Probabilistic Robotics and Particle Filtering

A Brief Intro

Justification

- Robotics is by nature a very messy subject
  - Sensors are noisy
  - Actuators are imperfect
- We rarely ever know anything “for sure.”
  - We can only collect evidence to try to make educated assumptions.

Justification

- Robotics is by nature a very messy subject
  - Sensors are noisy
  - Actuators are imperfect
- We rarely ever know anything "for sure."
  - We can only collect evidence to try to make educated assumptions.

For Example...

For Example...

- An IR rangefinder can tell us if we are likely to be near a wall or not
- A camera can tell us if there is a good chance of a colored stuffed doll being in front of us

However...

- We are making a big leap between sensing and perception here.
  - Just because a rangefinder gives us a certain voltage does not guarantee that there is an actual obstacle in our path
  - However, we can definitely say that if our rangefinder reports a certain voltage there may be a <u>better chance</u> of an obstacle being in our way.

- Instead of considering a sensor output as a <u>certainty</u>, we can think of the likelihood that it is correct

Probabilities

- We Will Use Probabilistic Representations For:
  - World State
  - Sensor Models
  - Action Models
- Use the calculus of probability theory to combine these models

Quick Probability Review

- “Probability of x” - $P(x)$
  - $P(x)$ is a real number between 0 and 1 representing the “percent chance” of x occurring
- “Probability of x *and* y” - $P(x,y)$
  - If x and y are independent - $P(x)P(y)$
- “Probability of x *or* y”
  - If x and y are mutually exclusive - $P(x)+P(y)$
  - Otherwise - $P(x)+P(y)-P(x,y)$

Quick Probability Review

- For Example:
  - P(Rain Tomorrow) = .05
  - P(I wear green next week) = .93
    - P(Rain tomorrow, and I wear green next week) = (.05)*(.93)=.046
      - Note that we can argue *independence* for these two events

Quick Probability Review

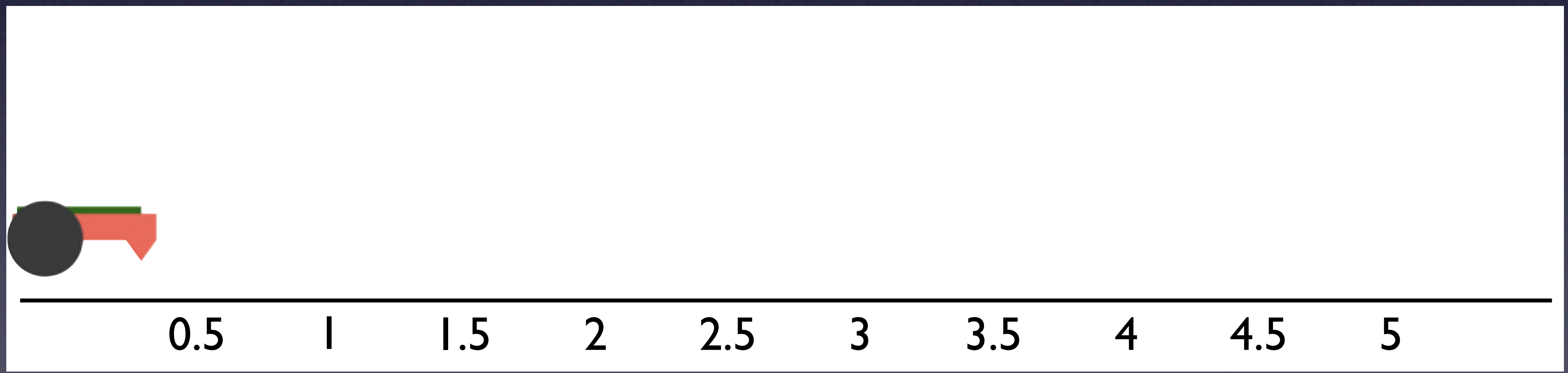
- “Probability of x *given* y” - P(x | y)
  - What is the probability of x, given that we have the prior knowledge of y being true
  - e.g. “P(Rand wearing raincoat | rainy outside)”

Probabilistic Localization

- The goal of today's lecture is to teach you to use probabilistic methods to represent both the motion and the perception of your robots.
- We will be using a 'Particle Filter' to represent these probability distributions.

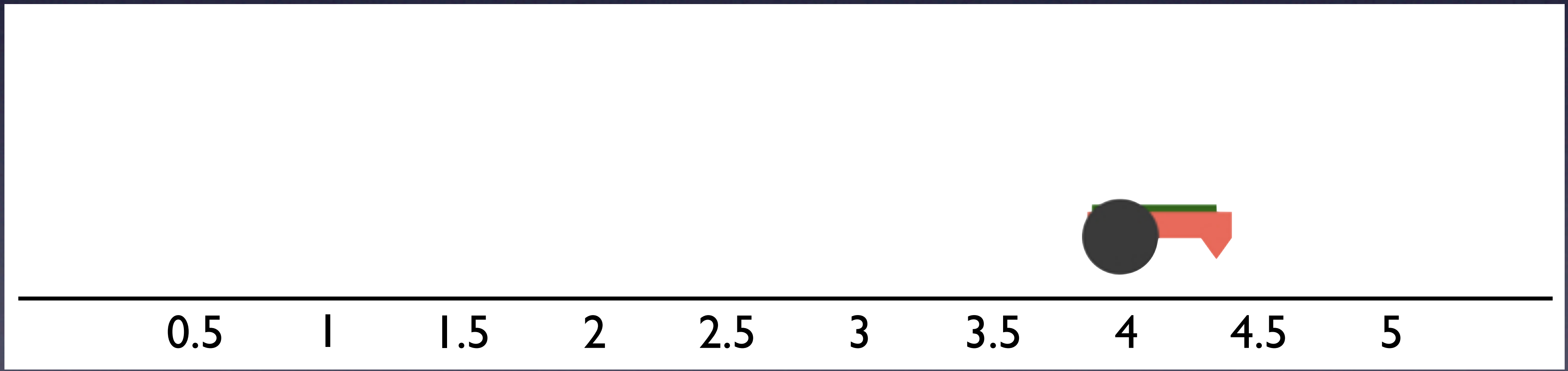
Particle Filter Motion Models

- We can describe every movement of your robots as a probability distribution.
  - For example, let's say we want our robot to just move forwards for 3.5 feet...



Particle Filter Motion Models

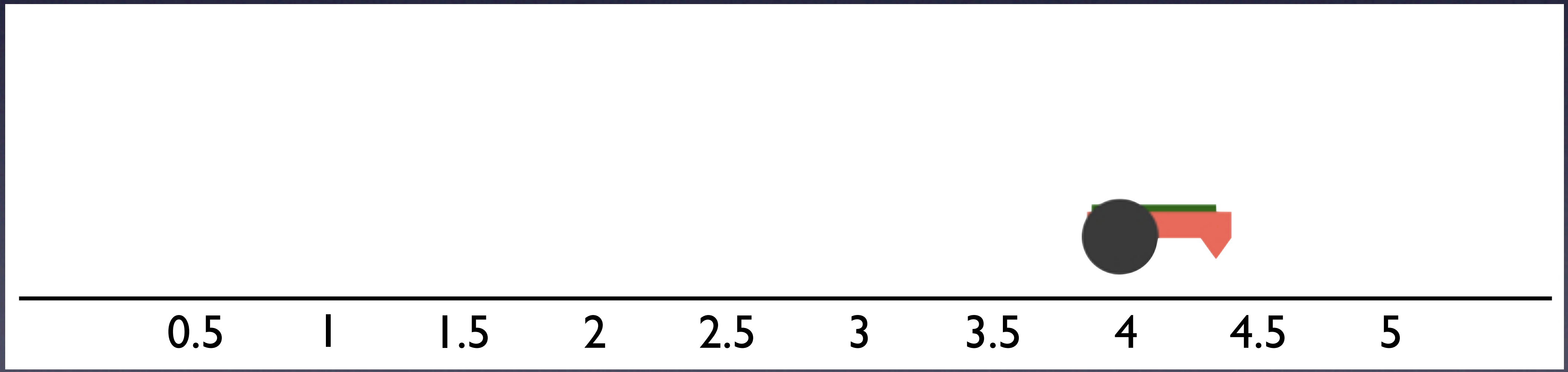
- We can describe every movement of your robots as a probability distribution.
  - For example, let’s say we want our robot to just move forwards for 3.5 feet...



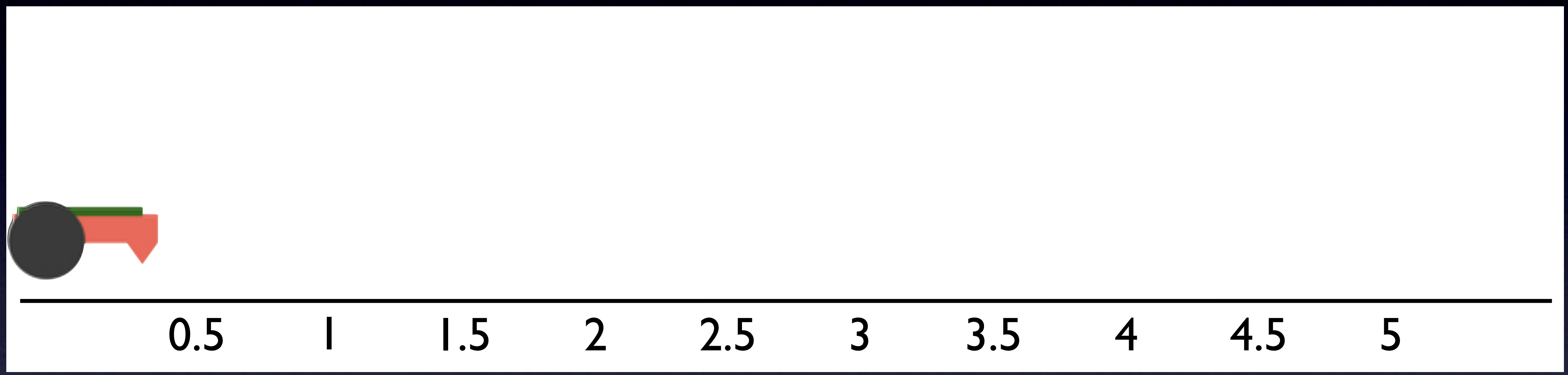
Particle Filter Motion Models

- We can describe every movement of your robots as a probability distribution.
  - For example, let's say we want our robot to just move forwards for 3.5 feet...

The motors are subject to noise!

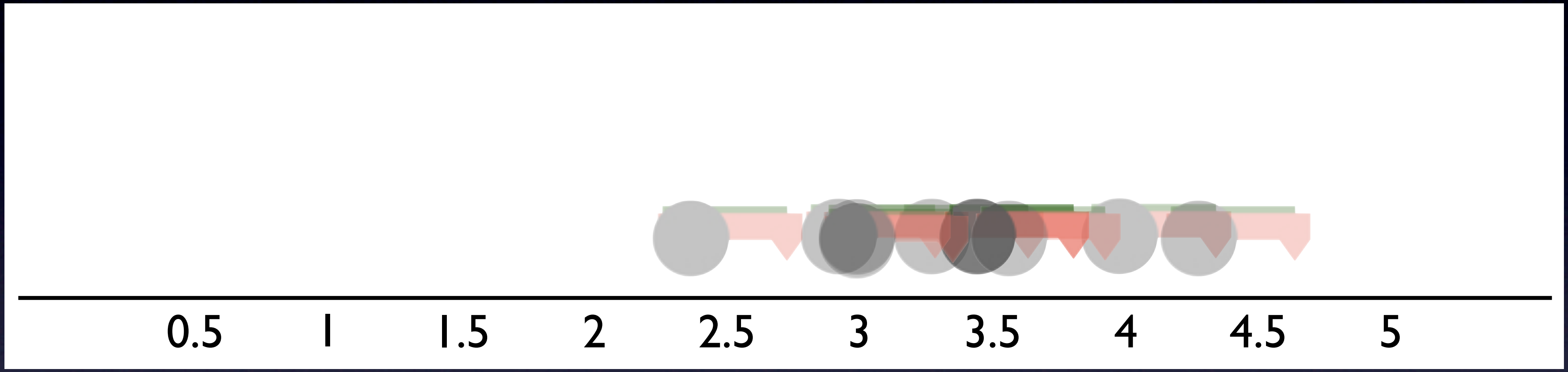


Particle Filter Motion Models



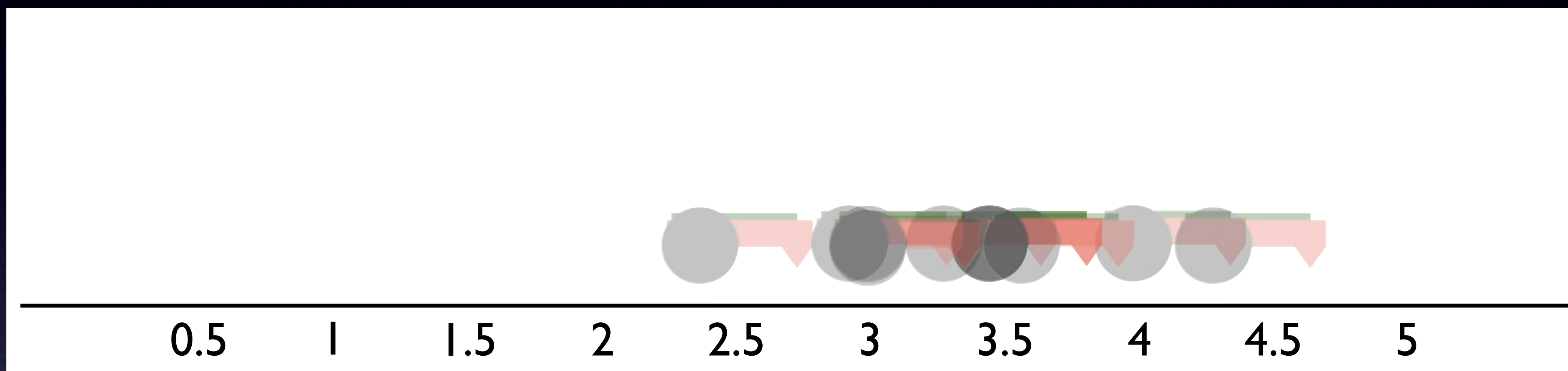
What if we were to tell our robot to move forwards 3.5 feet 100 different times?

Particle Filter Motion Models



What if we were to tell our robot to move forwards 3.5 feet 100 different times?

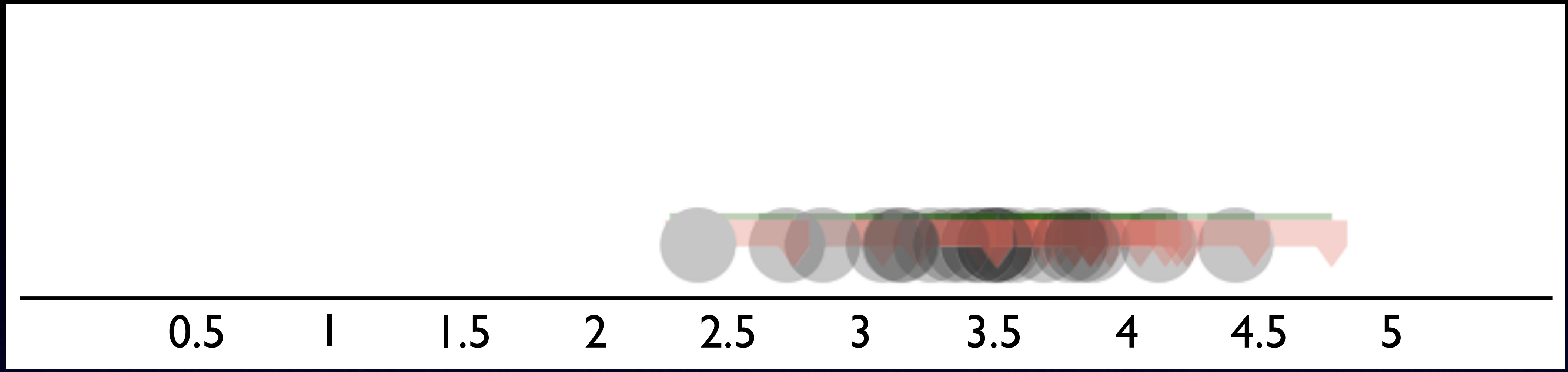
Particle Filter Motion Models

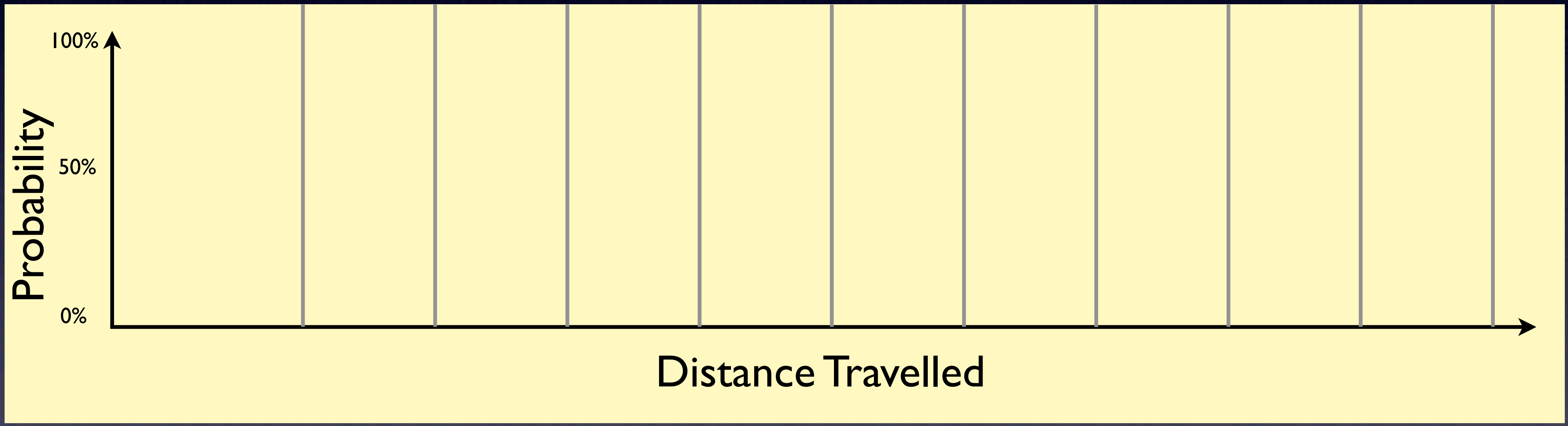


What if we were to tell our robot to move forwards 3.5 feet 100 different times?

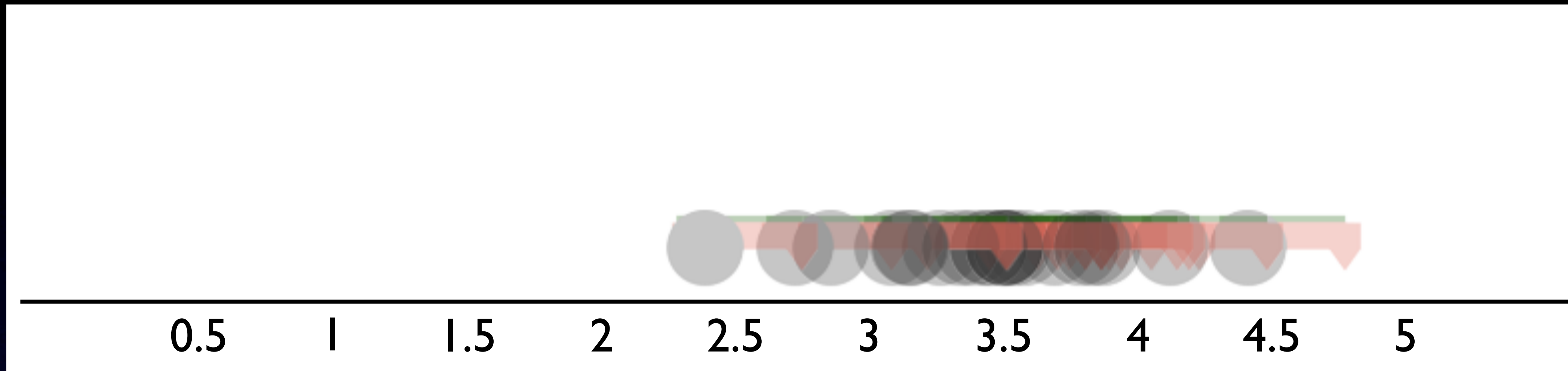
It would likely land in 100 different locations

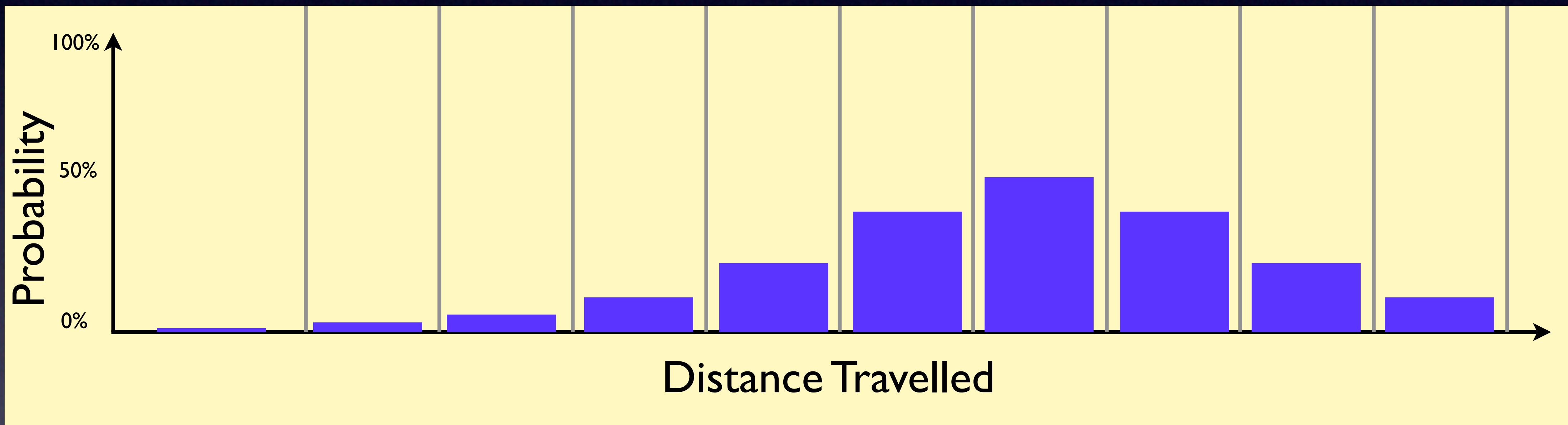
We can describe the chance nature of this movement with a probability distribution.



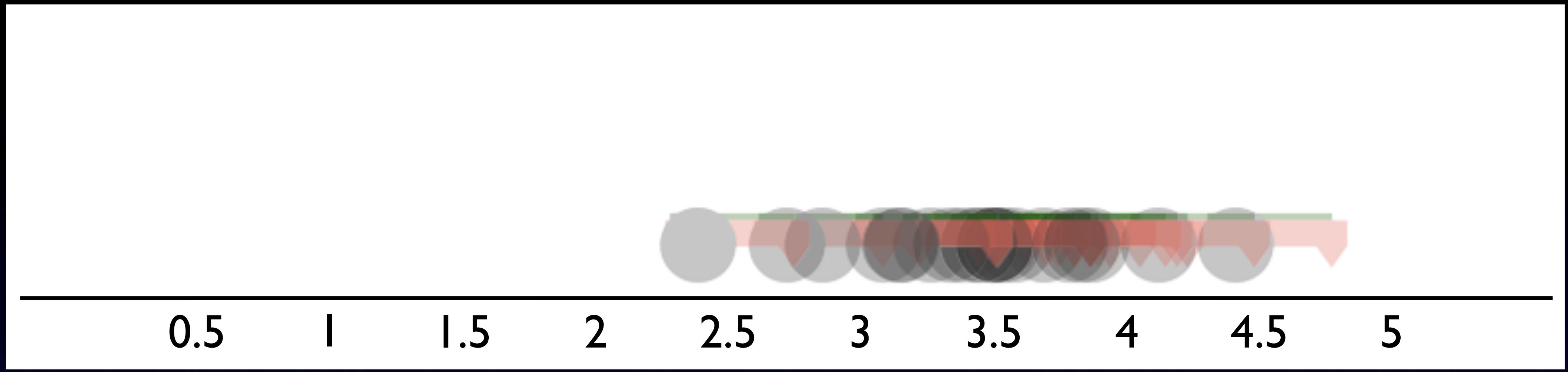


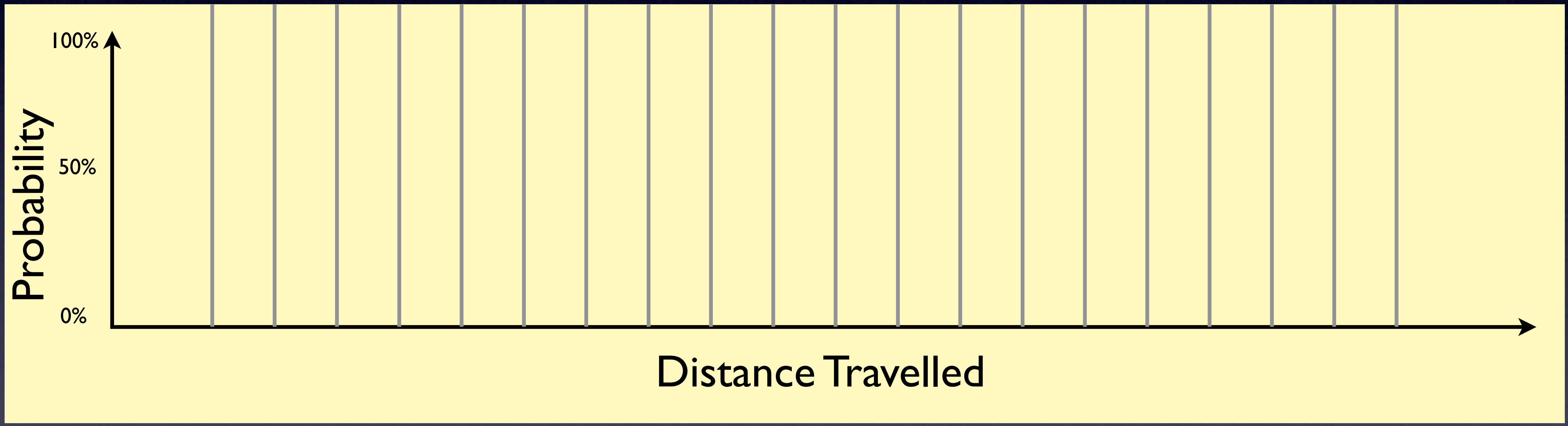
Let's break the track into 0.5 foot increments, and calculate the percentage of times our robot lands in each increment.



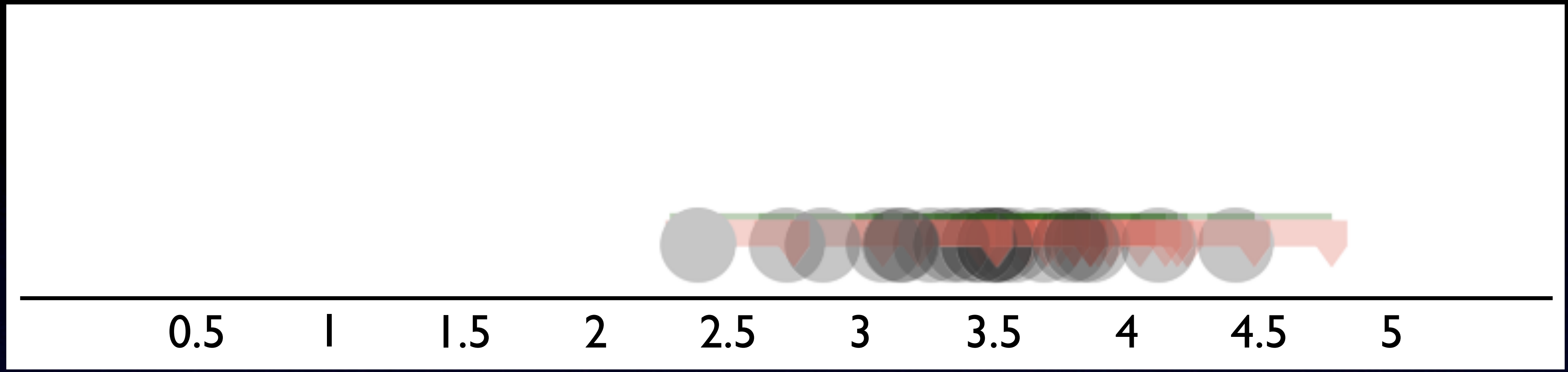


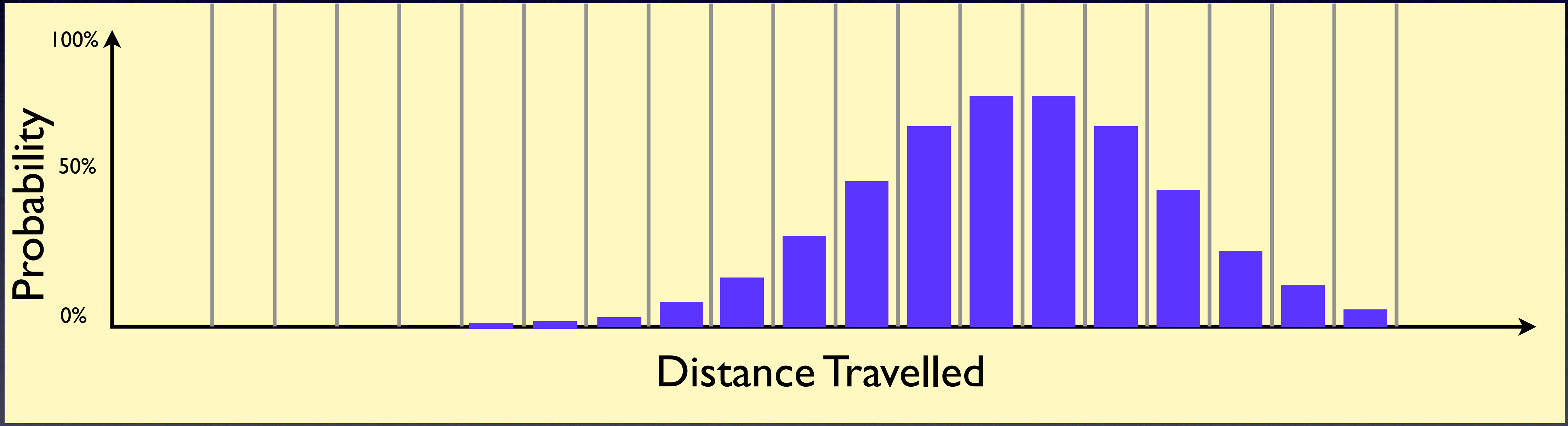
Let's break the track into 0.5 foot increments, and calculate the percentage of times our robot lands in each increment.



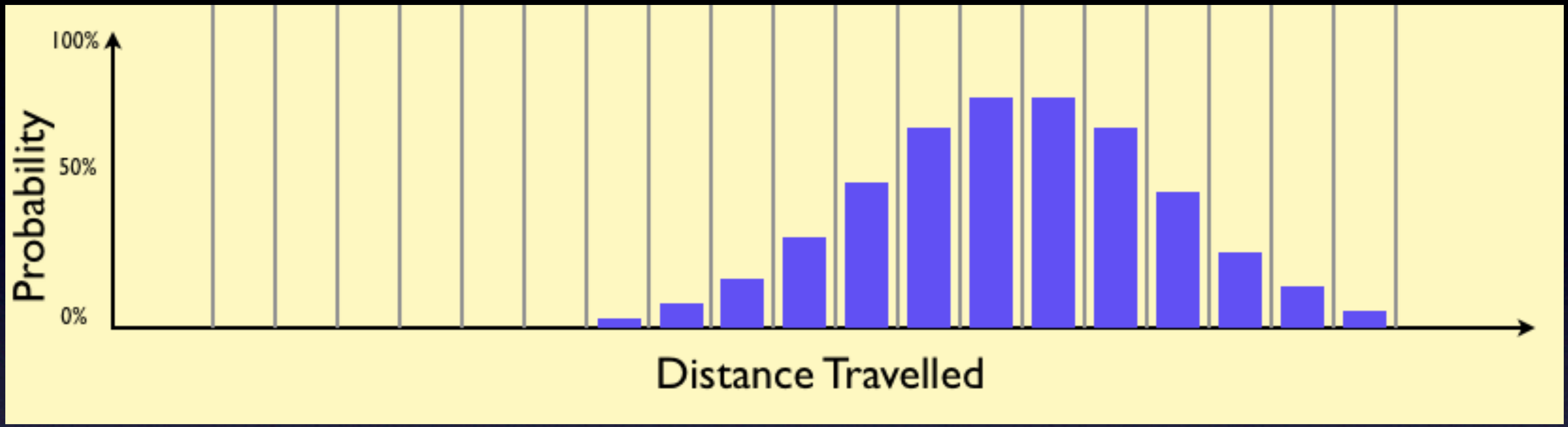


We can do the same thing with even smaller increments

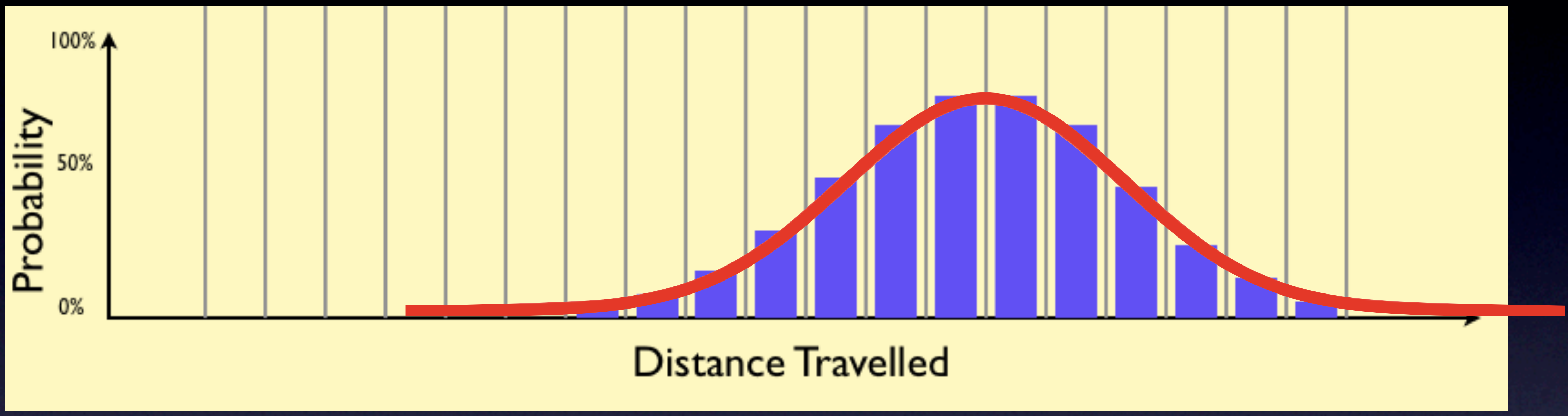




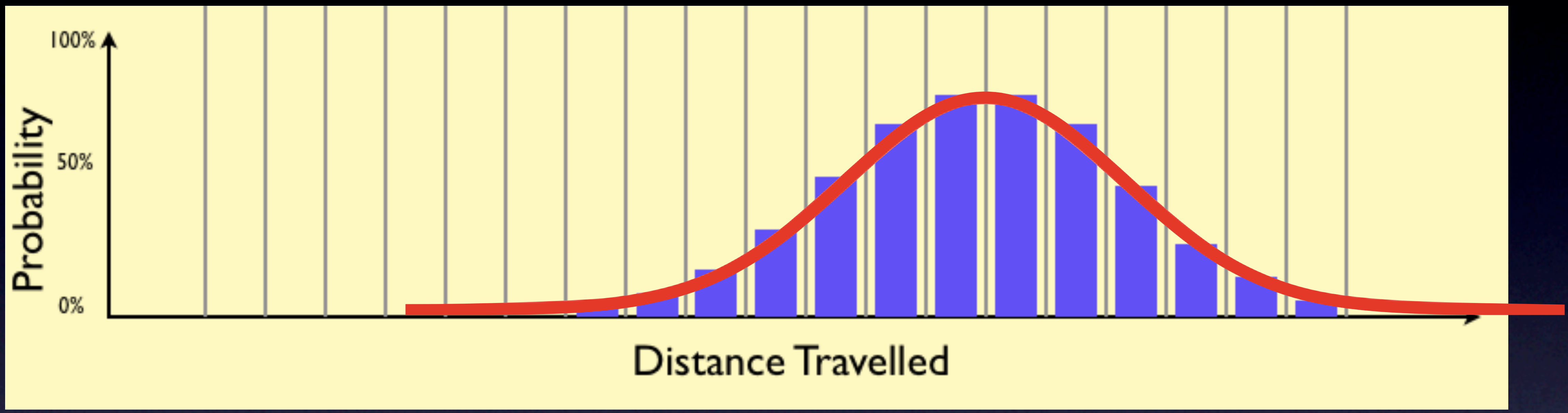
We can do the same thing with even smaller increments



This ‘bell curve’ shape is very common when describing noisy processes.



This ‘bell curve’ shape is very common when describing noisy processes.



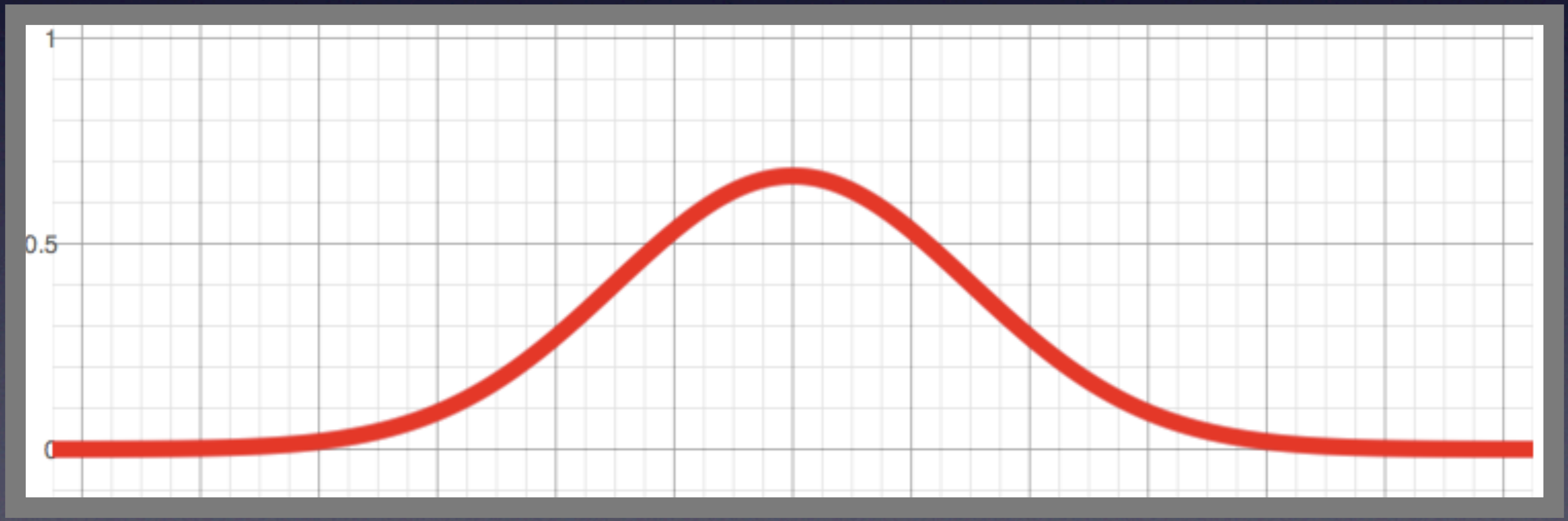
This ‘bell curve’ shape is very common when describing noisy processes.

It is called the ‘Gaussian’ or ‘Normal’ distribution

Gaussian Distribution

The Gaussian Distribution is not the only way to describe a random process, but it is one of the easiest and most flexible.

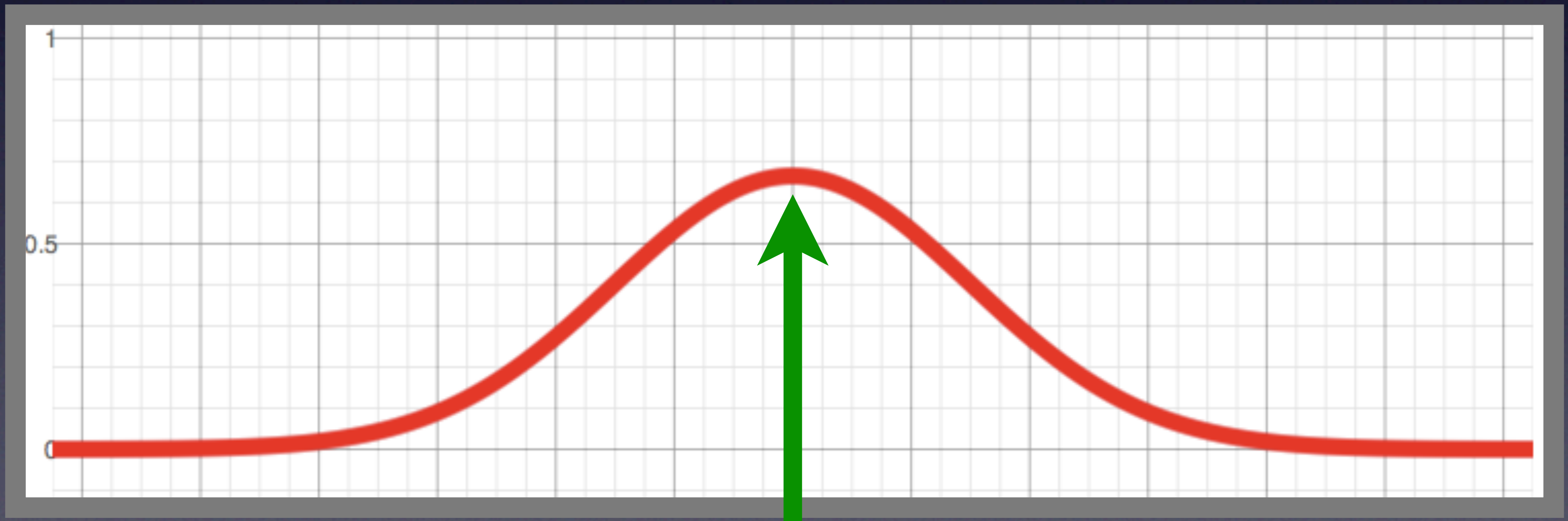
It often does a pretty good job of describing our physical systems



Gaussian Distribution

The Gaussian Distribution is not the only way to describe a random process, but it is one of the easiest and most flexible.

It often does a pretty good job of describing our physical systems

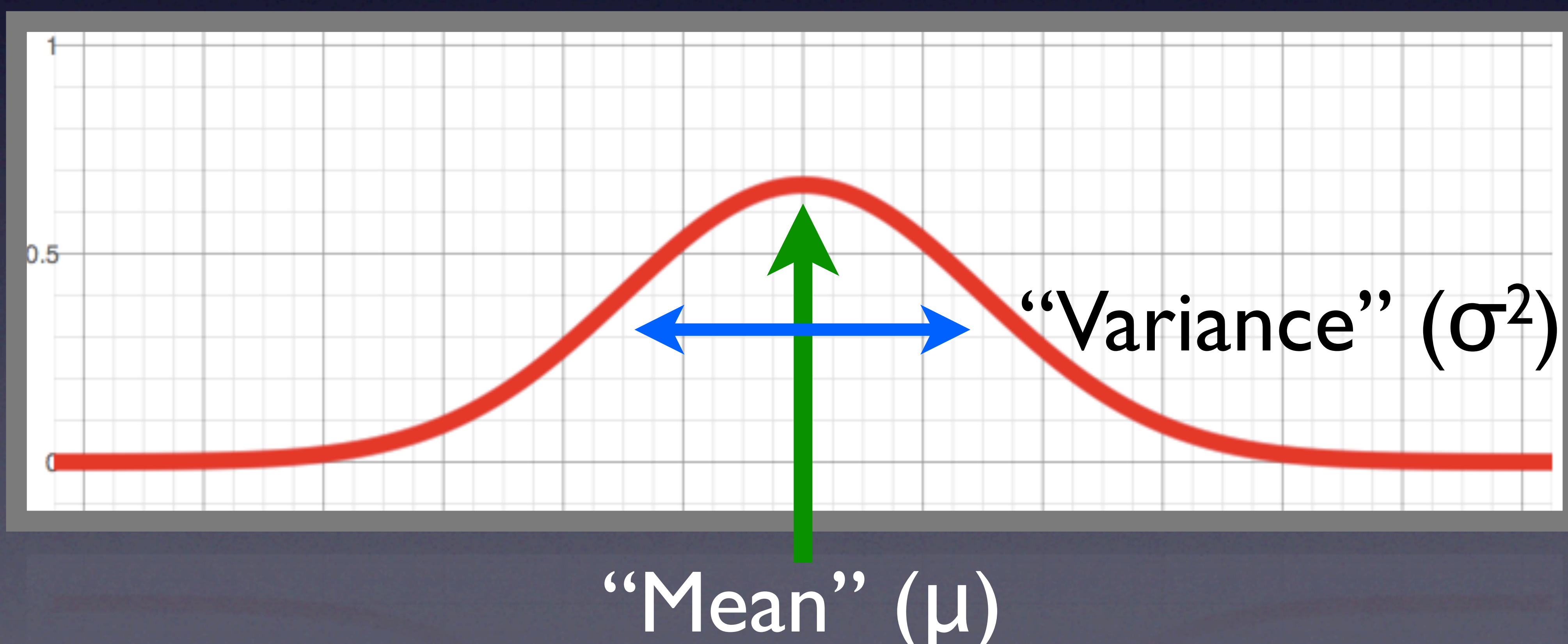


“Mean” (μ)

Gaussian Distribution

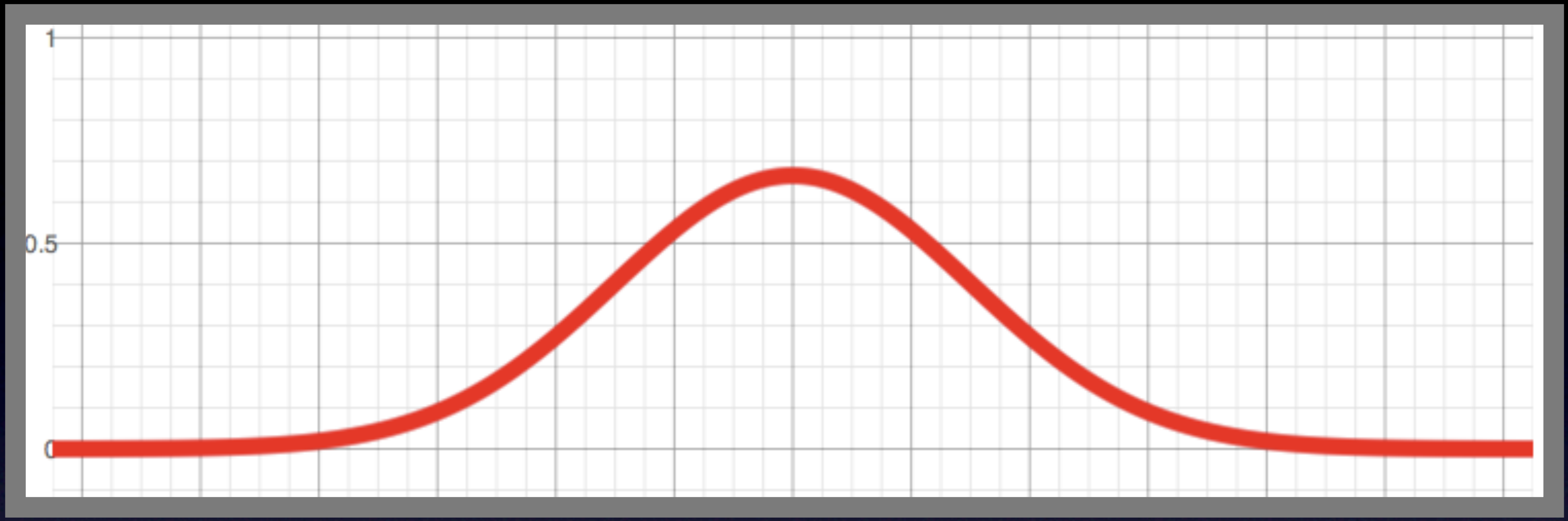
The Gaussian Distribution is not the only way to describe a random process, but it is one of the easiest and most flexible.

It often does a pretty good job of describing our physical systems

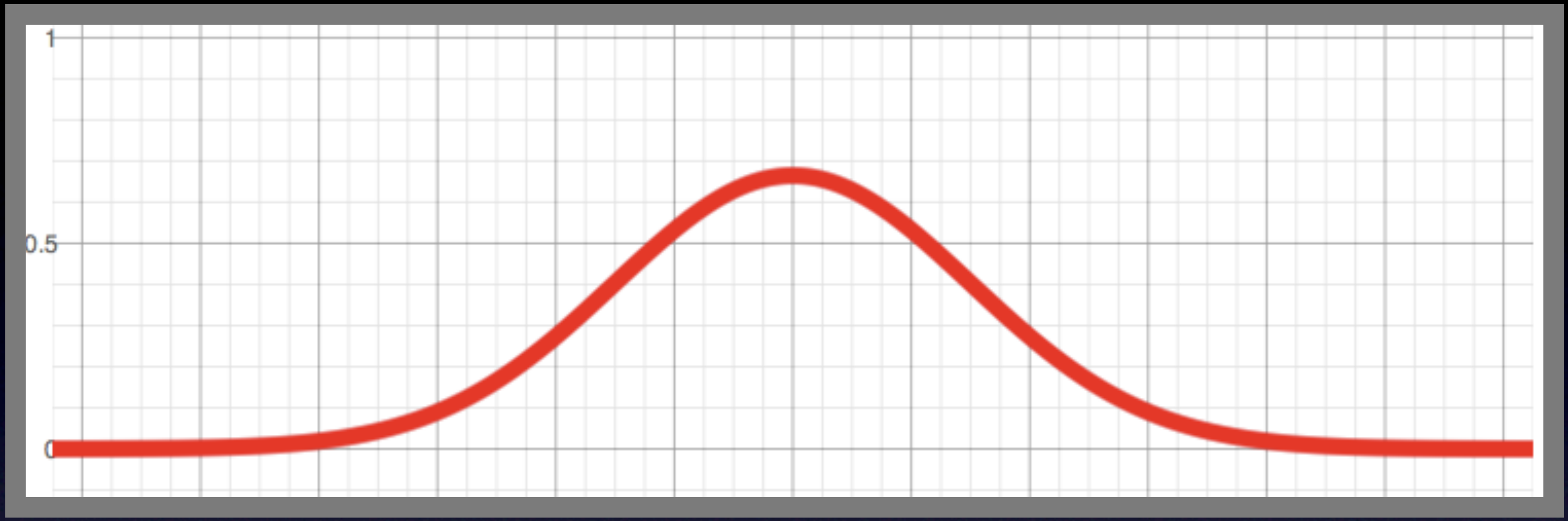


Just a few details...

Just a few details...

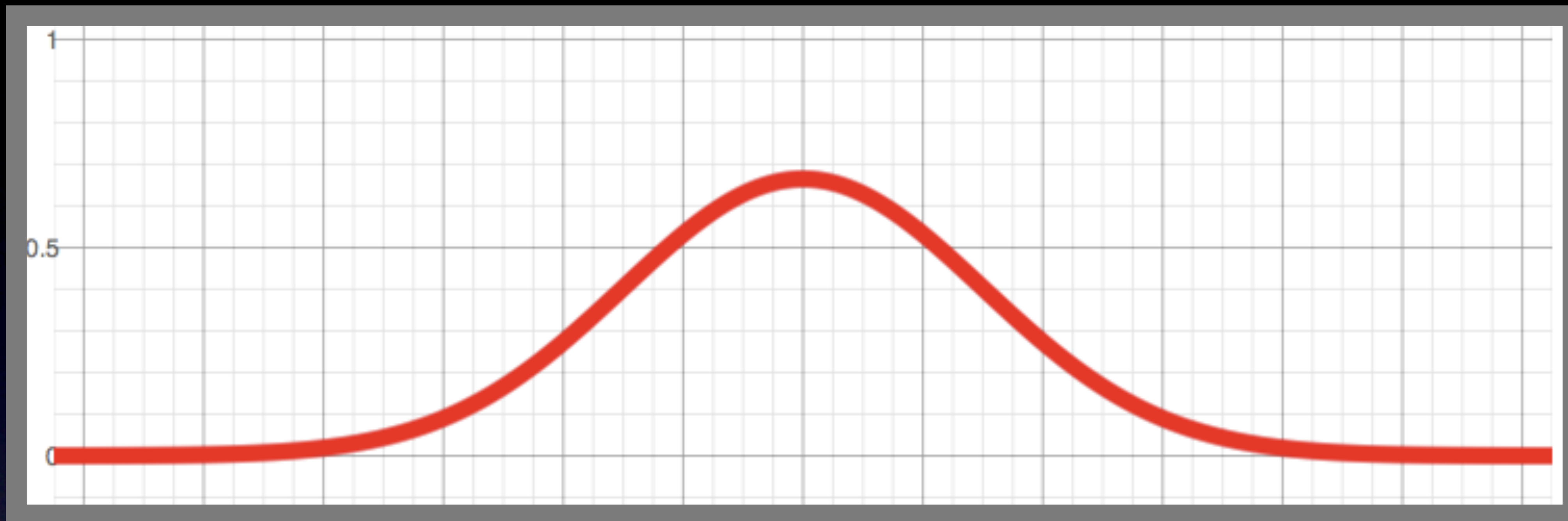


Just a few details...



$$\mathcal{N}(x;\mu,\sigma) = \left(\sqrt{2\pi\sigma^2}\right)^{-1} e^{-\frac{((x-\mu)2)}{(2\sigma 2)}}$$

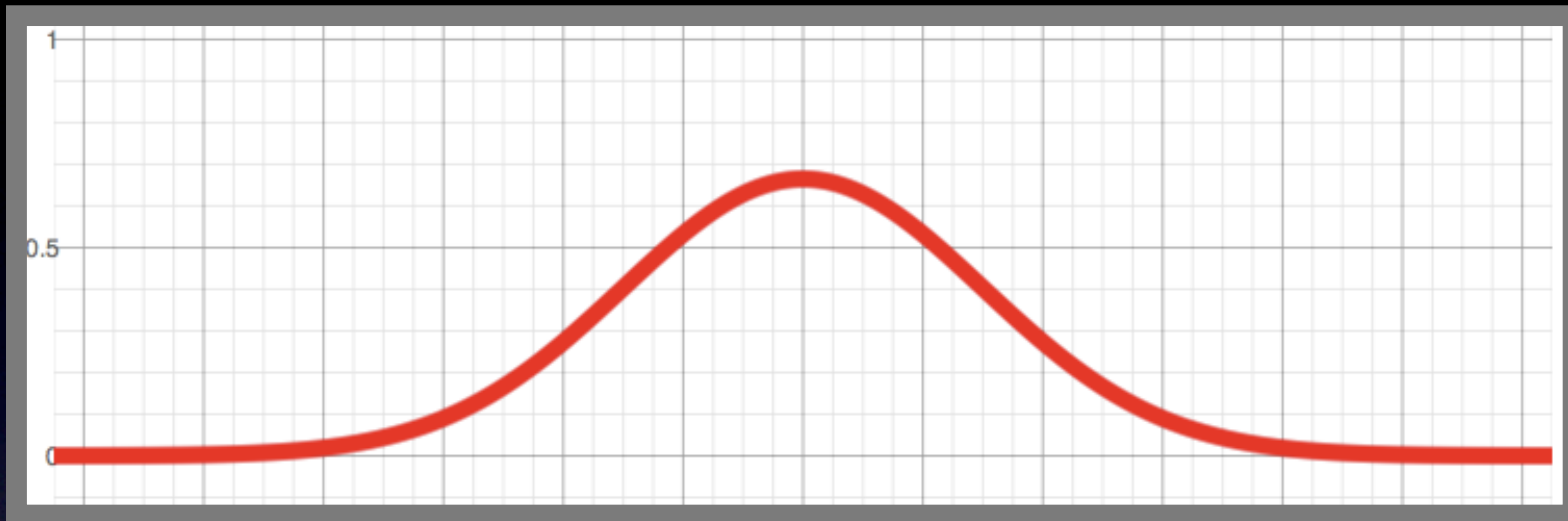
Just a few details...



$$\mathcal{N}(x;\mu,\sigma) = \left(\sqrt{2\pi\sigma^2}\right)^{-1} e^{-\frac{((x - \mu)2)}{(2\sigma 2)}}$$

This function is the 'probability density function' of the Normal Distribution

Just a few details...



$$\mathcal{N}(x;\mu,\sigma) = \left(\sqrt{2\pi\sigma^2}\right)^{-1} e^{-\frac{((x-\mu)^2)}{(2\sigma 2)}}$$

This function is the ‘probability density function’ of the Normal Distribution

It describes the probability of getting a value of ‘x’ if ‘x’ is sampled from a Normal Distribution with mean μ, and variance σ^2

Sampling From a Gaussian Distribution

Sampling From a Gaussian Distribution

- Generating samples from a normal distribution takes a little thought...

Sampling From a Gaussian Distribution

- Generating samples from a normal distribution takes a little thought...
  - The standard rand() function draws only from a uniform distribution...

Sampling From a Gaussian Distribution

- Generating samples from a normal distribution takes a little thought...
  - The standard rand() function draws only from a uniform distribution...

```
// From CSCI-445/gumstix/Util/MathFunctions.C
// ######################################################################
double randomDouble()
{
  return double(rand()) / (double(RAND_MAX) + 1.0);
}

// ######################################################################
double randomDoubleFromNormal(const double s)
{
  double sum = 0;
  for(int i=0; i<12; i++){
    sum += randomDouble()*2*s - s;
  }
  return sum/2;
}
```

Sampling From a Gaussian Distribution

- Generating samples from a normal distribution takes a little thought...
  - The standard rand() function draws only from a uniform distribution...

```
// From CSCI-445/gumstix/Util/MathFunctions.C
// ##########################################################################
double randomDouble()
{
  return double(rand()) / (double(RAND_MAX) + 1.0);
}

// ##########################################################################
double randomDoubleFromNormal(const double s)
{
  double sum = 0;
  for(int i=0; i<12; i++){
    sum += randomDouble()*2*s - s;
  }
  return sum/2;
}
```

This is a very close approximation to a normal..

An Example

Let’s say we have a robot with a compass.

An Example

Let's say we have a robot with a compass.
This robot accepts commands to turn to particular angles
(which is does very well)

An Example

Let’s say we have a robot with a compass.

This robot accepts commands to turn to particular angles (which is does very well)

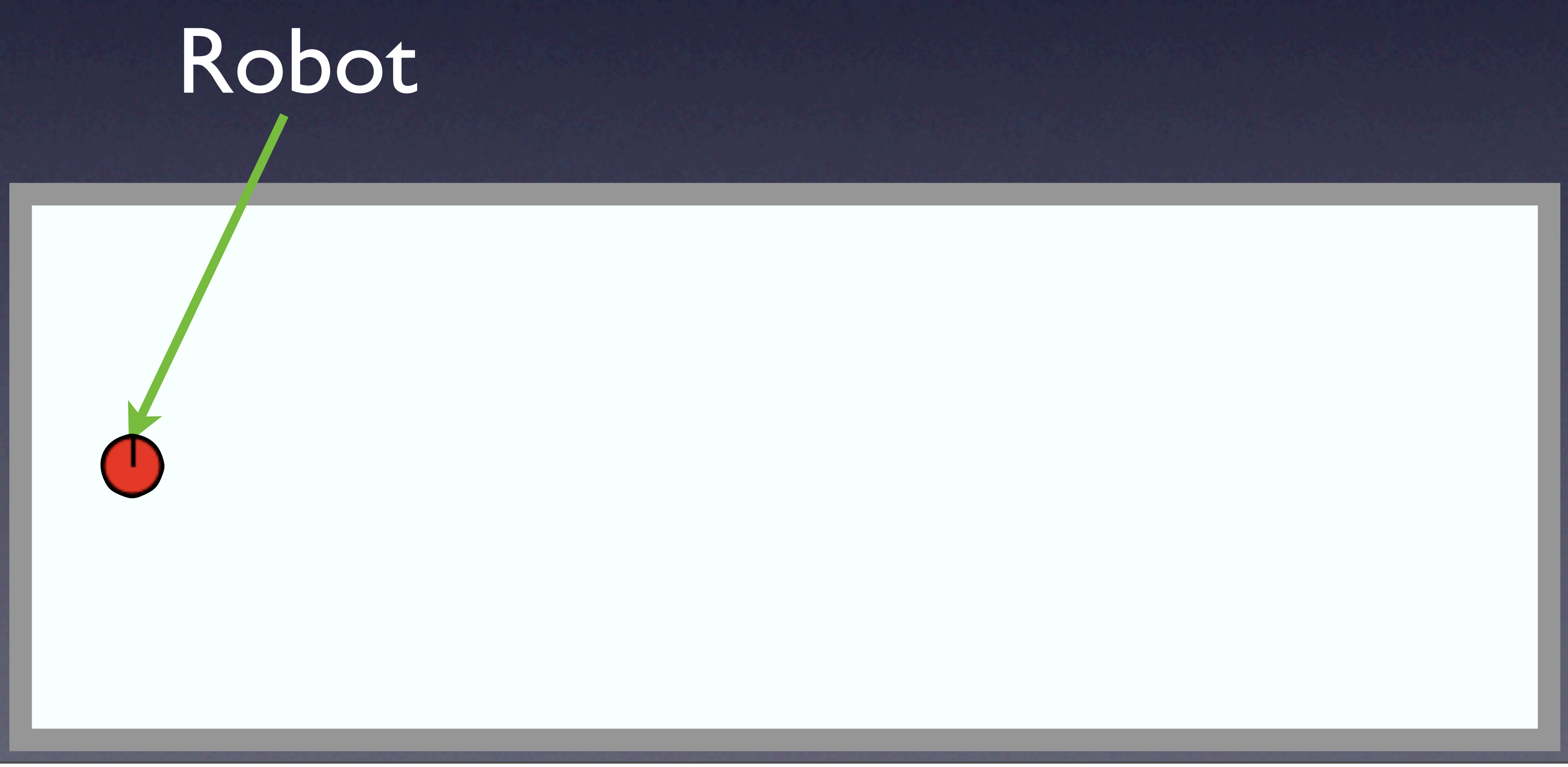
This robot also accepts commands to move forwards (which it does with some rotational and translational noise).

An Example

Let’s say we have a robot with a compass.

This robot accepts commands to turn to particular angles (which is does very well)

This robot also accepts commands to move forwards (which it does with some rotational and translational noise).

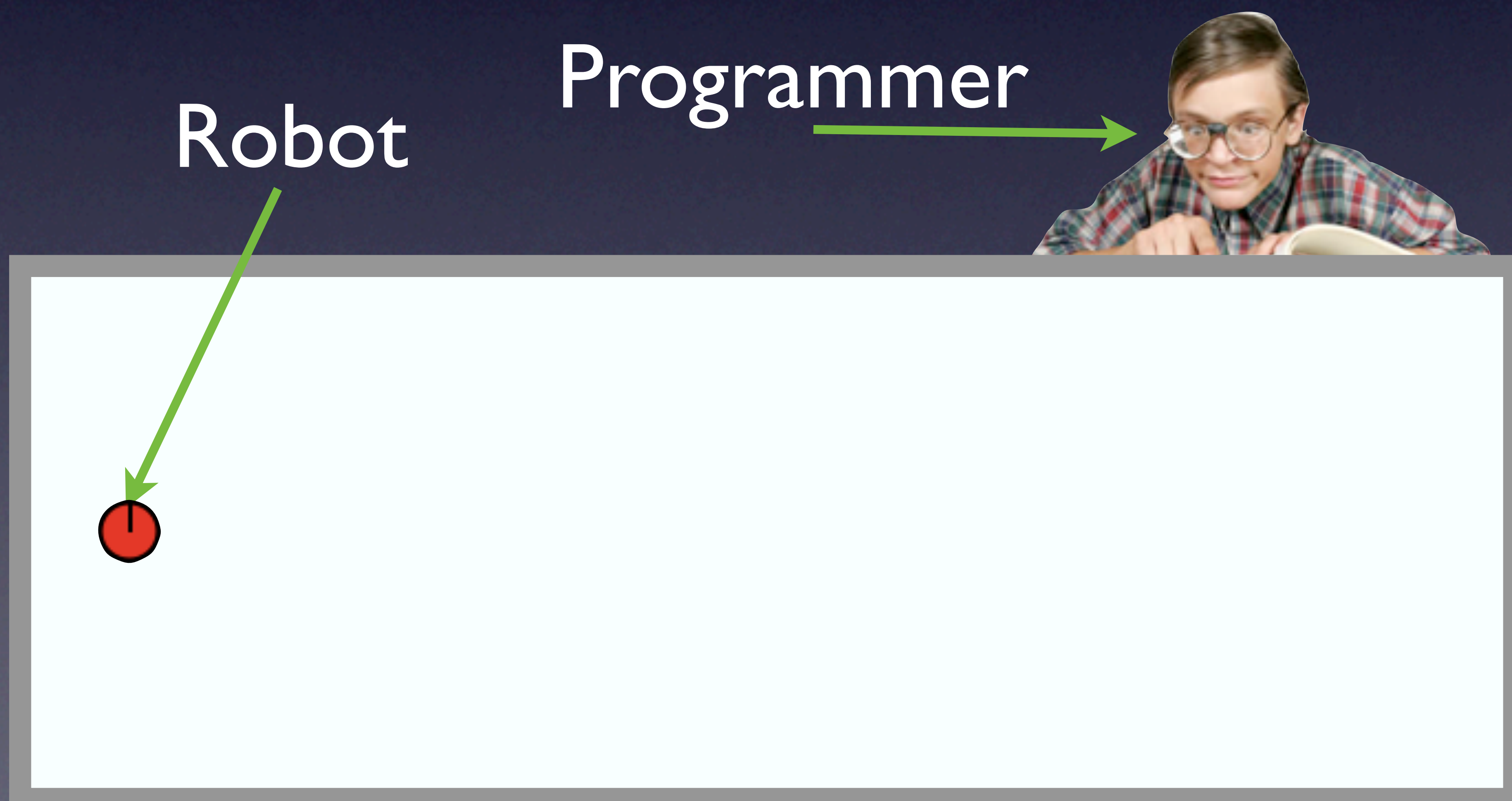


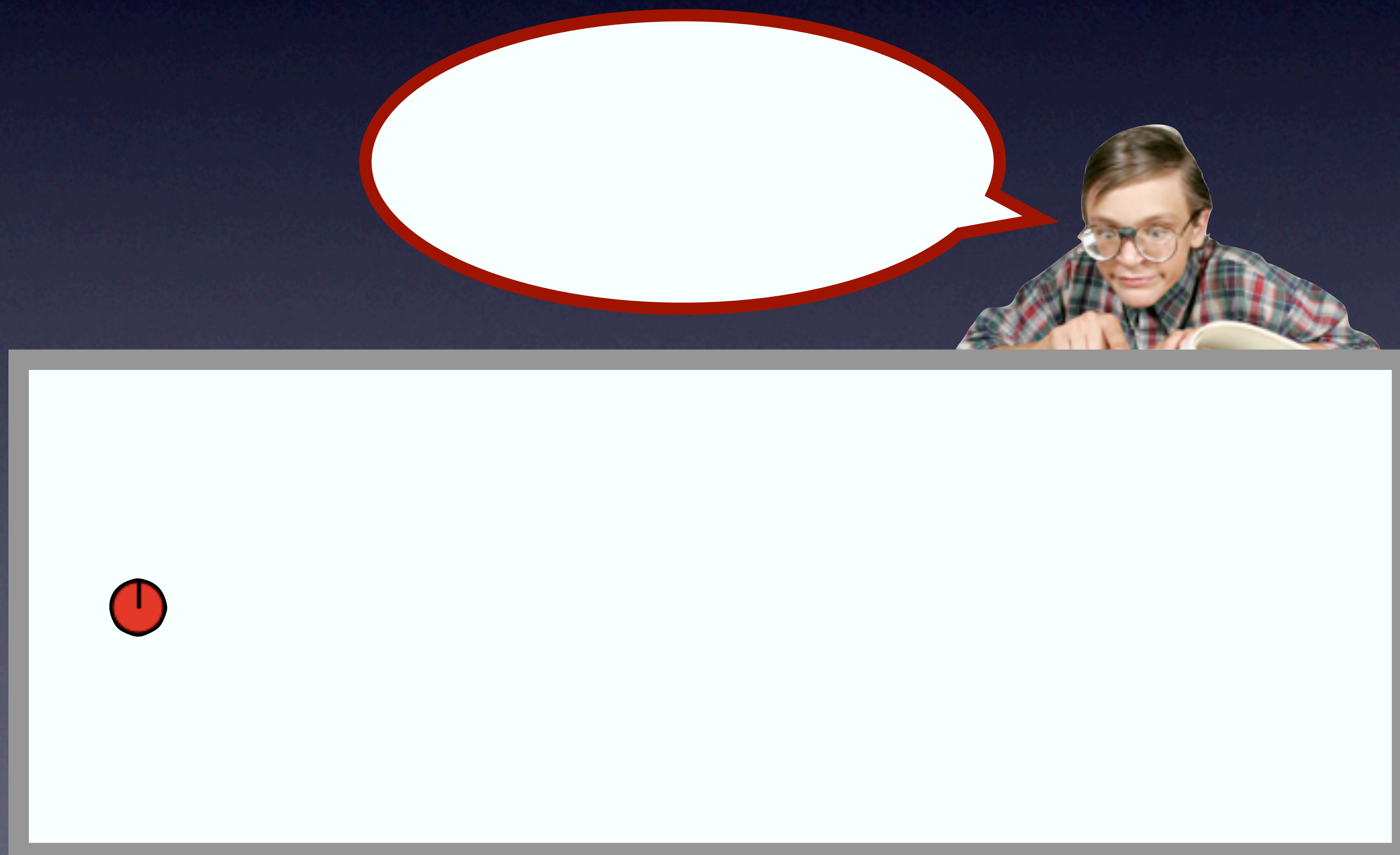
An Example

Let's say we have a robot with a compass.

This robot accepts commands to turn to particular angles (which is does very well)

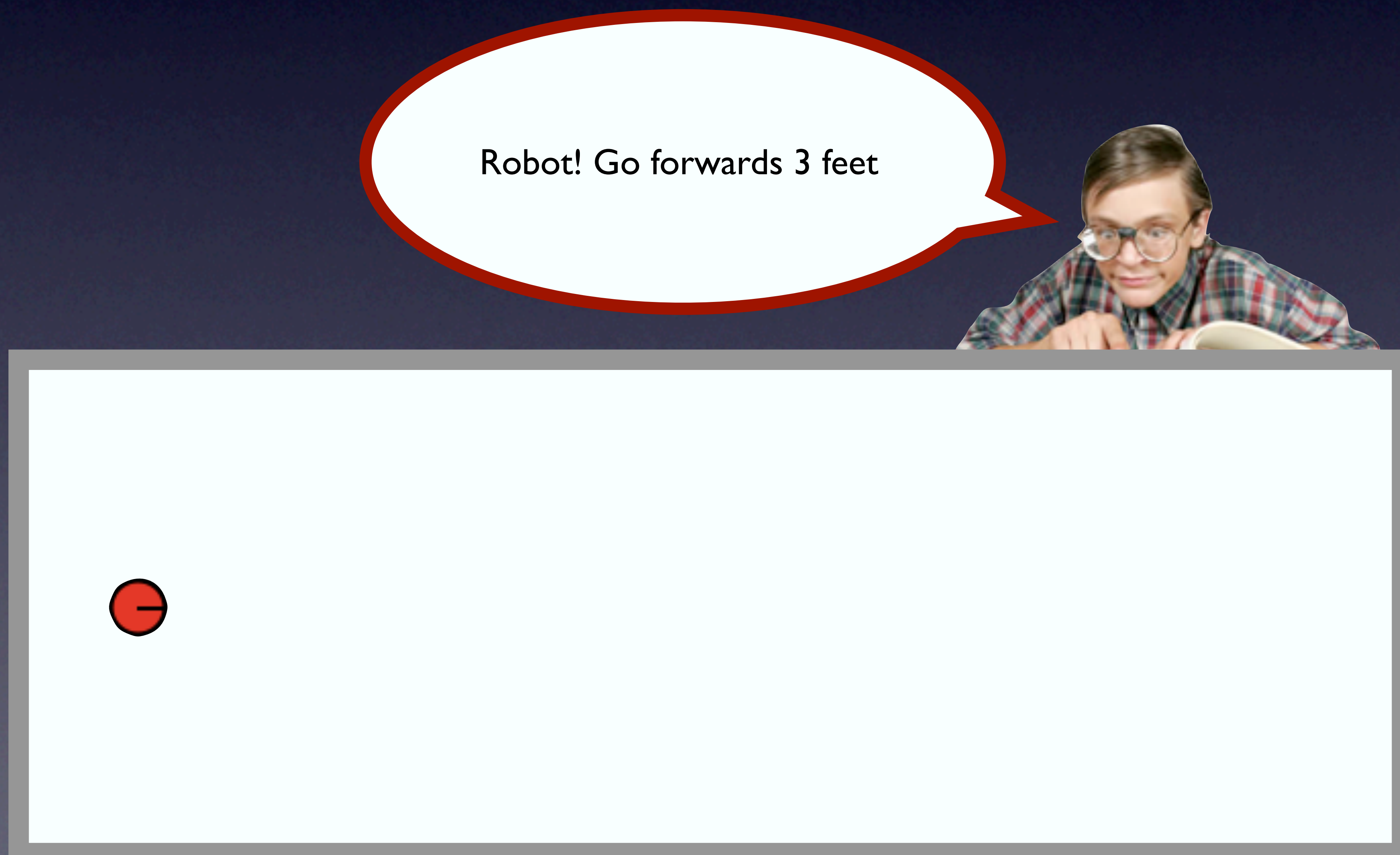
This robot also accepts commands to move forwards (which it does with some rotational and translational noise).

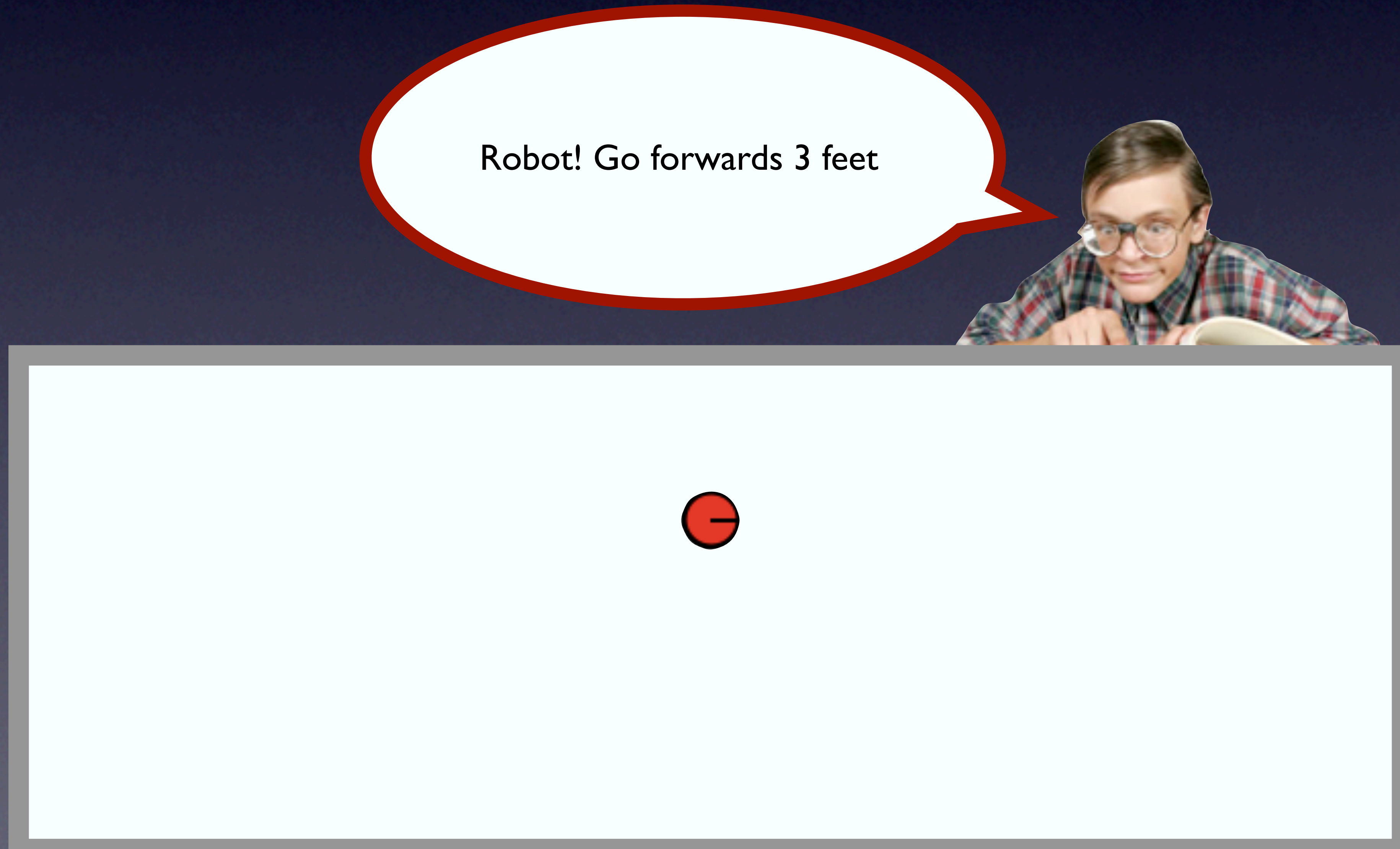


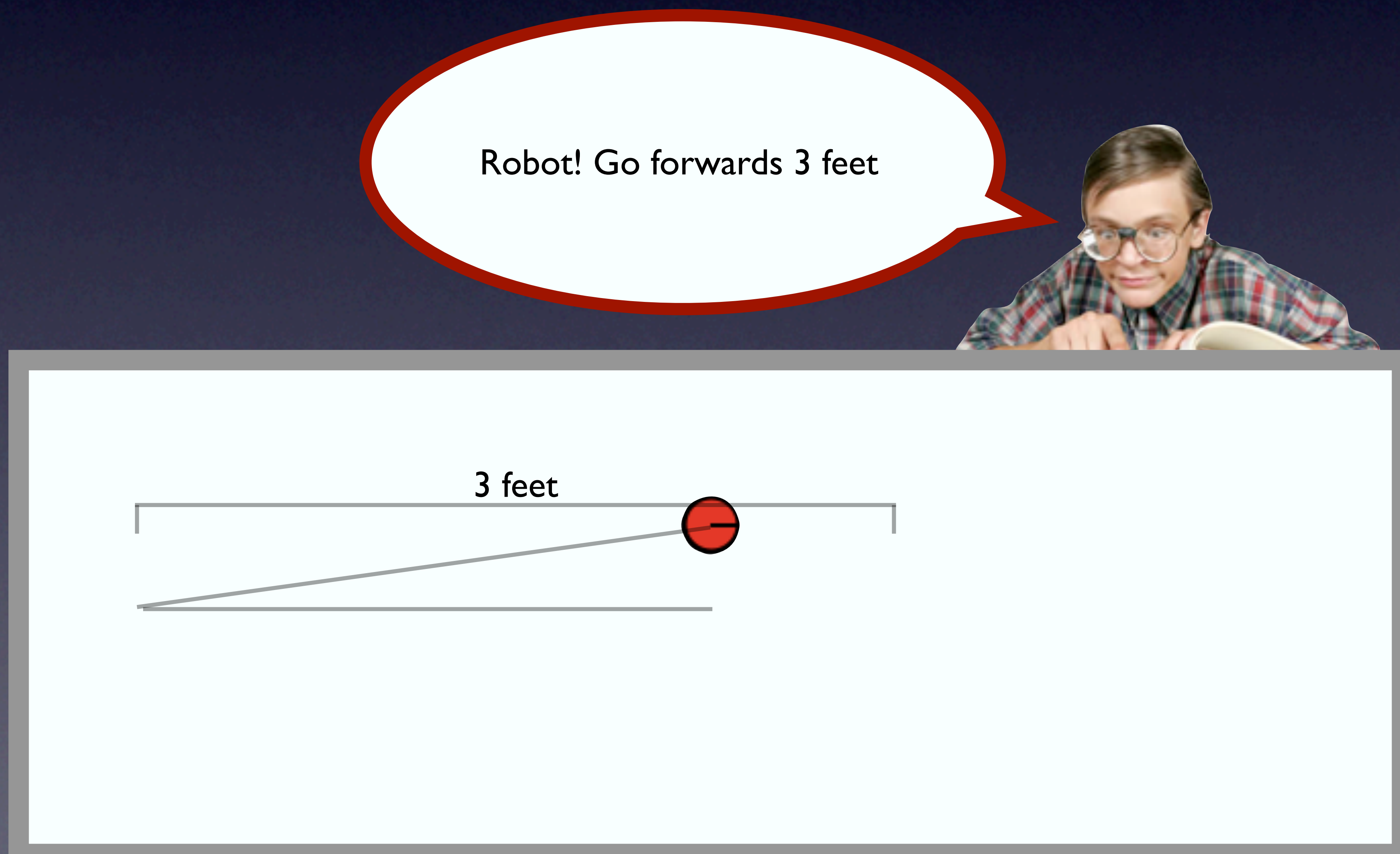



Robot! Turn 90 degrees clockwise

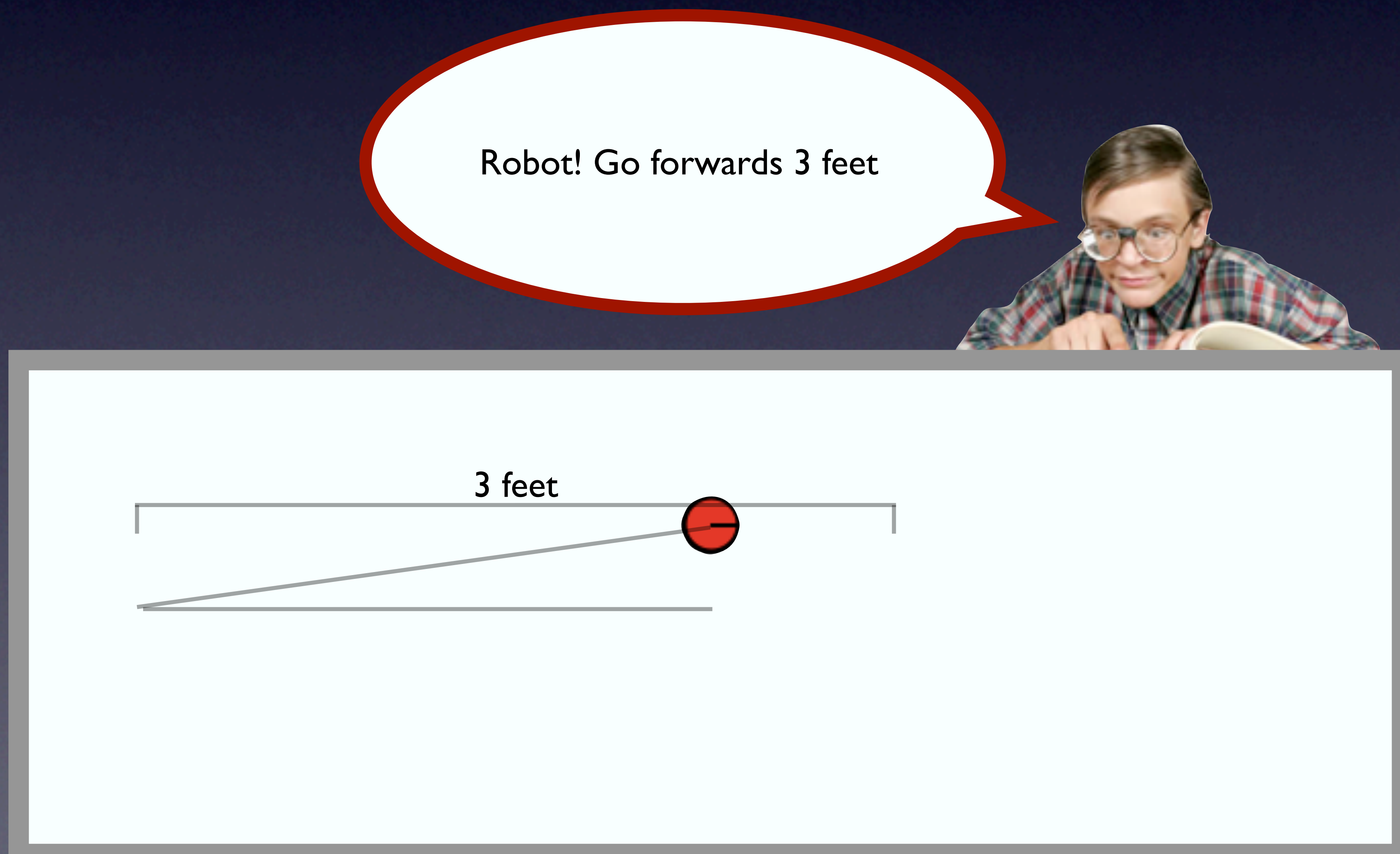

Robot! Turn 90 degrees clockwise


Robot! Go forwards 3 feet


Robot! Go forwards 3 feet

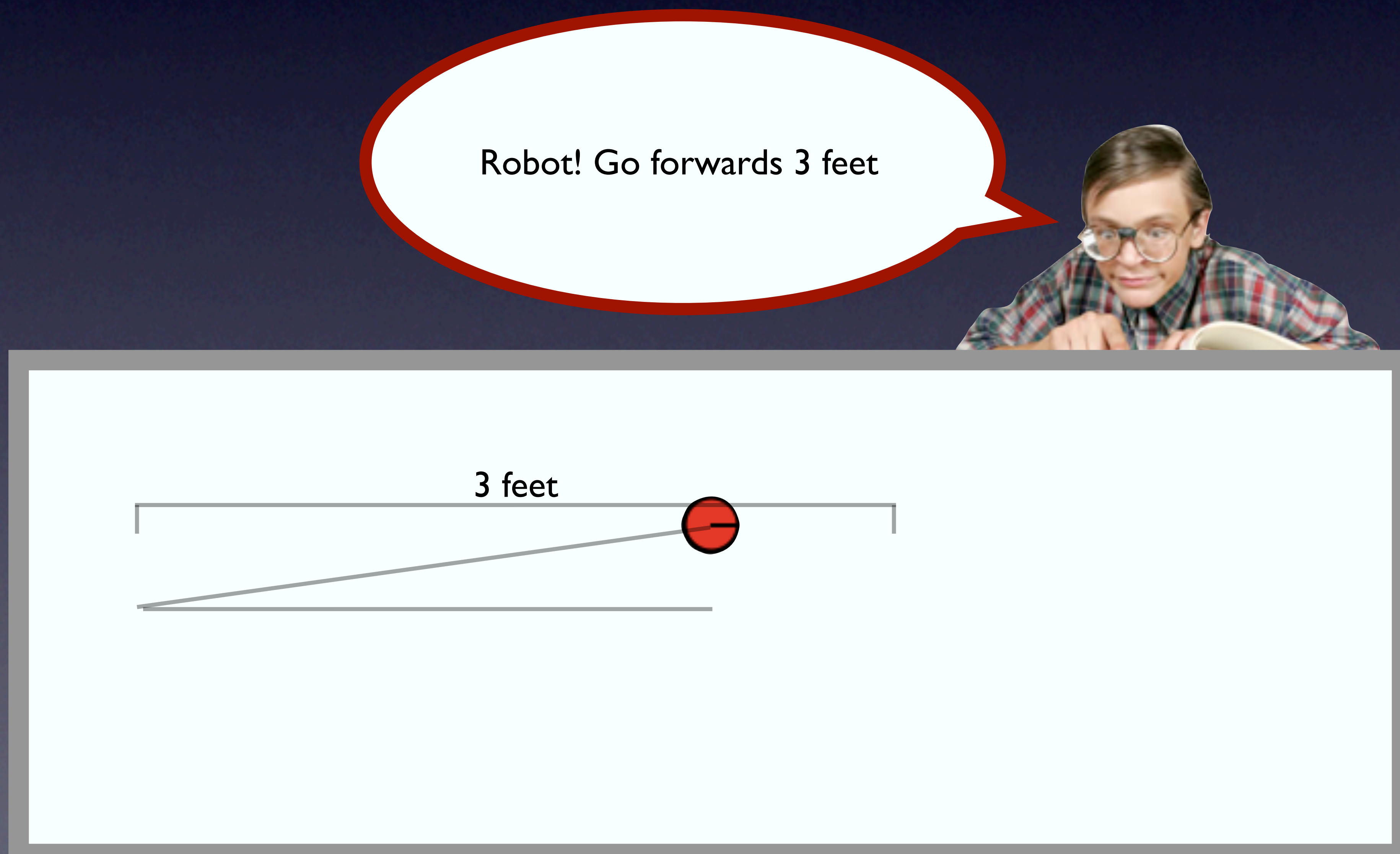

Robot! Go forwards 3 feet
3 feet

Our robot's motion is noisy!



Our robot’s motion is noisy!

Let’s choose a simple model from this noise, and say that each time we try to move in a straight line, our robot goes almost the right direction and distance, but with some Gaussian noise on both the direction and distance.



Θ = Desired Movement Direction
Θ' = Actual Movement Direction
d = Desired Movement Distance
d' = Desired Movement Distance
(x_t, y_t) = Current Robot Position
(x_{t+1}, y_{t+1}) = Simulated Noisy Robot Position
$\mathcal{N}(\mu, \sigma)$ = Normal Random Variable Sample
σ_Θ^2 = Direction Variance
σ_d^2 = Distance Variance

Θ = Desired Movement Direction
Θ' = Actual Movement Direction
d = Desired Movement Distance
d' = Desired Movement Distance
(x_t, y_t) = Current Robot Position
(x_{t+1}, y_{t+1}) = Simulated Noisy Robot Position
$\mathcal{N}(\mu, \sigma)$ = Normal Random Variable Sample
σ_Θ^2 = Direction Variance
σ_d^2 = Distance Variance

$$\Theta' = \Theta + \mathcal{N}(0, \sigma_\Theta)$$

$$d' = d + \mathcal{N}(0, \sigma_d)$$

Θ = Desired Movement Direction
Θ' = Actual Movement Direction
d = Desired Movement Distance
d' = Desired Movement Distance
(x_t, y_t) = Current Robot Position
(x_{t+1}, y_{t+1}) = Simulated Noisy Robot Position
$\mathcal{N}(\mu, \sigma)$ = Normal Random Variable Sample
σ_Θ^2 = Direction Variance
σ_d^2 = Distance Variance

$$\Theta' = \Theta + \mathcal{N}(0, \sigma_\Theta)$$

$$d' = d + \mathcal{N}(0, \sigma_d)$$

$$x_{t+1} = x_t + d' * \cos(\Theta')$$

$$y_{t+1} = y_t + d' * \sin(\Theta')$$

Θ = Desired Movement Direction
Θ' = Actual Movement Direction
d = Desired Movement Distance
d' = Desired Movement Distance
(x_t, y_t) = Current Robot Position
(x_{t+1}, y_{t+1}) = Simulated Noisy Robot Position
$\mathcal{N}(\mu, \sigma)$ = Normal Random Variable Sample
σ_Θ^2 = Direction Variance
σ_d^2 = Distance Variance

$$\Theta' = \Theta + \mathcal{N}(0, \sigma_\Theta)$$
$$d' = d + \mathcal{N}(0, \sigma_d)$$

$$x_{t+1} = x_t + d' * \cos(\Theta')$$
$$y_{t+1} = y_t + d' * \sin(\Theta')$$

This is the most simple way to predict our robots' motion.

There are much more complex and accurate models available, but we will use this one for simplicity.

An Example

Now, one easy way to estimate the 2D position of our robot is to simulate the movement of a whole bunch of robots, each with it's own random Gaussian noise.

An Example

Now, one easy way to estimate the 2D position of our robot is to simulate the movement of a whole bunch of robots, each with it's own random Gaussian noise.

These 'virtual' robots will scatter according to the amount of noise we add.

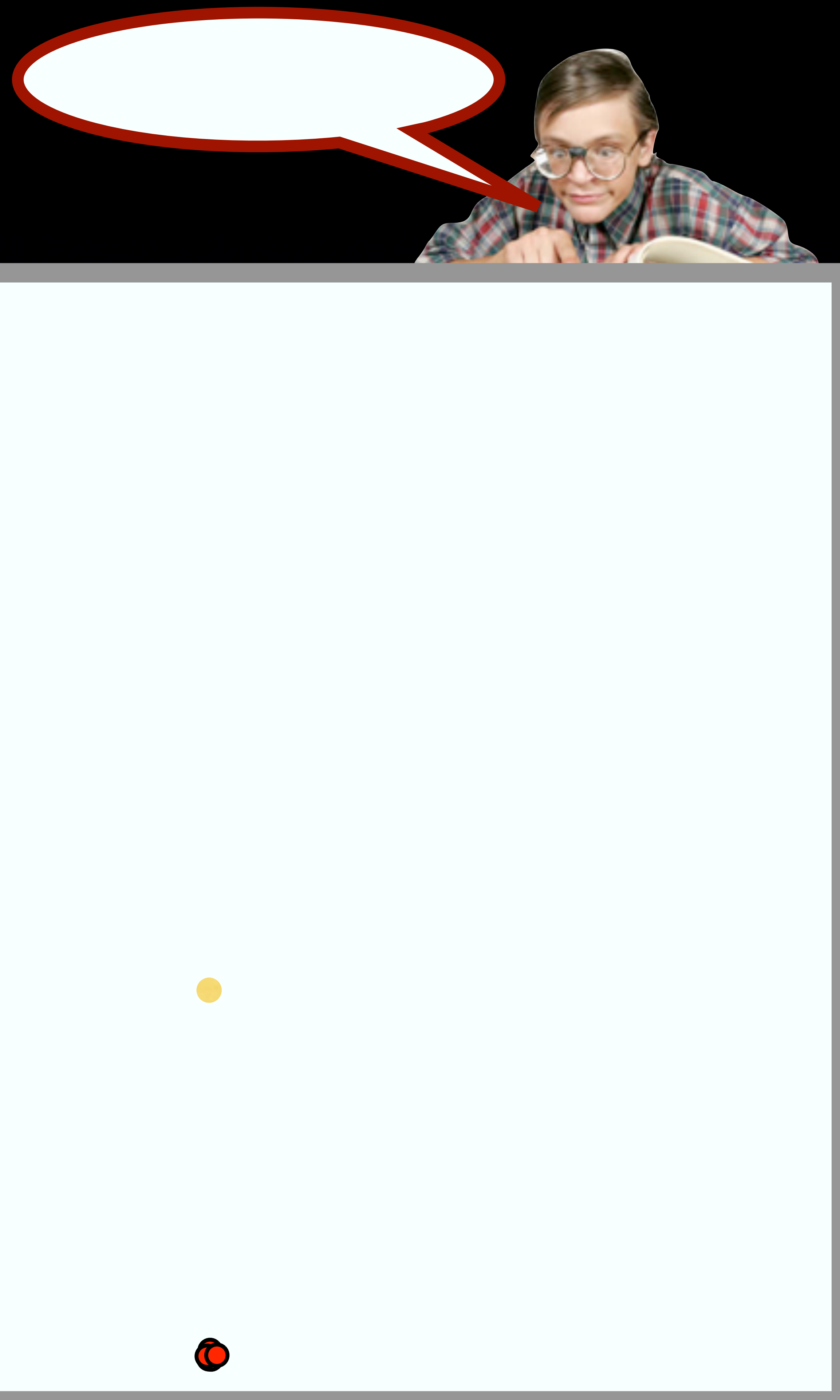
An Example

Now, one easy way to estimate the 2D position of our robot is to simulate the movement of a whole bunch of robots, each with it's own random Gaussian noise.

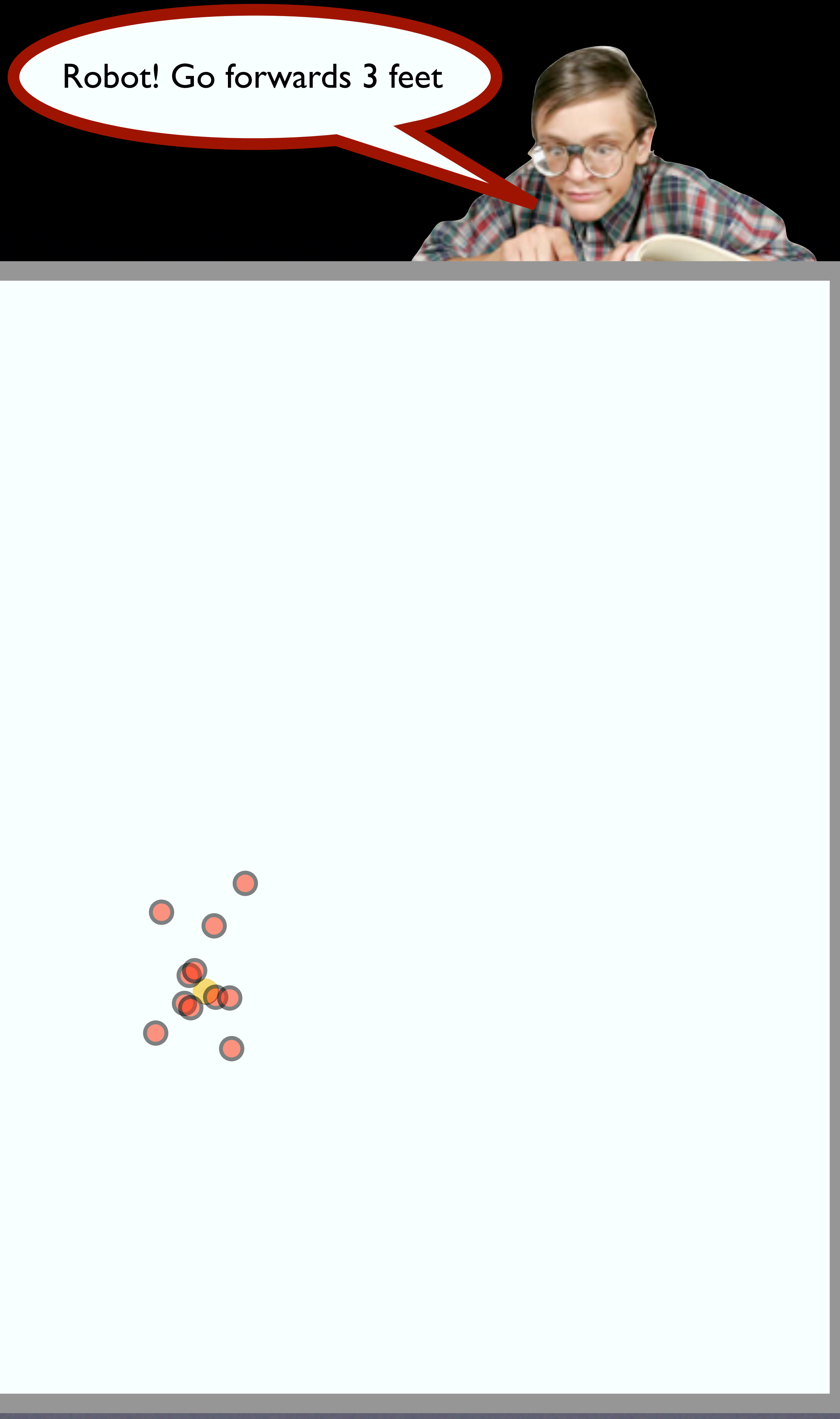
These 'virtual' robots will scatter according to the amount of noise we add.

If our noise model is a good approximation of real life, then the distribution of our virtual robots will describe the probability distribution of our real robot's location.

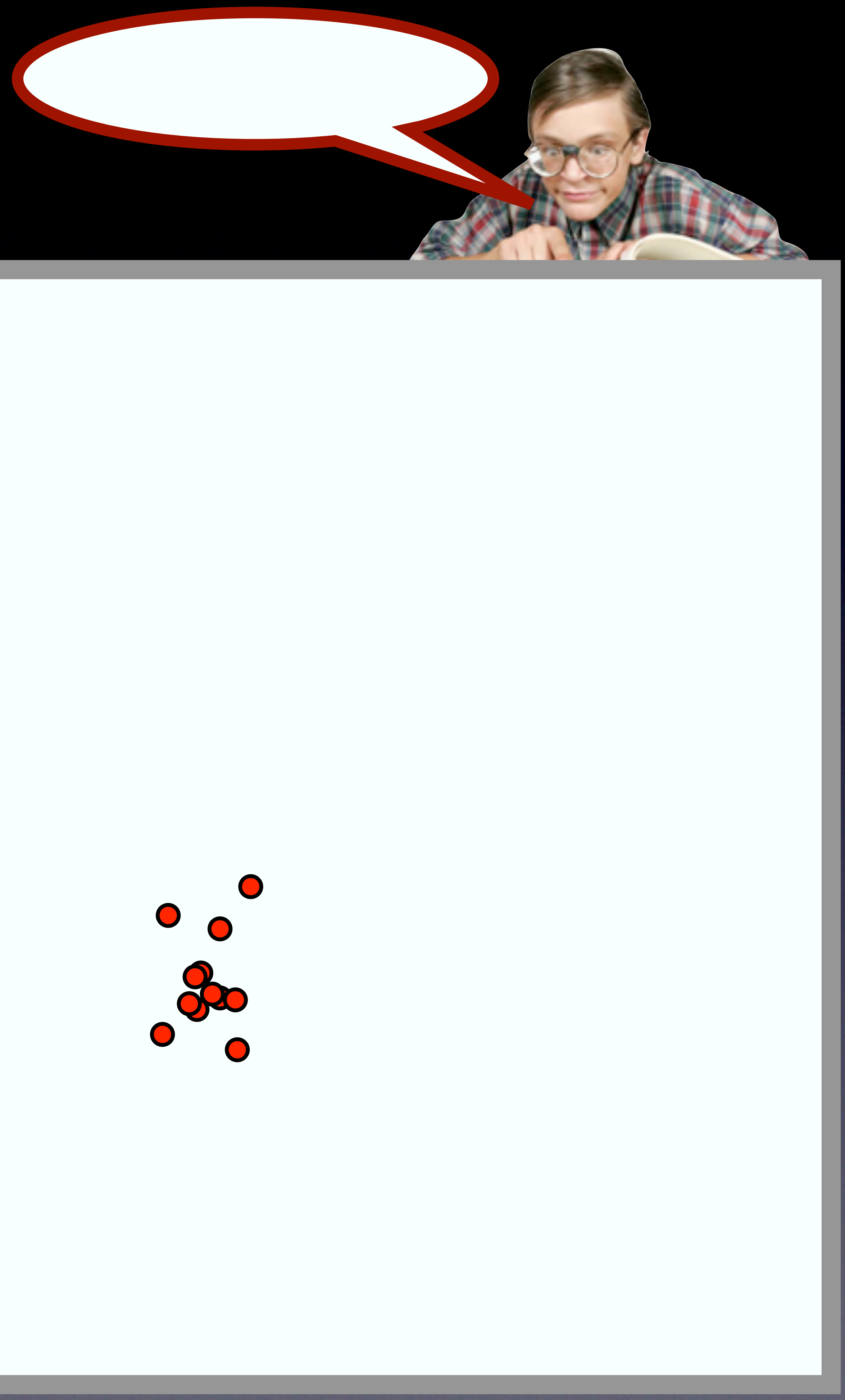
An Example



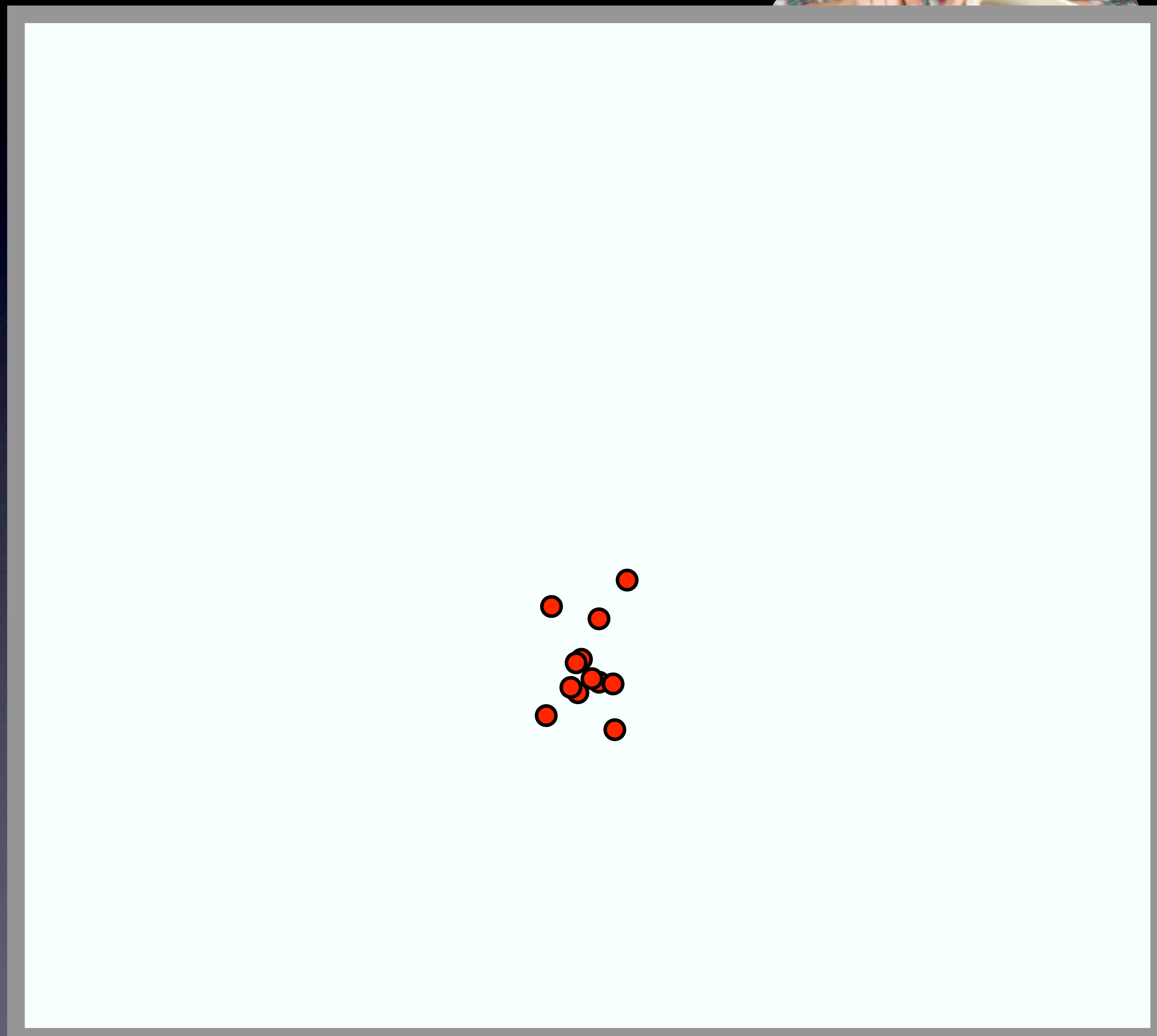
An Example



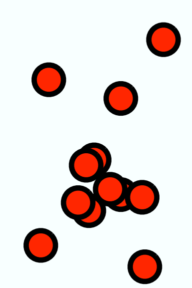
An Example



An Example

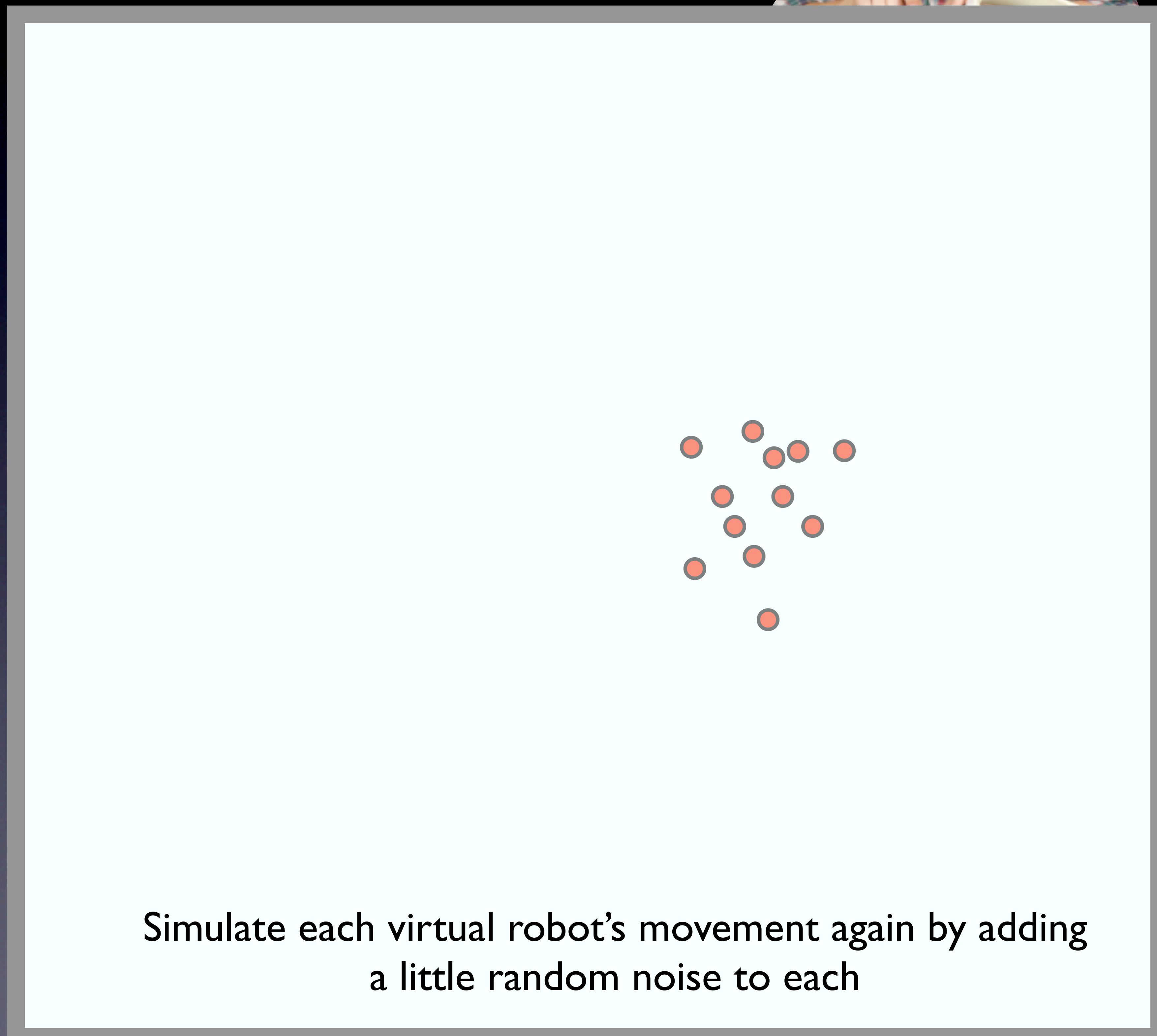


An Example

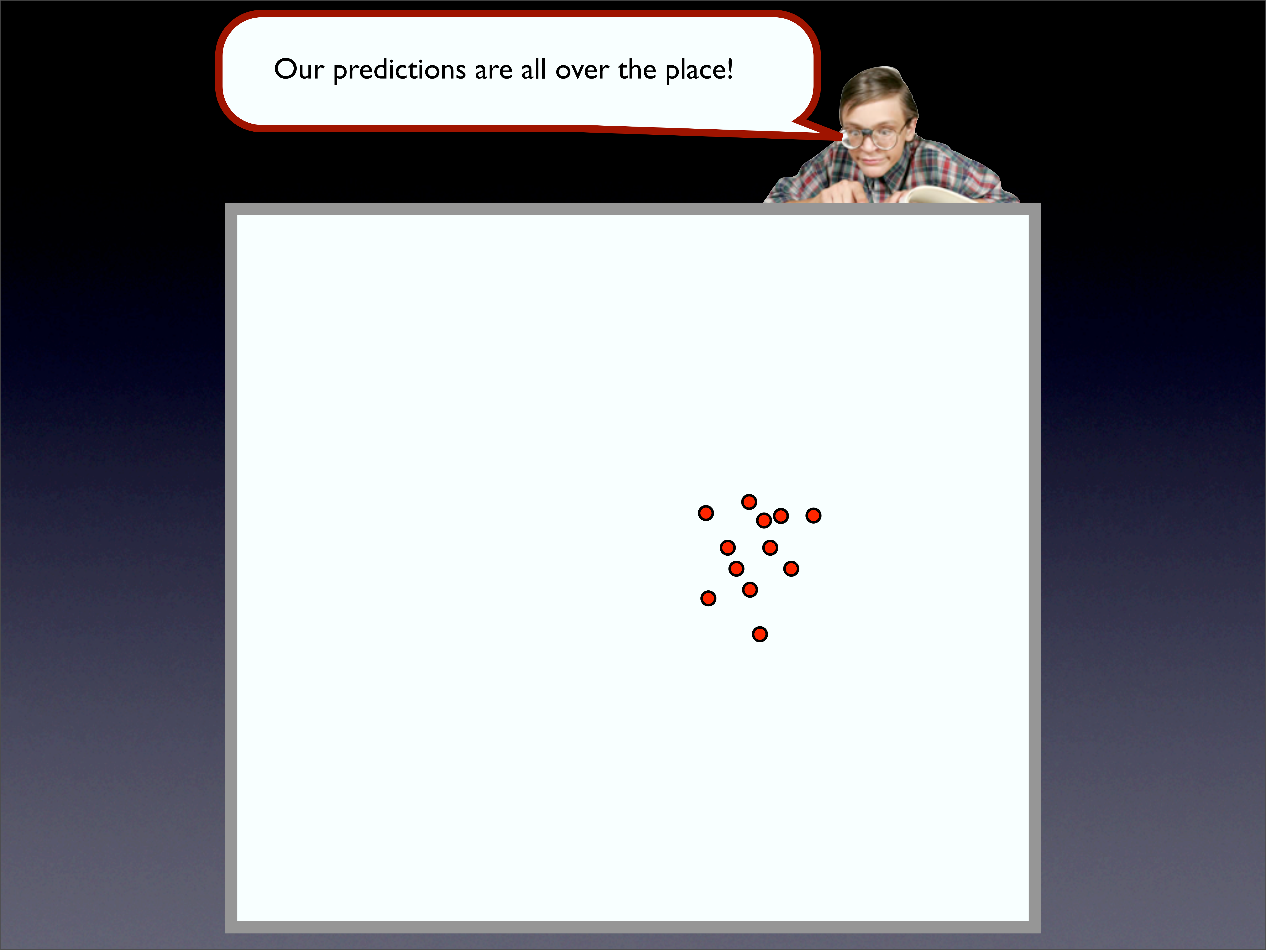


Simulate each virtual robot's movement again by adding a little random noise to each

An Example



Simulate each virtual robot's movement again by adding a little random noise to each


Our predictions are all over the place!

Measurement Models

Obviously, if we keep moving around and adding noise with each step, all of our virtual robots will eventually be completely scattered.

By taking measurements, hopefully we can assess the likelihood of each virtual robot

Measurement Models

So, we would like to assess the probability of our robot actually being at one of our simulated robots' positions given some new sensor reading.

Measurement Models

So, we would like to assess the probability of our robot actually being at one of our simulated robots' positions given some new sensor reading.

P(robot @ location | sensor reading)

Measurement Models

So, we would like to assess the probability of our robot actually being at one of our simulated robots' positions given some new sensor reading.

P(robot @ location | sensor reading)

How do we calculate this?

Measurement Models

So, we would like to assess the probability of our robot actually being at one of our simulated robots' positions given some new sensor reading.

P(virtual robot) = P(robot @ location | sensor reading)

How do we calculate this?

Measurement Models

P(robot @ location | sensor reading)

Measurement Models

P(robot @ location | sensor reading)

Bayes Law!

Measurement Models

P(robot @ location | sensor reading)

Bayes Law! $$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

Measurement Models

P(robot @ location | sensor reading)

Bayes Law!

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

$$P(\text{robot @ location} \mid \text{sensor reading}) = \frac{P(\text{sensor reading} \mid \text{robot @ location})\;P(\text{robot @ location})}{P(\text{sensor reading})}$$

$$P(\text{robot @ location} \mid \text{sensor reading}) = \frac{P\left(\text{sensor reading} \mid \text{robot @ location}\right) P\left(\text{robot @ location}\right)}{P\left(\text{sensor reading}\right)}$$

What does this mean, and how do we calculate it?

$$P(\text{robot @ location} \mid \text{sensor reading}) = \frac{\boxed{P(\text{sensor reading} \mid \text{robot @ location})}\; P(\text{robot @ location})}{P(\text{sensor reading})}$$

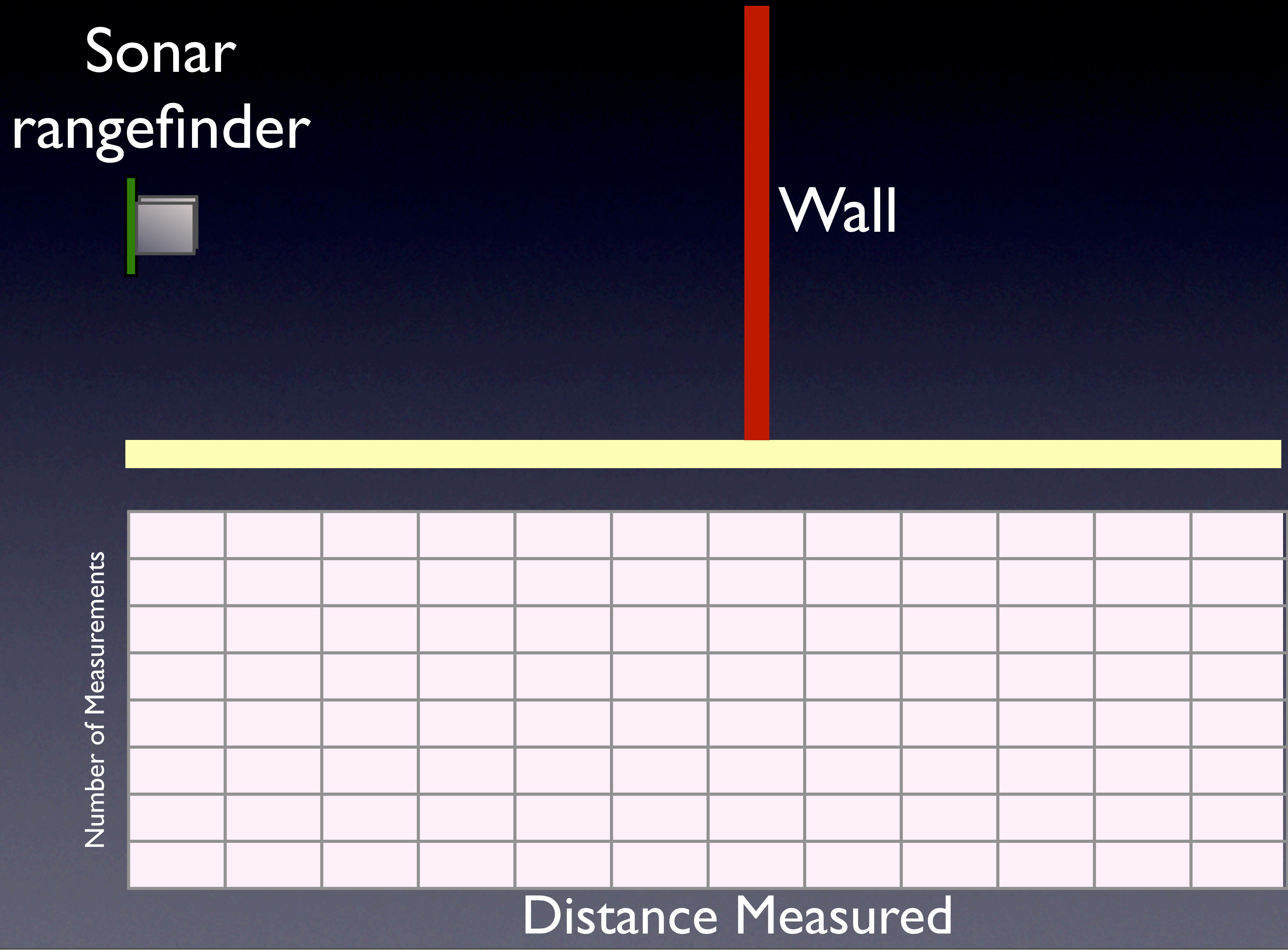
Measurement Models

P(sensor reading | robot @ location)

Just like our motion model, our sensors are subject to noise, and can be modeled as a probability distribution.

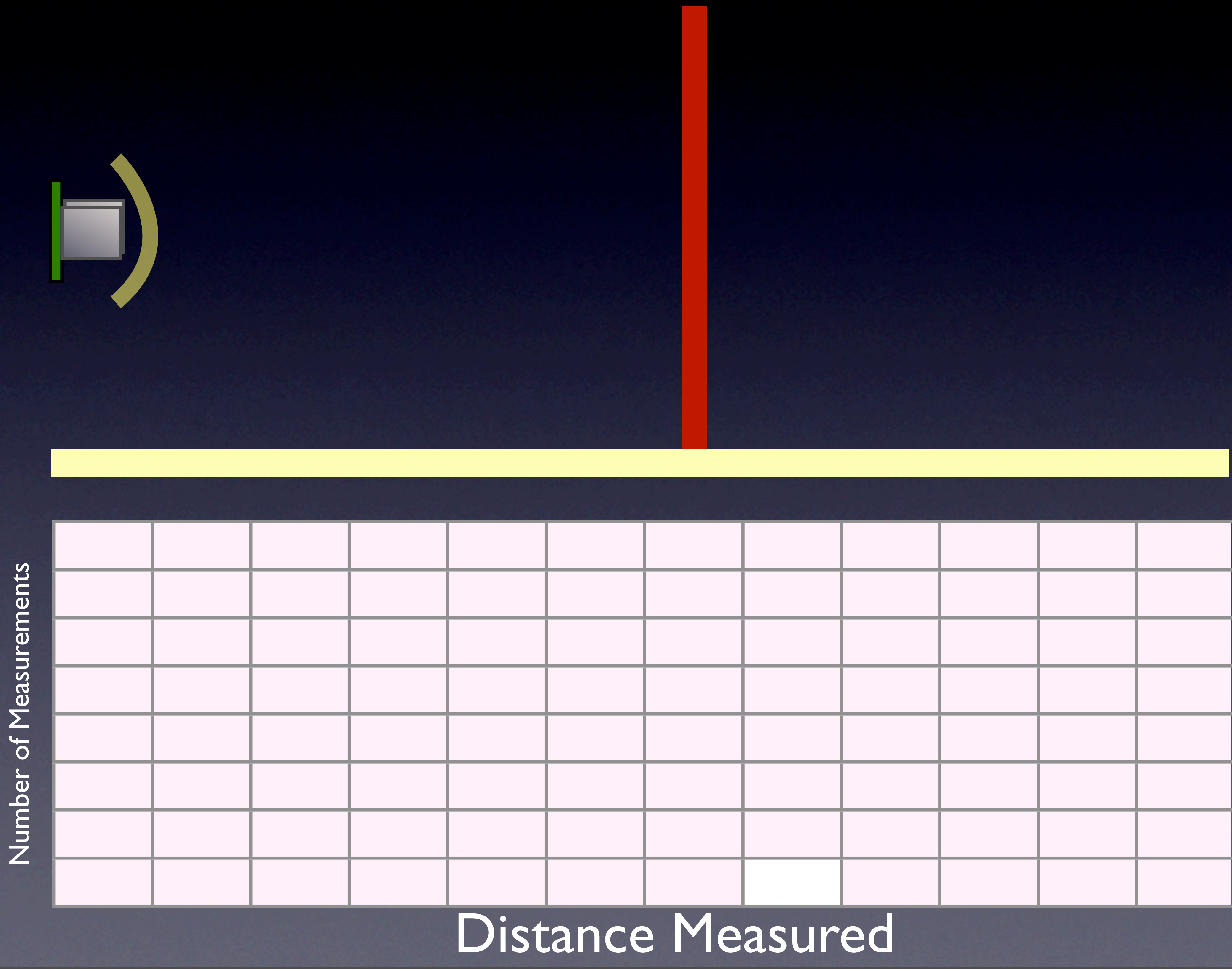
Measurement Models

P(sensor reading | robot @ location)



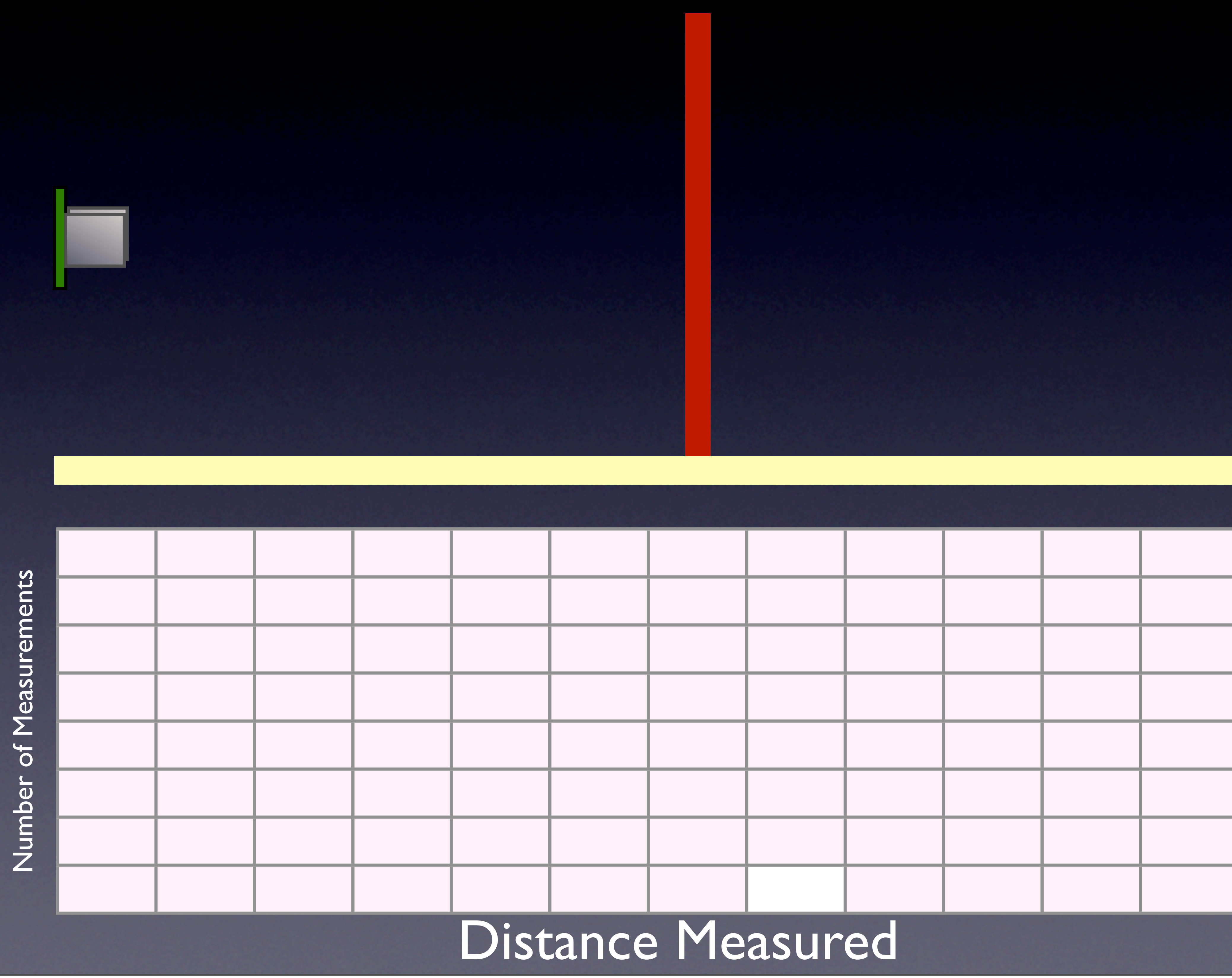
Measurement Models

P(sensor reading | robot @ location)



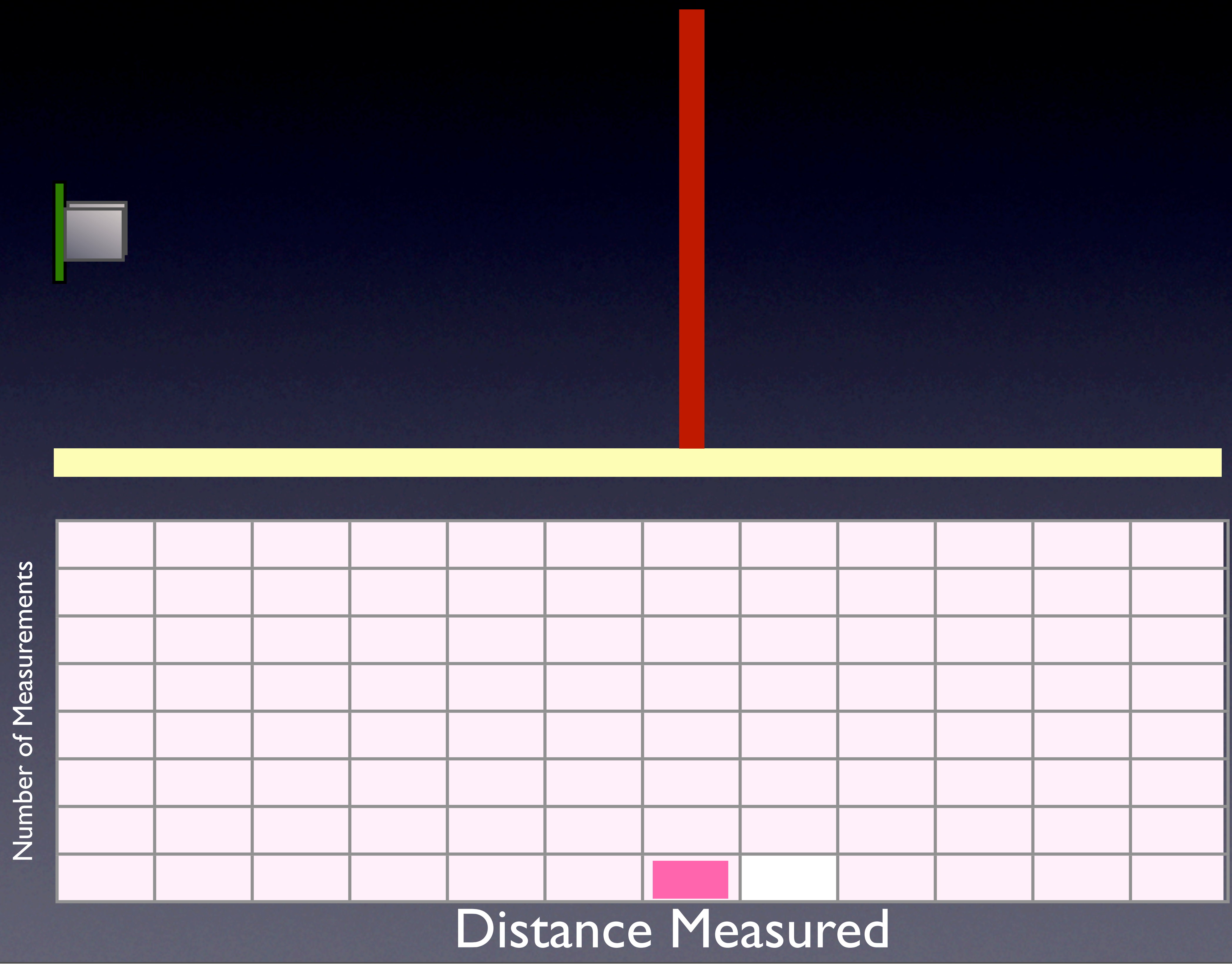
Measurement Models

P(sensor reading | robot @ location)



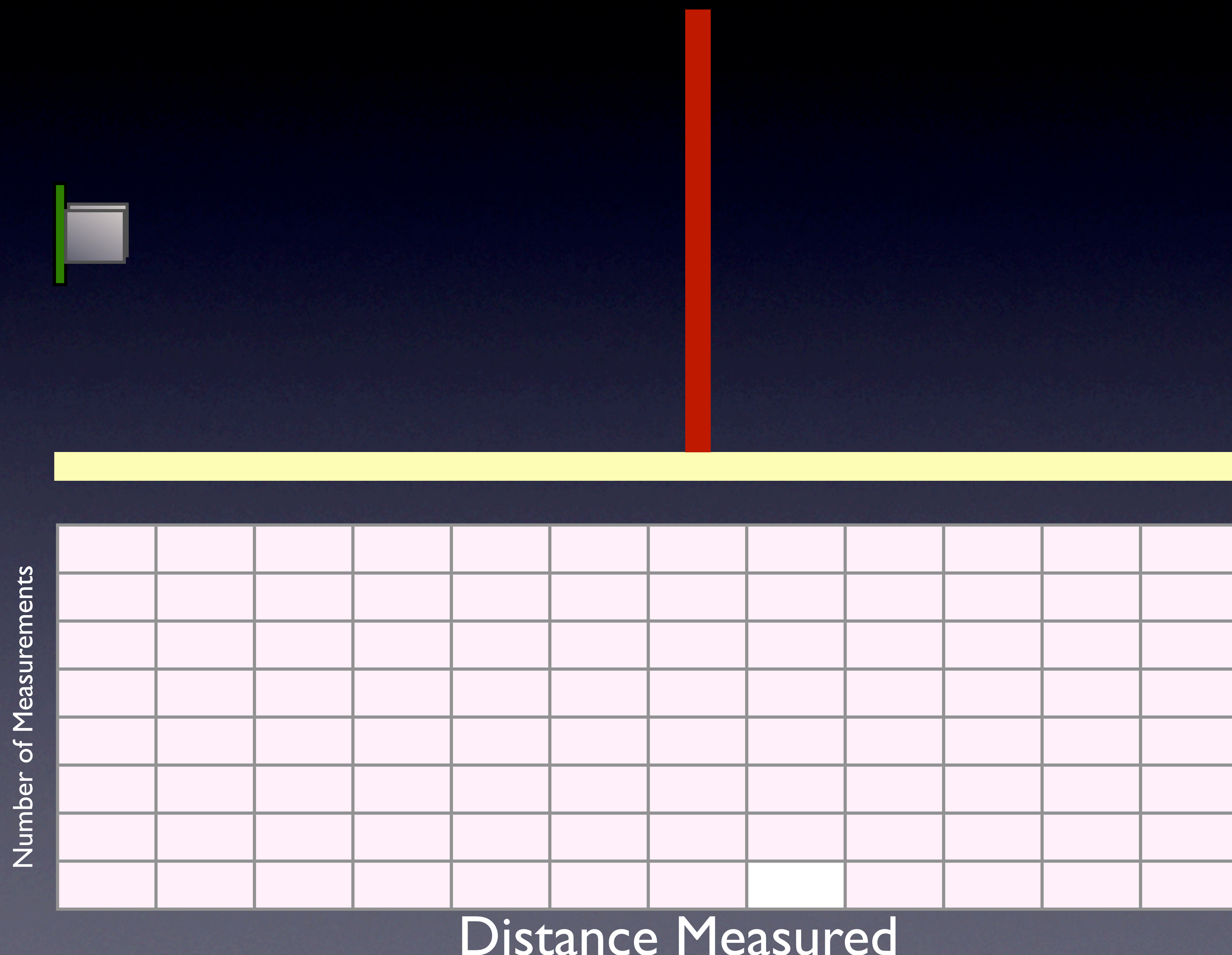
Measurement Models

P(sensor reading | robot @ location)



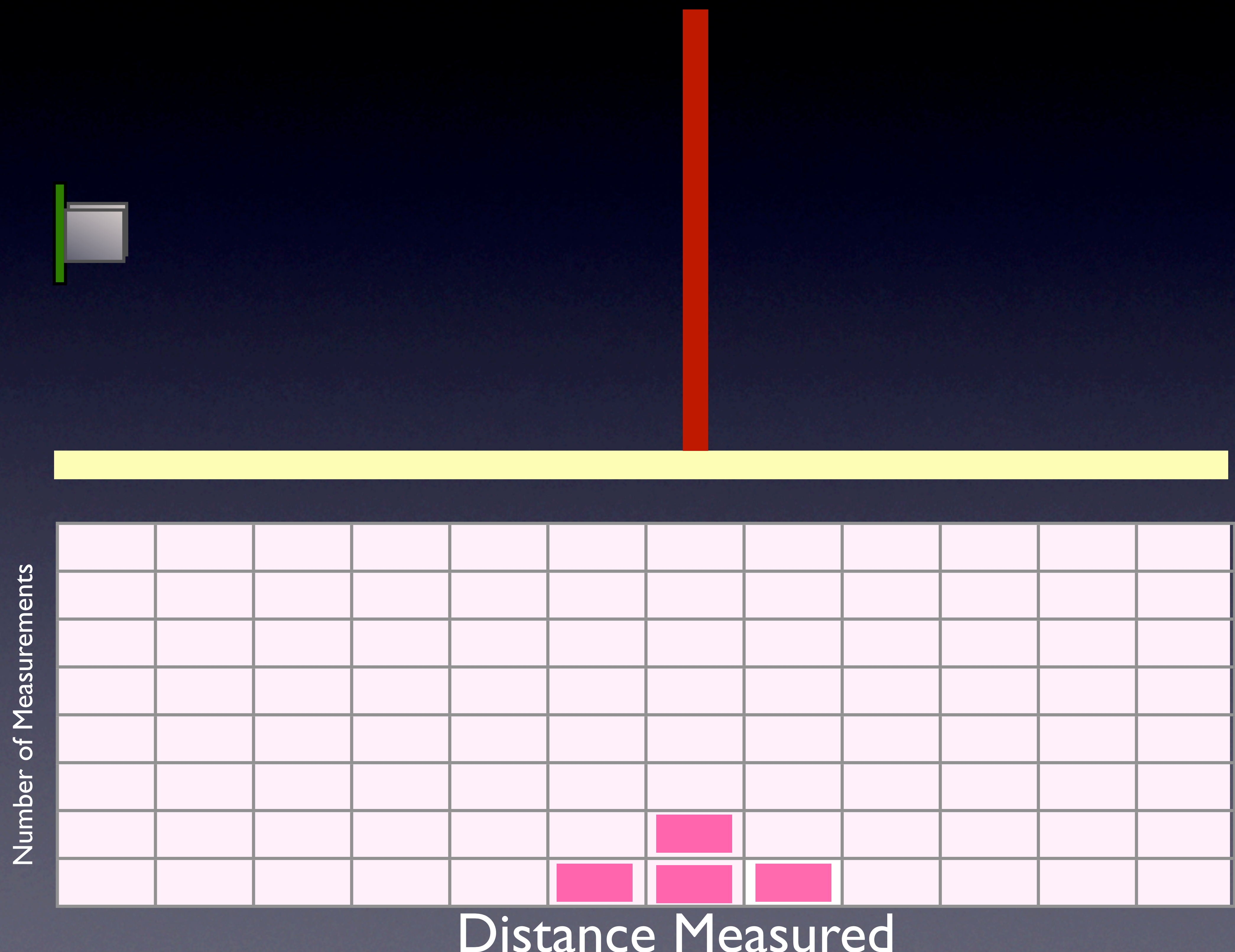
Measurement Models

P(sensor reading | robot @ location)



Measurement Models

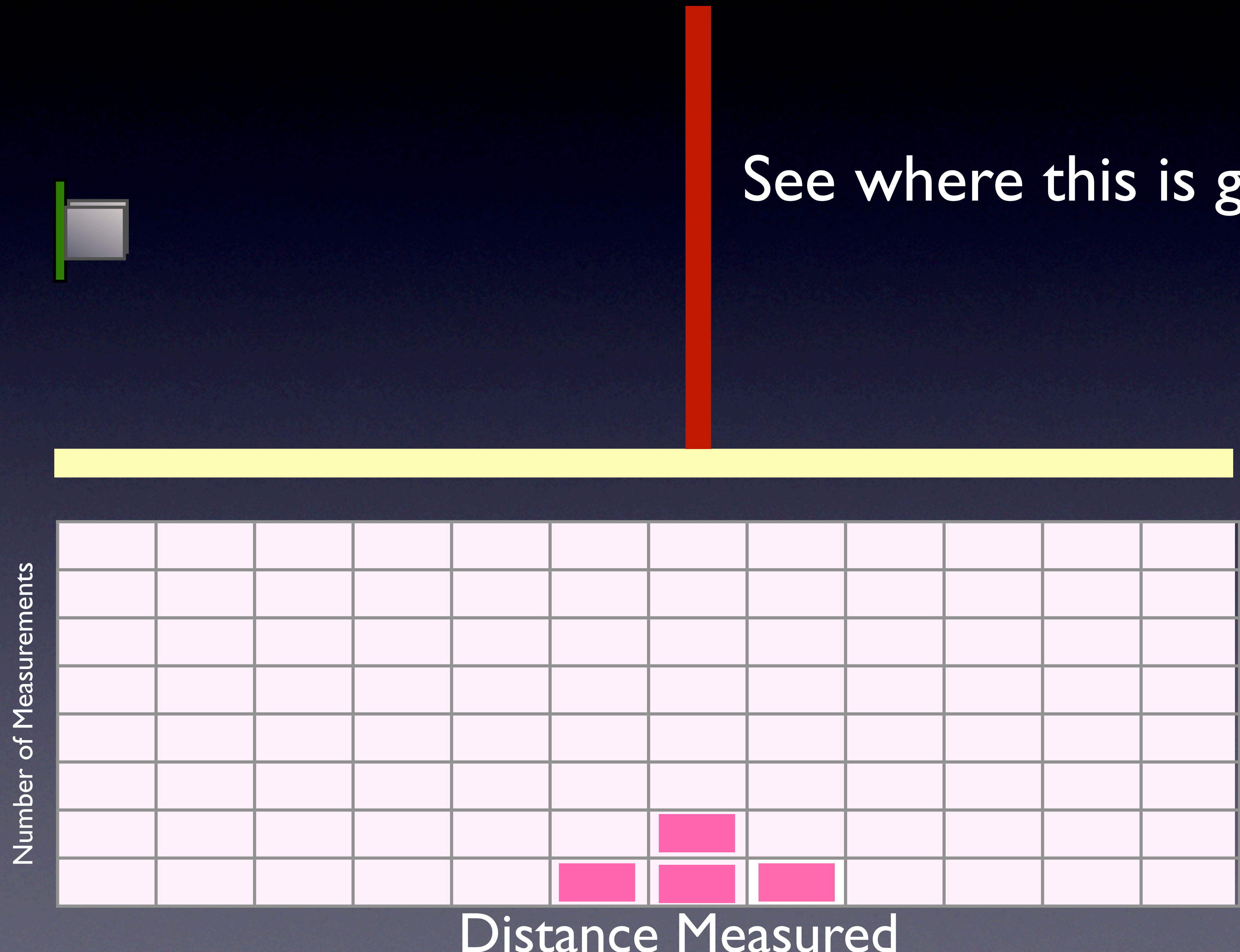
P(sensor reading | robot @ location)



Measurement Models

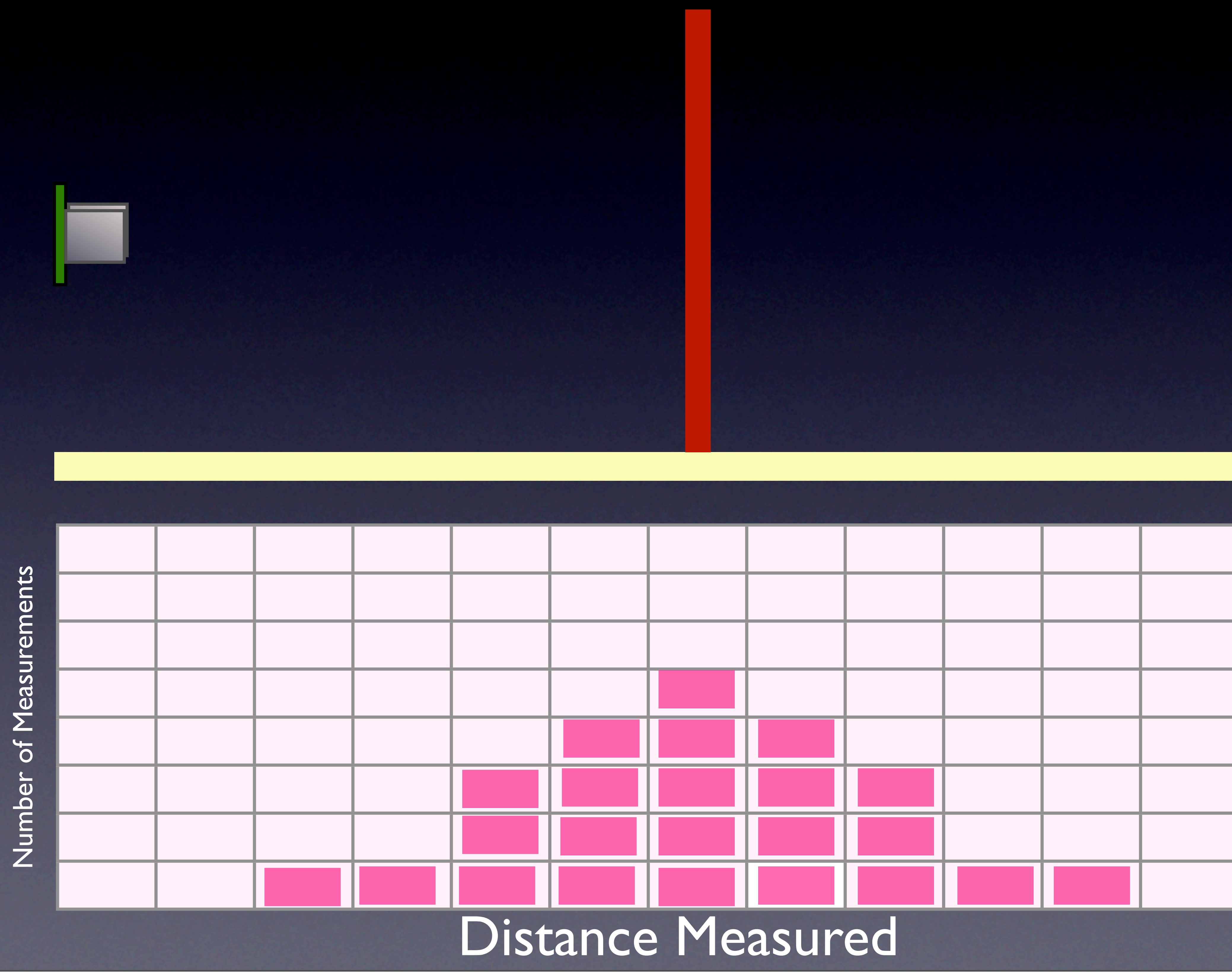
P(sensor reading | robot @ location)

See where this is going?



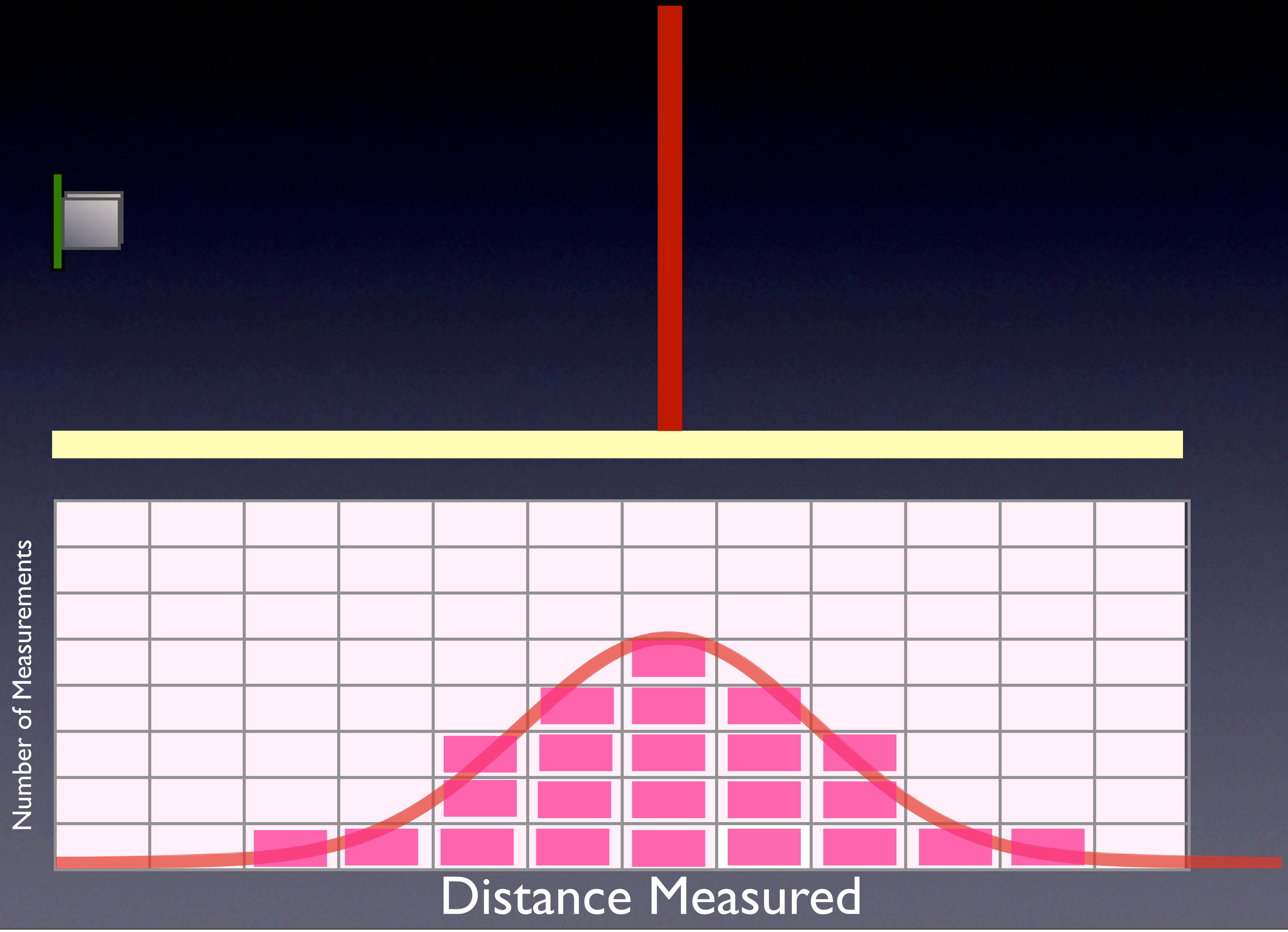
Measurement Models

P(sensor reading | robot @ location)

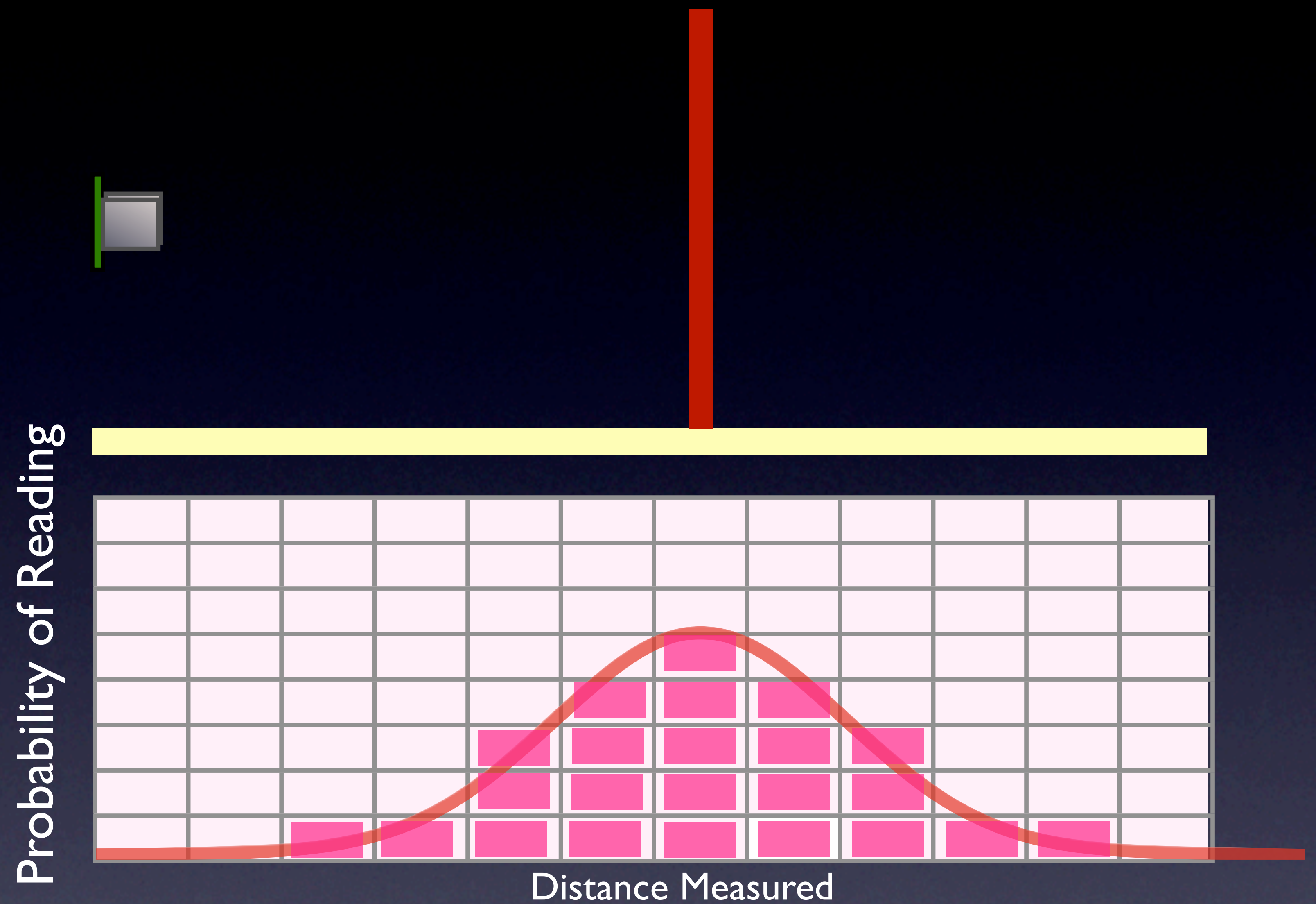


Measurement Models

P(sensor reading | robot @ location)



P(sensor reading | robot @ location)



Now, when we get some new reading, we know the probability of getting that reading given the actual distance to the object.

P(sensor reading | robot @ location) =

Every time we take a reading, we can assess the probability of getting that reading from each of our virtual robot positions

$P(\text{sensor reading} \mid \text{robot @ location}) = \mathcal{N}(\text{reading; location}, \sigma)$

Every time we take a reading, we can assess the probability of getting that reading from each of our virtual robot positions

Reading from sensor

$$P(\text{sensor reading} \mid \text{robot @ location}) = \mathcal{N}(\text{reading; location}, \sigma)$$

Every time we take a reading, we can assess the probability of getting that reading from each of our virtual robot positions

Location of a virtual robot

Reading from sensor

$$P(\text{sensor reading} \mid \text{robot @ location}) = \mathcal{N}(\text{reading}; \text{location}, \sigma)$$

Every time we take a reading, we can assess the probability of getting that reading from each of our virtual robot positions

Sensor Noise

Location of a virtual robot

Reading from sensor

$$P(\text{sensor reading} \mid \text{robot @ location}) = \mathcal{N}(\text{reading}; \text{location}, \sigma)$$

Every time we take a reading, we can assess the probability of getting that reading from each of our virtual robot positions

Sensor Noise

Location of a virtual robot

Reading from sensor

$$P(\text{sensor reading} \mid \text{robot @ location}) = \mathcal{N}(\text{reading; location}, \sigma)$$

Every time we take a reading, we can assess the probability of getting that reading from each of our virtual robot positions

$$P(\text{robot @ location} \mid \text{sensor reading}) = \frac{P(\text{sensor reading} \mid \text{robot @ location})\ P(\text{robot @ location})}{P(\text{sensor reading})}$$

$$P(\text{robot @ location} \mid \text{sensor reading}) = \frac{P(\text{sensor reading} \mid \text{robot @ location})\, P(\text{robot @ location})}{P(\text{sensor reading})}$$

If at each timestep, we calculate the probability of each virtual robot, then we can use those probabilities from the last timestep here

$$P(\text{robot @ location} \mid \text{sensor reading}) = \frac{P(\text{sensor reading} \mid \text{robot @ location})\; P(\text{robot @ location})}{P(\text{sensor reading})}$$

$$P(\text{robot @ location} \mid \text{sensor reading}) = \frac{P(\text{sensor reading} \mid \text{robot @ location})\ P(\text{robot @ location})}{P(\text{sensor reading})}$$

$$P(\text{robot @ location} \mid \text{sensor reading}) = \frac{P(\text{sensor reading} \mid \text{robot @ location})\ P(\text{robot @ location})}{P(\text{sensor reading})}$$

Now, all we have left is P(sensor reading)

$$P(\text{robot @ location} \mid \text{sensor reading}) = \frac{P(\text{sensor reading} \mid \text{robot @ location})\; P(\text{robot @ location})}{P(\text{sensor reading})}$$

Now, all we have left is P(sensor reading)

This value is a bit less understandable, so for the sake of simplicity let's just make this easy...

P(robot @ location x | sensor reading) = N * P(sensor reading | robot @ location) P(robot @ location)

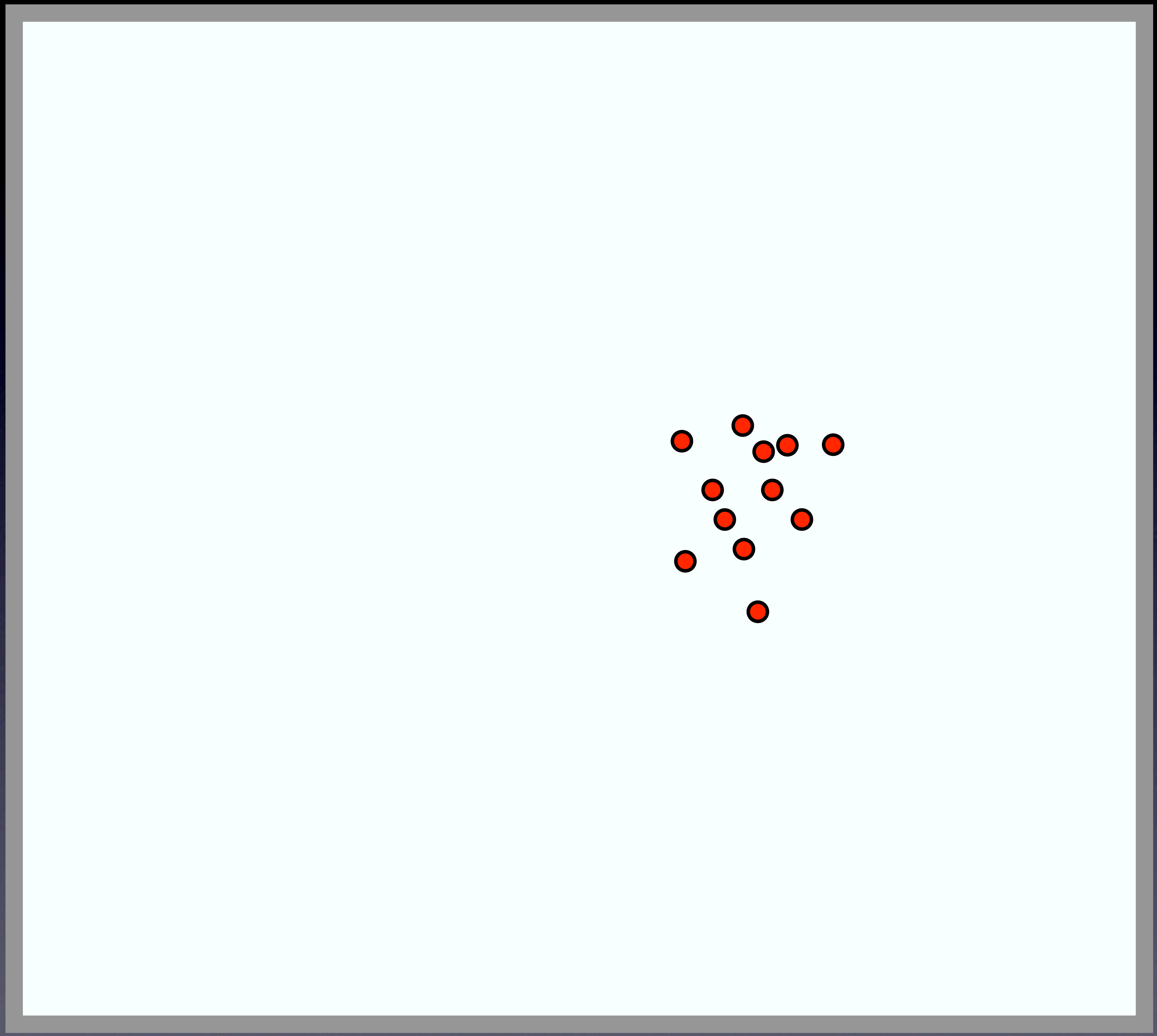
Each time we take a reading, and update the probability of each virtual robot, let's choose N such that all of our probabilities sum to 1.

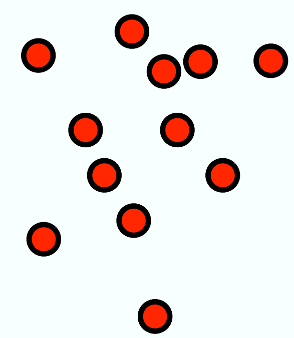
$$N = \frac{1}{\sum_x P(\text{robot @ location } x \mid \text{sensor reading})}$$

We now know everything needed to calculate

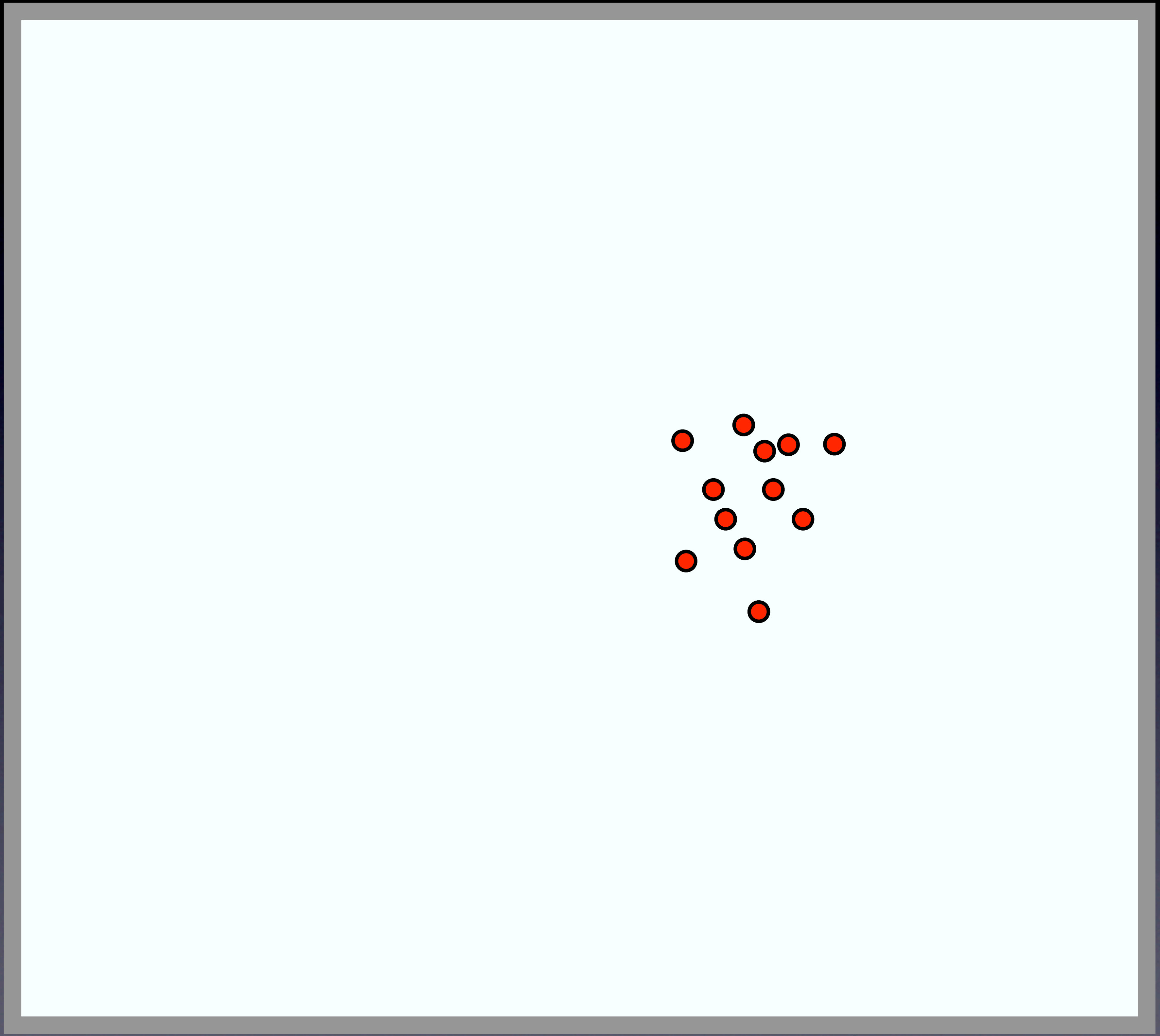
P(robot @ location x | sensor reading)

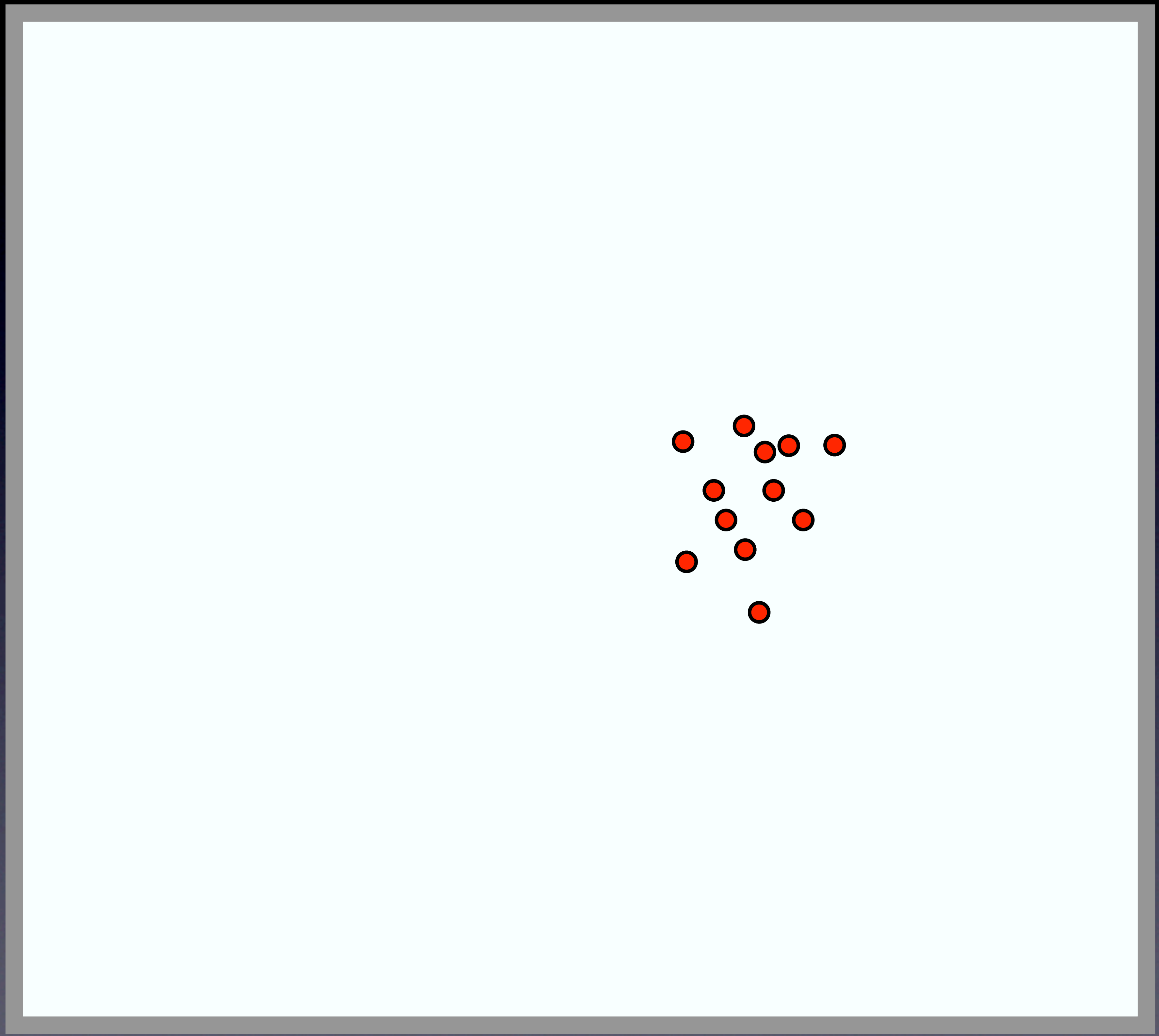
We call the result of this calculation the 'Posterior Probability'

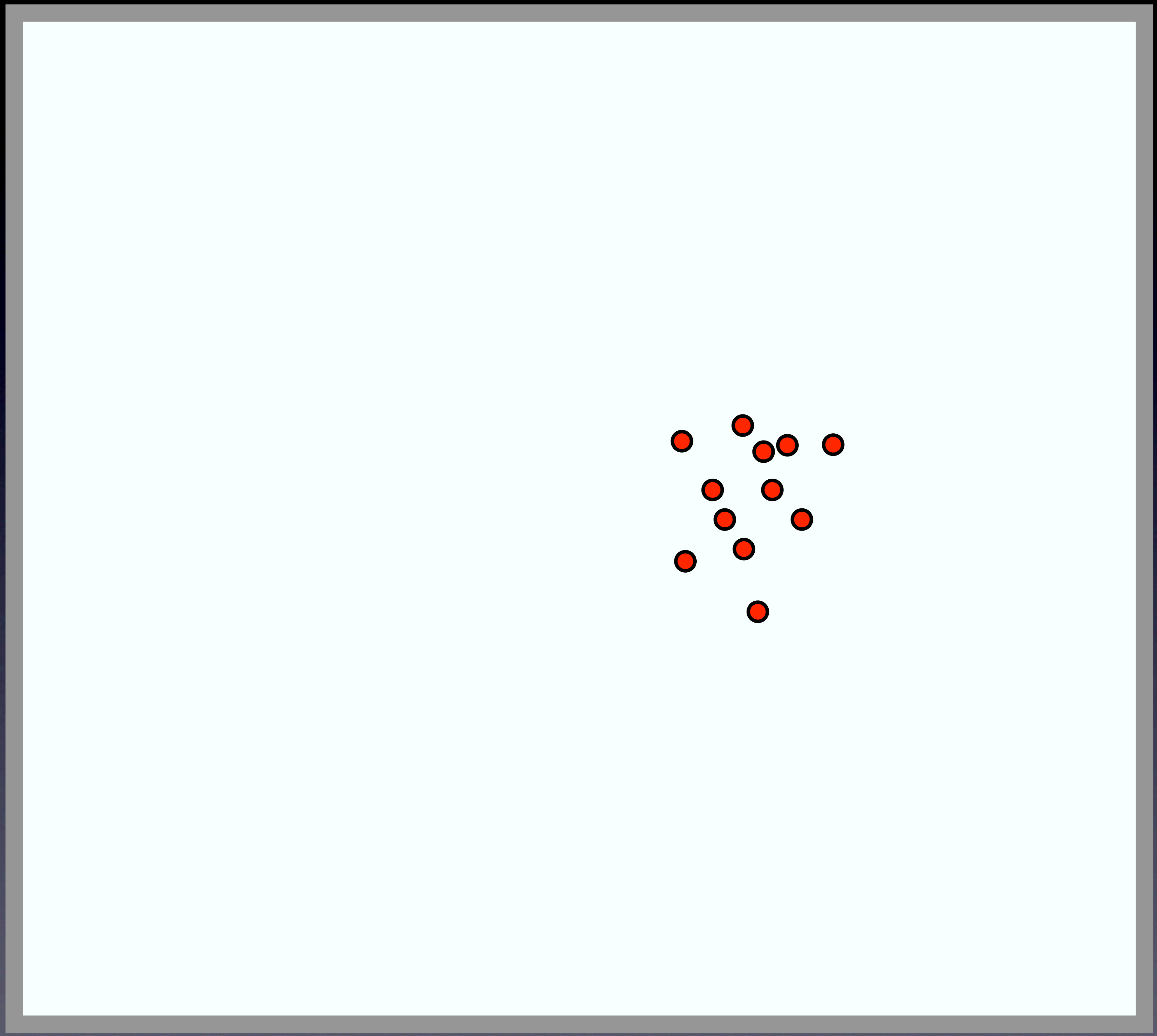


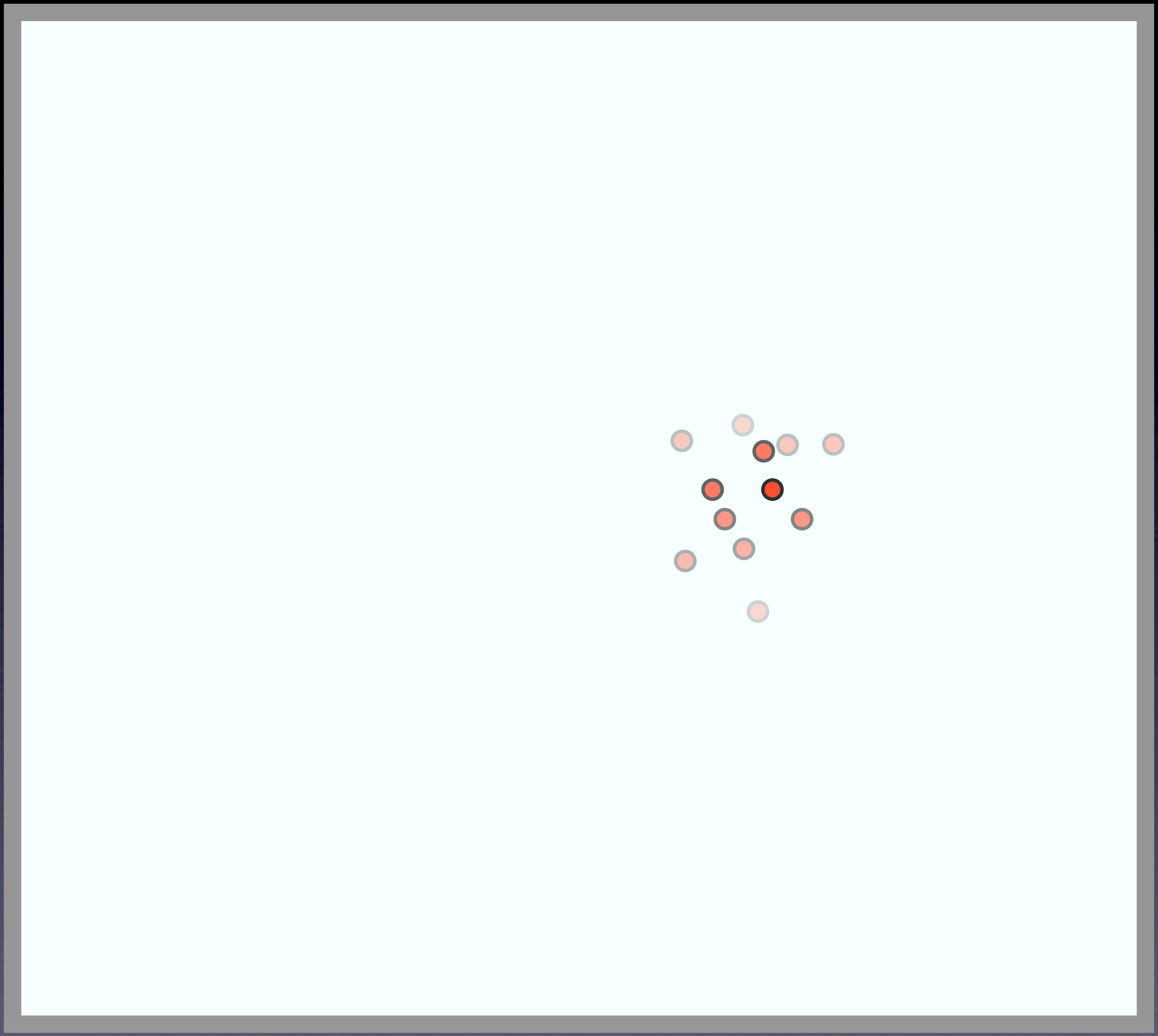


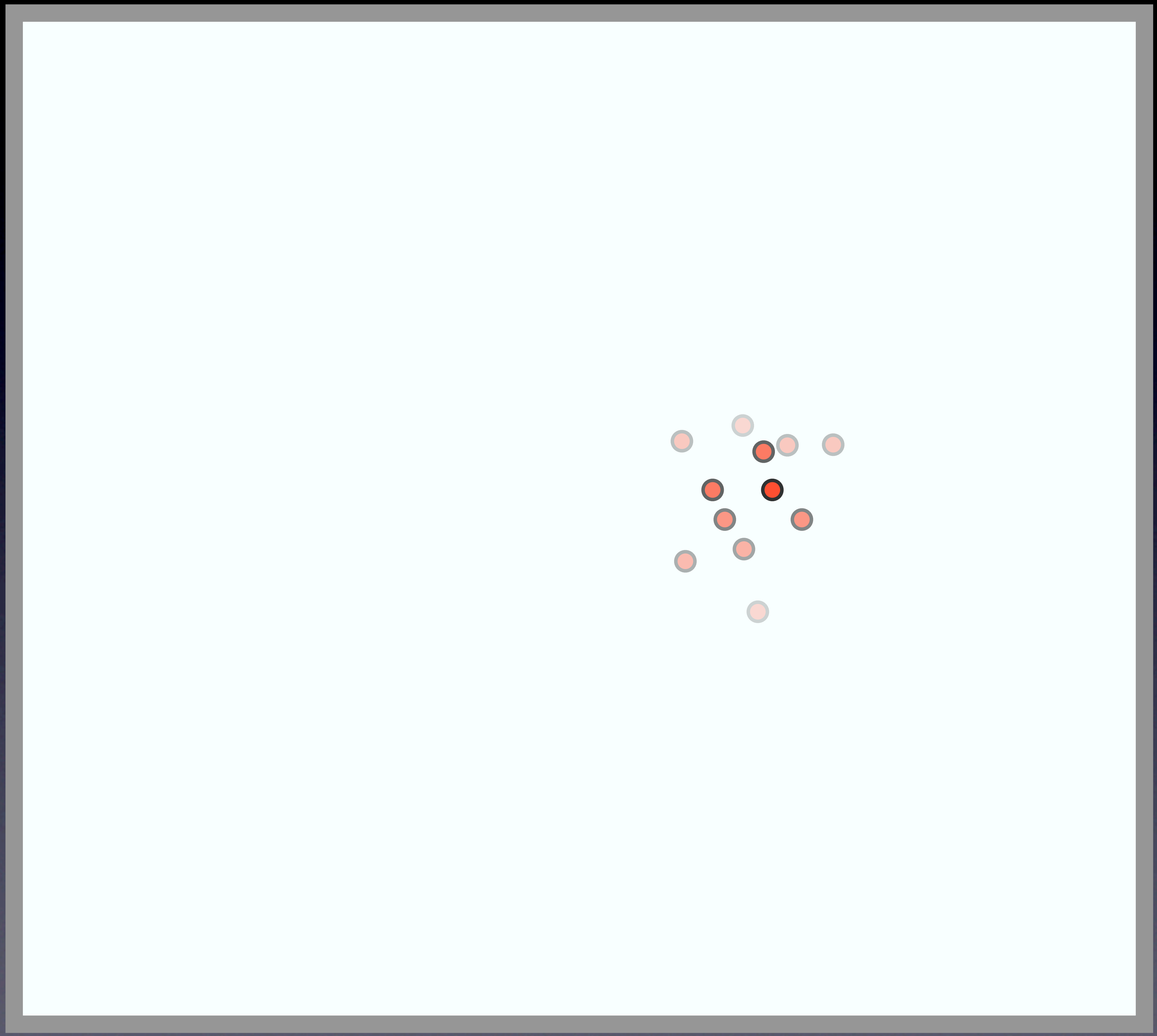
Take a single reading, and compute the probability of each virtual robot given that reading

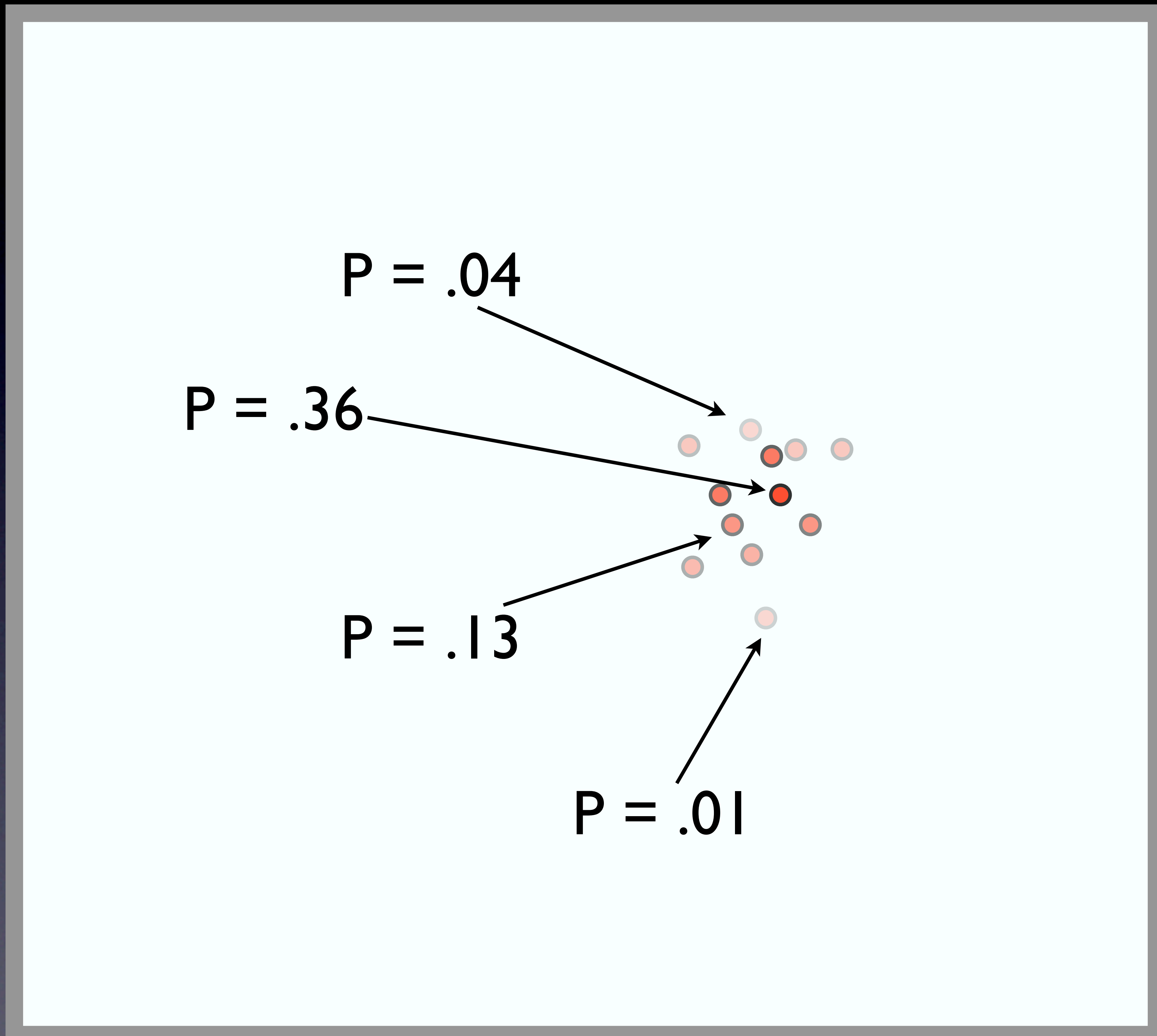











P = .04
P = .36
P = .13
P = .01

Resampling

Now that we can assess the probability of each of our virtual robots' positions given a new sensor measurement, we would like to kill off of some of the virtual robots with lower probabilities.

There are quite a few ways to do this, but I will provide you with an efficient method which you can use.

```cpp
//Use a roulette wheel to probabilistically duplicate particles with high weights,
//and discard those with low weights. A 'Particle' is some structure that has
//a weight element w. The sum of all w's in oldParticles should equal 1.
std::vector<Particle> resampleParticles(std::vector<Particle> oldParticles)
{
  std::vector<Particle> newParticles;

  //Calculate a Cumulative Distribution Function for our particle weights
  std::vector<float> CDF;
  CDF.resize(oldParticles.size());
  CDF.at(0) = oldParticles.at(0).w;
  for(uint i=1; i<CDF.size(); i++)
    CDF.at(i) = CDF.at(i-1) + oldParticles.at(i).w;

  //Loop through our particles as if spinning a roulette wheel.
  //The random u that we start with determines the initial offset
  //and each particle will have a probability of getting landed on and
  //saved proportional to it's posterior probability. If a particle has a very large
  //posterior, then it may get landed on many times. If a particle has a very low
  //posterior, then it may not get landed on at all. By incrementing by
  // 1/(numParticles) we ensure that we don't change the number of particles in our
  // returned set.
  uint i = 0;
  float u = randomDouble()* 1.0/float(oldParticles.size());
  for(uint j=0; j < oldParticles.size(); j++)
  {
    while(u > CDF.at(i))
      i++;

    Particle p = oldParticles.at(i);
    p.w     = 1.0/float(oldParticles.size());

    newParticles.push_back(p);

    u += 1.0/float(oldParticles.size());
  }

  return newParticles;
}
```

Particle Filter

Notice that I referred to our ‘virtual robots’ as ‘particles.’

The algorithm we have put together in this lecture is called a ‘Particle Filter’

Putting it all together

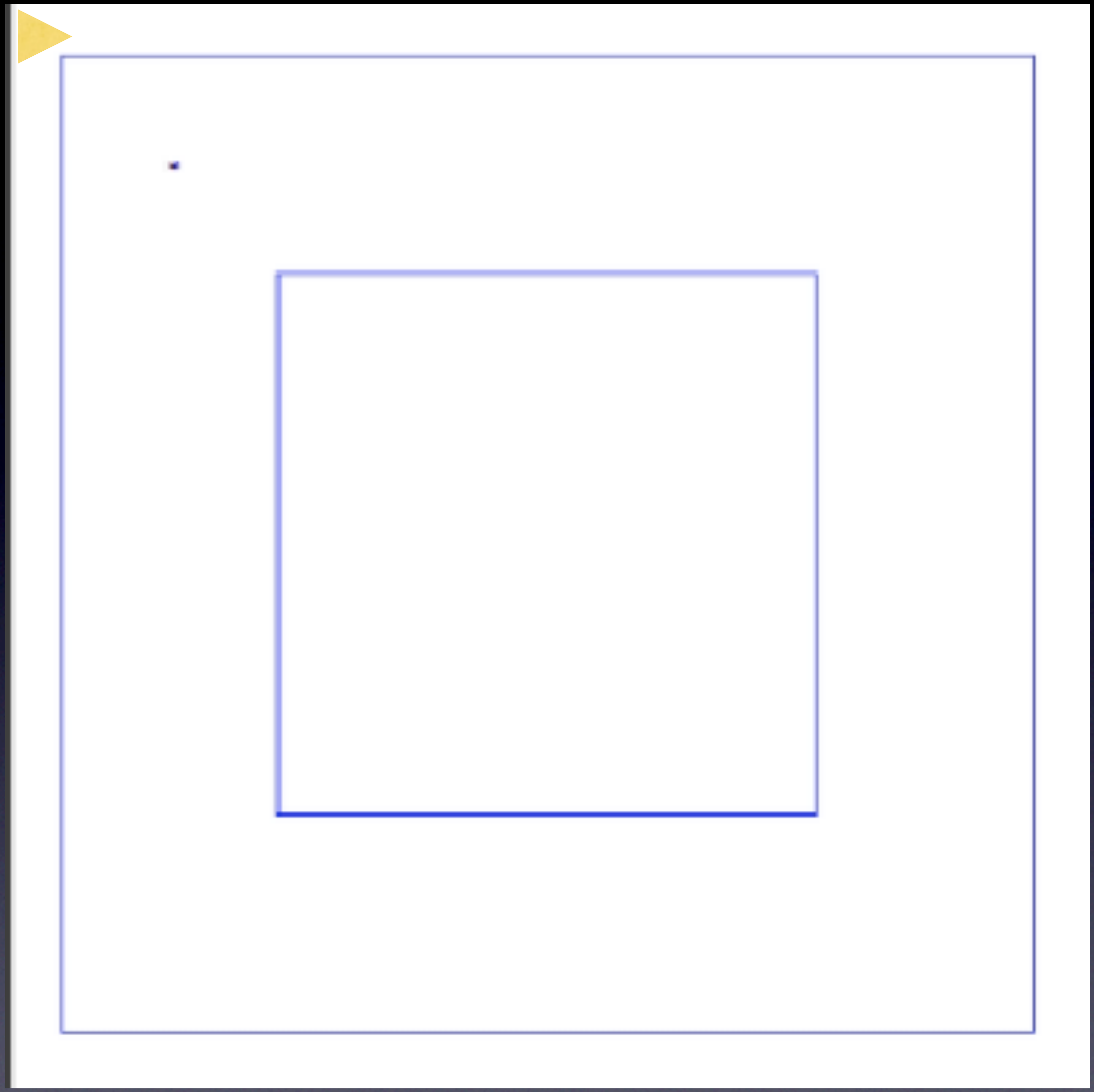
Particle Filtering Algorithm:

Create N particles at some starting location (or distributed randomly around the map), each with equal probability. Call this data structure 'Particles'.

When a new movement command (d, Θ) is issued:
- For each particle 'p' in 'Particles':
  - Generate a randomized movement command consisting of d' and and Θ'
  - Update the current position of 'p' according to the motion model applied to d' and Θ'
  - Optionally do bounds checking to ensure that our particles are not ghosting through walls.

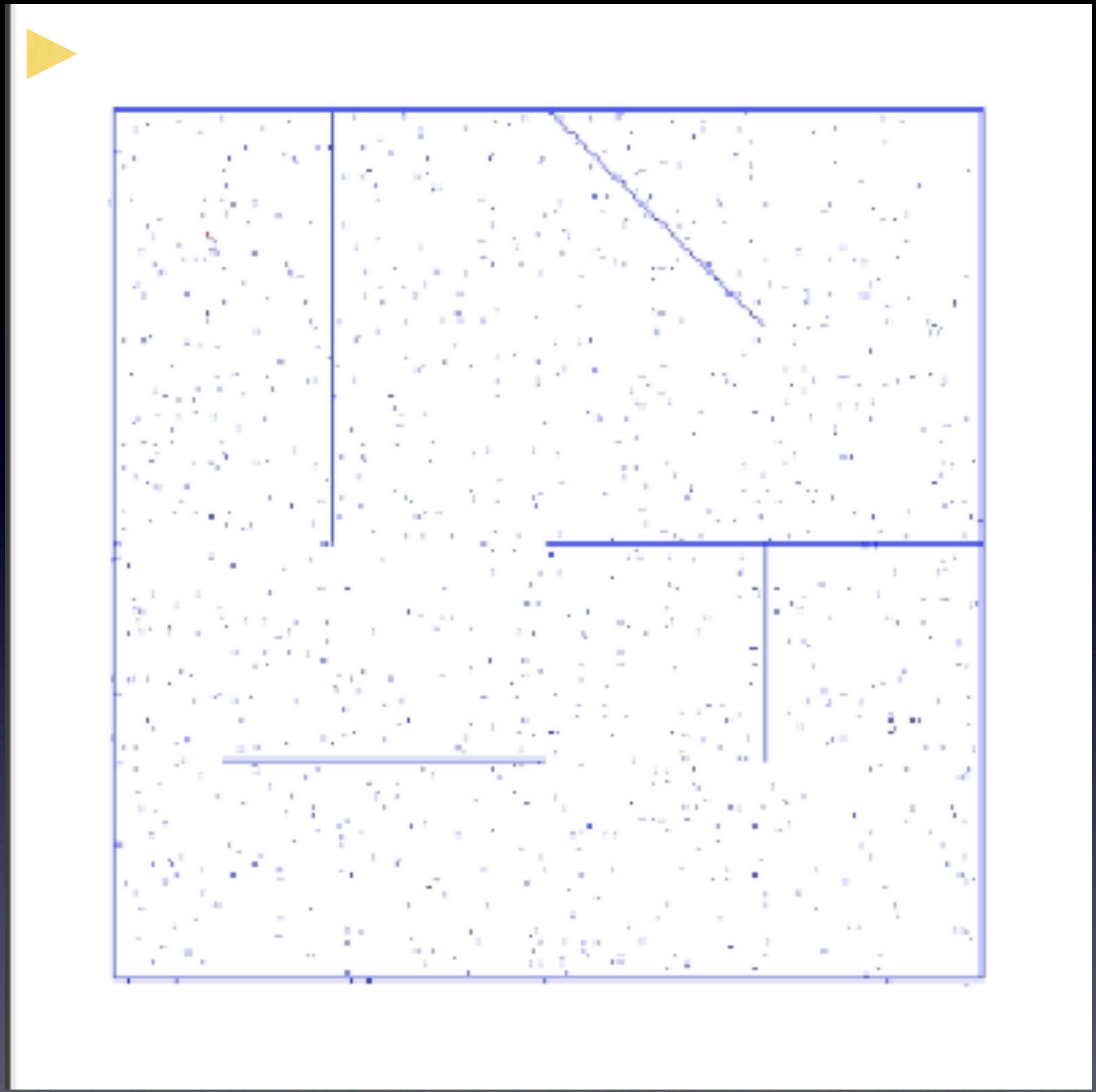
When a new sensor measurement ('z') is recieved
- For each particle 'p' in 'Particles':
  - Compute the posterior probability of each particle: P('p'.location | 'z')
- Resample the particles: 'Particles' = resampleParticles('Particles')

http://users.isr.ist.utl.pt/~vale/english/projects/pfilter/



http://users.isr.ist.utl.pt/~vale/english/projects/pfilter/

http://users.isr.ist.utl.pt/~vale/english/projects/pfilter/



http://users.isr.ist.utl.pt/~vale/english/projects/pfilter/

http://www.cs.washington.edu/ai/Mobile_Robotics/

http://www.cs.washington.edu/ai/Mobile_Robotics/

No Filtering Particle Filtering

http://www.it.uq.oz.au/~wyeth/NXT/

No Filtering

Particle Filtering

http://www.it.uq.oz.au/~wyeth/NXT/

Some Extras

- Do we really need to resample our particles every time we take a measurement?

Some Extras

- Do we really need to resample our particles every time we take a measurement?

No, we can calculate the number of effective particles:

$$N_{eff} = \frac{1}{\sum_{i}(\text{particle}_i.\text{prob})^2}$$

Some Extras

- Do we really need to resample our particles every time we take a measurement?

No, we can calculate the number of effective particles:

$$N_{eff} = \frac{1}{\sum_i (\text{particle}_i.\text{prob})^2}$$

And only resample when:

$$N_{eff} < N_{thresh}$$