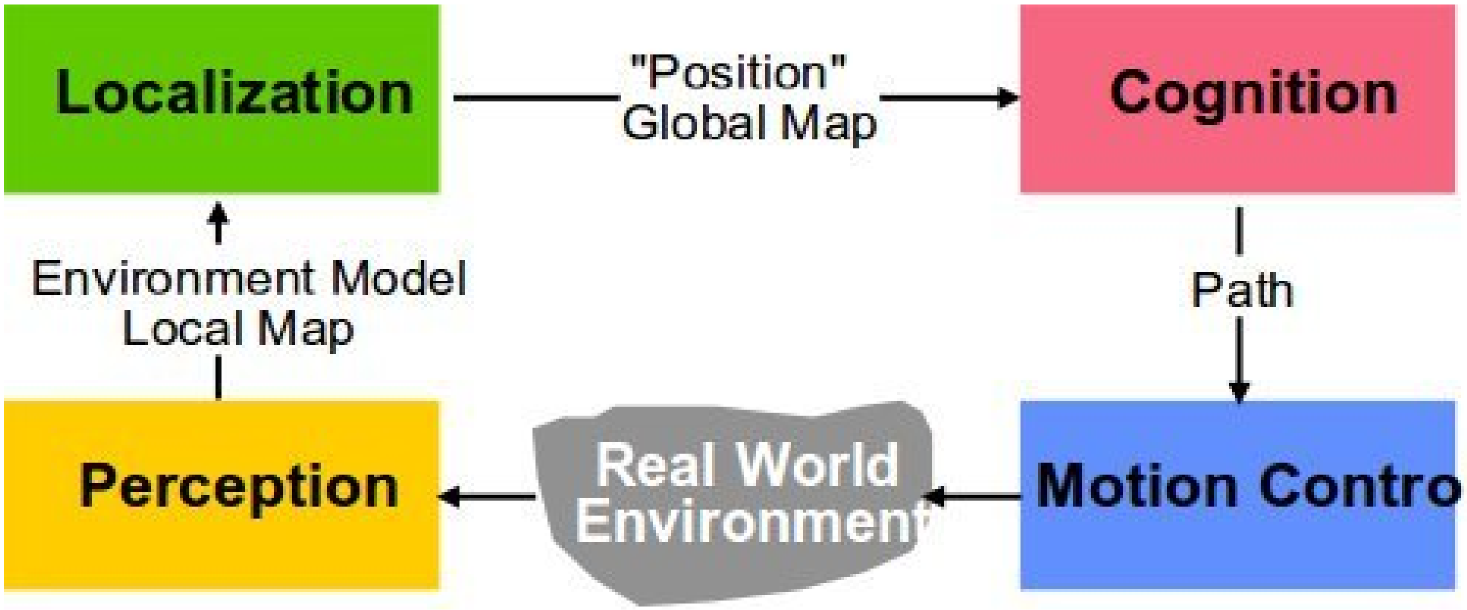
LOCALIZATION

Today



- Last time we looked at perception, and before that we looked at motion.
- This time we'll look at how using models of motion and perception we can figure out where the robot is.

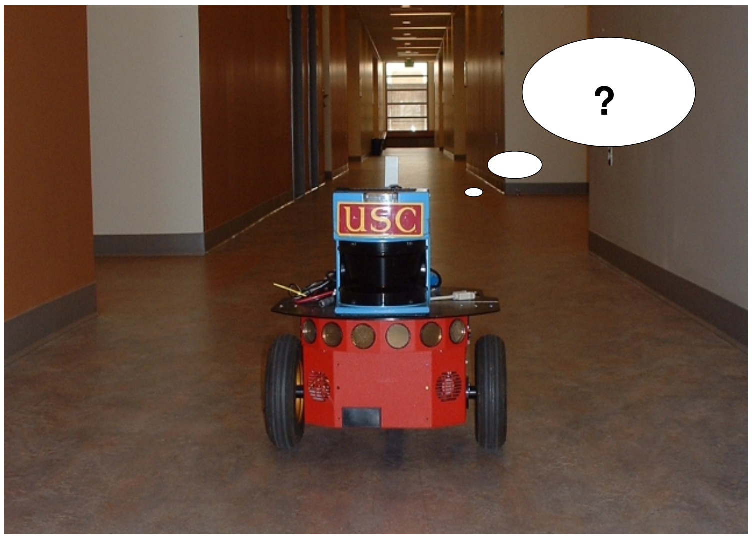
Acknowledgement

• The material on particle filters is heavily based on:

D. Fox, S. Thrun, F. Dellaert, and W. Burgard, Particle filters for mobile robot localization, in A. Doucet, N. de Freitas and N. Gordon, eds., Sequential Monte Carlo Methods in Practice. Springer Verlag, New York, 2000.

Localization

- We started this course with three questions:

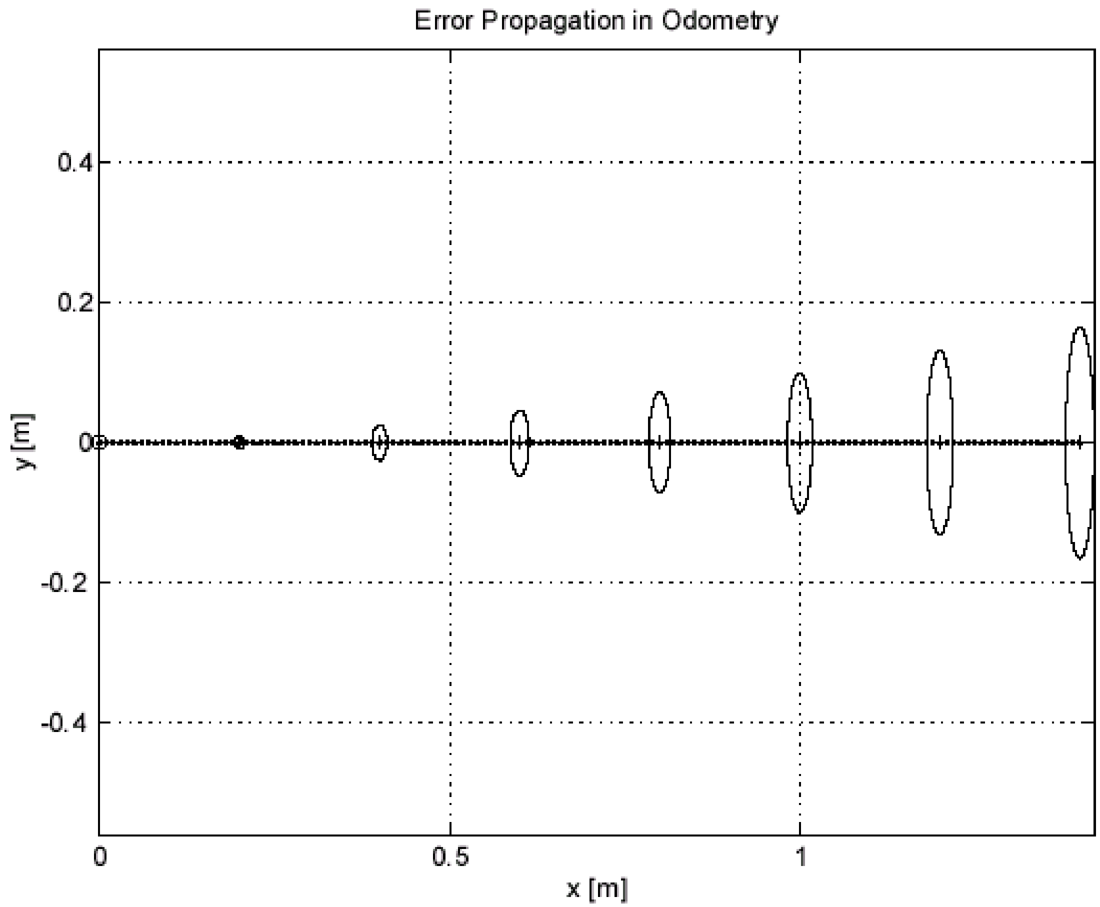


  - Where am I ?
  - Where am I going ?
  - How do I get there ?
- We are now at a point where we can answer the first of these.

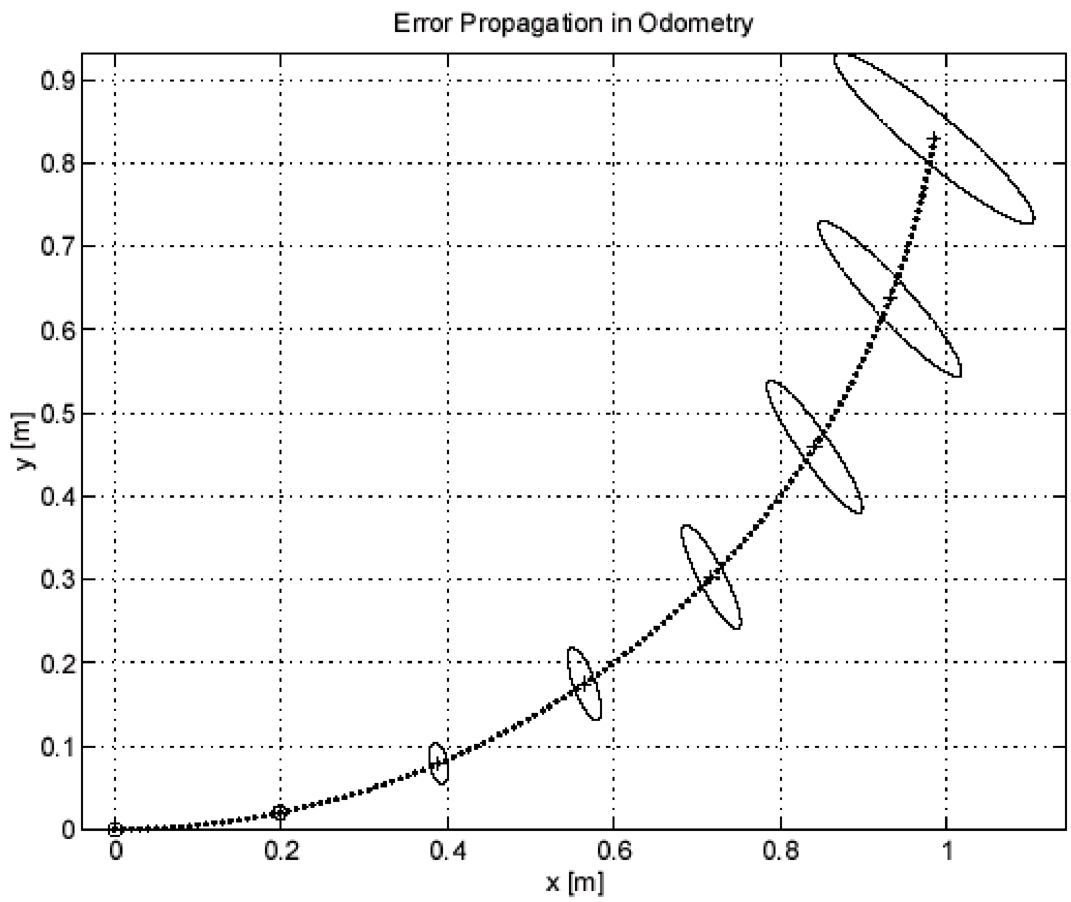
- The basic localization task is to compute current location and orientation (*pose*) given observations.
  - What constitues a pose depends on what kind of map we have.
  - But roughly speaking it is (x_I, y_I, θ).
  - The same things we worried about in the motion model
- Do we need to do any more than just use odometry?
  - After all, that seemed to work okay for the various lab exercises.

- In general odometry doesn't hold up well over long distances.
- *Range error*: integrated path length (distance) of the robots movement
  - Sum of the wheel movements
- *Turn error*: similar to range error, but for turns
  - Difference of the wheel motions
- *Drift error*: difference in the error of the wheels leads to an error in the robot's angular orientation.
- Over long periods of time, turn and drift errors far outweigh range errors!

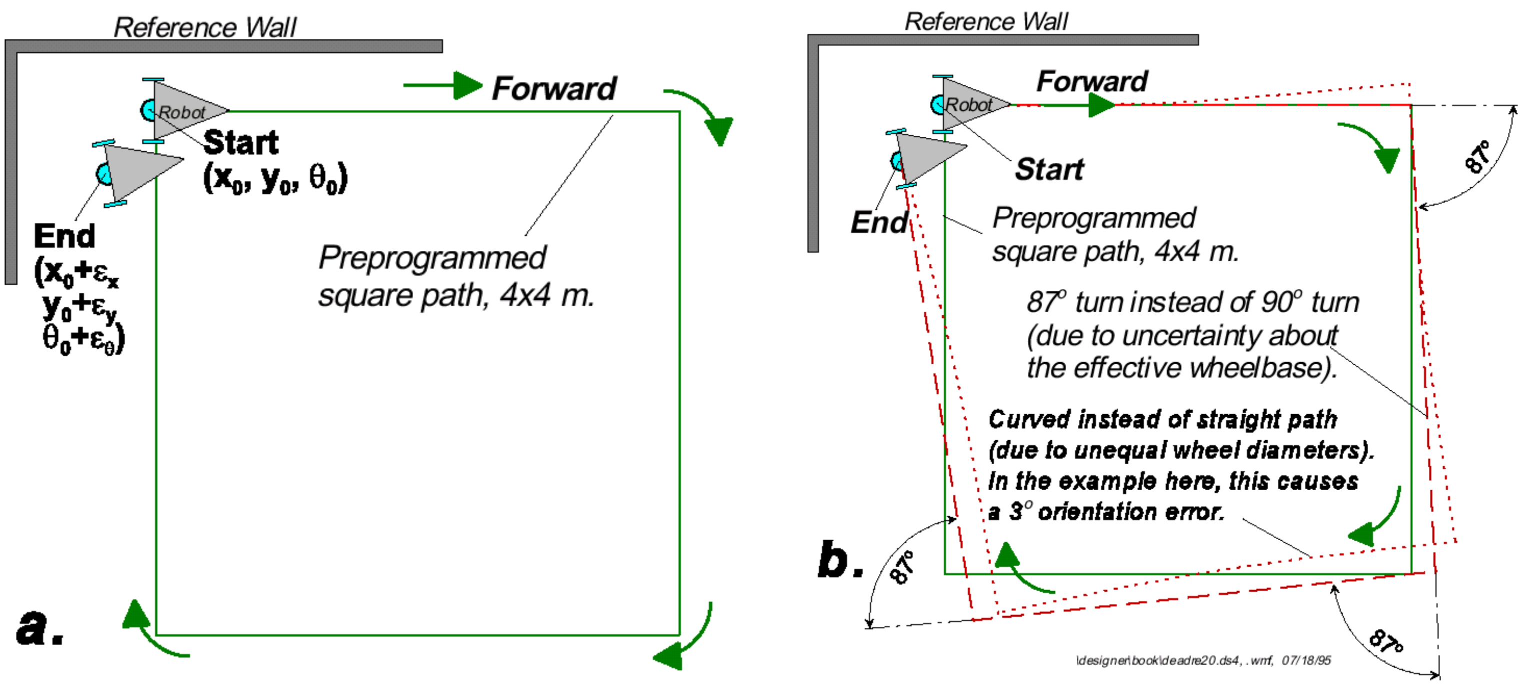
- A simple error model based on the kinematics predicts:



- Which is worse when we turn:

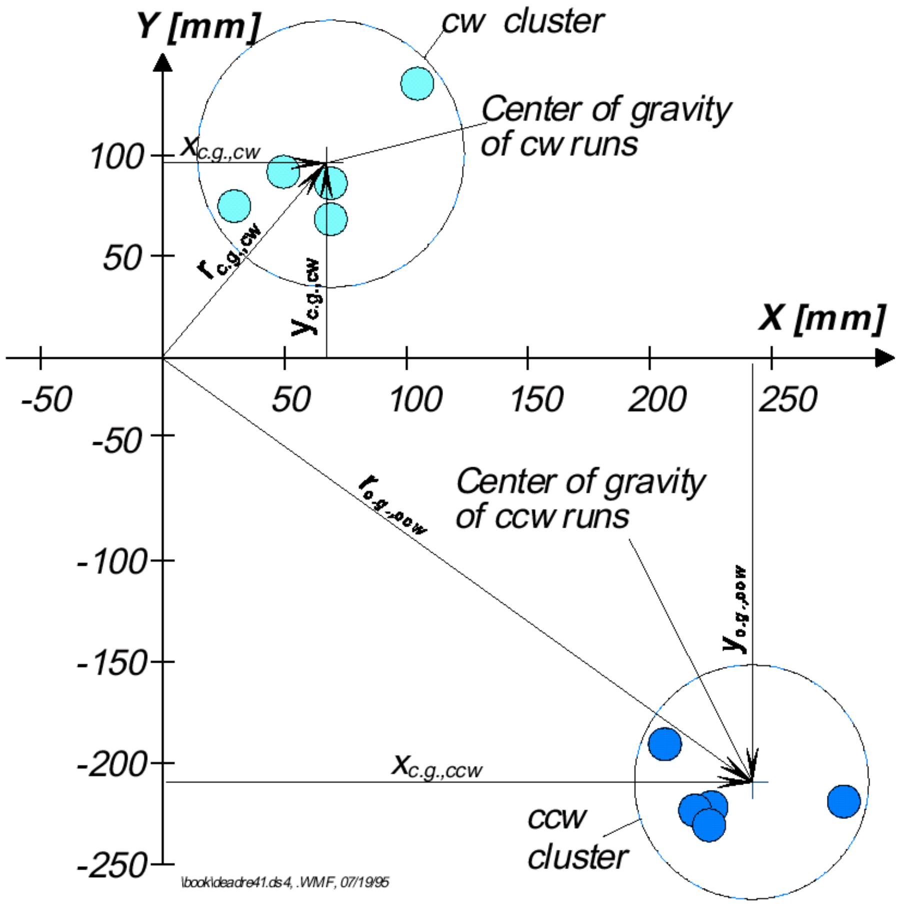


- Borenstein characterised this error:

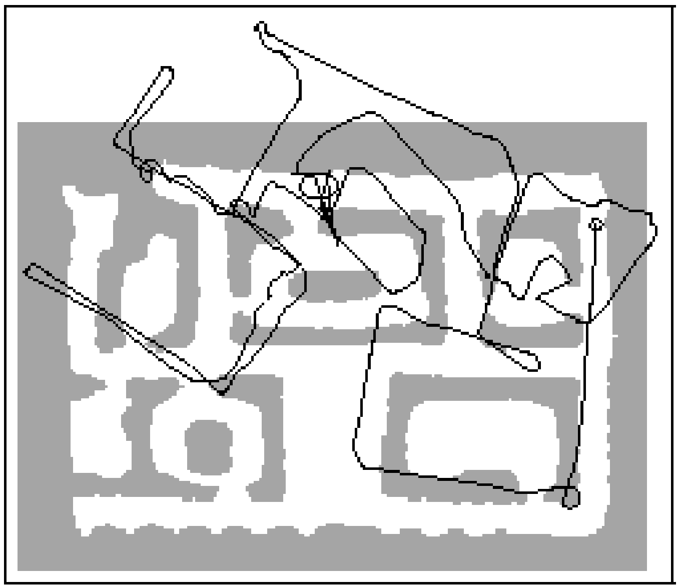
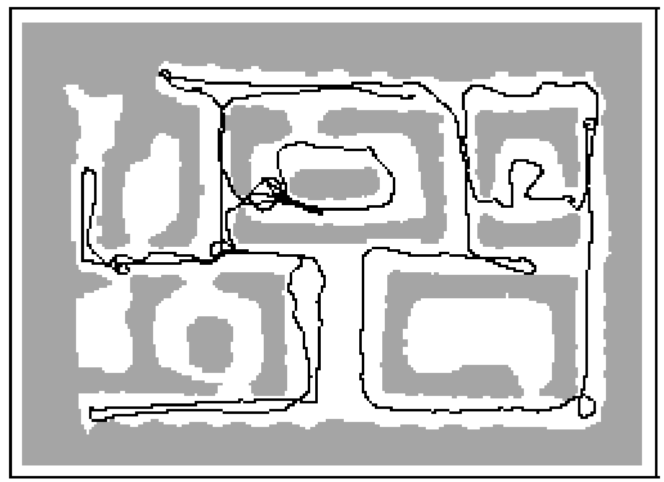


- Look familiar?

- Error distribution:

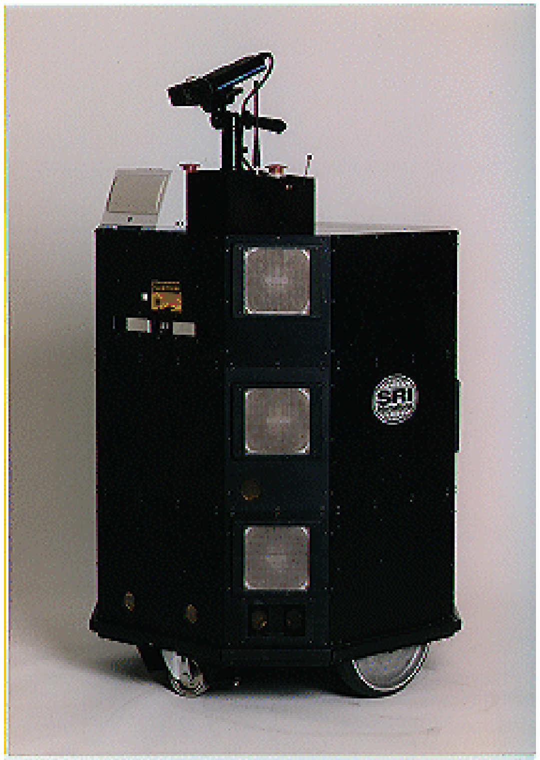


- Leading to:

- Images from Dieter Fox in his CMU days.

- The problems were memorably illustrated by:



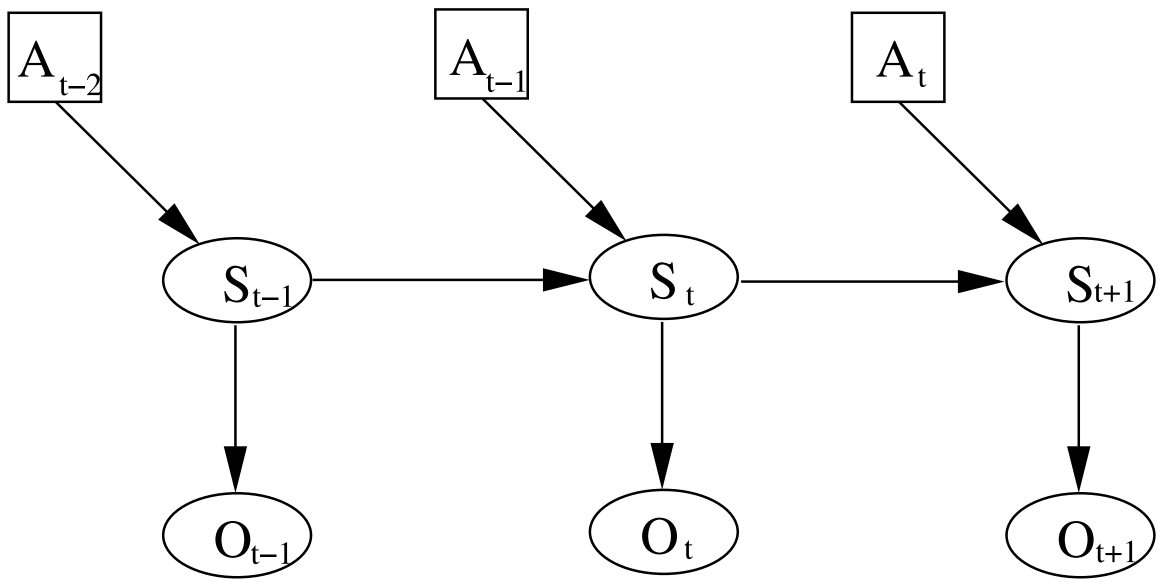
Flakey was custom-built at SRI. Differential drive gave it a maximum speed of 2 feet per second.

Sensors included a ring of 12 sonar, wheel encoders, video camera and a laser.

- If odometry alone doesn't help, what about GPS?
- Non-military GPS is not accurate enough to work on its own.
- Doesn't tend to work great indoors.

- Instead we try to use sensor data to identify where we are on a map.
- It is tempting to try and *triangulate*.
- But doing this is too prone to error.
  - Sensor noise.
  - Sensor aliasing.
- You get better results if you:
  - Combine data from multiple sensors.
  - Take into account previous estimates of where the robot is.

- General schema:

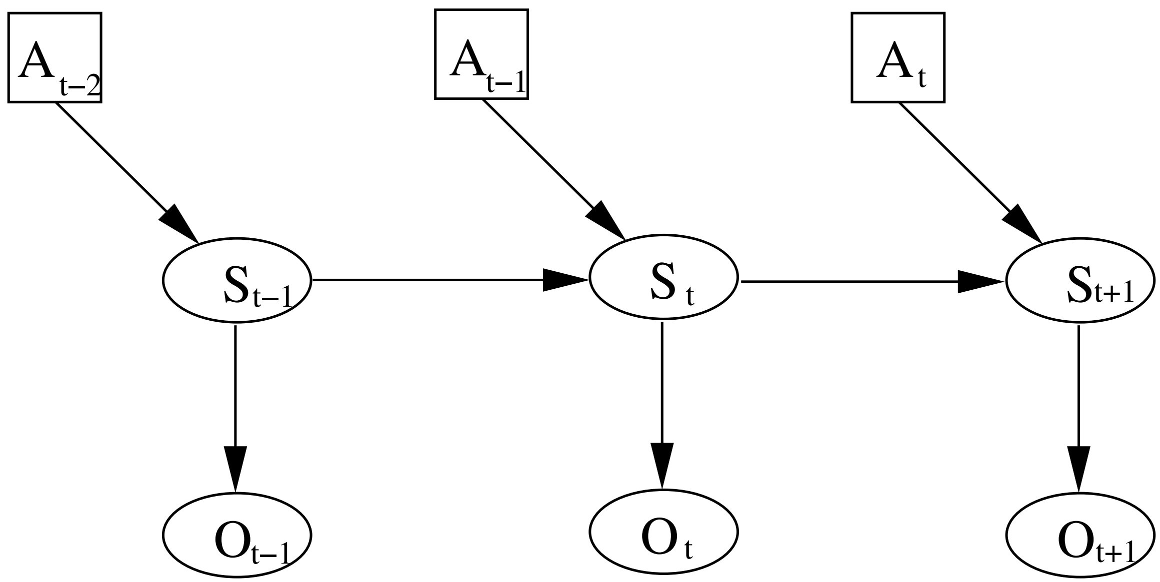


- Here A is action, S is pose and O is observation.
- The point is that position at one time depends on position at the previous time.

The localization problem(s)

- There are a number of flavors of localization:
  - Position tracking
  - Global localization
  - Kidnapped robot problem
  - Multi-robot localization
- All are hard, but variations of the technique we will look at helps to solve all of them.

- General schema (again):



- Here *A* is action, *S* is pose and *O* is observation.
- The point is that position at one time depends on position at the previous time.

- The pose at time t depends upon:
  - The pose at time $t-1$, and
  - The action at time $t-1$.
- The pose at time t determines the observation at time t.
- So, if we know the pose we can say what the observation is.

- But this is *backwards*...
- To help us out of this bind we need to bring in probabilities (as mentioned before they are also helpful because sensor data is noisy).

Probability theory

- Let's recap some probability theory
- We start with a *sample space* Ω.
- For instance, Ω for the action of rolling a die would be $\{1, 2, 3, 4, 5, 6\}$.
- Subsets of Ω then correspond to particular events. The set $\{2, 4, 6\}$ corresponds to the event of rolling an even number.
- We use S to denote the set of all possible events:

$$S = 2^{\Omega}$$

- It is sometimes helpful to think of the sample space in terms of Venn diagrams—indeed all probability calculations can be carried out in this way.

- A probability measure is a function:

$$\Pr : S \mapsto [0, 1]$$

such that:

$$\begin{aligned}\Pr(\emptyset) &= 0 \\ \Pr(\Omega) &= 1 \\ \Pr(E \cup F) &= Pr(E) + \Pr(F), \text{whenever } E \cap F = \emptyset\end{aligned}$$

- Saying $E \cap F = \emptyset$ is the same as saying that E and F cannot occur together.
- They are thus *disjoint* or *exclusive*.
- The meaning of a probability is somewhat fraught; both frequency and subjective belief (Bayesian) interpretations are problematic.

- If the occurrence of an event E has no effect on the occurrence of an event F, then the two are said to be *independent*.
- An example of two independent events are the throwing of a 2 on the first roll of a die, and a 3 on the second.
- If E and F are independent, then:

$$\Pr(E \cap F) = \Pr(E). \Pr(F)$$

- When E and F are not independent, we need to use:

$$\Pr(E \cap F) = \Pr(E). \Pr(F|E)$$

  where $\Pr(F|E)$ is the *conditional probability* of F given that E is known to have occurred.
- To see how $\Pr(F)$ and $\Pr(F|E)$ differ, consider F is the event "a 2 is thrown" and E is the event "the number is even".

- We can calculate conditional probabilities from:

$$\Pr(\boldsymbol{F}|\boldsymbol{E}) = \frac{\Pr(\boldsymbol{E} \cap \boldsymbol{F})}{\Pr(\boldsymbol{E})}$$

$$\Pr(\boldsymbol{E}|\boldsymbol{F}) = \frac{\Pr(\boldsymbol{E} \cap \boldsymbol{F})}{\Pr(\boldsymbol{F})}$$

  which, admittedly is rather circular.

- We can combine these two identities to obtain *Bayes' rule*:

$$\Pr(\boldsymbol{F}|\boldsymbol{E}) = \frac{\Pr(\boldsymbol{E}|\boldsymbol{F})\Pr(\boldsymbol{F})}{\Pr(\boldsymbol{E})}$$

- Also of use is *Jeffrey's rule*:

$$\Pr(\boldsymbol{F}) = \Pr(\boldsymbol{F}|\boldsymbol{E})\Pr(\boldsymbol{E}) + \Pr(\boldsymbol{F}|\neg\boldsymbol{E})\Pr(\neg\boldsymbol{E})$$

- More general versions are appropriate when considering events with several different possible outcomes.

Bayes Filtering

- The technique we will use for localization is a form of *Bayes filter*.
- The key idea is that we calculate a *probability distribution* over the set of possible poses.
- That is we compute the probability of each pose that is in the set of all possible poses.
- We do this informed by all the data that we have.
- This is what the paper means by:

  estimate the posterior probability density over the state space conditioned on the data.

- We call the probability that we calulate the *belief*.
- We denote the belief by:

$$Bel(s_t) = \Pr(x_t|d_{0,\ldots,t})$$

where $d_{0,\ldots,t}$ is all the data from time 0 to t.

- Two kinds of data are important:
  - Observations o_t
  - Actions a_t

just as in the general scheme.

- Note: the scheme on pages 14 and 16 uses S for state A for action and O for observation, just as the textbook does. The Fox paper uses u for action and y for observation.
- From here on I'll use s for pose, a for action and o for observation.

- Without loss of generality we assume actions and observations alternate:

$$Bel(s_t) = \Pr(s_t | o_t, a_{t-1}, o_{t-1}, a_{t-2}, \ldots, o_0)$$

- We figure out this belief by updating recursively.
- We can use Bayes' rule to write the above as:

$$Bel(s_t) = \frac{\Pr(o_t | s_t, a_{t-1}, \ldots, o_0) \Pr(s_t | a_{t-1}, \ldots, o_0)}{\Pr(o_t | a_{t-1}, \ldots, o_0)}$$

which reduces to:

$$Bel(s_t) = \eta \Pr(o_t | s_t, a_{t-1}, \ldots, o_0) \Pr(s_t | a_{t-1}, \ldots, o_0)$$

since the denominator is constant relative to s_t.

- Now, the basic principle behind using Bayes filters is that *if* we know the current state, then future states do not depend on past states.
- The *Markov* assumption.
- In this case the Markov assumption says that

$$\Pr(o_t | s_t, a_{t-1}, \ldots, o_0)$$

reduces to:

$$\Pr(o_t | s_t)$$

and the big expression can be written as:

$$\boldsymbol{Bel}(s_t) = \eta \Pr(o_t | s_t) \Pr(s_t | a_{t-1}, \ldots, o_0)$$

- A little more maths gets us a recursive equation for the belief.
- We integrate over the state at time $t-1$:

$$\mathbf{Bel}(s_t) = \eta \Pr(o_t|s_t) \int \Pr(s_t|s_{t-1}, a_{t-1}, \ldots, o_0) \Pr(s_{t-1}|a_{t-1}, \ldots, o_0) ds_{t-1}$$

- Then again exploiting the Markov assumption we reduce $\Pr(s_{t-1}|a_{t-1}, \ldots, o_0)$ and get:

$$\mathbf{Bel}(s_t) = \eta \Pr(o_t|s_t) \int \Pr(s_t|s_{t-1}, a_{t-1}) \Pr(s_{t-1}|a_{t-1}, \ldots, o_0) ds_{t-1}$$

- Finally we get:

$$\mathbf{Bel}(s_t) = \eta \Pr(o_t|s_t) \int \Pr(s_t|s_{t-1}, a_{t-1}) \mathbf{Bel}(s_{t-1}) ds_{t-1}$$

- This allows us to calculate the belief recursively based on:
  - The *next state density* or *motion model*

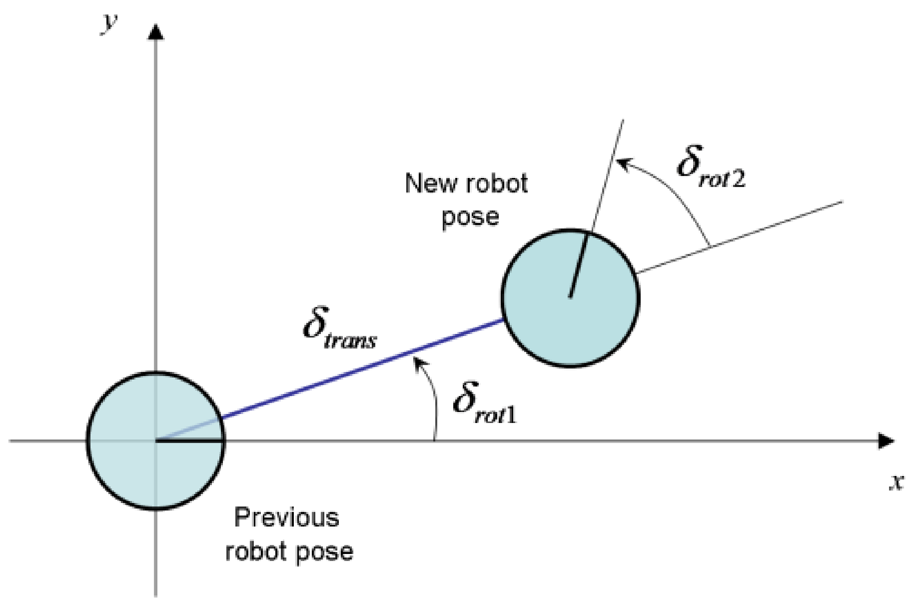
$$\Pr(s_t | s_{t-1}, a_{t-1})$$

  - The *sensor model*

$$\Pr(o_t | s_t)$$

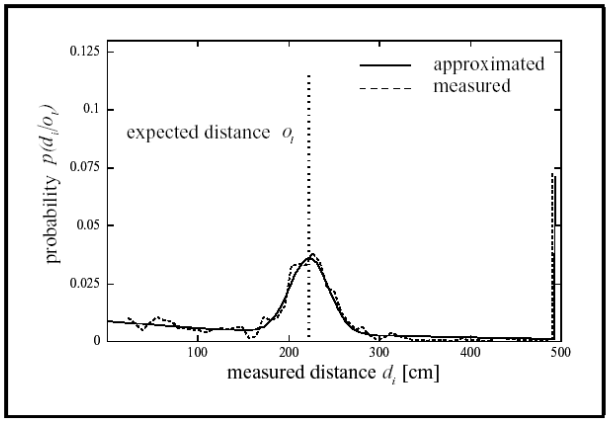
- In other words, belief about the current location is a function of belief about the previous location, what the robot did, and what the robot can see.

- The motion model, obviously enough, predicts how the robot moves.

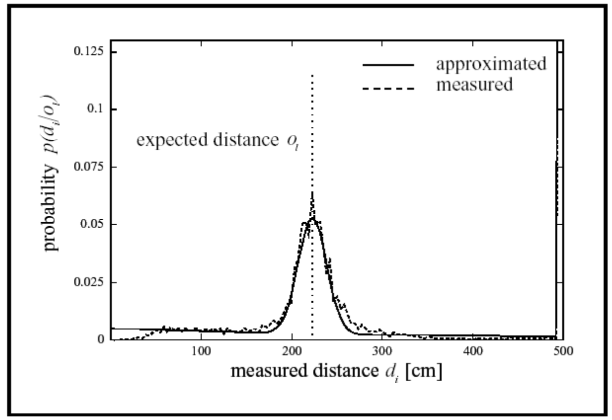


- The model should take into account the fact that the motion is uncertain.

- The sensor model captures both the *landmarks* the robot can see, and the lack of precise knowledge in where the robot must be to see them.



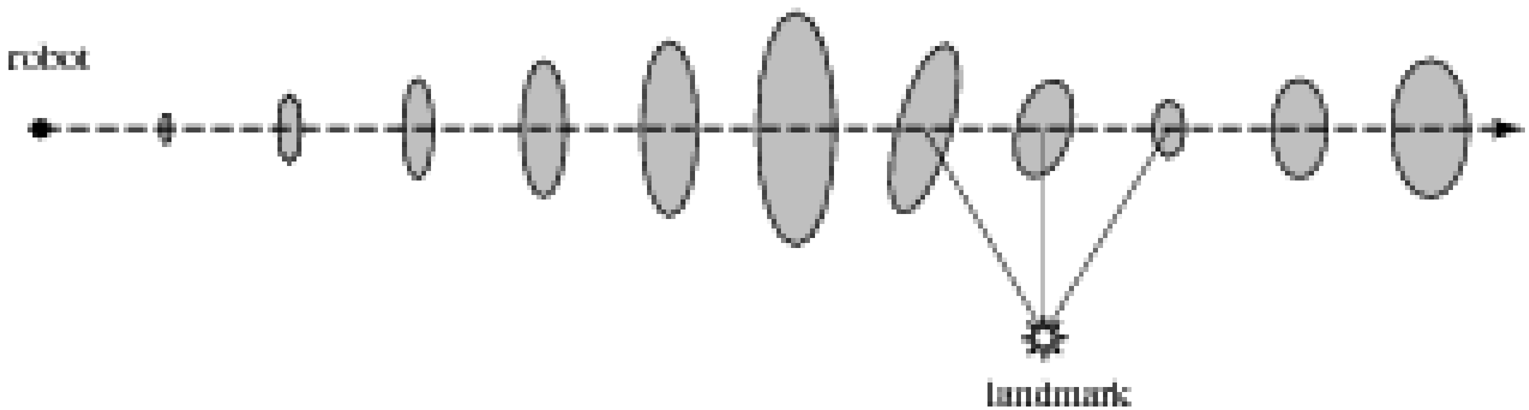
Ultrasound.



Laser range-finder.

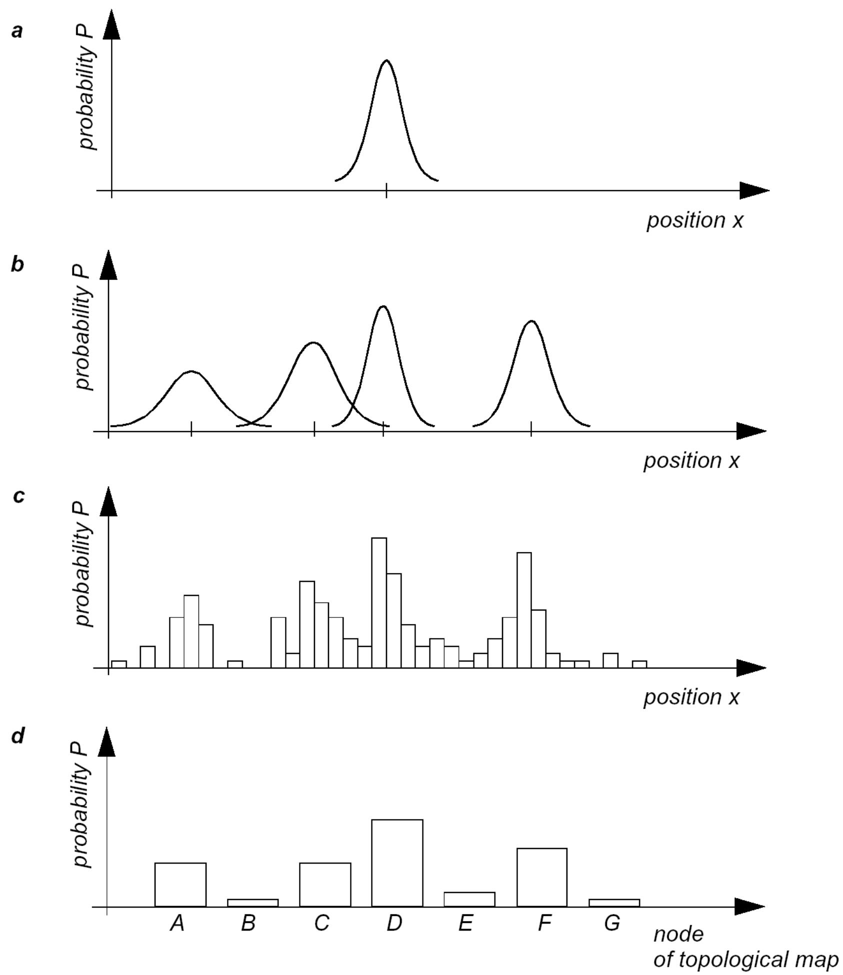
- o_t in the above is the distance the sensor says the object is away from the robot, d_t is the real distance.
- The map tells us how far the object is, d_t, and the graph tells us how likely this is.

- Overall, the filtering procedure works to reduce uncertainty of location when landmarks are observed.



- Diagram assumes that landmarks are *identifiable*—otherwise, *Bel* is multimodal

Models of belief



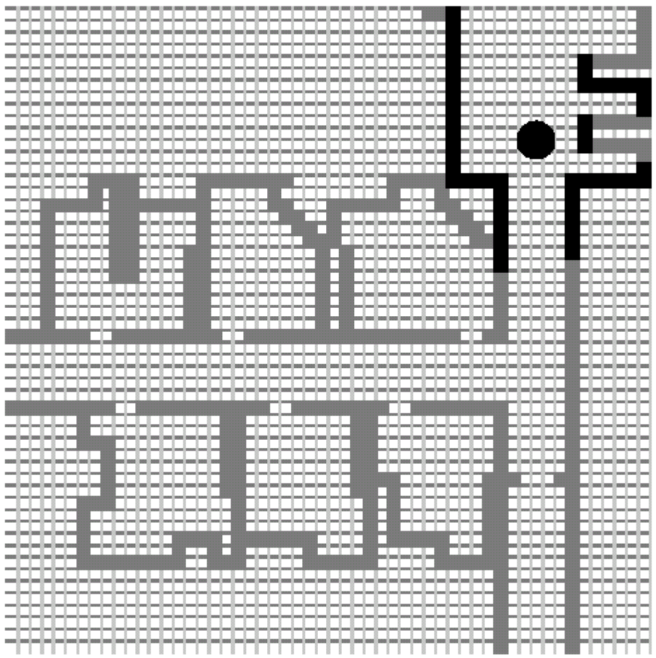
- Single hypothesis, continuous distribution
- Multiple hypothesis, continuous distribution
- Multiple hypothesis, discrete distribution
- Topological map, discrete distribution

- Handling the kind of probability distributions that the Bayes filter requires is a bit tricky.
- So we improvise.
- Three different approaches:
  - Assume everything is Gaussian.
  - Make the environment discrete.
  - Take a *sampling* approach.
- All are used with differing degrees of success.

- Assuming Gaussian distributions gives us *Kalman* filters.
  - Fast and accurate.
  - Only really work for position tracking.
- A discrete environment gives us *Markov* localization.
  - Simple.
  - Accuracy requires huge memory.
- We'll start by looking at Markov localization.

Markov Localization

- We start with a map that breaks the world into a grid:



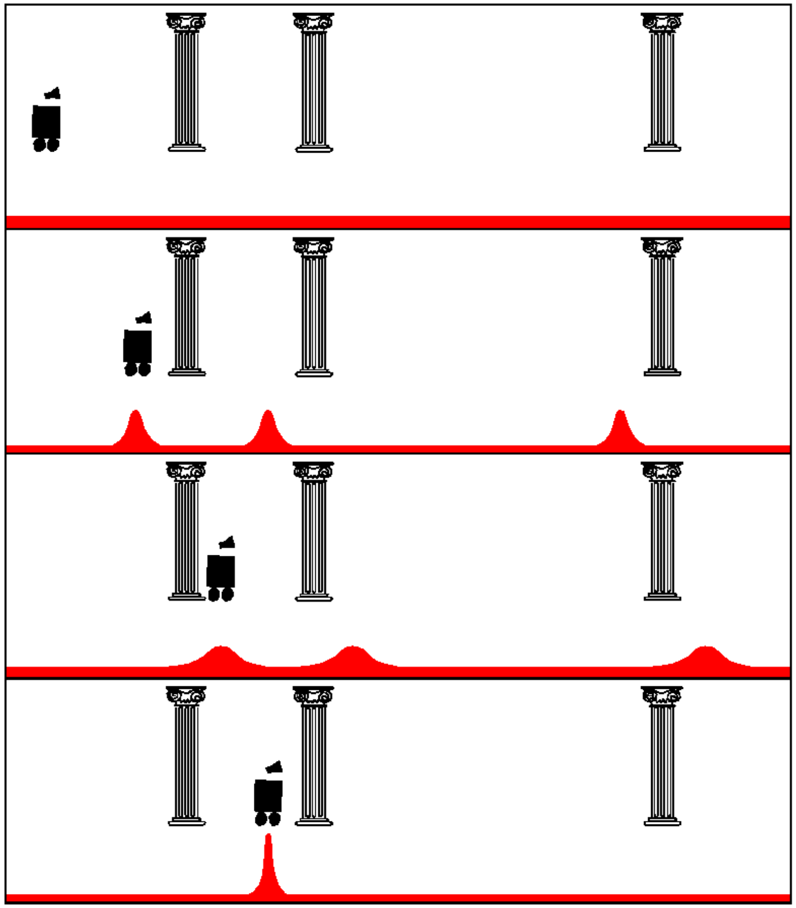
- There are many ways to do this, and we'll talk about some of them in the next lecture.

- Initally we have a uniform distribution over the possible locations.
- For every observation, for every location, we check what we observe against the map.
  - Apply the sensor model to find out how likely the observation is from that location.
  - Update the probability of the location.
- Then we normalize the probabilities — make sure they all add up to 1.

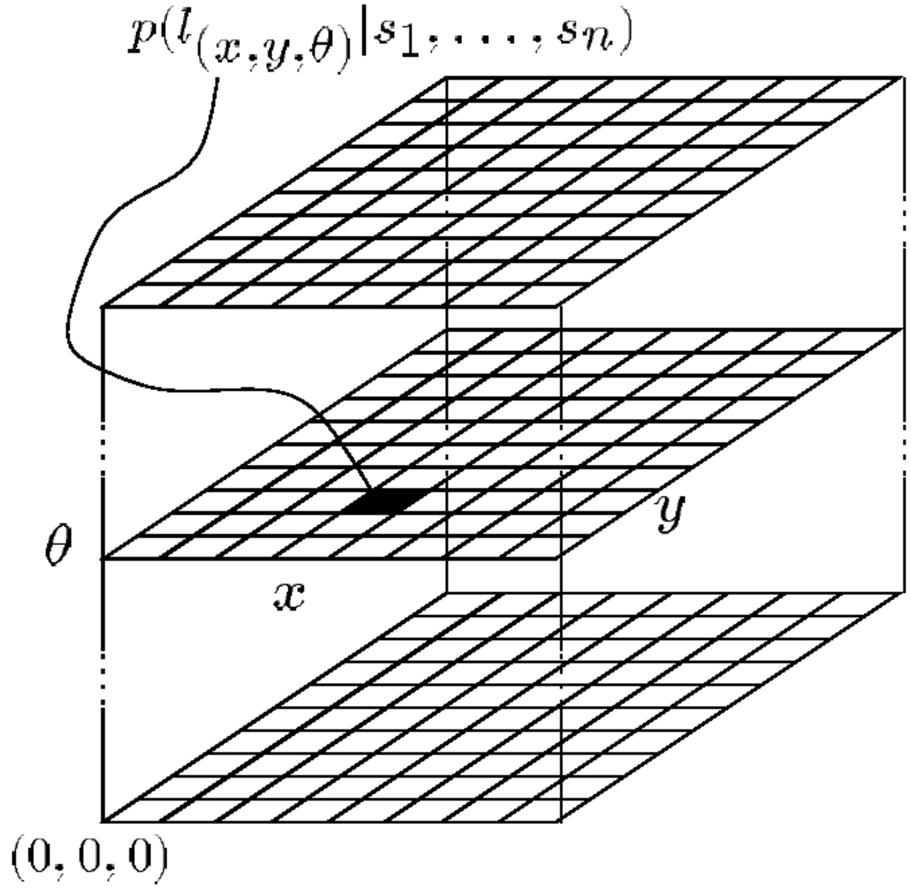
- For every motion, for every location
  - Apply the sensor model to find out what new locations are how likely.
  - Update the probability of those locations.
- Then we normalize the probabilities — make sure they all add up to 1.

- We repeat this process for every item of sensor data and every motion.

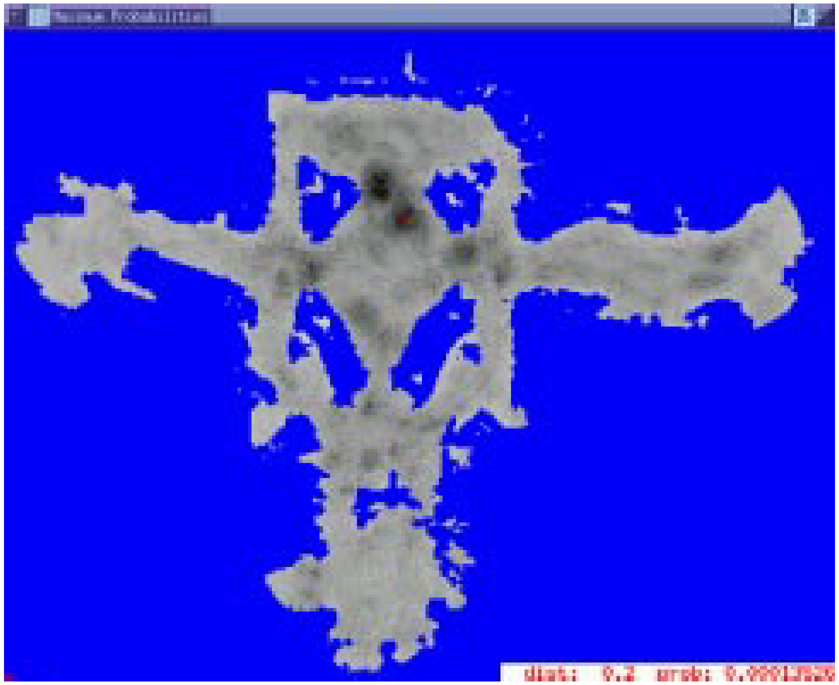
- Crudely what happens:

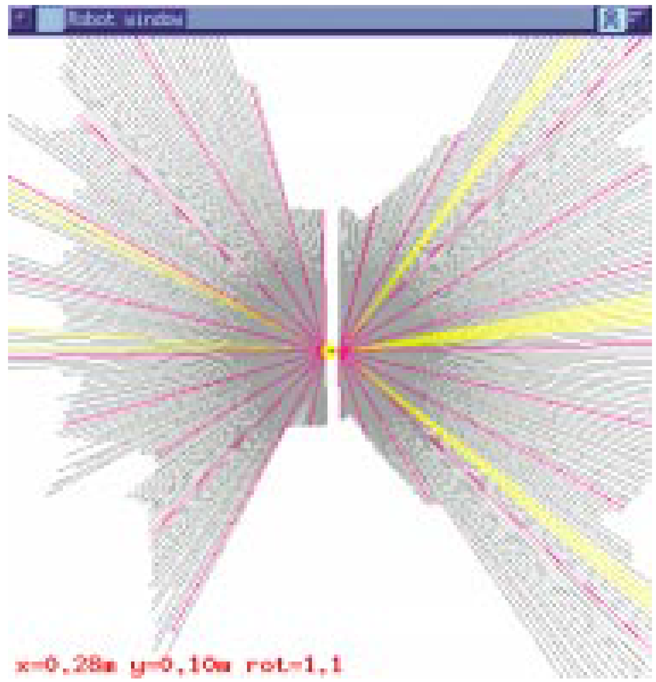


- How the grid updates:



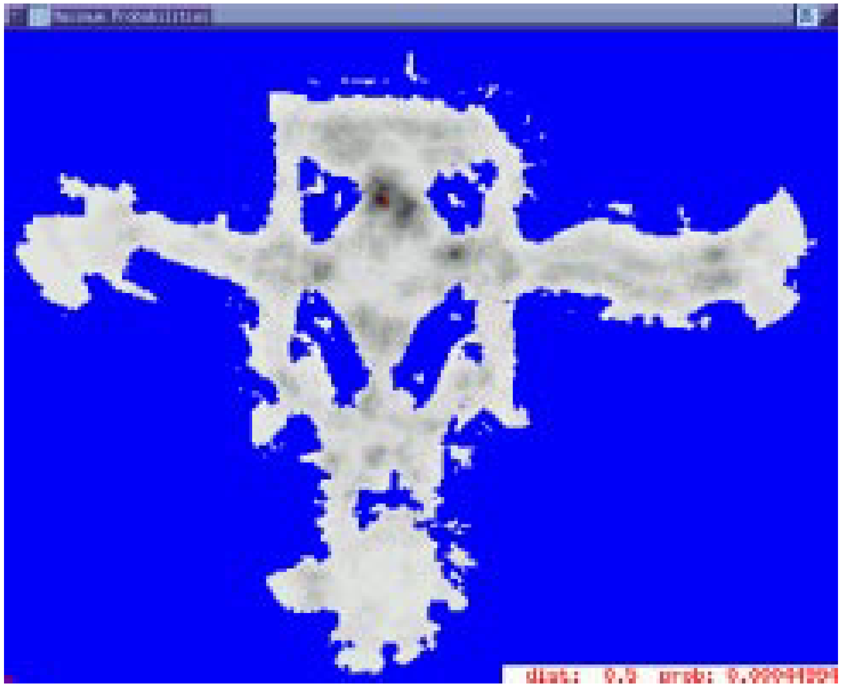
- After 1 scan.

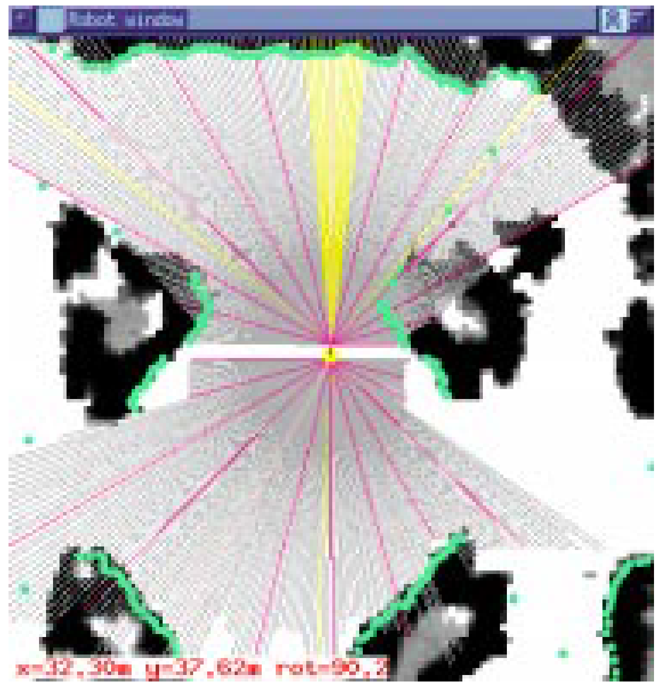




- W. Burgard

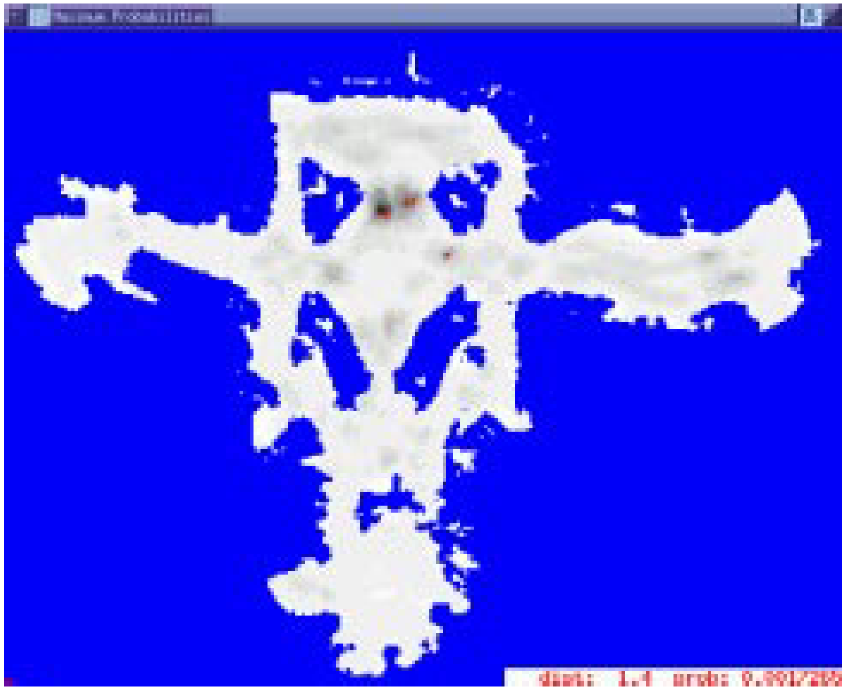
- After 2 scans.

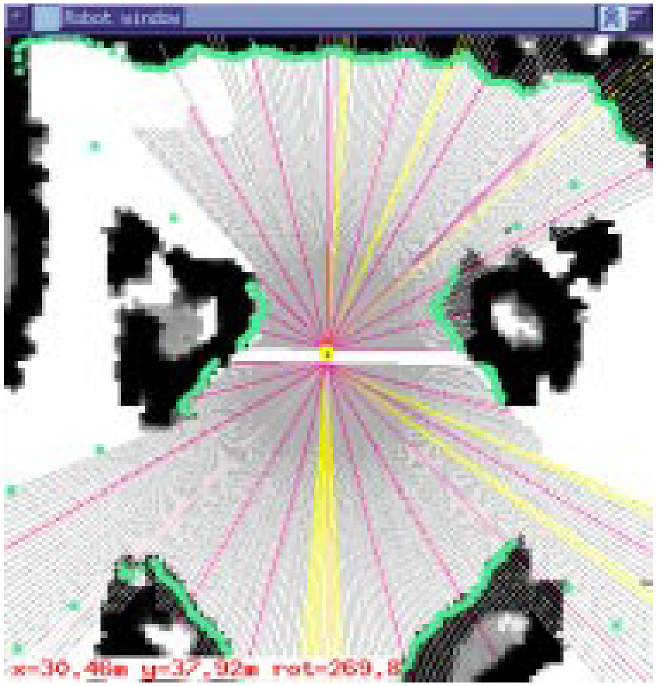




- W. Burgard

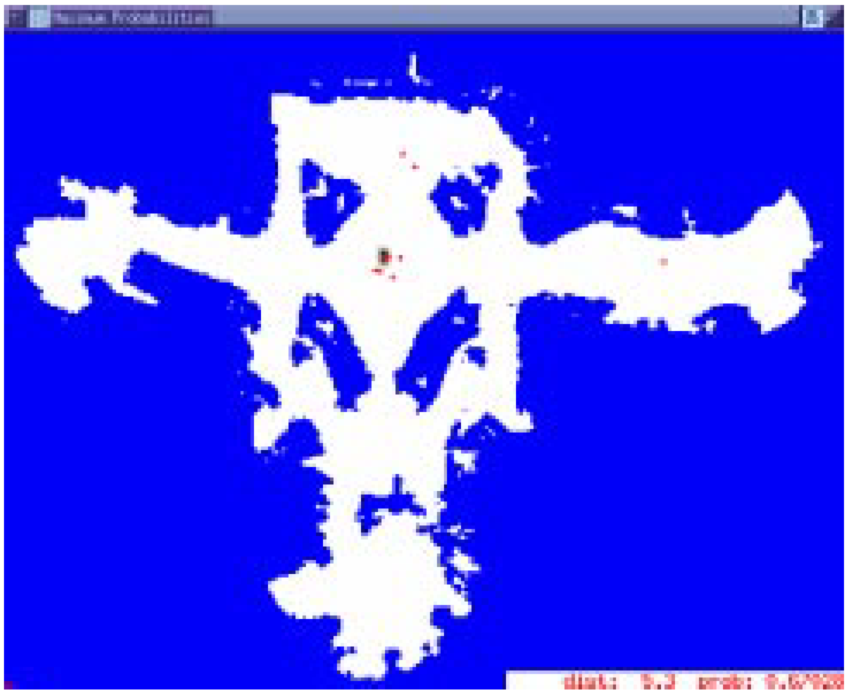
- After 3 scans.

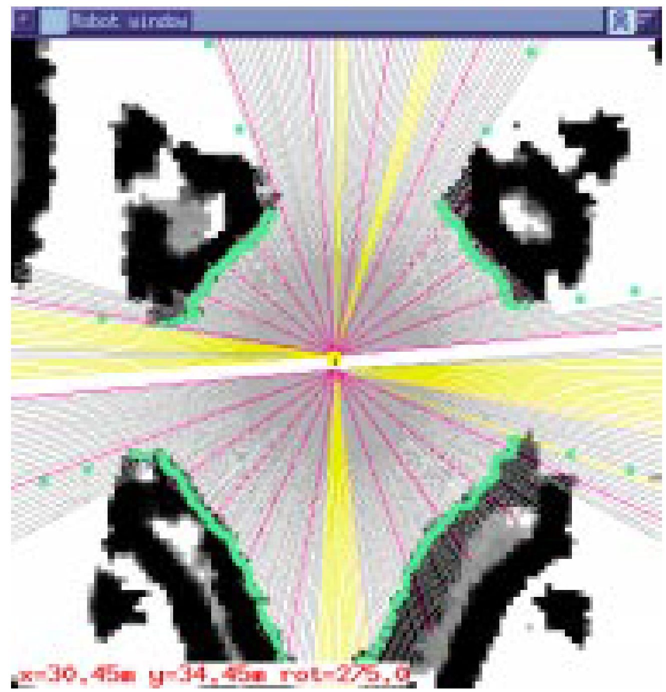




- W. Burgard

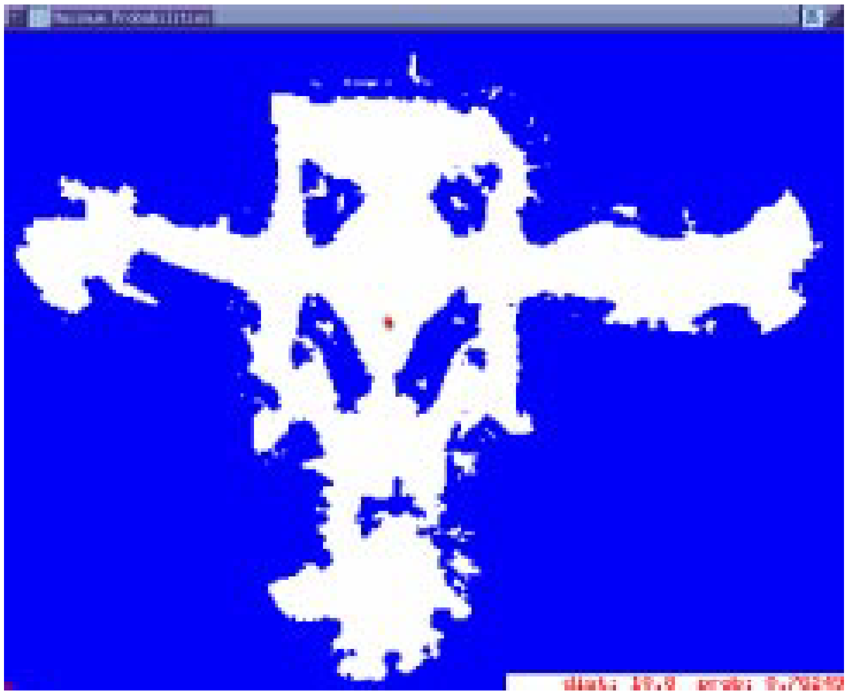
- After 13 scans.

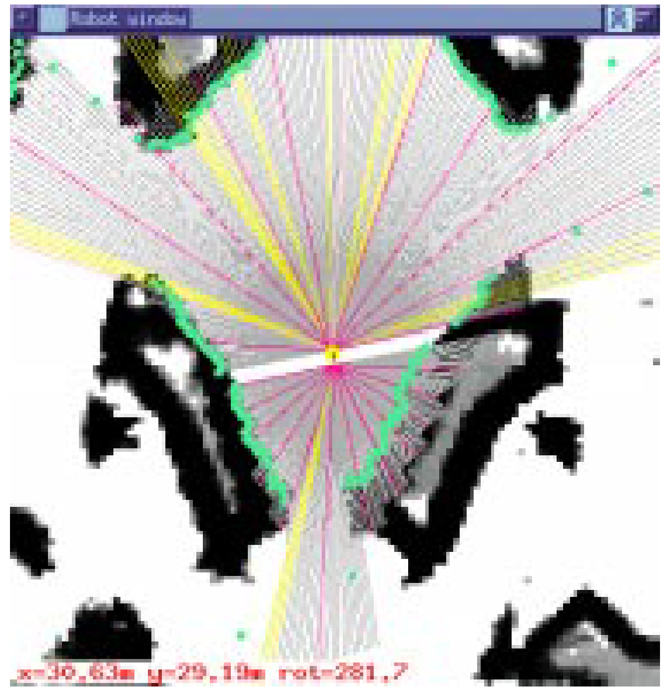




- W. Burgard

- After 21 scans.

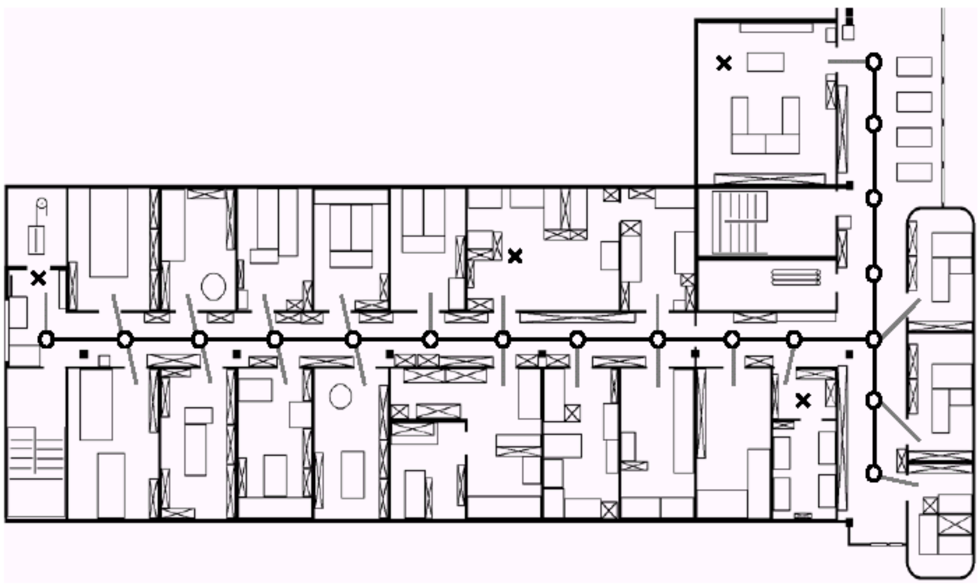




- W. Burgard

Topological maps

- Another way to make the map discrete is to use a topological map.



- Treat it the same way as the grid map.
- Fewer locations is good and bad.

Improving on Markov Localization

- The problem with Markov localization is that if the area is big, we need to consider a lot of possible locations.
  - Memory and processor intensive
- *Particle filters* use sampling techniques to reduce the number of possible positions, and hence the number of calculations.
- The sampling approach is what we will consider next.
- Rather than compute the whole distribution, we pick possible locations (samples) and do the calculations for them.
- This can work with surprisingly few samples (or *particles*).

Particle filter

- Also known as "Monte-Carlo Localization".
- We approximate $Bel(s_t)$by a set of samples:

$$Bel(s_t) \approx \{s_t^{(i)}, w_t^{(i)}\}_{i=1,\ldots,m}$$

- Each $s_t^{(i)}$ is a possible pose, and each $w_t^{(i)}$ is the probability of that pose (also called an *importance factor*).
- Initially we have a set of samples (typically uniform) that give us $Bel(s_o)$.
- Then we update with the following algorithm.

$s_{t+1} = \emptyset$

for $j = 1$ to m

// `apply the motion model`

generate a new sample $s_{t+1}^{(j)}$ from $s_t^{(j)}$, a_t and $\Pr(s_{t+1}|s_t, a_t)$

// `apply the sensor model`

compute the weight $w_{t+1}^{(j)} = \Pr(o_{t+1}|s_{t+1})$

// `pick points randomly but biased by their weight`

for $j = 1$ to m

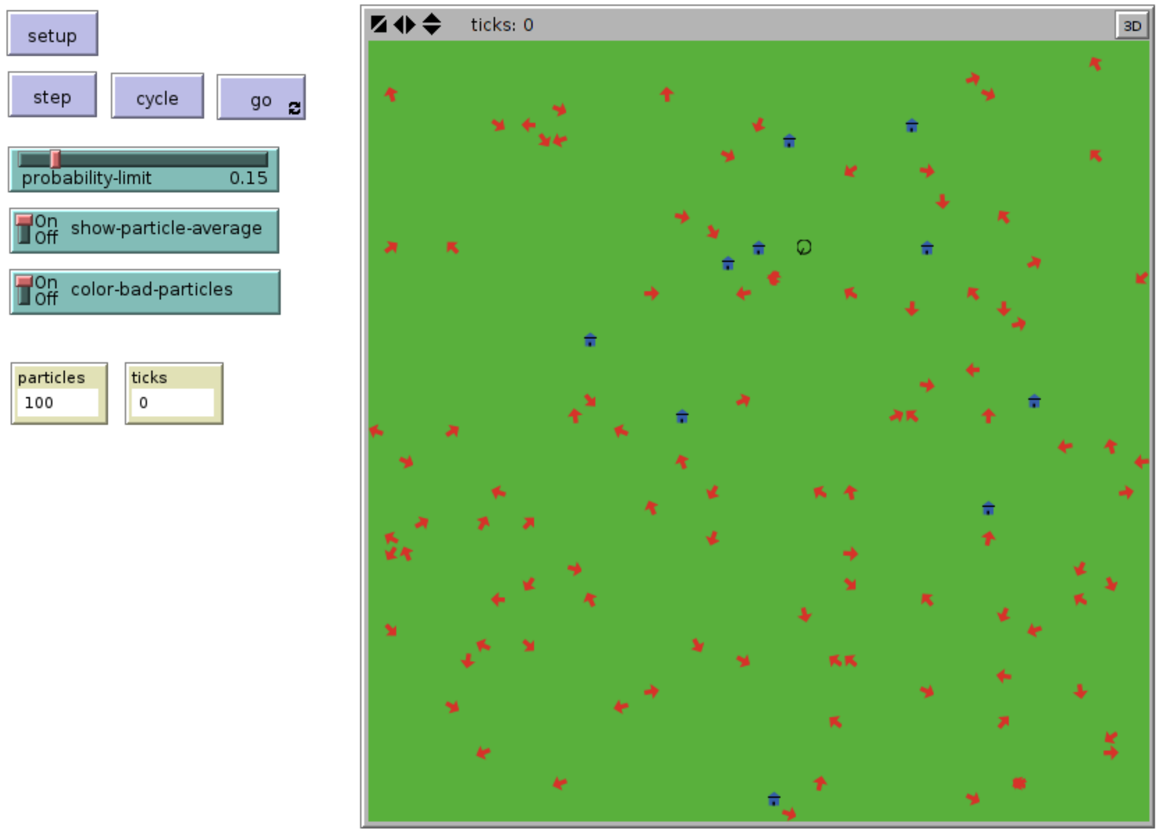
pick a random $s_{t+1}^{(i)}$ from s_{t+1} according to $w_{t+1}^{(1)}, \ldots, w_{t+1}^{(m)}$

normalize w_{t+1} in s_{t+1}

return s_{t+1}

- And that is all it takes.

- How does this work?



Effectiveness

- All localization is limited by the noise in sensors:
  - There are techniques for reducing noise by modelling spurious measurements.
  - Cannot remove all uncertainty.
- Discrete, grid-based approaches can reduce average error below 5cm.
  - However this is hard to do in real-time.
  - Requires huge amounts of memory.
- Particle filters with feasible sample sizes (≈ 1000) have comparable error rates.

- With much smaller numbers of particles (≈ 100) we have average errors of around 10cm.
- This is sufficient for many tasks.

Kidnapped robot

- Markov localization has no problem with the kidnapped robot
  - Always considers all possible poses.
- A well-localized particle filter cannot easily recover from kidnapping.
- Solution: seed the particle set with some random particles.
  - simple: fixed percentage of particles.
  - sensor resetting: larger number of particles the less well-localized the robot is.

Sensor resetting

```
averageProb = totalProb/PARTICLES;

case SENSOR_RESETTING:

  return (int)floor(PARTICLES *
          max(0, (1 - (averageProb/P_THRESHOLD))));
  break;

case SENSOR_RESETTING_PLUS:

  longAverageProb
      += ETA_LONG * (averageProb - longAverageProb);
  shortAverageProb
      += ETA_SHORT * (averageProb - shortAverageProb);
  return (int)floor(PARTICLES
       * max(0,(1 - NU * (shortAverageProb/longAverageProb))));
  break;
```

Summary

- This lecture looked at the problem of localization
  - How we have the robot figure out where it is.
- We discussed why odometry is not sufficient.
- We then described probabilistic localization techniques, concentrating on:
  - Markov localization
  - Particle filters
- Next lab we'll start to play with localization.