa-xb)² + (ya-yb)²))/2

    FOR a = -σ/2 TO σ/2 BY ω DO {
        objX = xs + (r * cos(φ + a))
        objY = ys + (r * sin(φ + a))
        set grid at (objX, objY) = 1
    }
}
```

Divide by 2 to get rid of speckles!



Start the Lab …