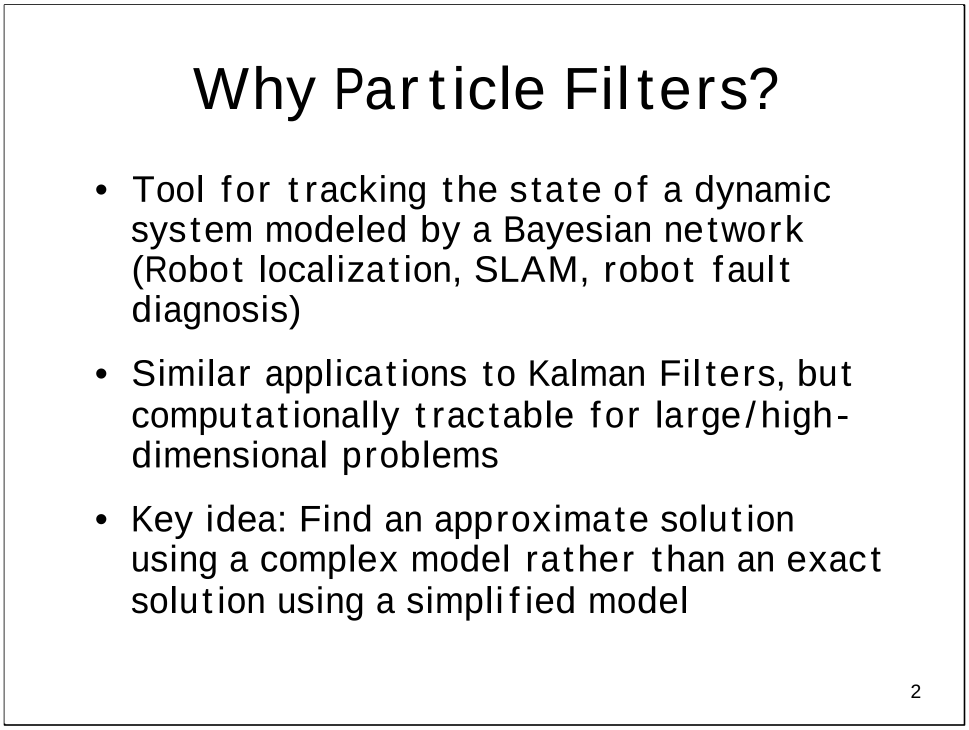
Particle Filters and Their Applications

Kaijen Hsiao
Henry de Plinval-Salgues
Jason Miller

Cognitive Robotics
April 11, 2005



Why Particle Filters?

- Tool for tracking the state of a dynamic system modeled by a Bayesian network (Robot localization, SLAM, robot fault diagnosis)
- Similar applications to Kalman Filters, but computationally tractable for large/high-dimensional problems
- Key idea: Find an approximate solution using a complex model rather than an exact solution using a simplified model

Why should you be interested in particle filters?

Because, like Kalman filters, they're a great way to track the state of a dynamic system for which you have a Bayesian model. That means that if you have a model of how the system changes in time, possibly in response to inputs, and a model of what observations you should see in particular states, you can use particle filters to track your belief state.

Applications that we've seen in class before, and that we'll talk about today, are Robot localization, SLAM, and robot fault diagnosis.

So why should you use particle filters instead of Kalman filters?

Well, the main reason is that for a lot of large or high-dimensional problems, particle filters are tractable whereas Kalman filters are not.

The key idea is that a lot of methods, like Kalman filters, try to make problems more tractable by using a simplified version of your full, complex model. Then they can find an exact solution using that simplified model. But sometimes that exact solution is still computationally expensive to calculate, and sometimes a simplified model just isn't good enough. So then you need something like particle filters, which let you use the full, complex model, but just find an approximate solution instead.

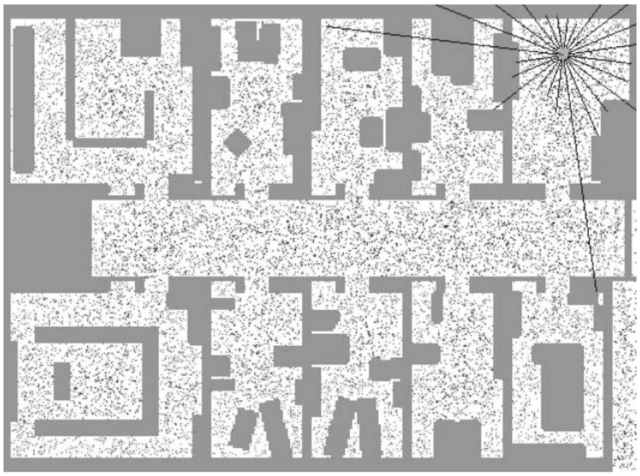
Outline

- Introduction to Particle Filters (Kaijen)
- Particle Filters in SLAM (Henry)
- Particle Filters in Rover Fault Diagnosis (Jason)

Outline

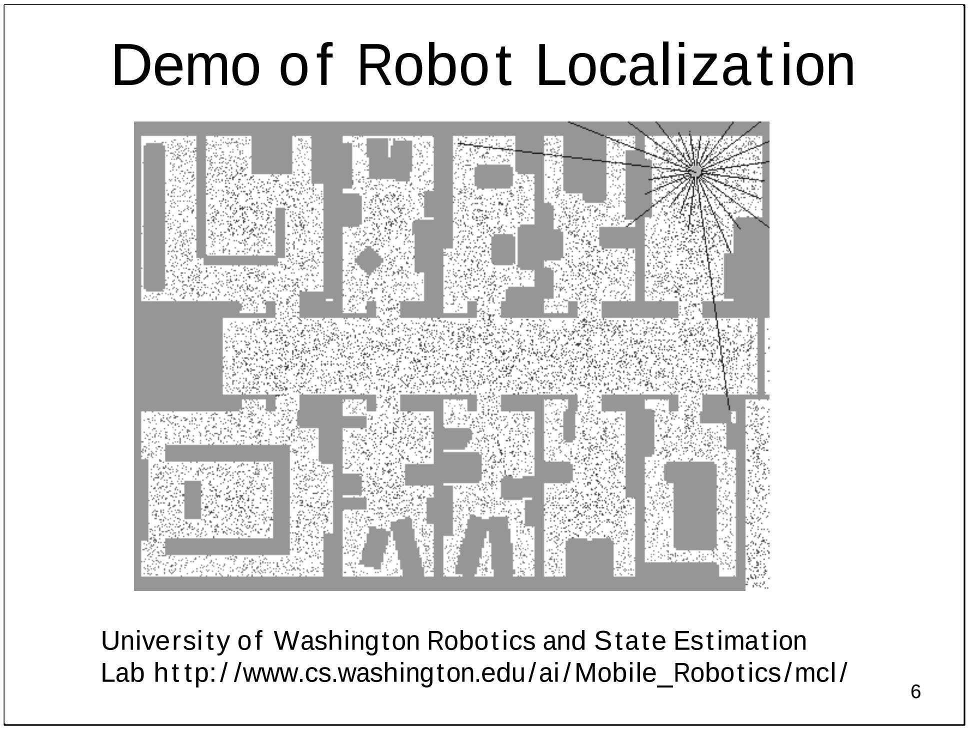
- Introduction to Particle Filters
  - Demo!
  - Formalization of General Problem: Bayes Filters
  - Quick Review of Robot Localization/Problem with Kalman Filters
  - Overview of Particle Filters
  - The Particle Filter Algorithm Step by Step
- Particle Filters in SLAM
- Particle Filters in Rover Fault Diagnosis

Demo of Robot Localization



University of Washington Robotics and State Estimation Lab http://www.cs.washington.edu/ai/Mobile_Robotics/mcl/

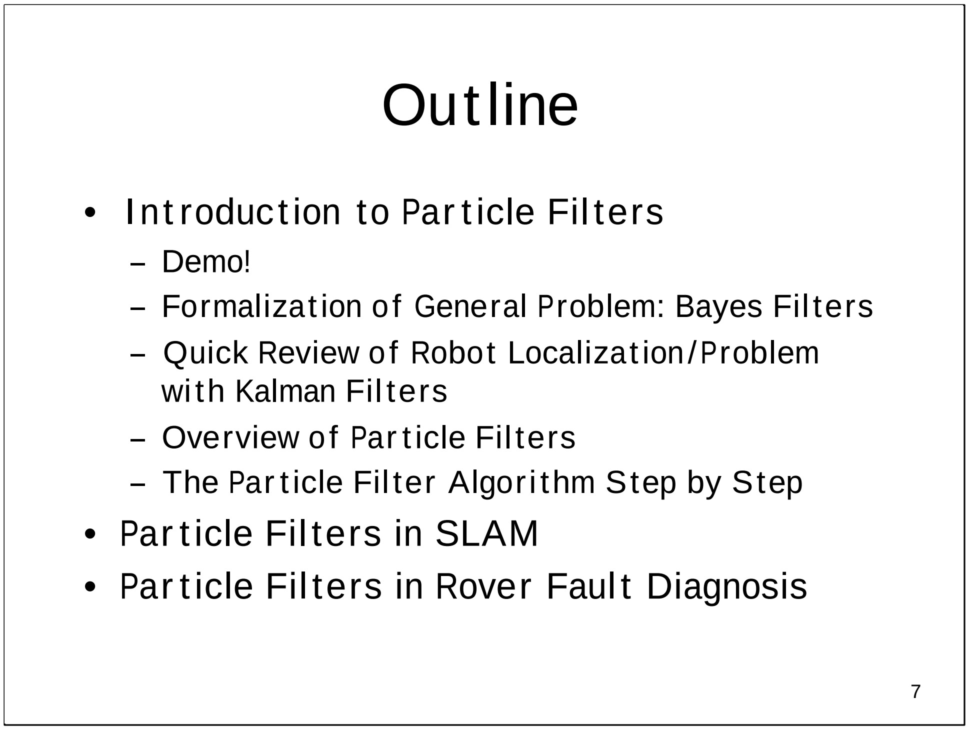
What you see here is a demo from the University of Washington Robotics and State Estimation Lab. This is a frozen panel of the beginning of a robot localization task. The little blue circle is our best guess as to where the robot is now. The little red dots are different hypotheses for where the robot might be—at the beginning of the task, we have no idea where the robot is, so the hypotheses cover the entire space. As we'll see later, each hypothesis is called a 'particle'. The lines extending from the robot are sensor measurements taken by a laser rangefinder. The reason the lines extend well past the walls on the map is because the robot isn't actually in that location. The robot movement comes from a person driving the robot manually; there is no automatic exploration going on.



Demo of Robot Localization

University of Washington Robotics and State Estimation Lab http://www.cs.washington.edu/ai/Mobile_Robotics/mcl/

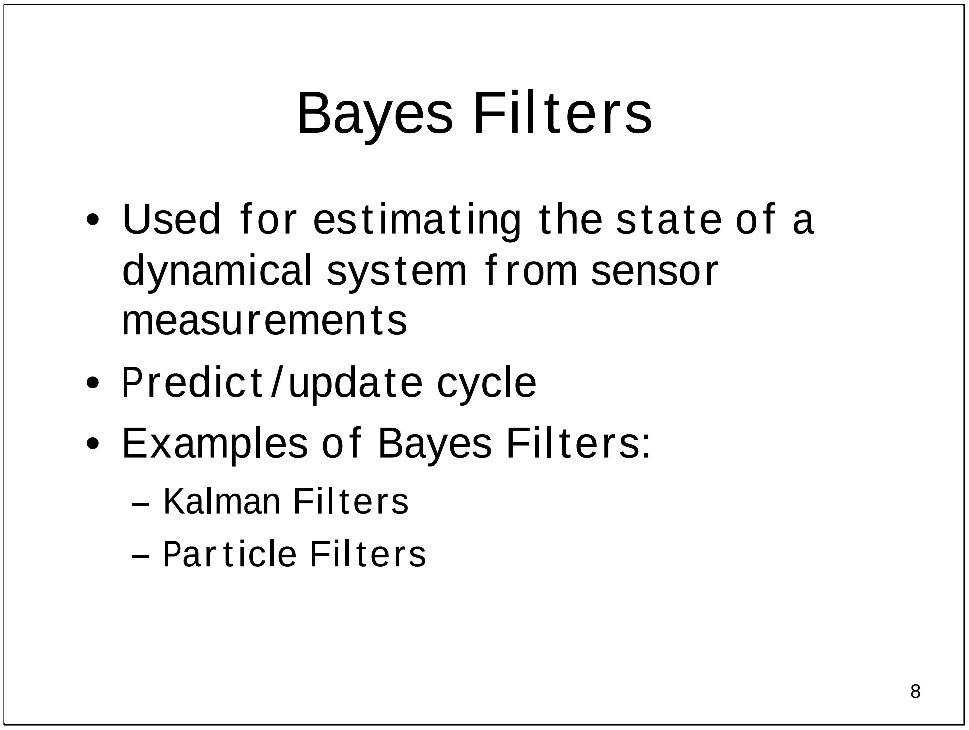
As you watch the animated gif, the best-guess location of the robot will jump around as the most likely hypothesis changes. As the robot moves and takes measurements, it figures out that most of the hypotheses it started with are pretty unlikely, so it gets rid of those. Pretty soon, the number of hypotheses is reduced to a few clouds in the hallway; the robot is actually in the hallway, but there's a lot of symmetry there, so it's not sure exactly where. Then it's down to two hypotheses, and when the robot finally enters a room and looks around, it becomes clear that its current best hypothesis was actually correct.



Outline

- Introduction to Particle Filters
  - Demo!
  - Formalization of General Problem: Bayes Filters
  - Quick Review of Robot Localization/Problem with Kalman Filters
  - Overview of Particle Filters
  - The Particle Filter Algorithm Step by Step
- Particle Filters in SLAM
- Particle Filters in Rover Fault Diagnosis

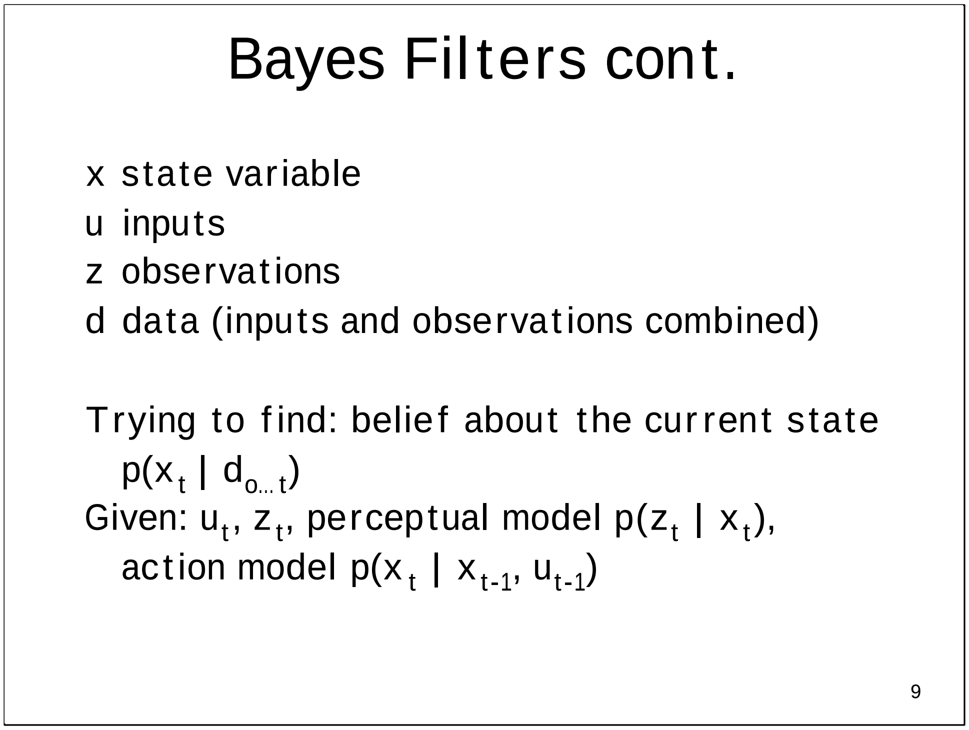
Now I will discuss the formalization of the general problem that both particle filters and Kalman filters solve, which is called Bayes Filtering.



Bayes Filters

- Used for estimating the state of a dynamical system from sensor measurements
- Predict/update cycle
- Examples of Bayes Filters:
  - Kalman Filters
  - Particle Filters

Bayes Filtering is the general term used to discuss the method of using a predict/update cycle to estimate the state of a dynamical system from sensor measurements. As mentioned, two types of Bayes Filters are Kalman filters and particle filters.



Bayes Filters cont.

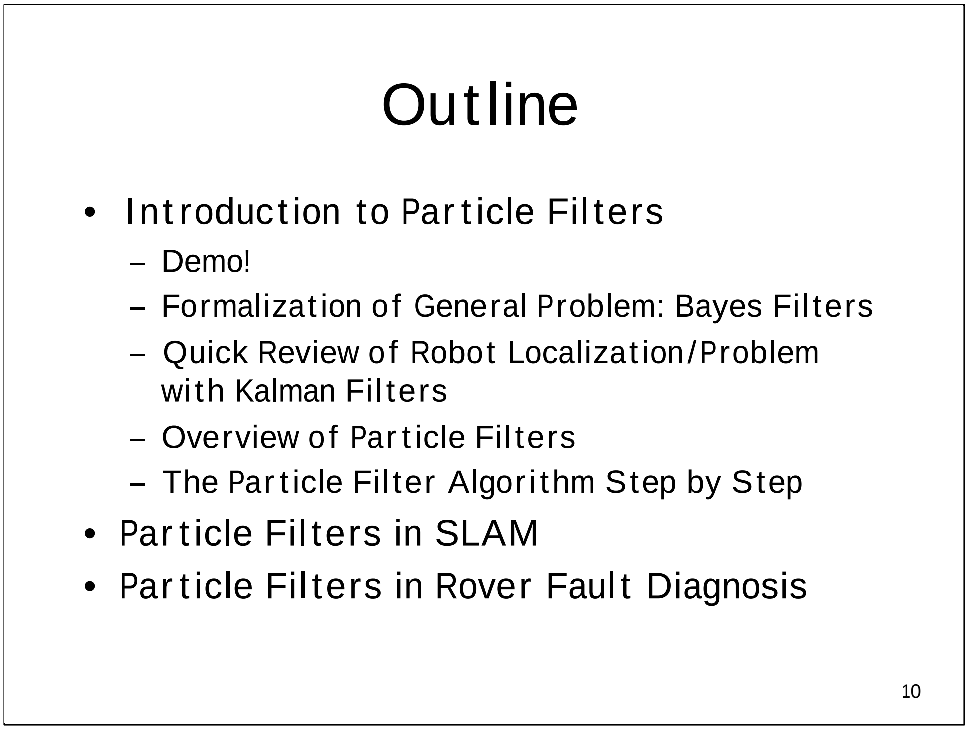
x state variable
u inputs
z observations
d data (inputs and observations combined)

Trying to find: belief about the current state $p(x_t \mid d_{o \ldots t})$

Given: u_t, z_t, perceptual model $p(z_t \mid x_t)$, action model $p(x_t \mid x_{t-1}, u_{t-1})$

Now we introduce the variables we will be using. X is the state variable, and X_t is the state variable at time t. U is the inputs to your system, z is the observations made by the sensors, and d just refers to inputs and observations together. What the Bayes Filter is trying to find at any point in time is the belief about the current state, which is the probability of x_t given all the data we've seen so far.

What we are given is the inputs, the observations, the perceptual model, which is the probability that you'll see a particular observation given that you're in some state at time t, and the action model, which is the probability that you'll end up in state x_t at time t, assuming that you started in state x_{t-1} at time t-1, and input u_{t-1} to your system.

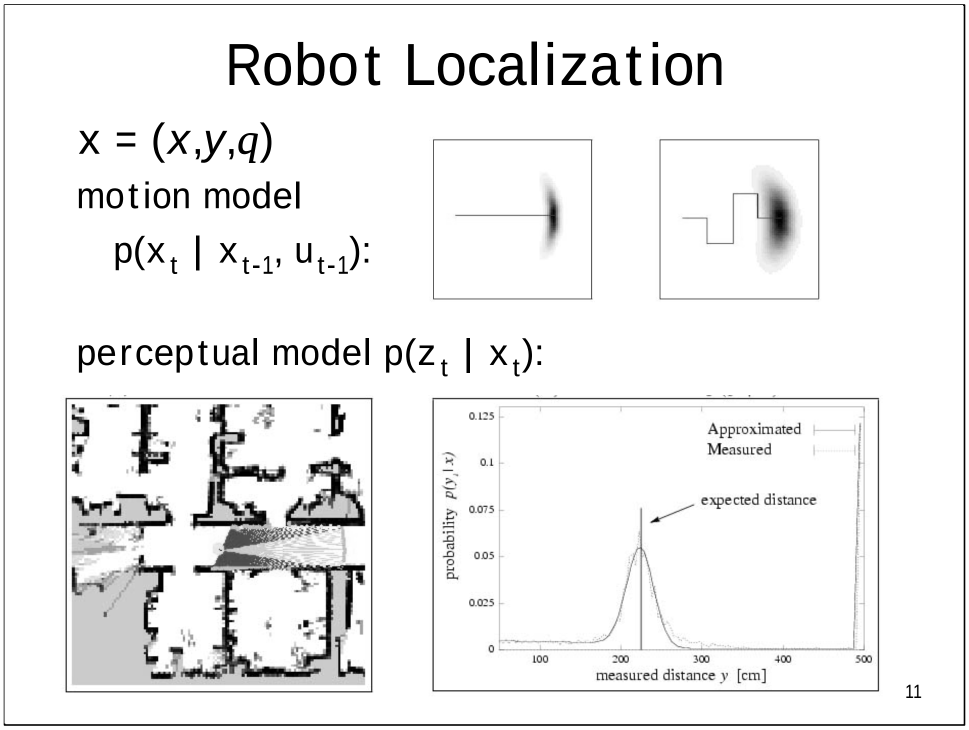


Outline

- Introduction to Particle Filters
  - Demo!
  - Formalization of General Problem: Bayes Filters
  - Quick Review of Robot Localization/Problem with Kalman Filters
  - Overview of Particle Filters
  - The Particle Filter Algorithm Step by Step
- Particle Filters in SLAM
- Particle Filters in Rover Fault Diagnosis

10

Now I will give a quick review of robot localization and show what the problem is with doing localization with Kalman filters.



Robot Localization

x = (x,y,q)

motion model $p(x_t \mid x_{t-1}, u_{t-1})$:

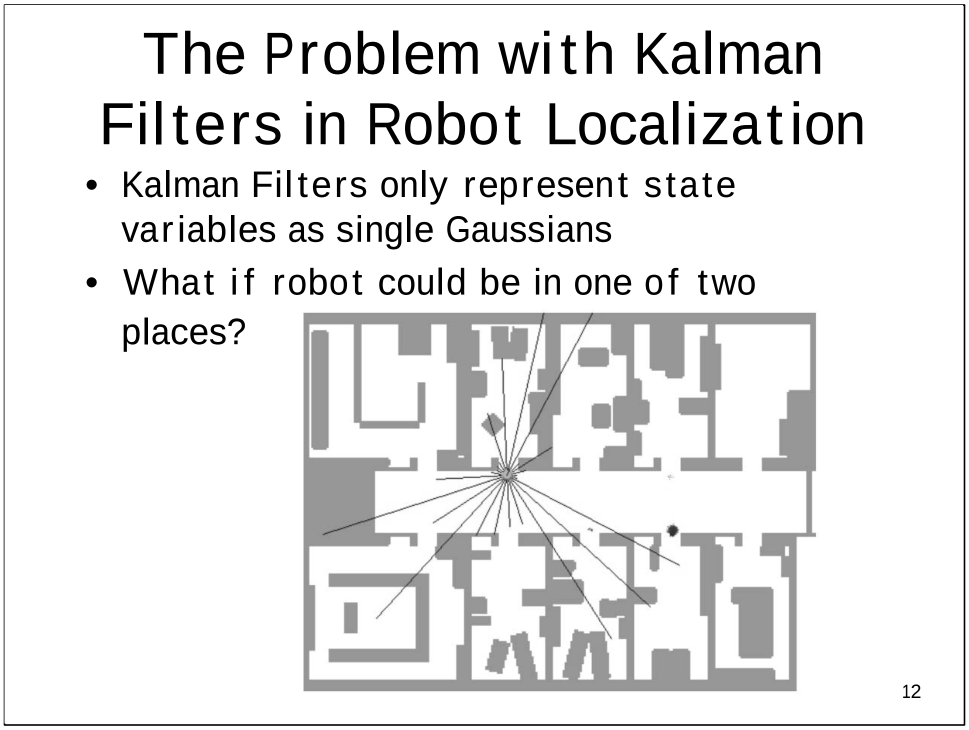
perceptual model $p(z_t \mid x_t)$:

0.125
0.1
0.075
0.05
0.025
0
probability p(y | x)
Approximated
Measured
expected distance
100
200
300
400
500
measured distance y [cm]

So here's the robot localization problem. You're trying to track the state x, which is made up of the (x,y) position of the robot as well as its orientation, theta.

You have a motion model for the robot, which looks like the two figures in the top right. If you start at the left end of the straight red line, pointed to the right, and tell your robot to move forward some distance, you expect it to end up somewhere in that cloud due to wheel slippage and the like. Darker regions have higher probability. If you start at the left end of the wiggly red line, your robot will have even more wheel slippage while turning (and it's going a farther distance), and so the resulting position uncertainty cloud is larger.

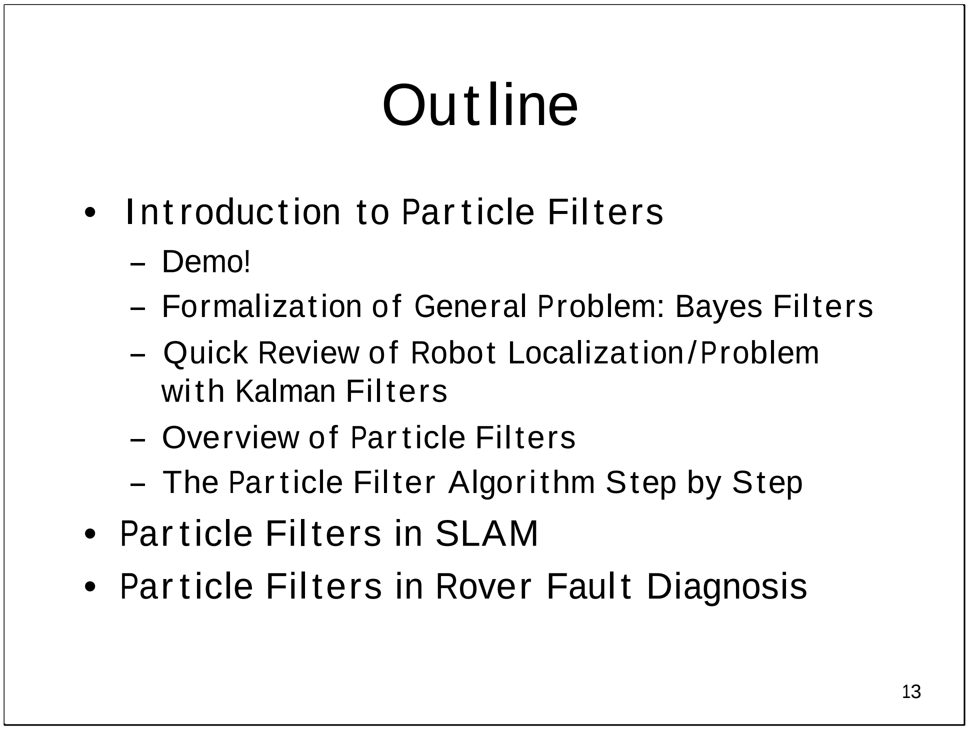
You also have a perceptual model for your robot, which is the probability that you'll see certain observations when you're in a particular state x_t. On the bottom left is a picture of a robot in a map getting measurements from its laser rangefinders. Given a position and a map, you can use ray-tracing to get expected measurements for each rangefinder angle. Then you can look at a graph like the one on the bottom right, which is the result of characterizing your sensor. As you can see, for a particular expected distance, your sensor will give you a value near that distance with some reasonable probability. But rangefinders often miss objects and report seeing something at the maximum distance, so with some probability you expect the sensor to give you the max distance instead. So given an actual measurement and an expected distance, you can find the probability of getting that measurement using the graph.



The Problem with Kalman Filters in Robot Localization

- Kalman Filters only represent state variables as single Gaussians
- What if robot could be in one of two places?

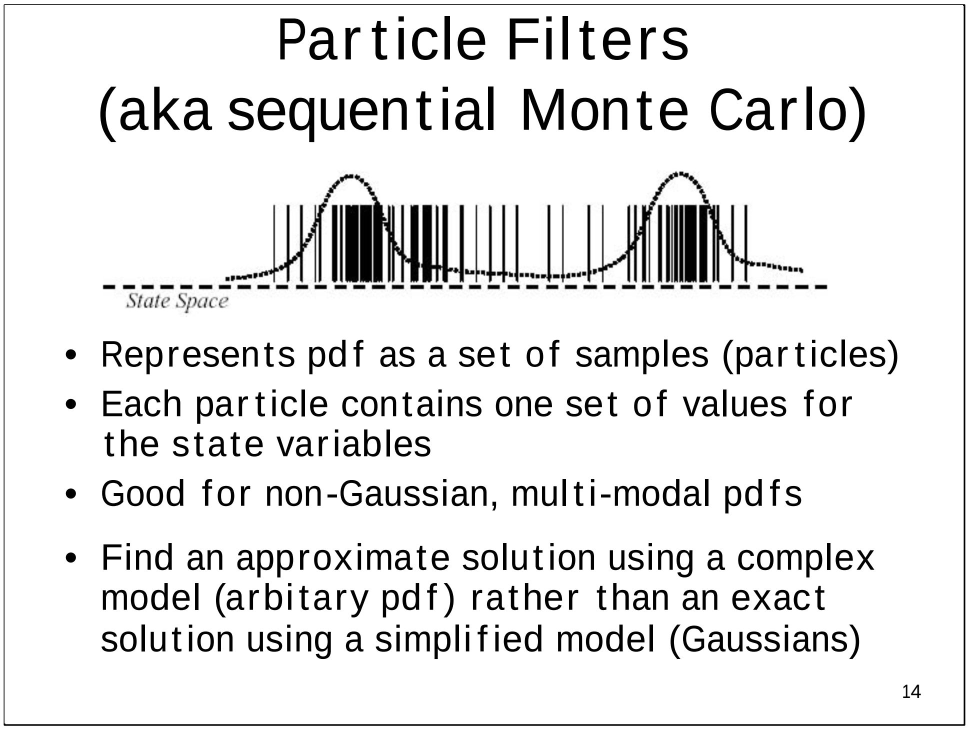
The problem with Kalman filters is that they represent the state of the system using only single Gaussians. As you can see in the diagram excerpted from the demo just showed, sometimes it is necessary to have multimodal hypotheses about where the robot might be. If you can only choose one of the two possibilities (the most likely one), and you choose incorrectly, then it is extremely difficult to recover from your mistake. Particle filters, on the other hand, can keep track of as many hypotheses as there are particles, so if new information shows up that causes you to shift your best hypothesis completely, it is easy to do.



Outline

- Introduction to Particle Filters
    - Demo!
    - Formalization of General Problem: Bayes Filters
    - Quick Review of Robot Localization/Problem with Kalman Filters
    - Overview of Particle Filters
    - The Particle Filter Algorithm Step by Step
- Particle Filters in SLAM
- Particle Filters in Rover Fault Diagnosis

Now I will give an overview of the basic premise of particle filters.

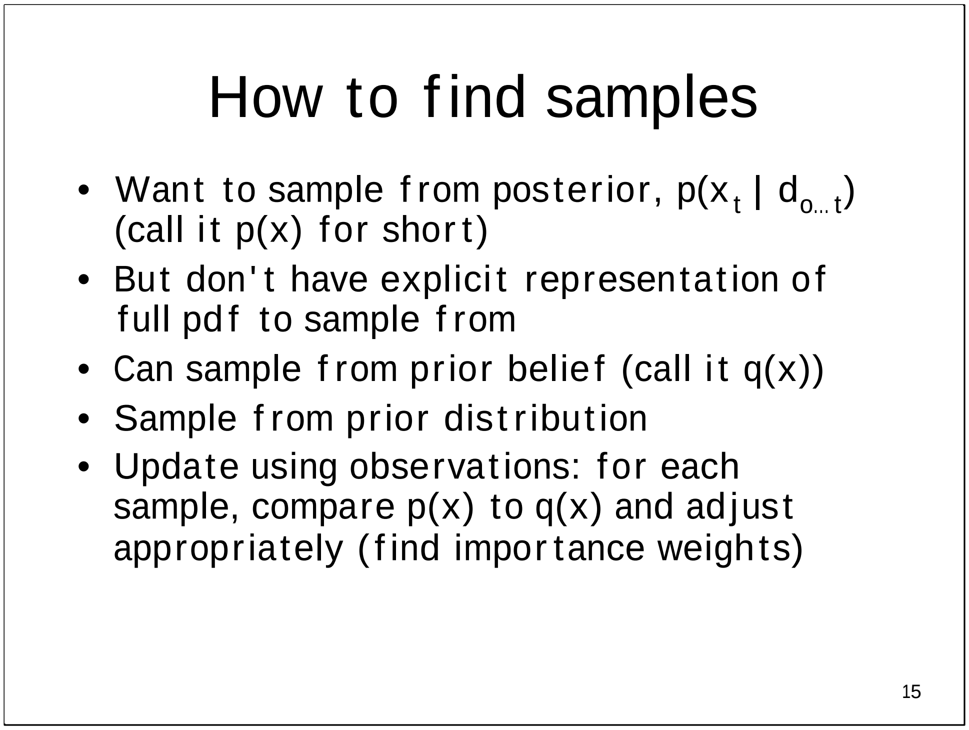


Particle Filters (aka sequential Monte Carlo)

State Space

- Represents pdf as a set of samples (particles)
- Each particle contains one set of values for the state variables
- Good for non-Gaussian, multi-modal pdfs
- Find an approximate solution using a complex model (arbitary pdf) rather than an exact solution using a simplified model (Gaussians)

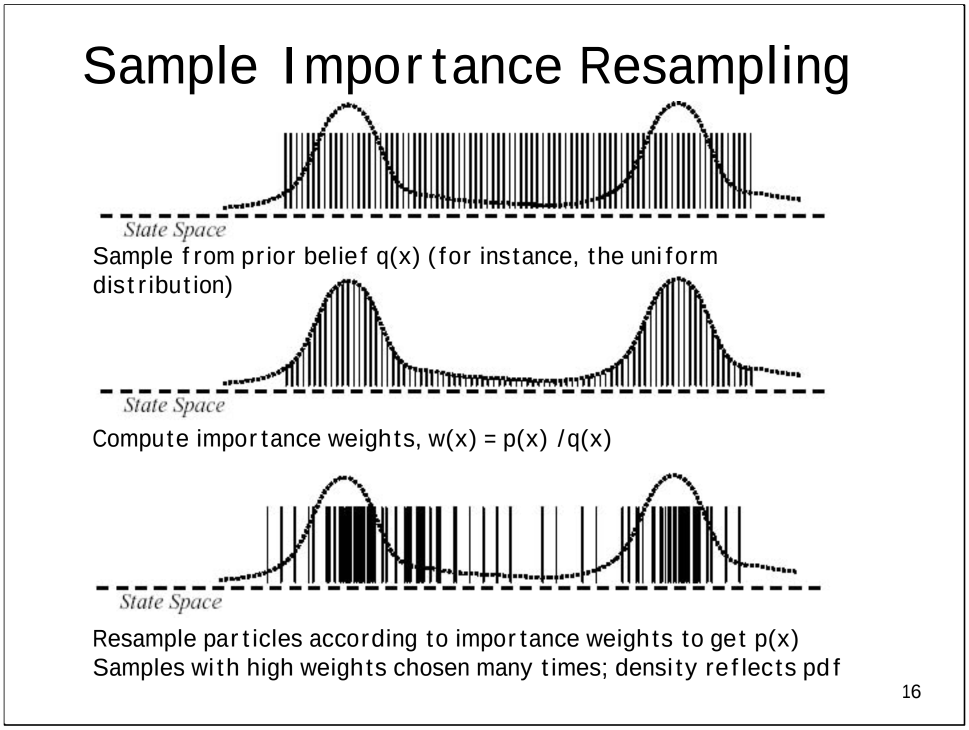
The basic idea of particle filters is that any pdf can be represented as a set of samples (particles). If your pdf looks like the two-humped line in the figure, you can represent that just by drawing a whole lot of samples from it, so that the density of your samples in one area of the state space represents the probability of that region. Each particle has one set of values for the state variables. This method can represent any arbitrary distribution, making it good for non-Gaussian, multi-modal pdfs. Again, the key idea is that you find an approximate representation of a complex model (any arbitrary pdf) rather than an exact representation of a simplified mode (Gaussians).



How to find samples

- Want to sample from posterior, $p(x_t \mid d_{o\ldots t})$ (call it p(x) for short)
- But don't have explicit representation of full pdf to sample from
- Can sample from prior belief (call it q(x))
- Sample from prior distribution
- Update using observations: for each sample, compare p(x) to q(x) and adjust appropriately (find importance weights)

So what you actually want samples of is your posterior, which we will call p(x) for short. But how do you sample from your posterior? You don't have an explicit representation of your posterior to draw points from. But you do know how to sample from your prior belief, because you had some belief from the last time step that you know how to update with your motion model. Let's call the prior belief q(x). And you do know how to find, for any one x, what the posterior probability is, based on your prior belief and your observations. So, sample from q(x), and then for each sample that you made, update it using what we will call an 'importance weight', based on the observations made.



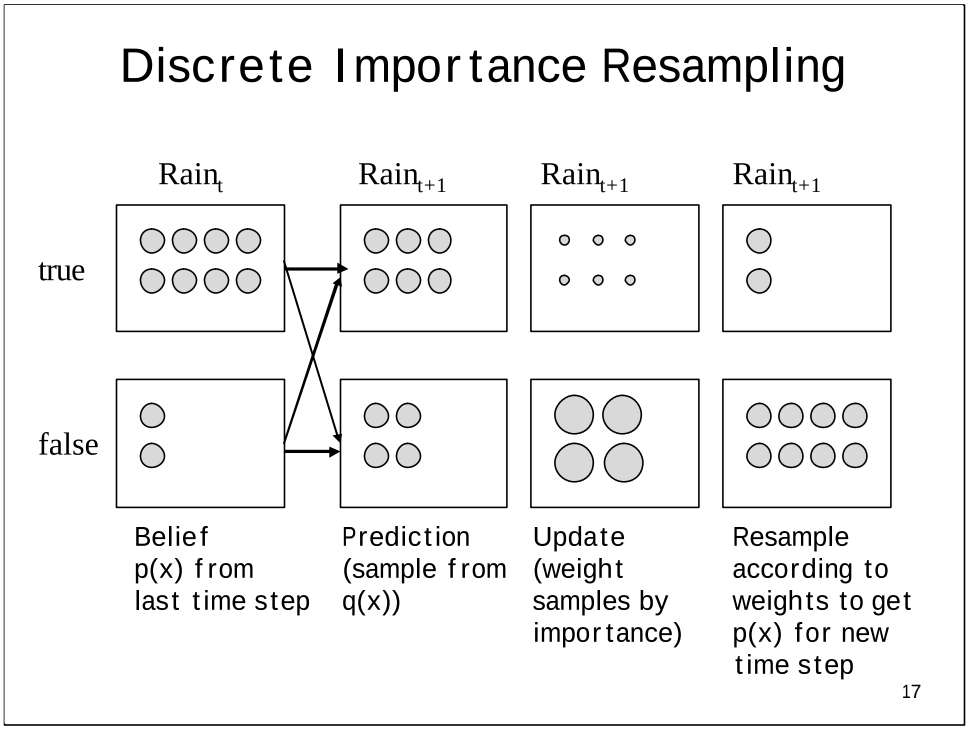
Sample Importance Resampling

Sample from prior belief q(x) (for instance, the uniform distribution)

Compute importance weights, w(x) = p(x) /q(x)

Resample particles according to importance weights to get p(x)
Samples with high weights chosen many times; density reflects pdf

Here is a graphical visualization of the importance resampling process. Let's say the posterior you're trying to represent, as before, is the two-humped dotted line. Even if you have no information to start, and your prior is just the uniform distribution, you can still recover the properly sampled pdf of your posterior p(x). First you sample from your prior (the uniform distribution). For each of those samples, you can find the value of the posterior p(x). So for each sample, you assign that sample a weight, w(x), equal to p(x)/q(x). At this point, when the particles are weighted, you can use your highest-weighted (highest-probability) sample as your best-guess state, or you can use the weighted sum of particles to get a mean-equivalent, or you can use the average of particles within some distance from your best particle for a more intelligent best-guess. To represent the pdf properly with samples, though, we want the density of the particles in any segment of the state space to be proportional to the probability of that segment. As you can see in the middle panel, the particles are still spaced evenly from uniform sampling. So in order to adjust the densities properly, we resample the particles. That means we want to keep the total number of particles the same, while increasing the number of particles in the high-probability regions and decreasing the number of particles in low-probability regions. So we draw particles (with replacement) from the set of weighted particles according to their importance weights (probabilities). High-weighted particles can be chosen a lot of times, whereas low-weighted particles are likely not to be chosen at all. The result looks like the third figure, in which the particles go back to being unweighted, and the density of the particles properly represents the pdf.



Discrete Importance Resampling

Belief p(x) from last time step

Prediction (sample from q(x))

Update (weight samples by importance)

Resample according to weights to get p(x) for new time step

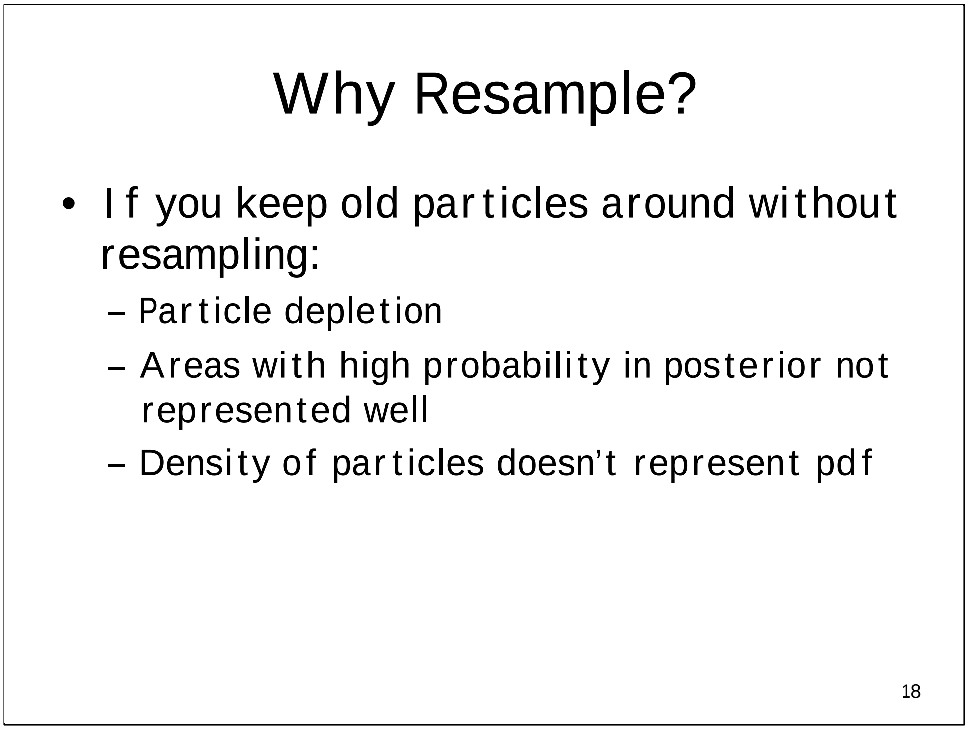
Another way to visualize the importance resampling process is to look at a discrete example.

Let’s say you have a dynamic Bayes’ net with two states: Rain = true or Rain = false. You’re trying to figure out whether or not it’s raining, but you can’t see outside because you’re in an office with no windows. But let’s say that every hour, your boss stops by, and either he brings an umbrella, or he doesn’t. If he brings an umbrella, it’s likely raining, but maybe not, since some people bring umbrellas for no reason. Likewise, if he doesn’t bring an umbrella, it’s probably not raining, but it might be. So you have some model about the probability that it’s raining, given that you think it was raining an hour ago and your boss brings an umbrella, or doesn’t bring an umbrella, and so on. And you also have some model about the transition probabilities of rain/not-rain, saying that if it was raining an hour ago, it might have stopped with some probability, and so on.

So we start, in the first column of boxes, $Rain_t$, with some belief p(x) from the last time step. There are 8 particles in Rain=true and only 2 in Rain=false, meaning that p(rain=true) is 8/(2+8) = 4/5, and p(rain=false) is 2/(2+8) = 1/5.

Next, we make a prediction about what the state will be in the next time step based on our transition model, before looking at any observations. This is our prior belief, q(x), and letting particles transition with some probability to each of the two states gives us the new sample set from q(x). Now we have 6 particles in Rain=true, and 4 particles in Rain=false.

Then let’s say the boss comes in, and he’s not carrying an umbrella. Now, we can find the probability of each particle based on our observation, according to our perceptual model. So the Rain=true particles have low probabilities, so we

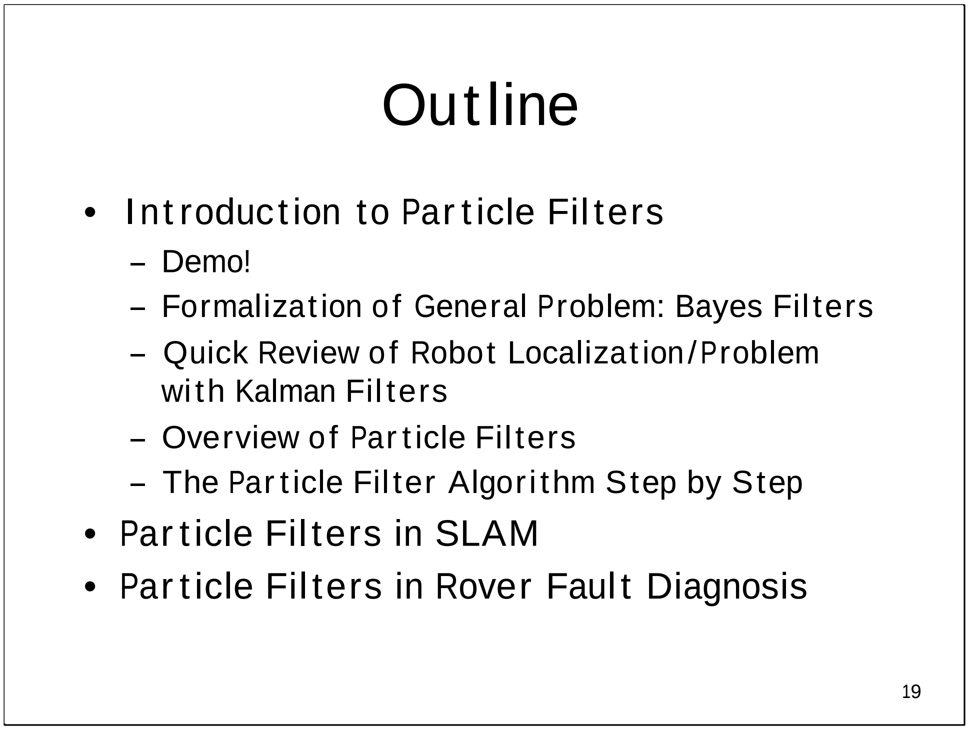


Why Resample?

- If you keep old particles around without resampling:
  - Particle depletion
  - Areas with high probability in posterior not represented well
  - Density of particles doesn’t represent pdf

So, just to make clear why it is necessary to resample the particles:

If you just keep your old particles around forever without resampling them, what happens is that your particles drift around according to your motion model (transition probabilities for the next time step), but other than their weights, they are unaffected by your observations. Highly unlikely particles will be kept around and transitioned to more unlikely states, and you might only have say, one particle in the area of high probability of your posterior. So what you end up with is one particle with a way higher likelihood than any of the other particles, and a whole lot of particles with almost-nil probability. This is what we call ‘particle depletion’, because you in effect have only one particle. And one particle doesn’t represent a pdf very well. If you don’t have a lot of particles in the areas of your pdf with high probability, you won’t represent the pdf very well. The density of your particles should be high in high-probability areas, and low in low-probability areas. And so you have to resample the particles, so that they continue to represent the pdf accurately and keep track of many high-probability hypotheses, instead of tracking lots of useless, low-probability hypotheses.



Outline

- Introduction to Particle Filters
  - Demo!
  - Formalization of General Problem: Bayes Filters
  - Quick Review of Robot Localization/Problem with Kalman Filters
  - Overview of Particle Filters
  - The Particle Filter Algorithm Step by Step
- Particle Filters in SLAM
- Particle Filters in Rover Fault Diagnosis

Now we will show the particle filter algorithm step by step, using the example of robot localization.

The Algorithm

- To start: Sample from initial pdf, $p(x_0)$
  - For localization, no idea where robot is -> throw particles everywhere!
- For each time step, loop with three phases:
  1) Prediction
  2) Update
  3) Resample

To start the algorithm, we need the initial belief state, $p(x_0)$. This is just our initial guess of the pdf. For robot localization, if we have no idea, we can just scatter particles all over the map, as in the demo shown earlier.

For each time step, we then loop with three phases: prediction, update, and resample. These will be explained in more detail in the next few slides.

Calculation of Belief for Robot Localization:

$$\underbrace{\overset{\text{Pdf of current state}}{p(x_t \mid d_{o\ldots t})}}_{\text{Resample:}} = \underbrace{\underset{\text{Normalization constant}}{\eta}\ \overset{\text{Perception model}}{p(z_t \mid x_t)}}_{\text{Update:}}\ \underbrace{\int \underset{\text{Motion model}}{p(x_t \mid u_{t-1}, x_{t-1})}\ \overset{\text{Pdf from last time step}}{p(x_{t-1} \mid d_{o\ldots t-1})}\ dx_{t-1}}_{\text{Prediction:}}$$

p(x) – the posterior probability, our belief about the state variables

w(x) – the importance weights

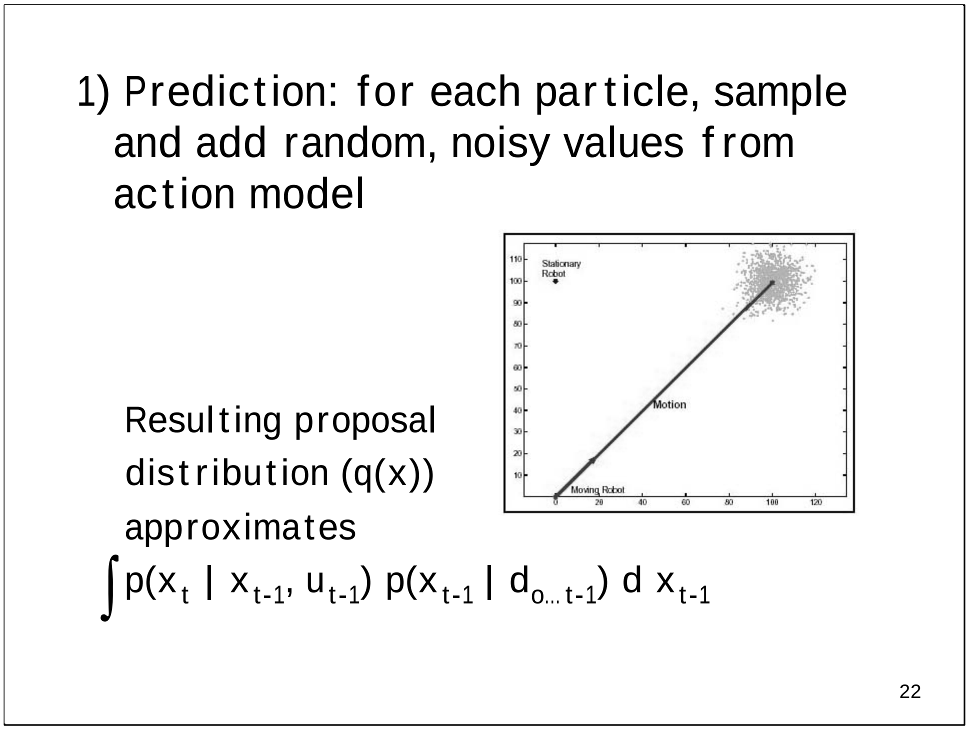
q(x) – the prior probability

The equation on this slide shows the formalization of the steps taken in the particle filter algorithm. It is derived from applying Bayes rule to the posteior, and then using the Markov assumption. While executing the particle filter algorithm, we are calculating this equation from right to left.

First we start with the pdf from the last time step, and then we multiply it by the motion model in the ‘prediction’ step to get q(x), the prior probability. The integral is there only to say that we can end up in the same state in time t from more than one state in time t-1, and thus we have to integrate over the states from time t-1. But we do not have to worry about this detail in the particle filter algorithm, since the particle representation takes care of the integral.

Next, we find the importance weights w(x) using the perception model and normalize them so that they sum to 1.

q(x) times w(x) = p(x), the posterior probability, which we use resampling based on the importance weights to achieve.



1) Prediction: for each particle, sample and add random, noisy values from action model

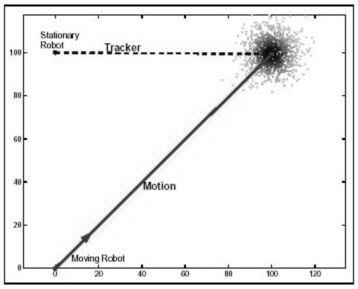
Stationary Robot
Motion
Moving Robot
0
20
40
60
80
100
120
10
20
30
40
50
60
70
80
90
100
110

Resulting proposal distribution (q(x)) approximates

$$\int p(x_t \mid x_{t-1}, u_{t-1})\, p(x_{t-1} \mid d_{o\ldots t-1})\, d\, x_{t-1}$$

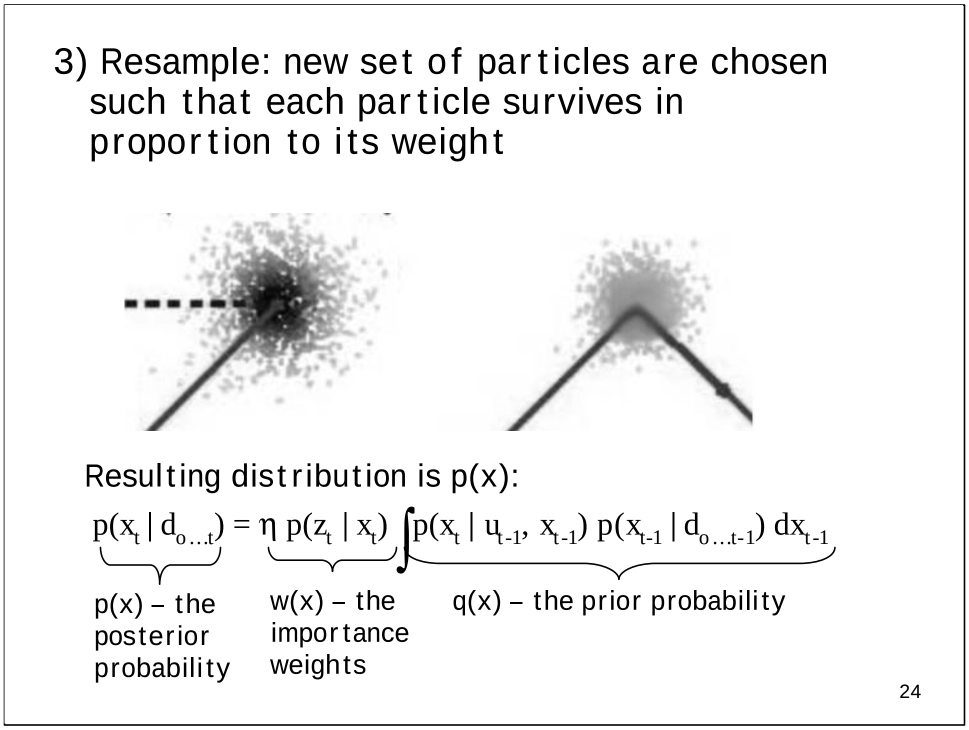
In the prediction step, we take each particle and add a random sample from the motion model. In the figure, the robot starts from the lower left, and moves to the upper right. The resulting position, from the motion model, will be somewhere in that cloud of particles. The resulting distribution of particles approximates the prior distribution.

2) Update: each particle’s weight is the likelihood of getting the sensor readings from that particle’s hypothesis



The weight associated with each particle is $w(x) = p(x)/q(x) = p(z_t \mid x_t)$, normalized so that all the weights sum to 1

During the update step, we take the sensor measurements and assign each particle a weight that is equal to the probability of observing the sensor measurements from that particle’s state. Those weights are then normalized so that they sum to 1. In the figure, the robot has observed the ‘stationary robot’ landmark at the top left, and based on that measurement, it has assigned weights to each particle. Darker particles have higher weights.



3) Resample: new set of particles are chosen such that each particle survives in proportion to its weight

Resulting distribution is p(x):

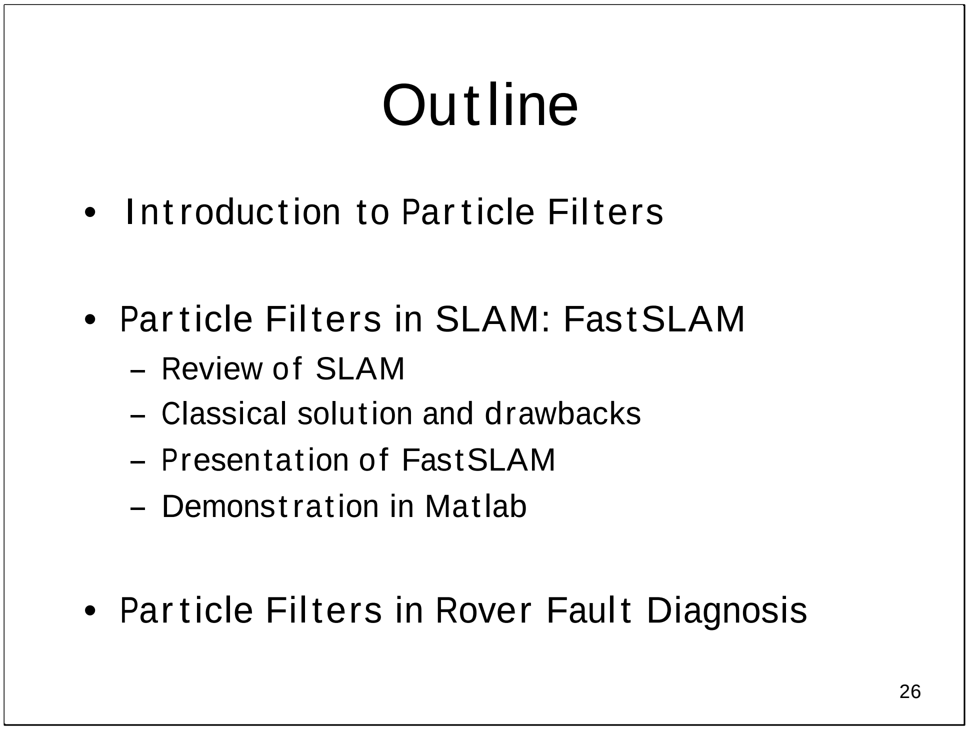
$$\underbrace{p(x_t \mid d_{o\ldots t})}_{\text{p(x) – the posterior probability}} = \eta\, \underbrace{p(z_t \mid x_t)}_{\text{w(x) – the importance weights}} \int \underbrace{p(x_t \mid u_{t-1}, x_{t-1})\, p(x_{t-1} \mid d_{o\ldots t-1})\, dx_{t-1}}_{\text{q(x) – the prior probability}}$$

24

Finally, in the resample step, a new set of particles is chosen so that each particle survives in proportion to its weight. As you can see in the picture, the weighted cloud of particles turns into the somewhat more condensed and smoother cloud of unweighted particles on the right. Highly unlikely particles at the fringe are not chosen, and the highly likely particles near the center of the cloud are replicated so that the high-probability region has a high density, correctly representing p(x), our posterior distribution.

Intro Summary

- Particle filters represent pdfs using sample densities
- New belief can be found using the prediction-update-resample loop
- Particle filters can be better than Kalman filters for robot localization



Outline

- Introduction to Particle Filters
- Particle Filters in SLAM: FastSLAM
  - Review of SLAM
  - Classical solution and drawbacks
  - Presentation of FastSLAM
  - Demonstration in Matlab
- Particle Filters in Rover Fault Diagnosis

In this part, I will present you a particular application of particle filtering to the SLAM problem: Fast SLAM.

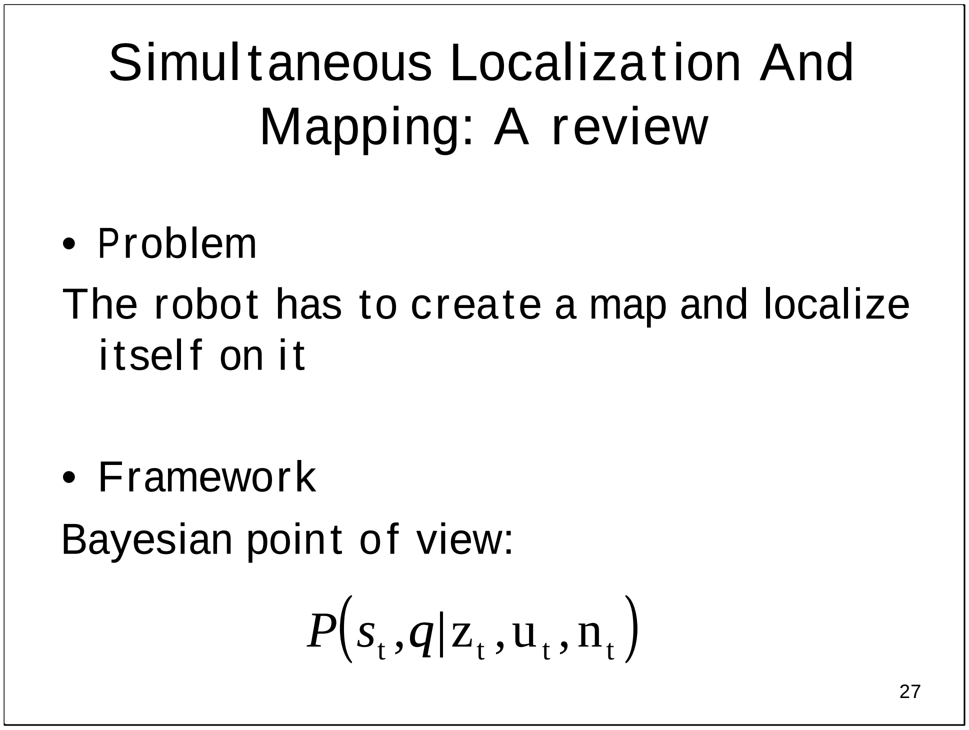
I will show you how this algorithm applies the principles explained by Kaijen in her part, and how they differ somehow with these general principles.

I will first review briefly the SLAM problem, since we have seen it in this class several weeks ago.

I will then present you the classical solution to the SLAM problem and its main drawbacks.

Then, I will explain you how FastSLAM works.

Finally, I will end up with a short demo I wrote in Matlab.



Simultaneous Localization And Mapping: A review

- Problem

The robot has to create a map and localize itself on it

- Framework

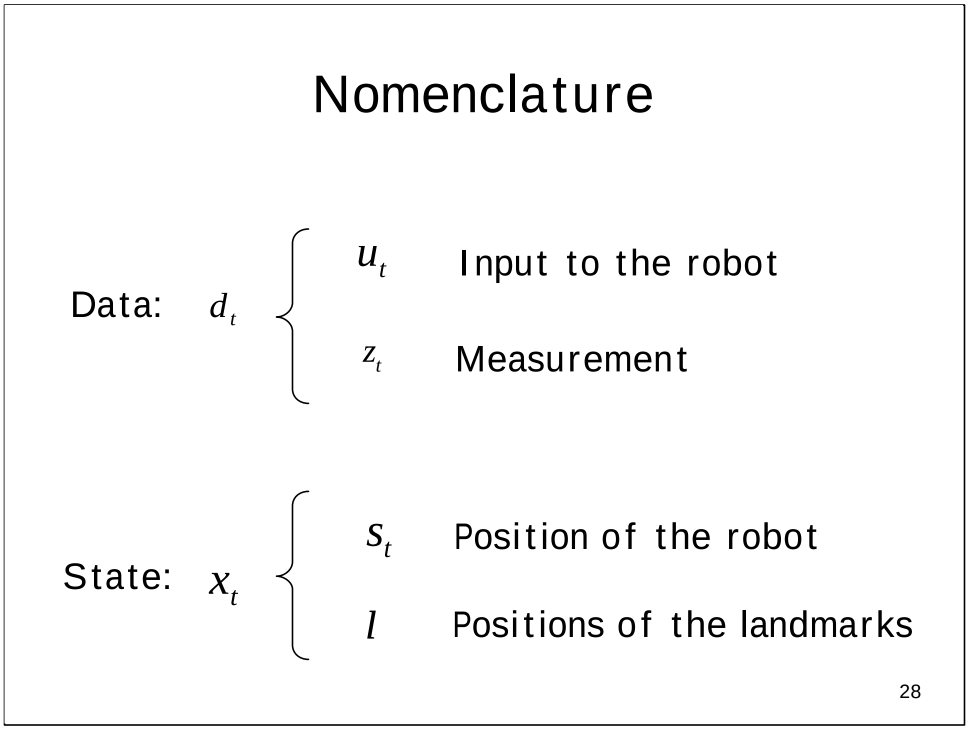
Bayesian point of view:

$$P\left(s_t, q | z_t, u_t, n_t\right)$$

The SLAM problem consists of creating a map and localizing on it.

The framework used for it is the bayesian point of view, as described by Kaijen.

In the next slide, I will explain the nomenclature for FastSLAM.



Nomenclature

Data: d_t

- u_t Input to the robot
- z_t Measurement

State: x_t

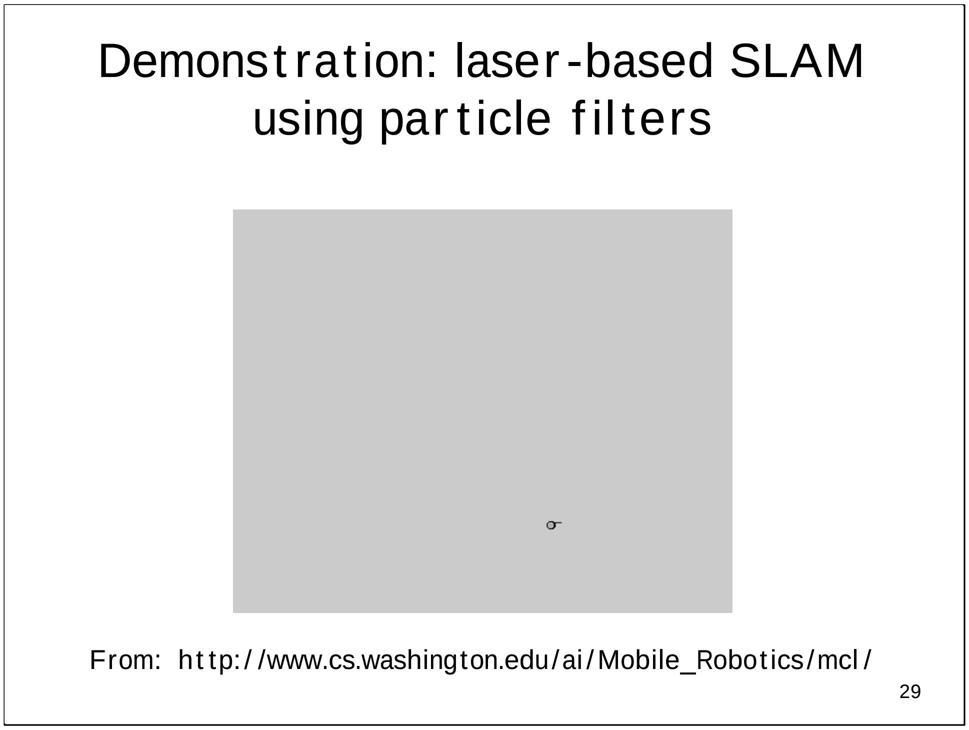
- s_t Position of the robot
- l Positions of the landmarks

(The top part appears first)

As Kaijen mentionned, a typical particle filtering algorithm takes some data, and outputs an estimate on the state of the system.

In FastSLAM, the data are the same as in the localization problem described by Kaijen: they consist of both the input to the robot (go left, right, up, for instance), and the measurements at each time step (there is a landmark at 3 feet in direction pi/4 with respect to the robot’s path).

Now, the state is slightly different from the localization problem. Indeed, here, we want both to estimate the position of the robot and the positions of all the landmarks surrounding it. As a result, the state consists of both of them. Note that the landmarks are considered motionless in this case (no underscore ‘t’).



Demonstration: laser-based SLAM using particle filters

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

Upfront, to show you guys how FastSLAM can perform, I show you a demo I found on the web.

(I start the demo, and stop it after ten seconds to explain).

On this demo, you see a robot moving in the corridor of a building, while mapping the walls, and localizing itself.

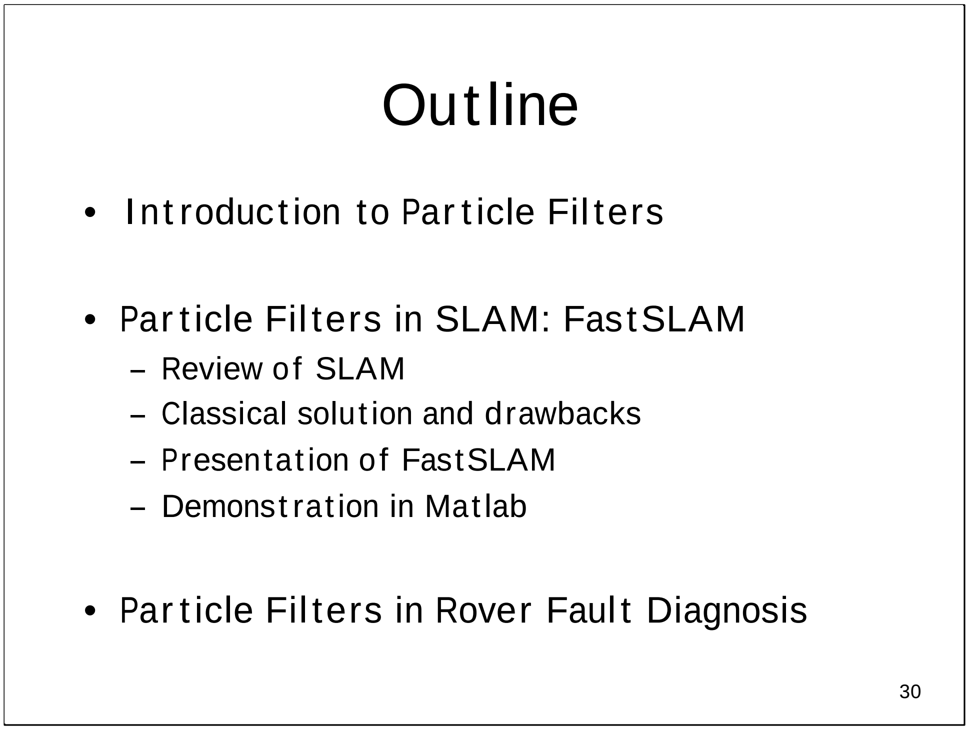
The green dot represents the actual position of the robot. The red lines are the different guesses on the robot's path (particles). The red dot, which we cannot see very well, is the best estimate on the robot's position. Finally, we see the map of the building's walls, as the robot moves along the corridors.

(starting demo again)

As the robot moves, you see the lines diverging, because the uncertainty on the robot's position is increasing as time evolves.

When the loop is closed, from the extra-information that the robot was actually at the same place a while ago, it can reduce the uncertainty (we see the lines gathering).

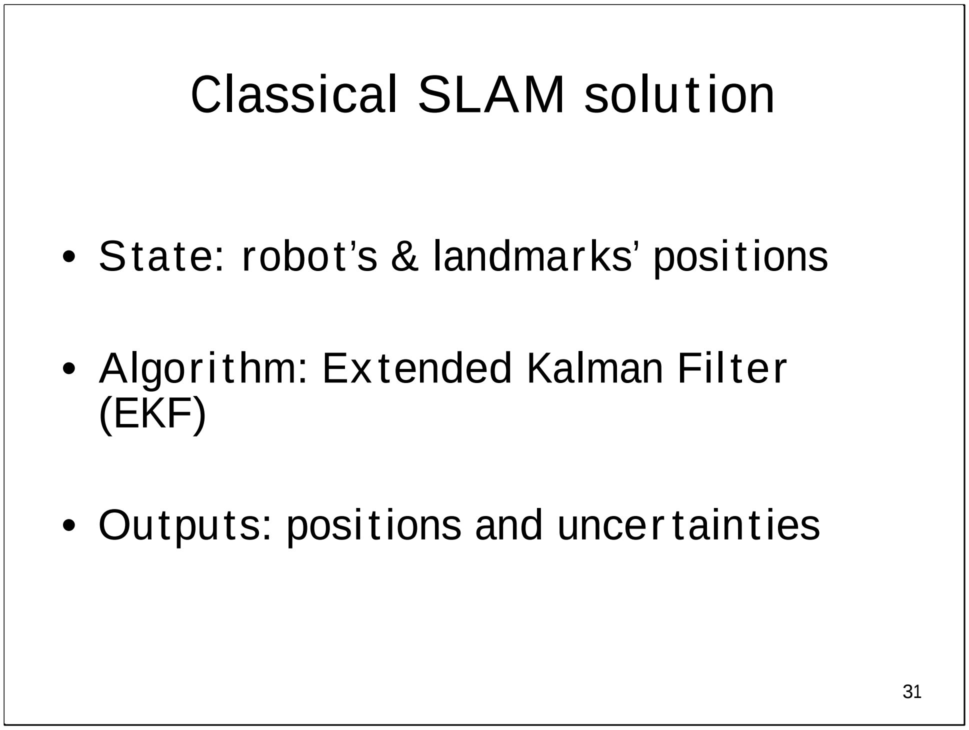
Finally, we see how robustly the map is improved over time, as the robot moves. This is a feature of particle filtering: it is very robust.



Outline

- Introduction to Particle Filters
- Particle Filters in SLAM: FastSLAM
  - Review of SLAM
  - Classical solution and drawbacks
  - Presentation of FastSLAM
  - Demonstration in Matlab
- Particle Filters in Rover Fault Diagnosis

(Introducing the second part of my talk: the classical solution to the SLAM problem.)



Classical SLAM solution

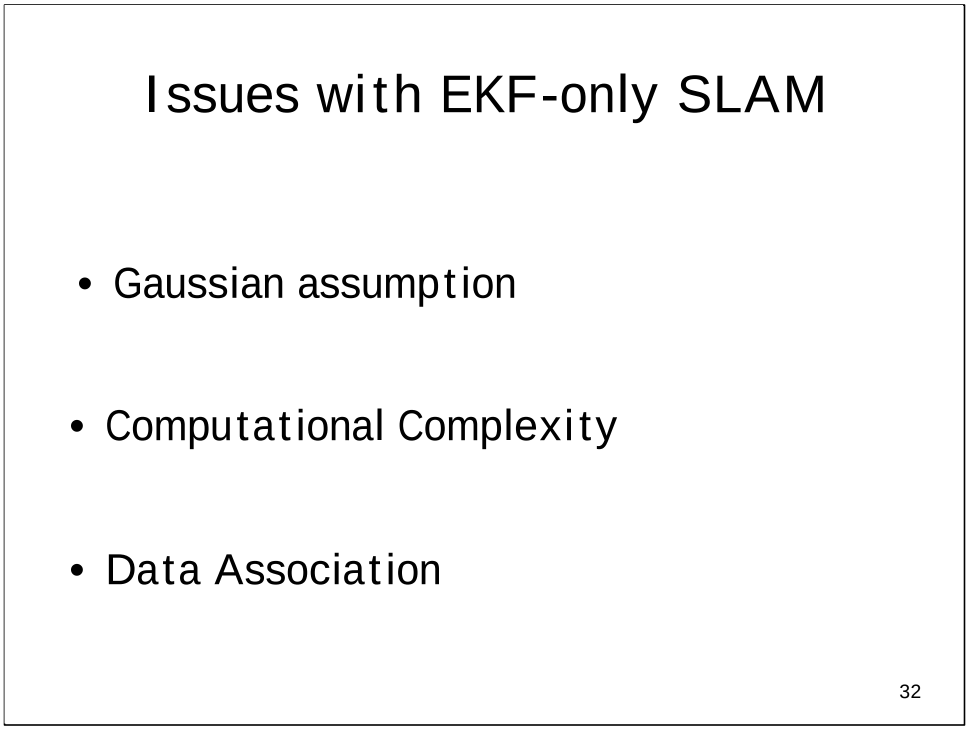
- State: robot’s & landmarks’ positions
- Algorithm: Extended Kalman Filter (EKF)
- Outputs: positions and uncertainties

The classical SLAM solution uses:

A *huge* state, consisting of the robot’s position, and all the landmarks’ positions, in a single huge vector

The algorithm used is the extended kalman filter. We have already seen in this class the kalman filter. The extended kalman filter is the same algorithm , but extended to handle also non-linear models.

The output of the algorithm is the estimate on this huge state, together with the uncertainty on it.



Issues with EKF-only SLAM

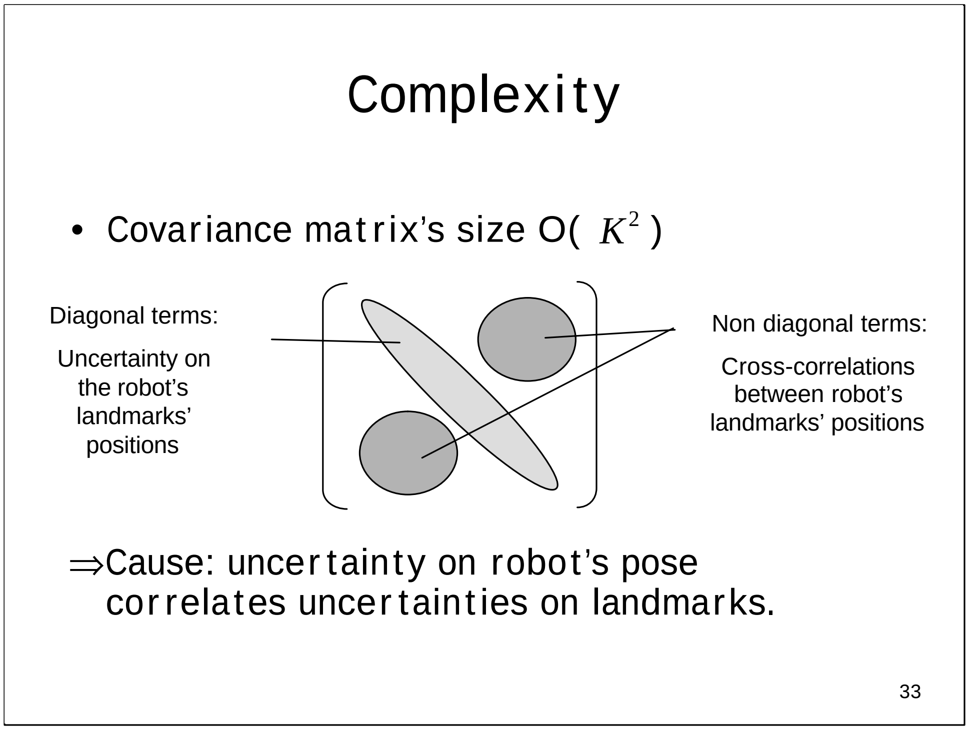
- Gaussian assumption
- Computational Complexity
- Data Association

What are the main issues with this classical way of solving the SLAM problem ?

First, as Kaijen mentionned, the Extended Kalman Filter assumes that the probability density functions are gaussian, which may not be accurate. Since Kaijen mentionned it, I won’t go any further on this point.

The second issue is the computational complexity of this approach (I will detail this point in the next slide)

The third issue is the so-called ‘data association problem’, which I will also explain.



Complexity

- Covariance matrix's size O(K^2)

Diagonal terms:

Uncertainty on the robot's landmarks' positions

Non diagonal terms:

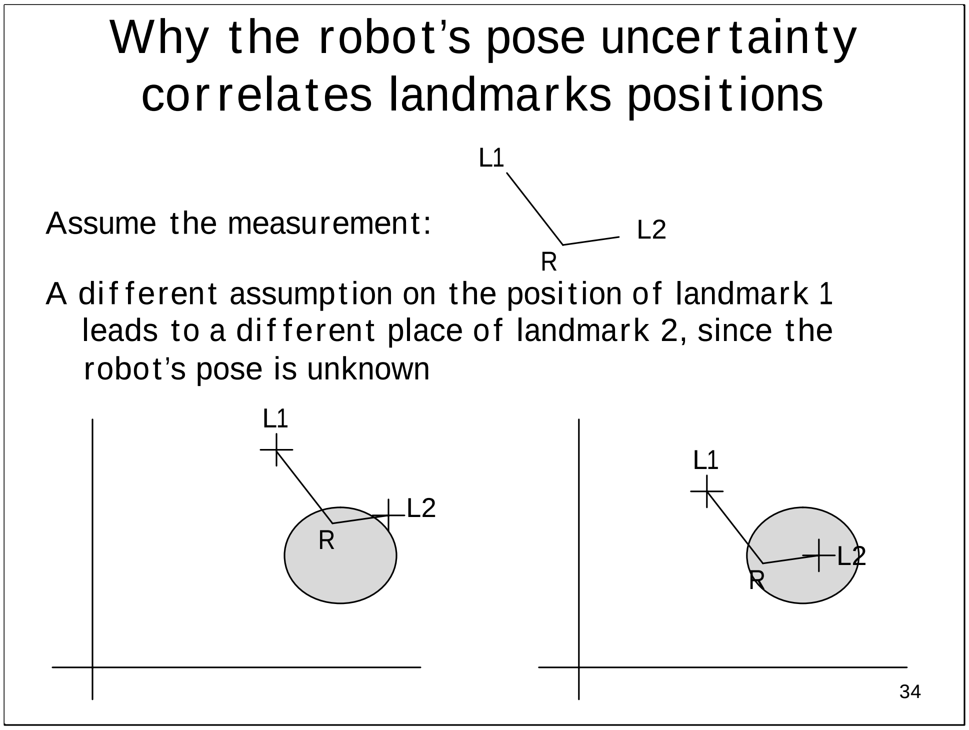
Cross-correlations between robot's landmarks' positions

⇒Cause: uncertainty on robot's pose correlates uncertainties on landmarks.

The reason why the classical solution to SLAm is complex is that, since the landmarks' position uncertainty are correlated (I explain that in the next slide), the algorithm must keep track of all the cross-correlations between them (non-diagonal term of the covariance matrix, in green here).

As a result, we must compute O(K^2) terms each time step.

With as many as 10000 landmarks, this can be computationally unaffordable.



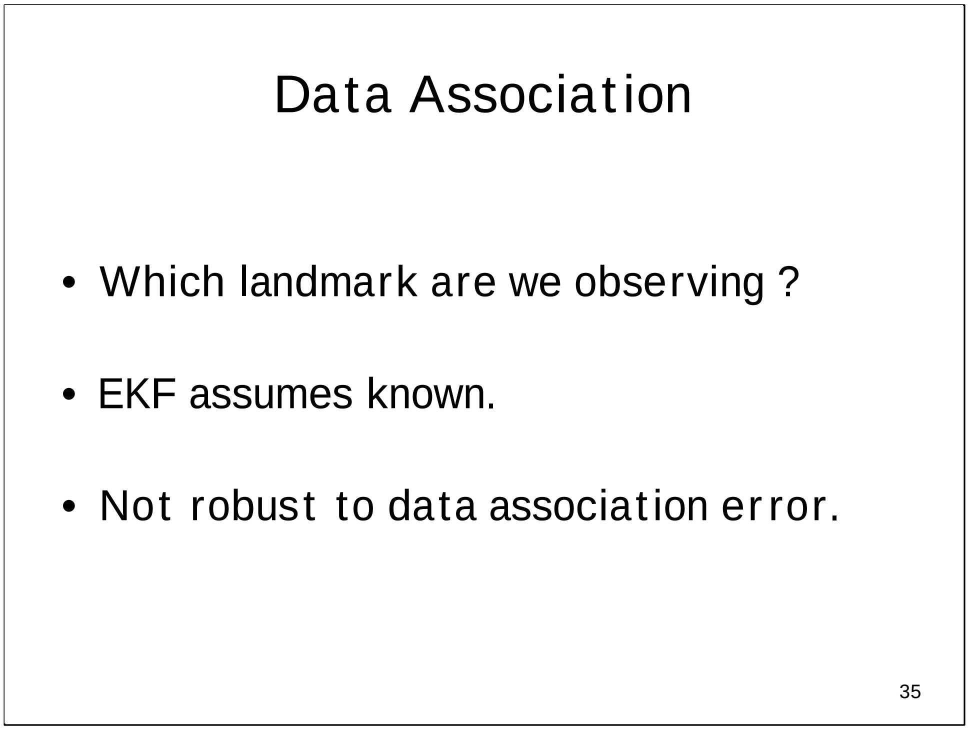
Why the robot's pose uncertainty correlates landmarks positions

Assume the measurement:

A different assumption on the position of landmark 1 leads to a different place of landmark 2, since the robot's pose is unknown

L1
L2
R
L1
L2
R
L1
L2
R

This slide explains in further detail why the landmark's positions are correlated through the uncertainty on the robot's position. I take an illustration. Supposing we have an observation from the robot (L1, L2). Then, if we assume, for instance from another observation, a given position for L1, through this observation, we end up to a given position for L2. Now, if the assumption on L1 is different, the conclusion on L2 is also. This is because the position of the robot is not known precisely. As a result, this lack of knowledge on the robot's position correlates the positions of the landmarks.

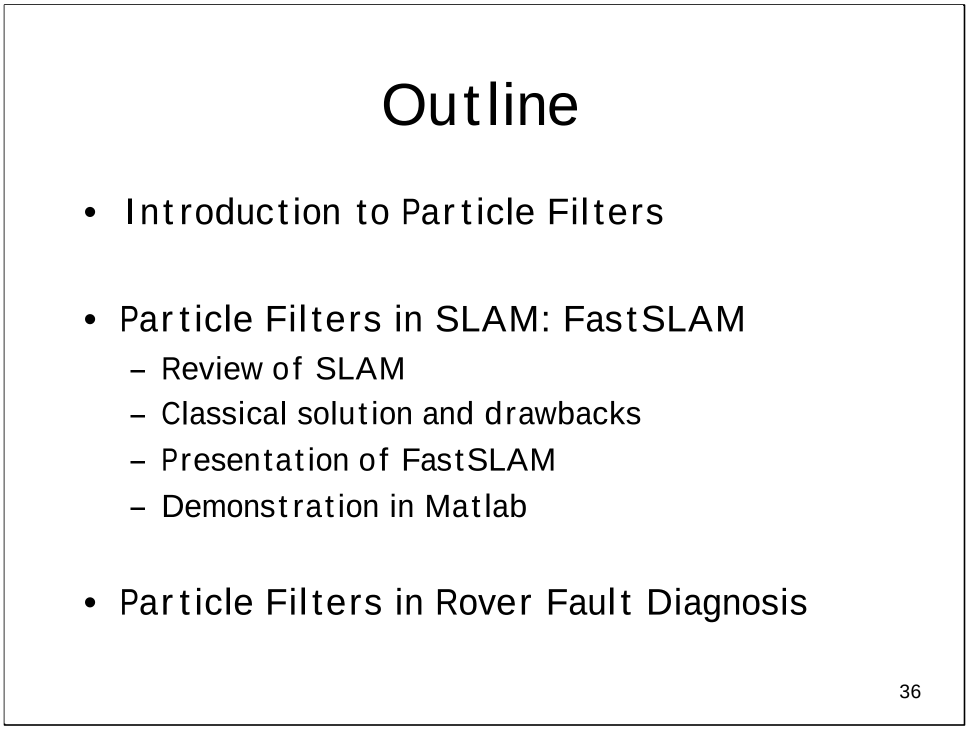


Data Association

- Which landmark are we observing ?
- EKF assumes known.
- Not robust to data association error.

In real application, we may not know which landmark we are observing (is it the one I have seen 10sec ago, or a new one ??), since many can have the same shape and color, and even because we may not be able to use an object recognition software.

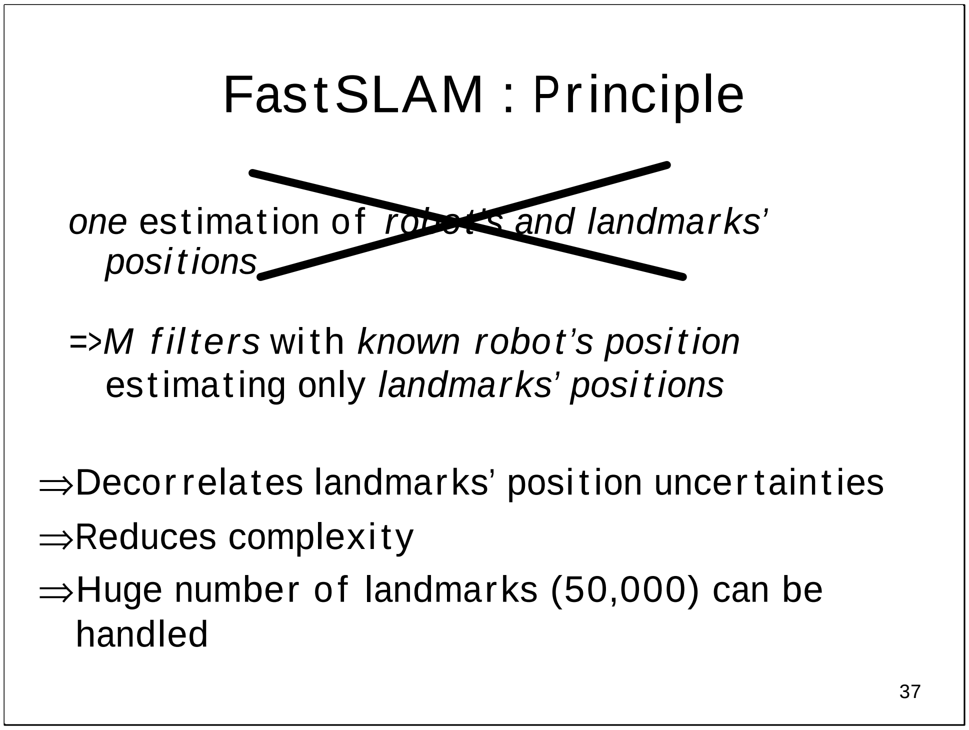
The problem with the EKF-SLAM is that it assumes that this association is known, and that it is very sensitive to errors in this association: it can diverge when an error is made.



Outline

- Introduction to Particle Filters
- Particle Filters in SLAM: FastSLAM
  - Review of SLAM
  - Classical solution and drawbacks
  - Presentation of FastSLAM
  - Demonstration in Matlab
- Particle Filters in Rover Fault Diagnosis

Moving to the presentation of FastSLAM as a way to solve the drawbacks of the EKF SLAM.



FastSLAM : Principle

~~*one* estimation of *robot's and landmarks' positions*~~

=>*M filters* with *known robot's position* estimating only *landmarks' positions*

⇒Decorrelates landmarks' position uncertainties

⇒Reduces complexity

⇒Huge number of landmarks (50,000) can be handled

37

This slide presents the principle of particle-filtering slam, as opposed to EKF-SLAM.

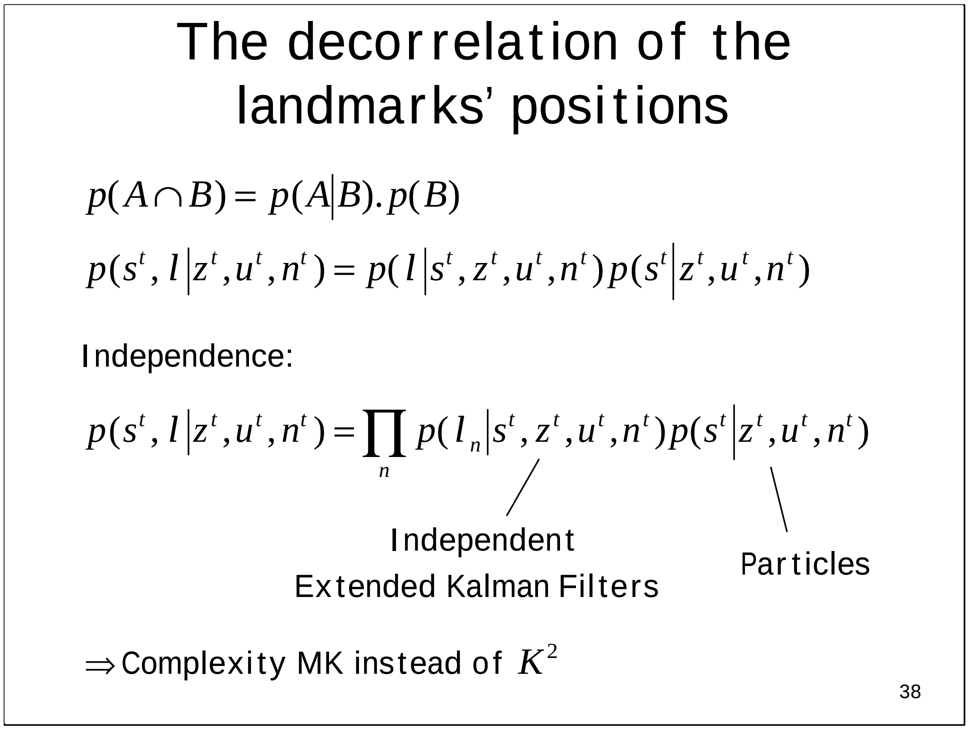
Instead of having ONE big estimation process on this huge state, keeping track of all the cross-correlations,

…

(make the cross and second line appear)

…

We replace it by M filters, each of which consider the robot's position is perfectly known, and estimates the landmarks' positions only: the great advantage of this idea is that it decorrelates the landmarks' positions uncertainties.



The decorrelation of the landmarks’ positions

$$p(A \cap B) = p(A|B).p(B)$$

$$p(s^t, l | z^t, u^t, n^t) = p(l | s^t, z^t, u^t, n^t)\, p(s^t | z^t, u^t, n^t)$$

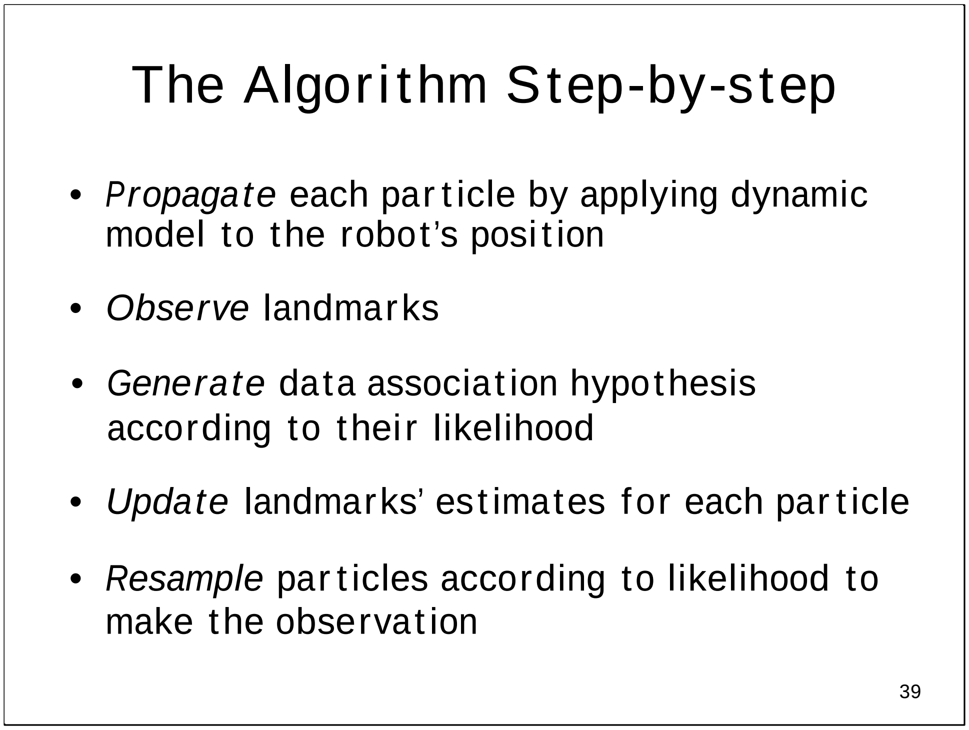
Independence:

$$p(s^t, l | z^t, u^t, n^t) = \prod_n \underbrace{p(l_n | s^t, z^t, u^t, n^t)}_{\text{Independent Extended Kalman Filters}} \underbrace{p(s^t | z^t, u^t, n^t)}_{\text{Particles}}$$

$\Rightarrow$ Complexity MK instead of K^2

This slides shows the maths behind the decorrelation of the landmarks’ positions.

Starting from computation of the probability density function of the state, we arrive to the structure of FastSLAM, in which one pdf is represented by the particles, and another, by the EKFs.

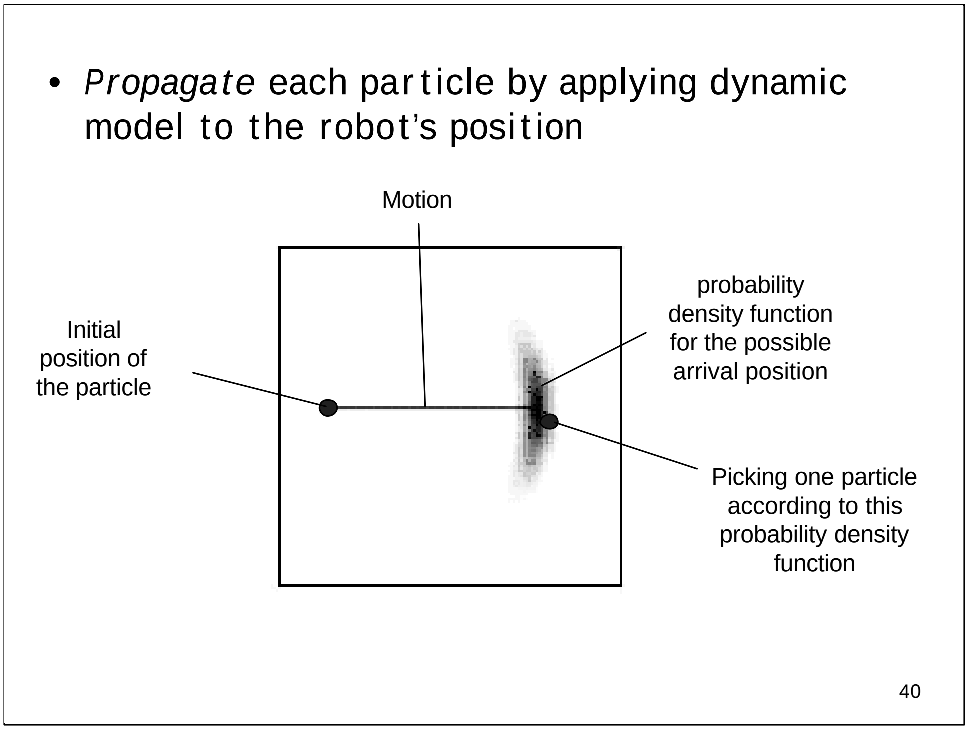


The Algorithm Step-by-step

- *Propagate* each particle by applying dynamic model to the robot’s position
- *Observe* landmarks
- *Generate* data association hypothesis according to their likelihood
- *Update* landmarks’ estimates for each particle
- *Resample* particles according to likelihood to make the observation

Here, I present step by step what the algorithm does.

I go through each point in detail in the following slides.



- *Propagate* each particle by applying dynamic model to the robot's position

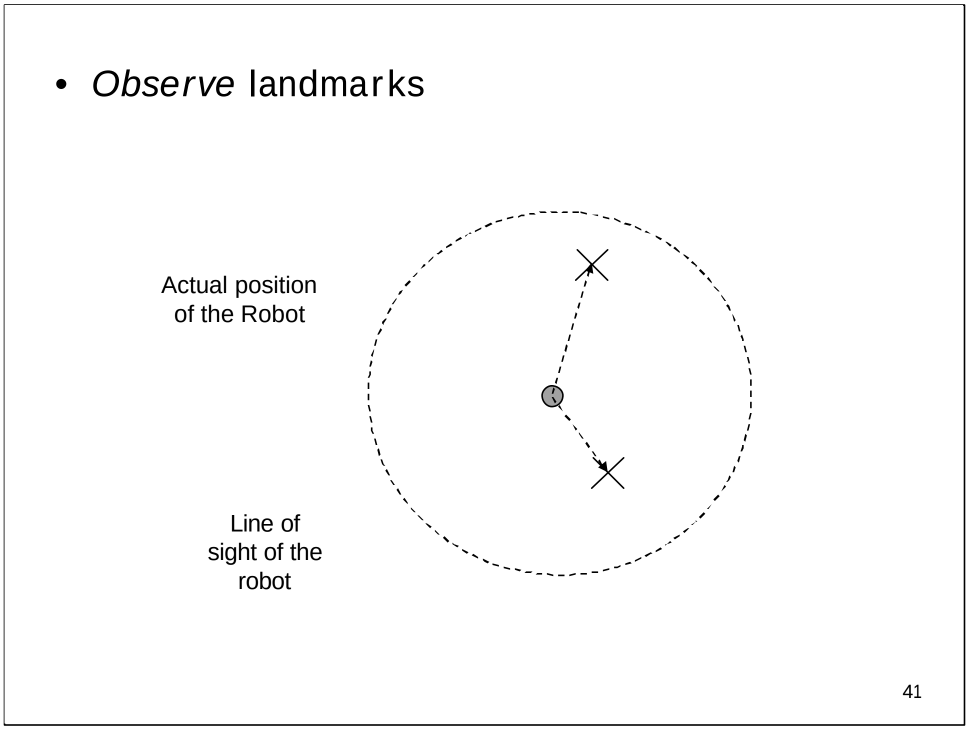
Motion

Initial position of the particle

probability density function for the possible arrival position

Picking one particle according to this probability density function

This slide is a detailed explanation of the propagation point of the previous slide: when the robot moves, from the motion model, we know the posterior probability density function. We pick one particle to replace the former one, according to this pdf.

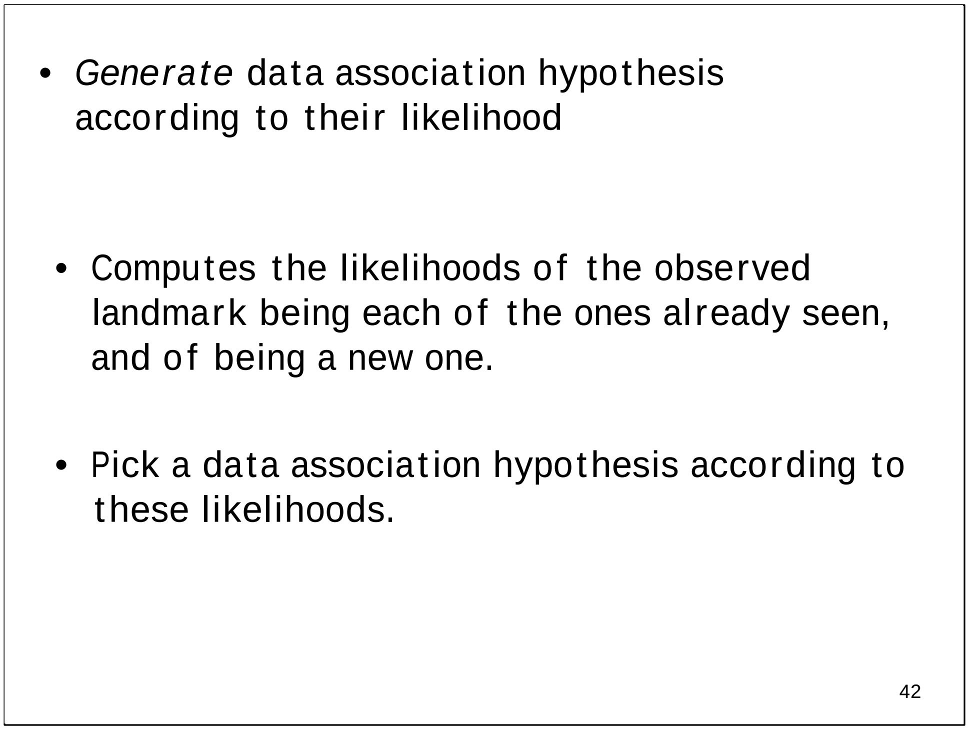


- *Observe* landmarks

Actual position
of the Robot

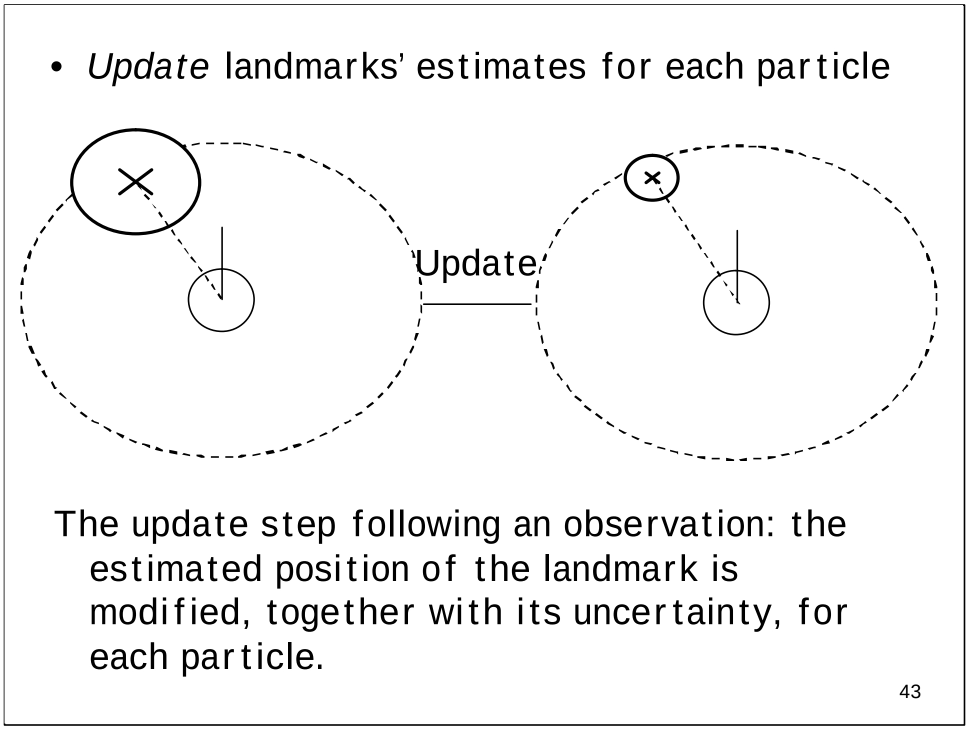
Line of
sight of the
robot

Detailed explanation of the observation: the robot observes the landmarks in its line of sight.



- *Generate* data association hypothesis according to their likelihood
  - Computes the likelihoods of the observed landmark being each of the ones already seen, and of being a new one.
  - Pick a data association hypothesis according to these likelihoods.

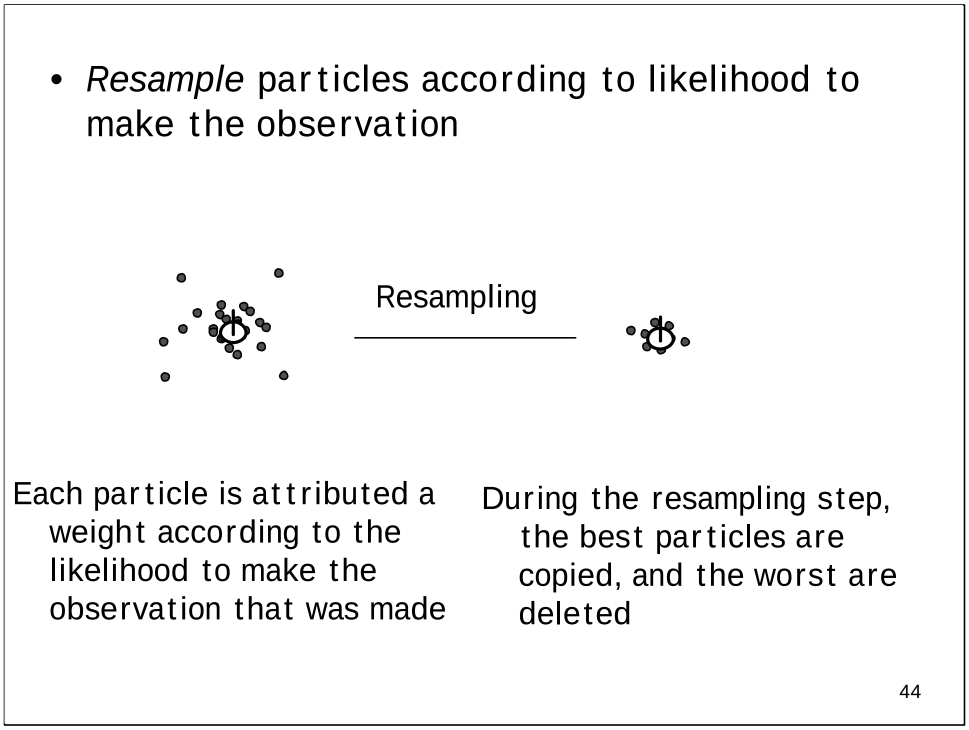
Detail of the data association hypothesis generation: each hypothesis is given a weight according to its likelihood, and each particle’s hypothesis is taken according to this weight.



- *Update* landmarks' estimates for each particle

The update step following an observation: the estimated position of the landmark is modified, together with its uncertainty, for each particle.

Detail of the update step: after an observation, the robot updates its estimate concerning this landmark, and the uncertainty of this estimate.



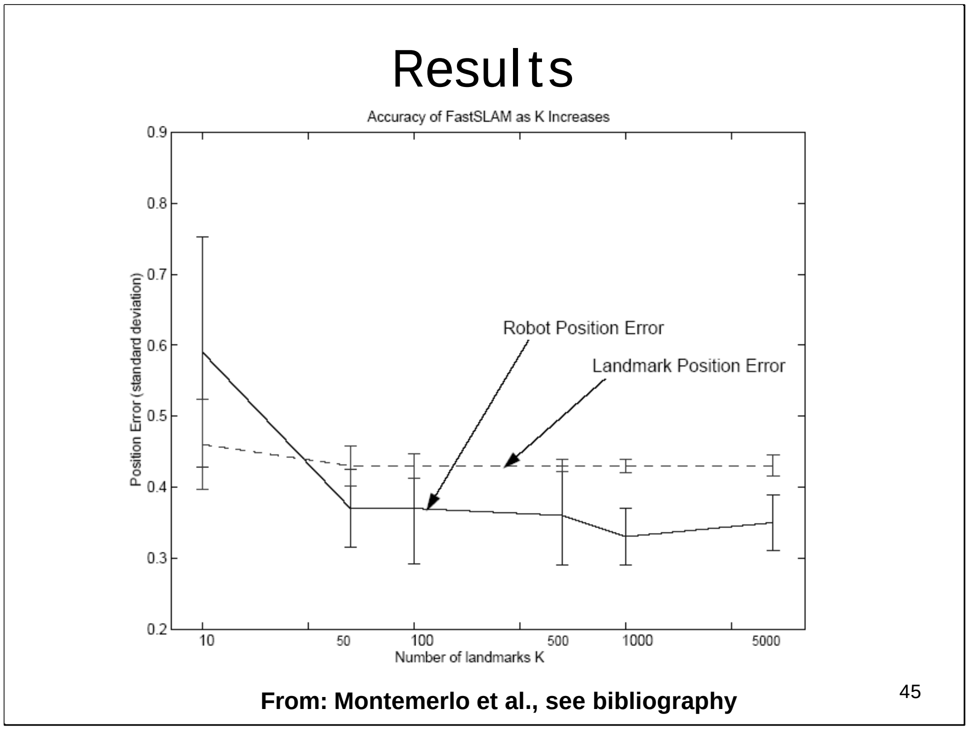
- *Resample* particles according to likelihood to make the observation

Resampling

Each particle is attributed a weight according to the likelihood to make the observation that was made

During the resampling step, the best particles are copied, and the worst are deleted

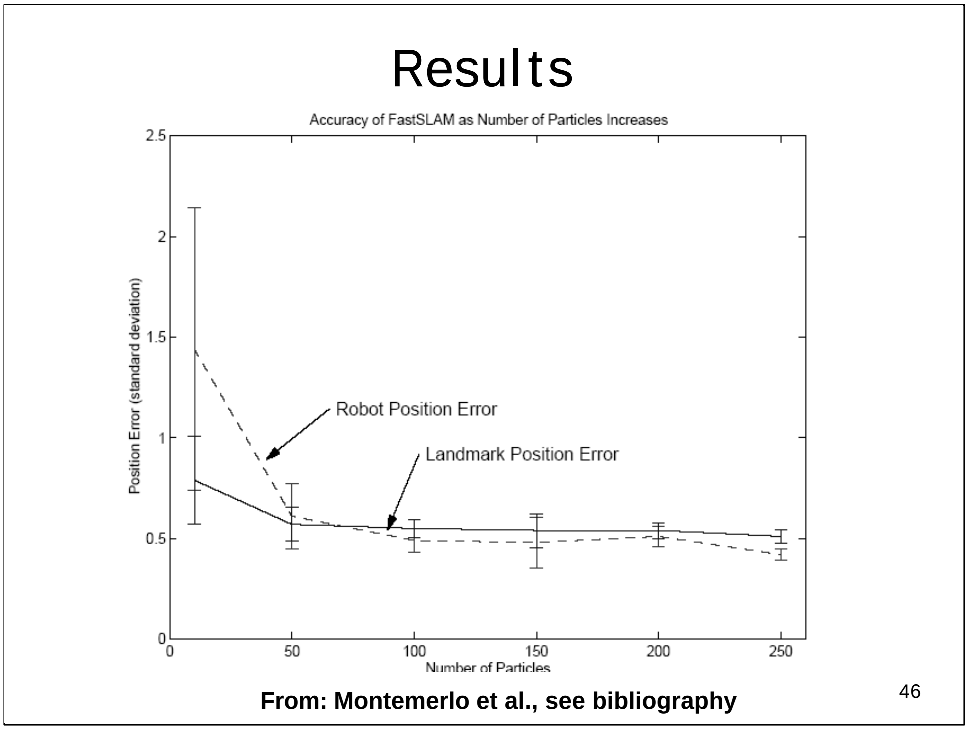
Detail of the resampling step: during this step, the particles are given weights according to their likelihood (based on the observation that was made). Then, the best ones are selected and copied, while the worst ones are deleted.



Results

From: Montemerlo et al., see bibliography

This slide presents a first result obtained by Montemerlo’s team. It represents the evolution of the error in the positions of the robot and the landmarks as the number of landmarks increases: the robo’s position error decreases, and the landmarks’ error does not.



Results

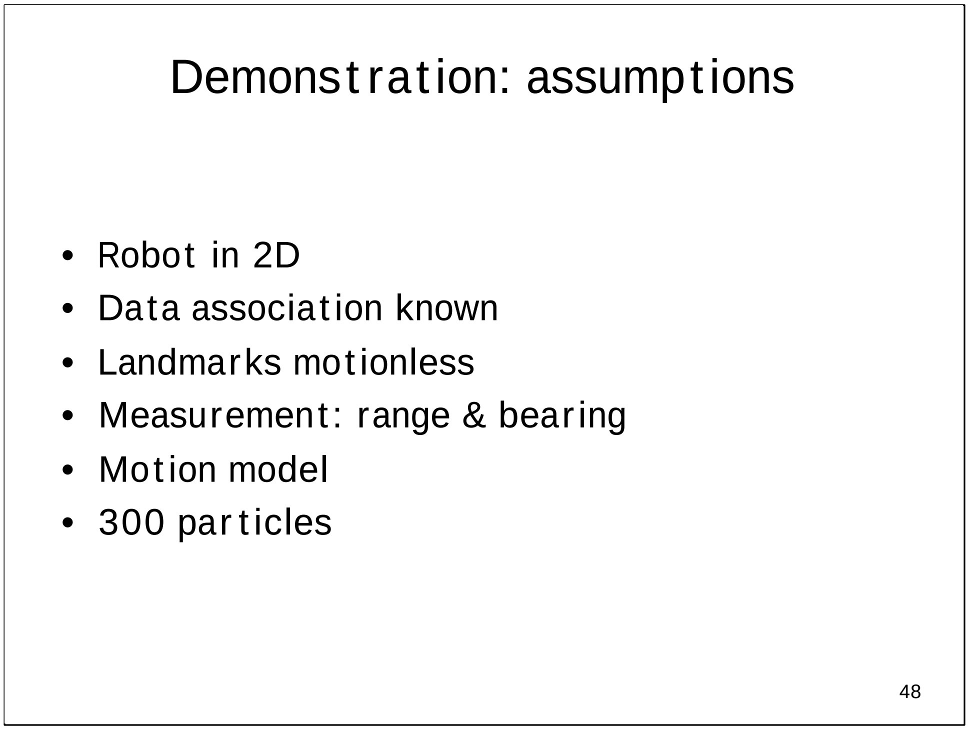
From: Montemerlo et al., see bibliography

Another result from Montemerlo’s experiments. This is the evolution of the error in the positions of the robot and the landmarks as the number of particle increases. Interestingly enough, it does not change much as the number increases beyond 100. As a result, they chose to use 100 particles for their experiments.

Outline

- Introduction to Particle Filters
- Particle Filters in SLAM: FastSLAM
  - Review of SLAM
  - Classical solution and drawbacks
  - Presentation of FastSLAM
  - Demonstration in Matlab
- Particle Filters in Rover Fault Diagnosis

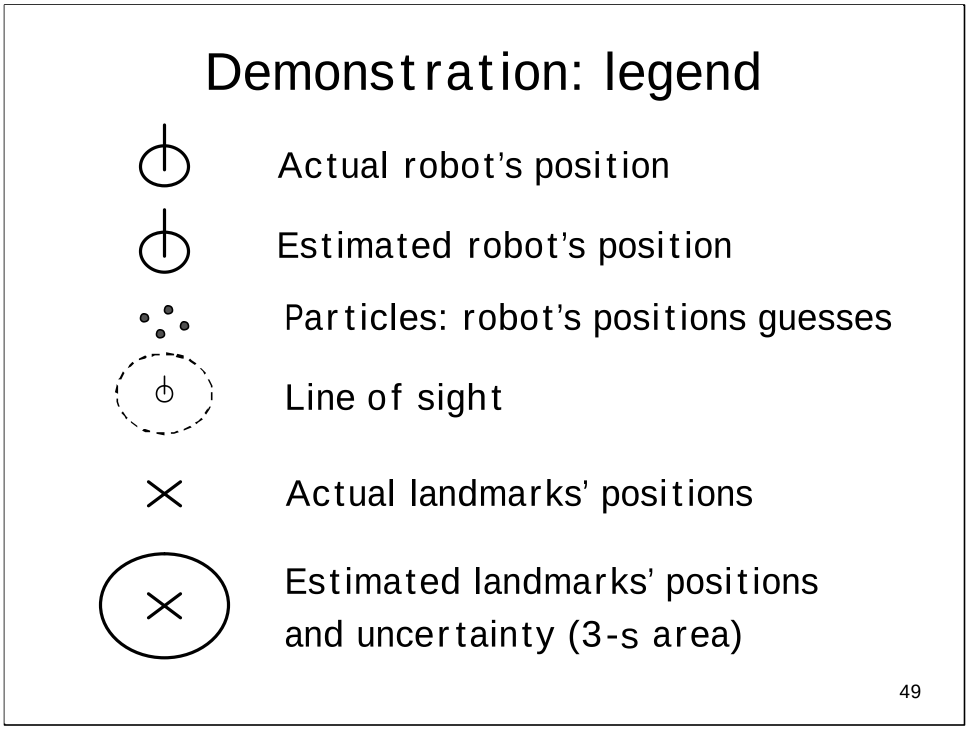
Moving to the demo.



Demonstration: assumptions

- Robot in 2D
- Data association known
- Landmarks motionless
- Measurement: range & bearing
- Motion model
- 300 particles

This slide introduces my Matlab demo, stating the different assumptions I made for it.



Demonstration: legend

Actual robot's position

Estimated robot's position

Particles: robot's positions guesses

Line of sight

Actual landmarks' positions

Estimated landmarks' positions and uncertainty (3-s area)

Here is the legend for the demo, since it contains many elements, and may be confusing at first glance.

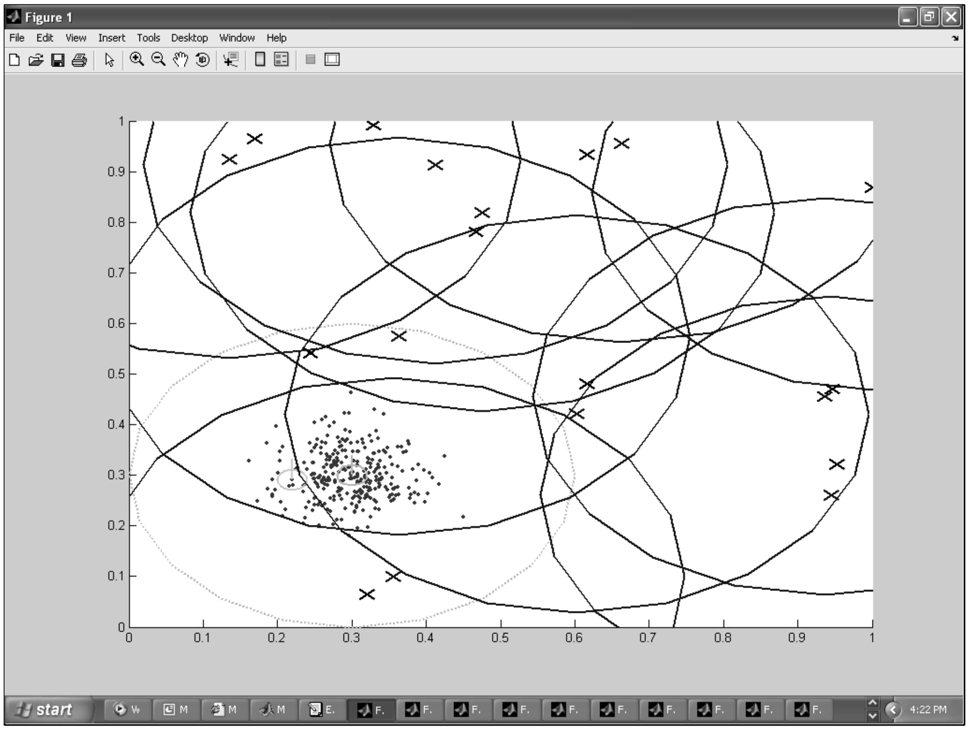


Figure 1
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:22 PM

Here is the initial situation. It is messy since the initial uncertainties are huge, to make you see the improvement over time. One can see the robot’s actual and estimated position, its line of sight, the different particles, and the actual and estimated –together with uncertainties- positions of the landmarks.

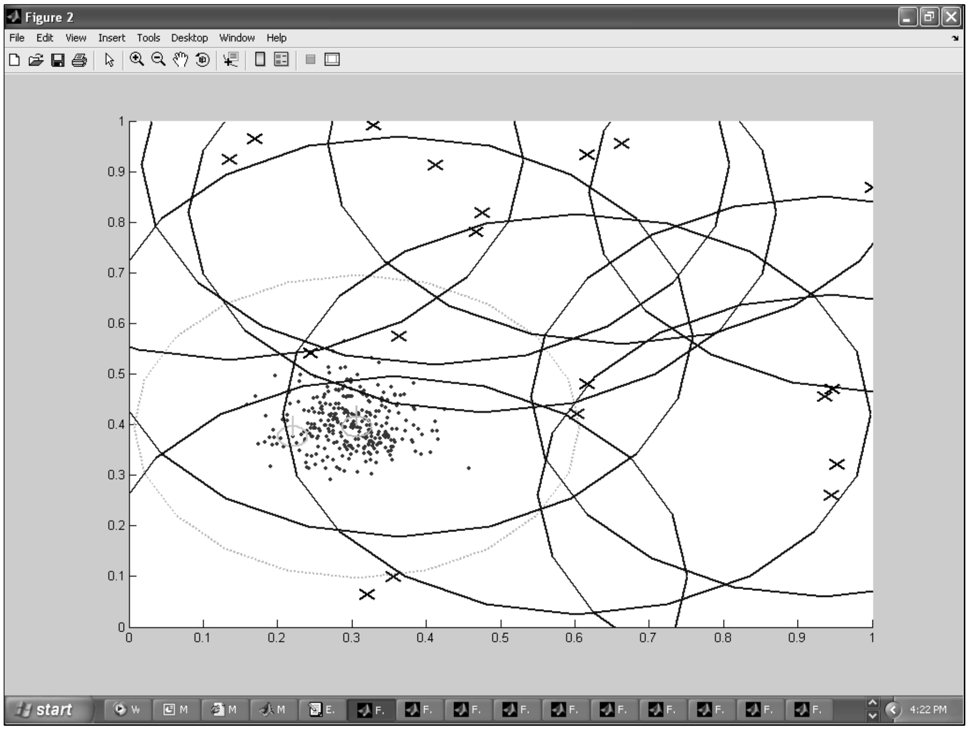


Figure 2
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:22 PM

The robot moves one step ahead

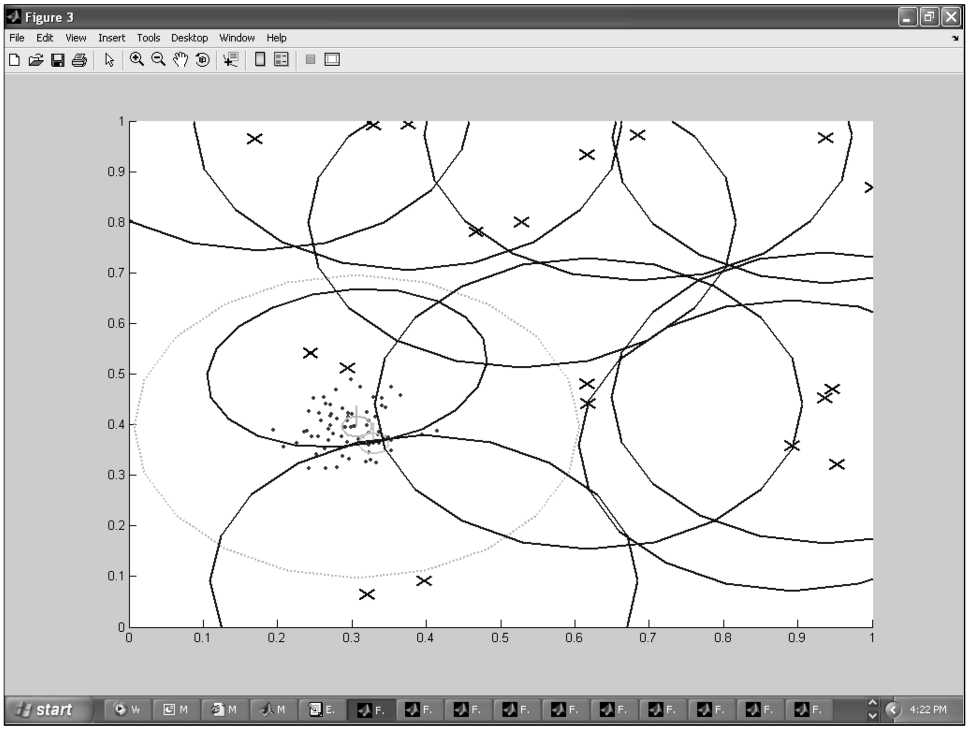


Figure 3
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:22 PM

During the first steps, I represented two steps: after moving (the particles spread out because of the uncertainty in the motion), and after observation+update+resampling (the particles gather since only the best ones are selected through the resampling step.)

Here, after moving, the update+resampling steps had taken place. As a result, the uncertainty on the landmarks' positions is reduced for those within the line of sight. The number of particles is also reduced, since, through resampling, only the best ones are kept.

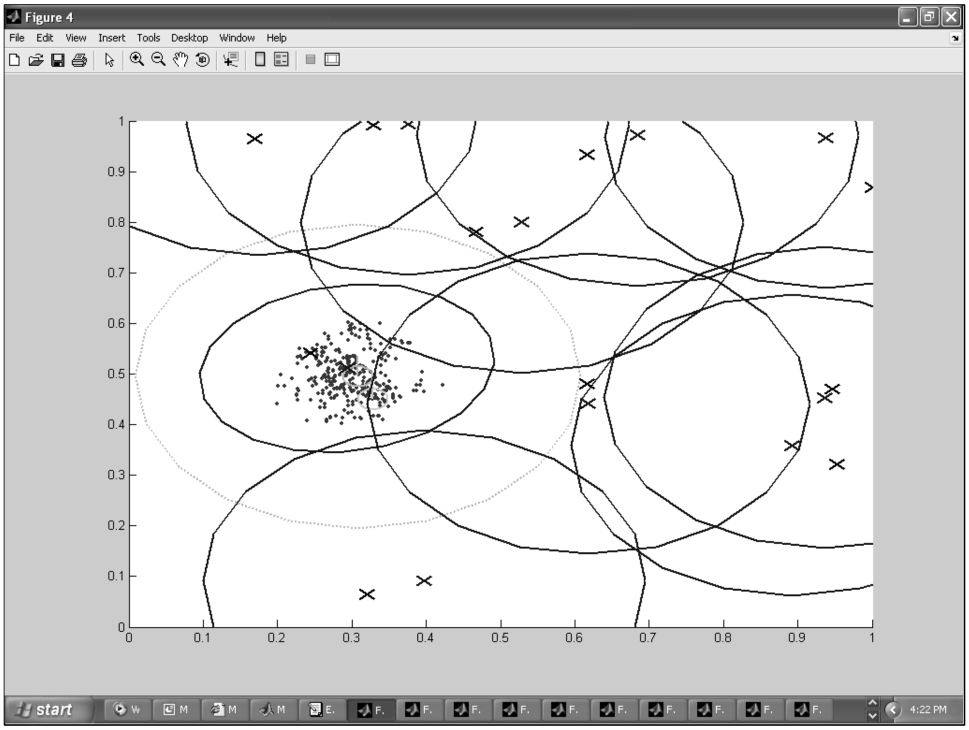


Figure 4
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
start
4:22 PM

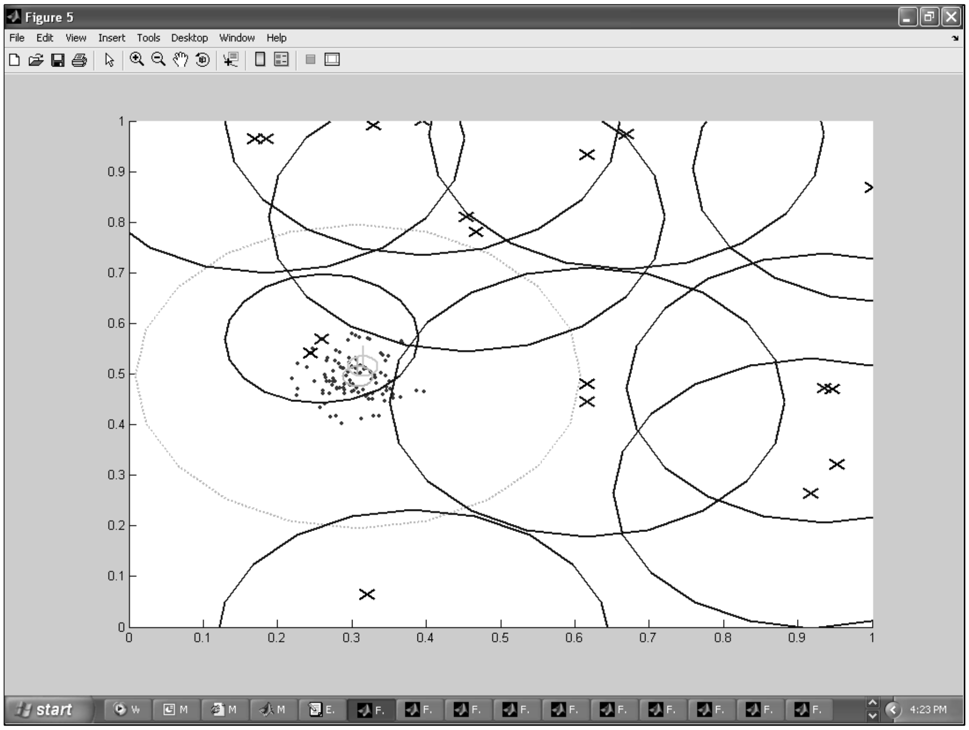


Figure 5
File
Edit
View
Insert
Tools
Desktop
Window
Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:23 PM

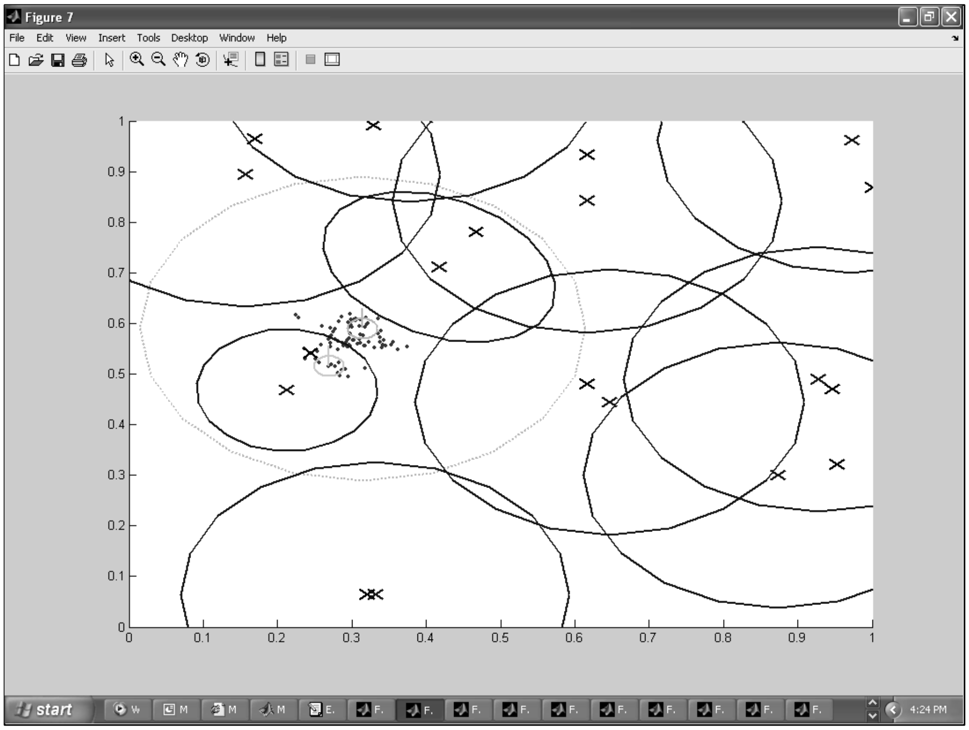


Figure 7
File
Edit
View
Insert
Tools
Desktop
Window
Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:24 PM

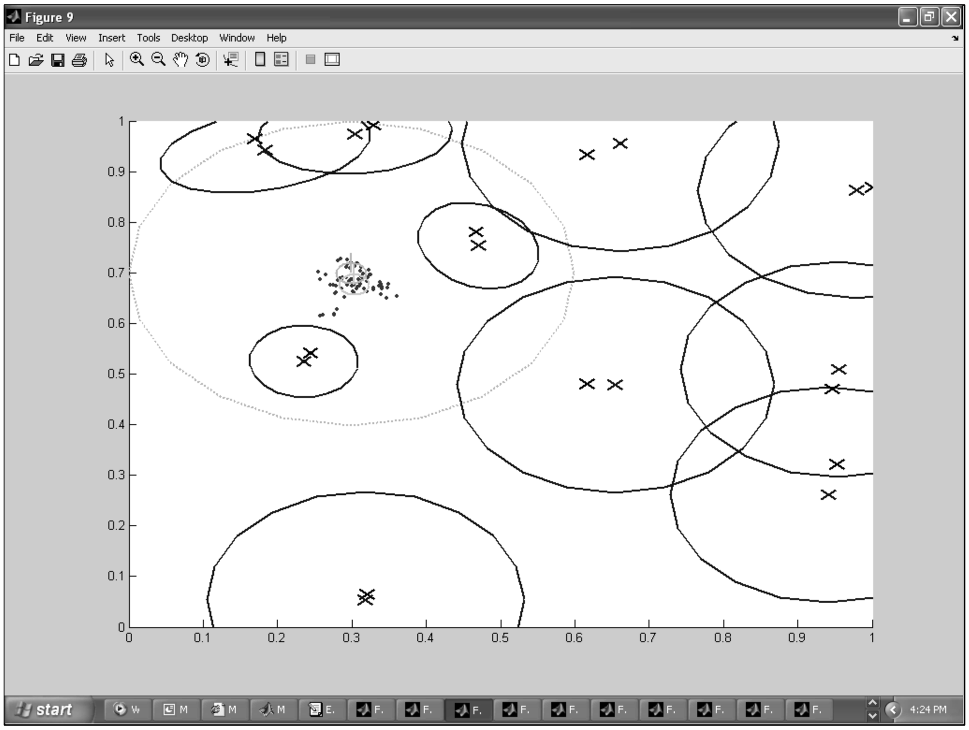


Figure 9
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
start
4:24 PM

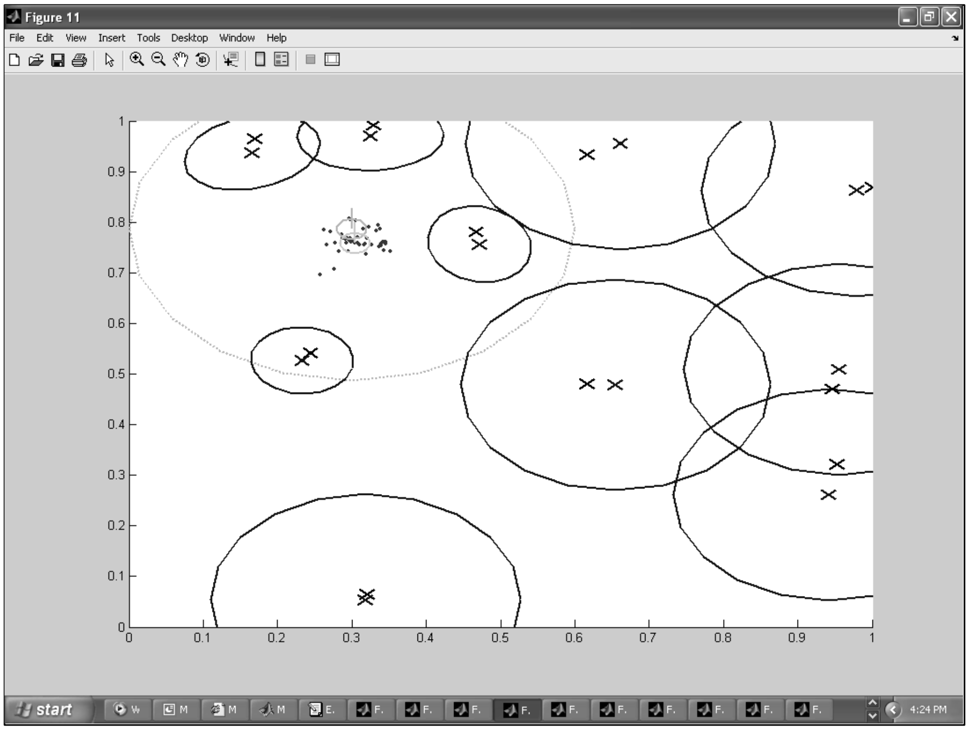


Figure 11
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:24 PM

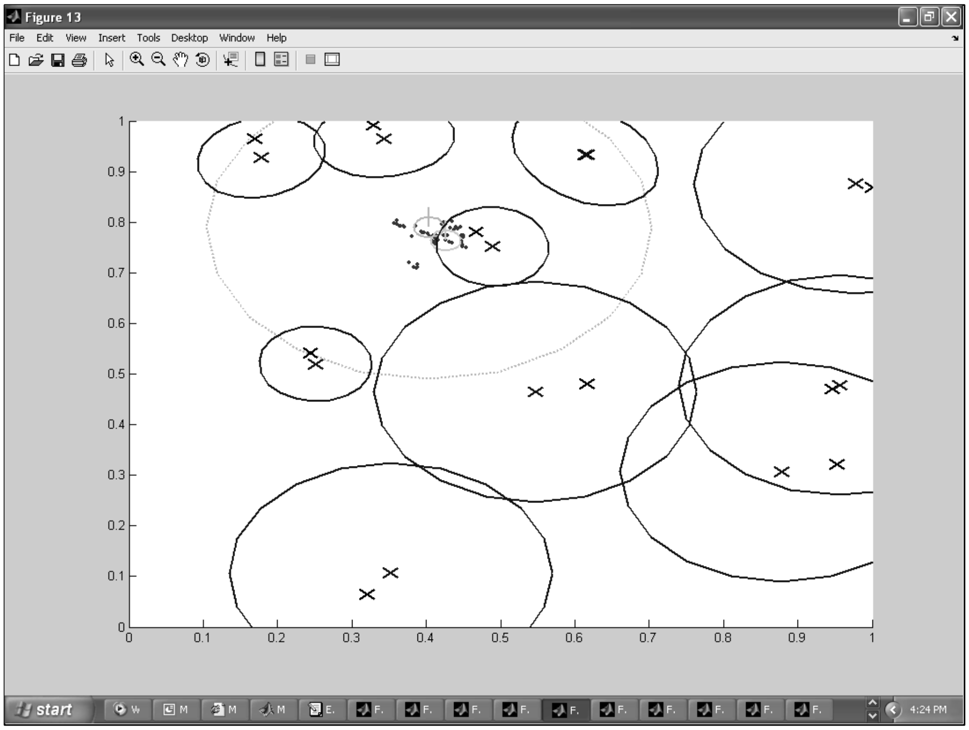


Figure 13
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:24 PM

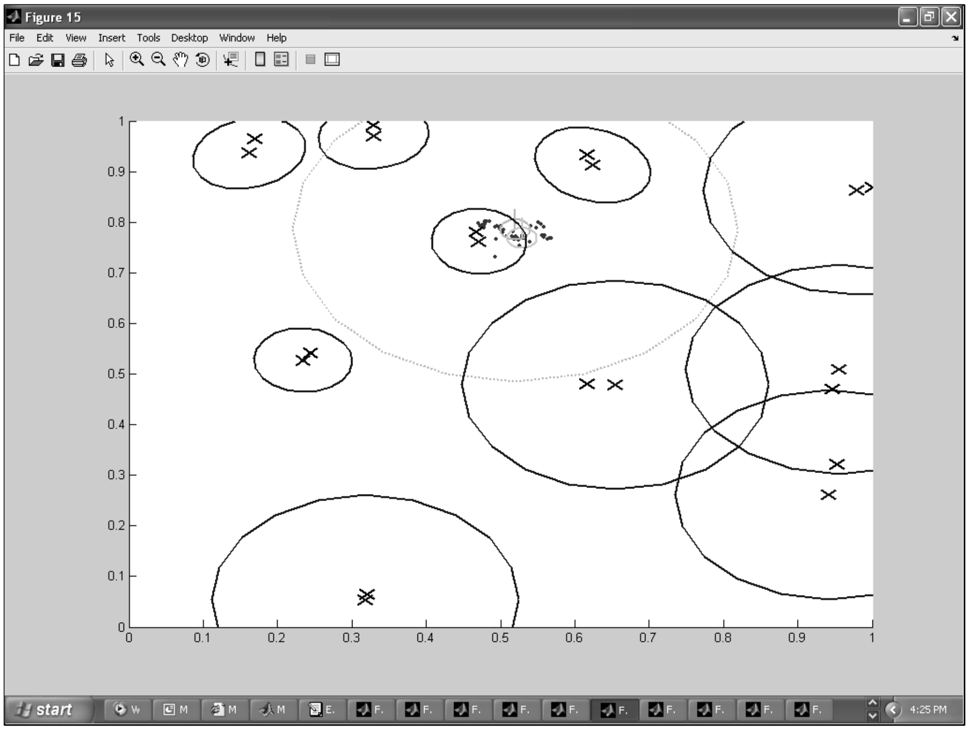


Figure 15
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:25 PM

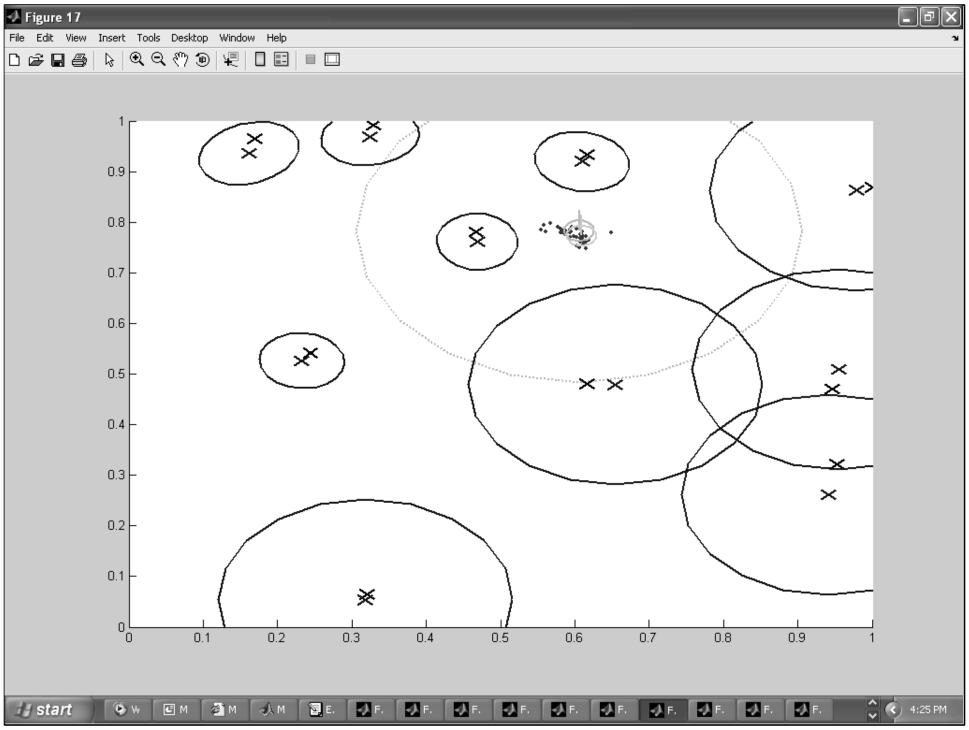


Figure 17
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:25 PM

As the robot moves, one can see the uncertainty on the landmarks' positions decrease, and the spreading of the particles as well.

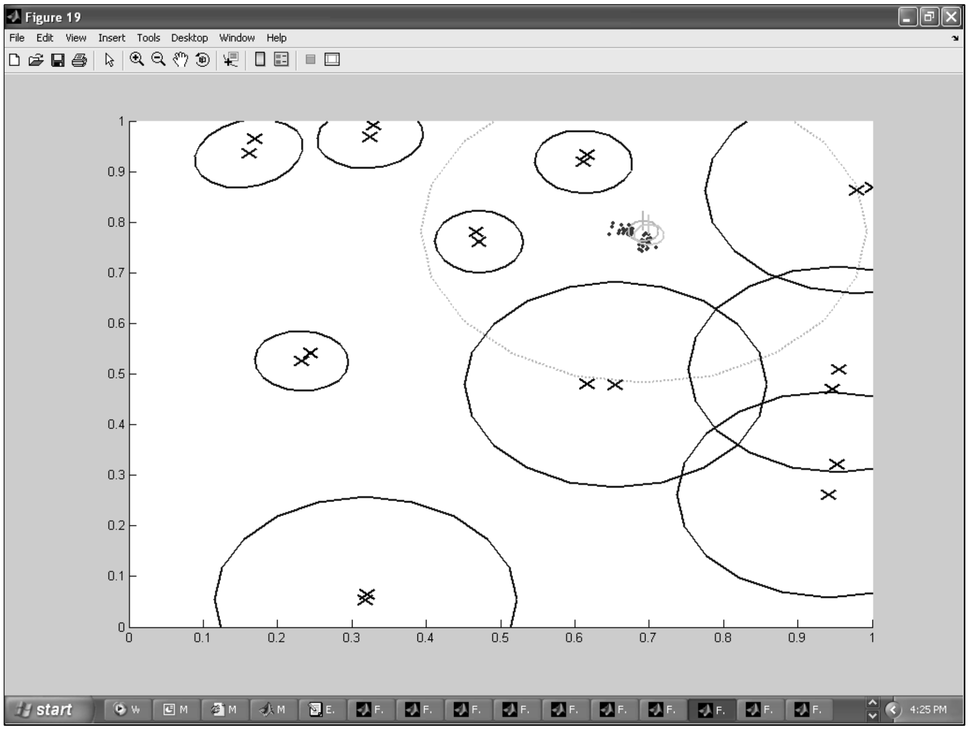


Figure 19
File
Edit
View
Insert
Tools
Desktop
Window
Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:25 PM

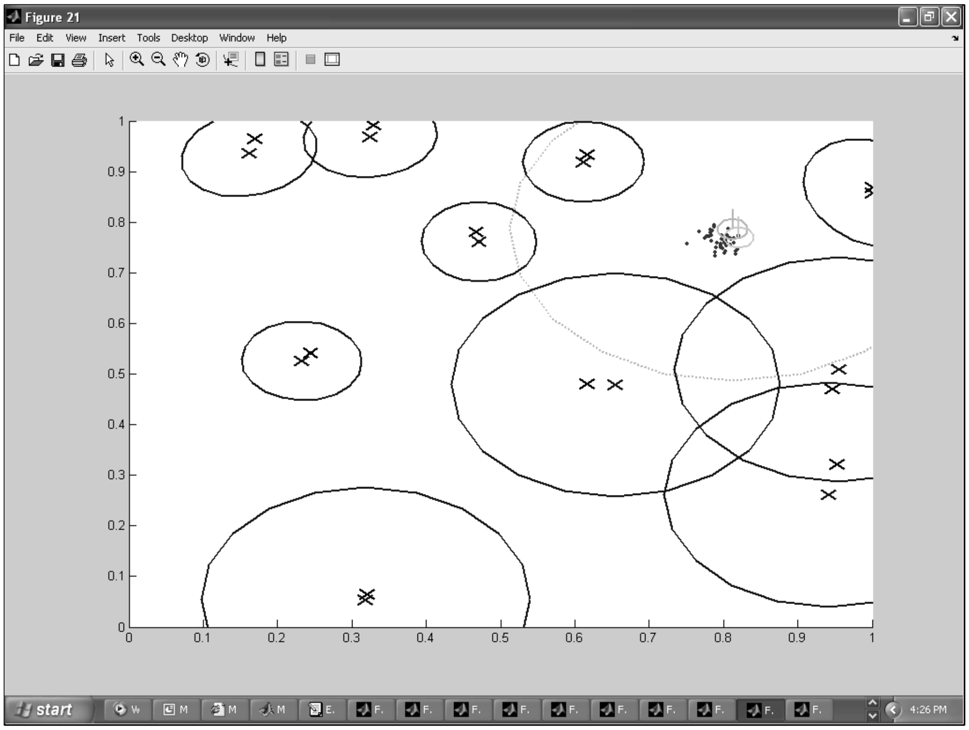


Figure 21
File Edit View Insert Tools Desktop Window Help
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
start
4:26 PM

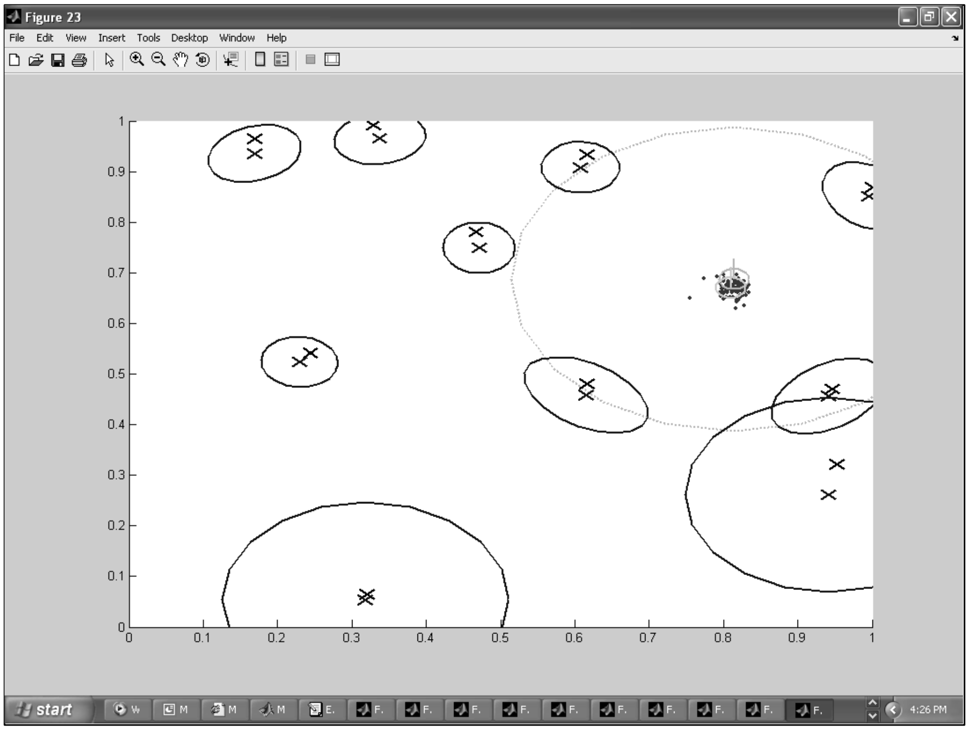


Figure 23
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:26 PM

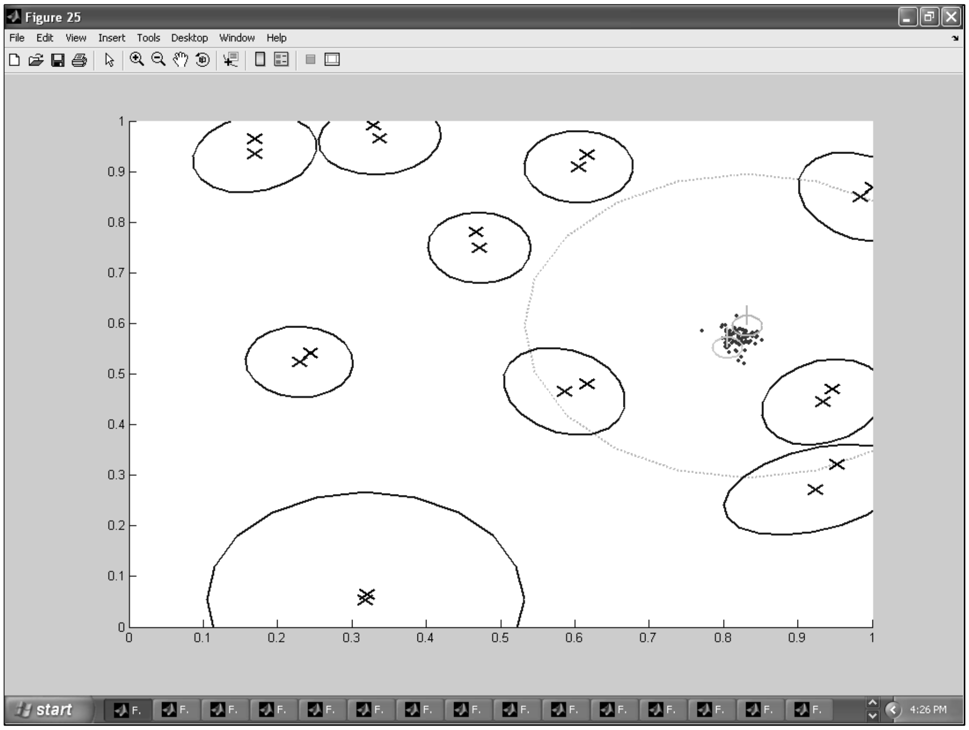


Figure 25
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:26 PM

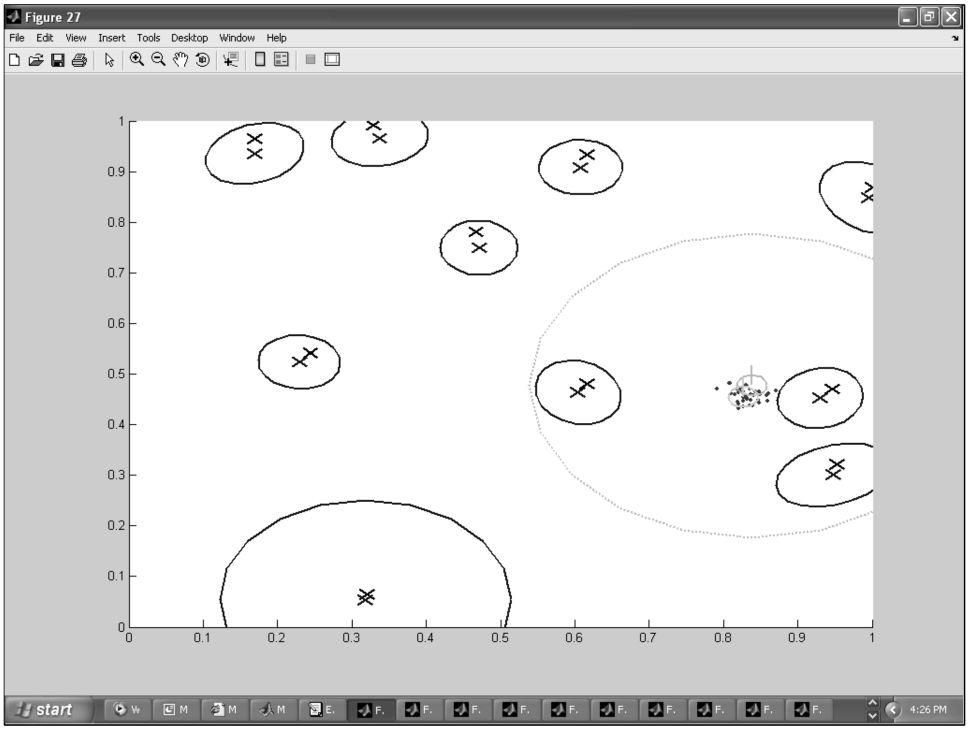


Figure 27
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:26 PM

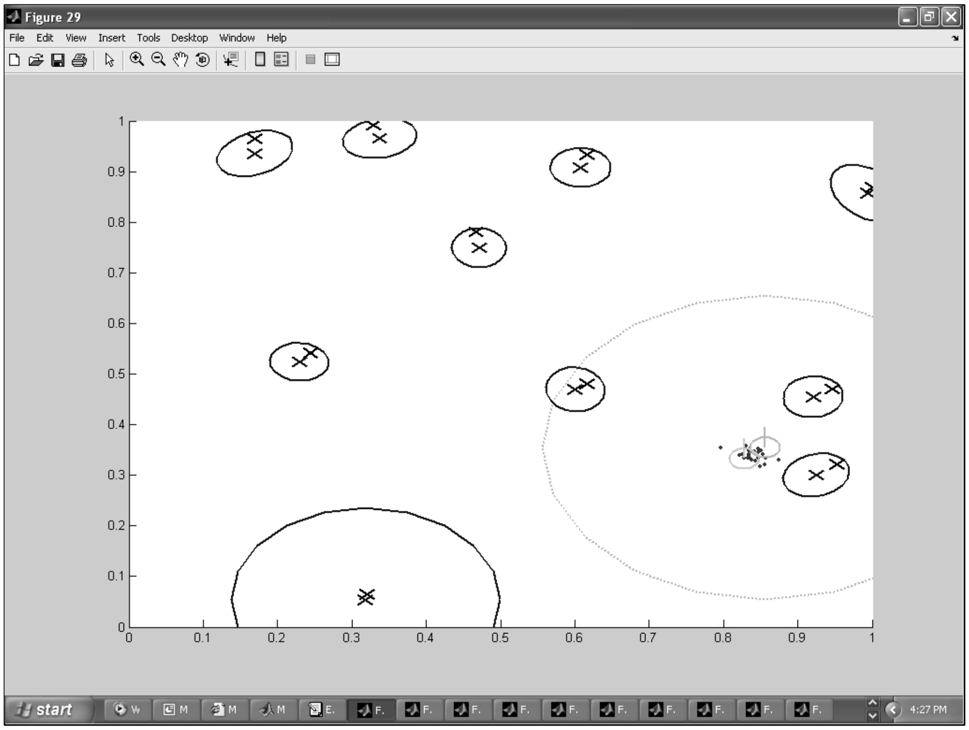


Figure 29
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:27 PM

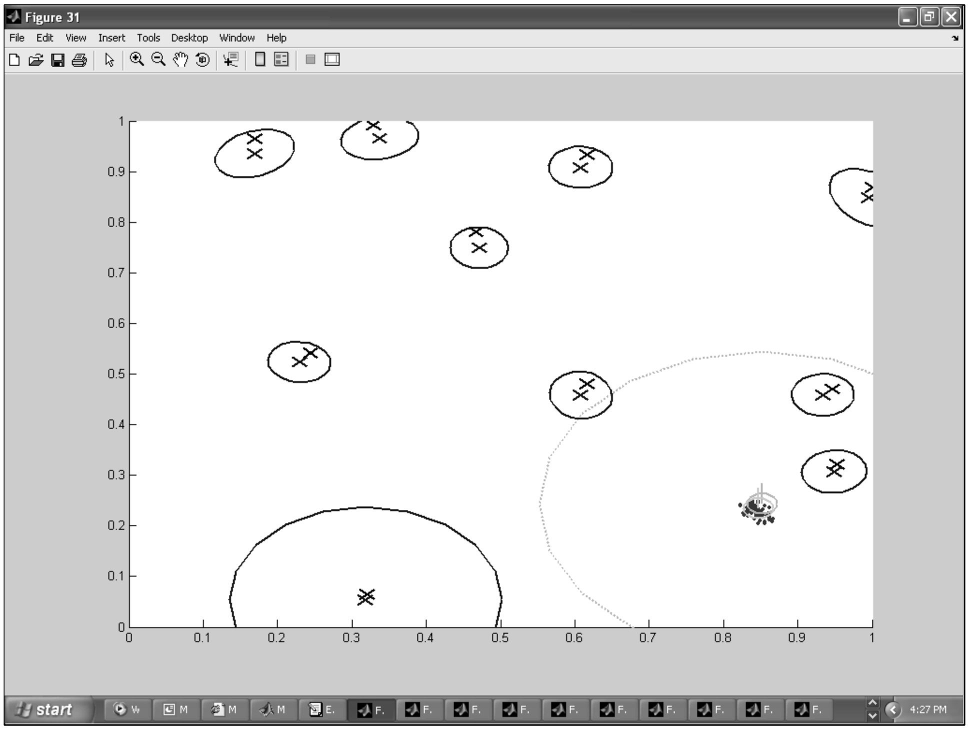


Figure 31
File
Edit
View
Insert
Tools
Desktop
Window
Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
4:27 PM

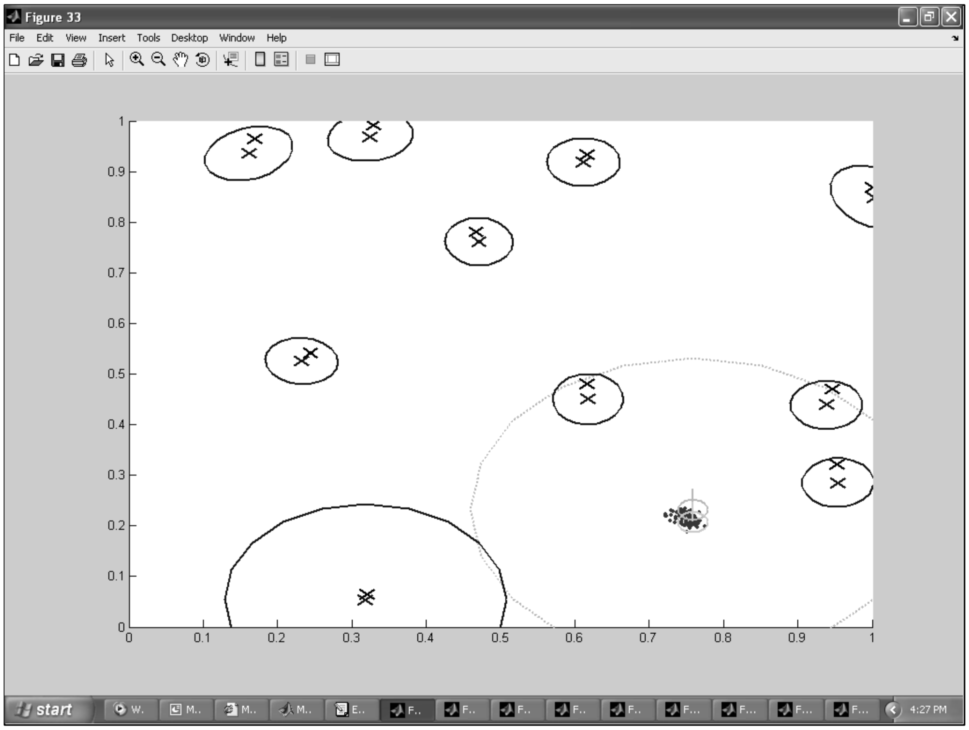


Figure 33
File
Edit
View
Insert
Tools
Desktop
Window
Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
W.
M..
M..
M..
E..
F..
F...
4:27 PM

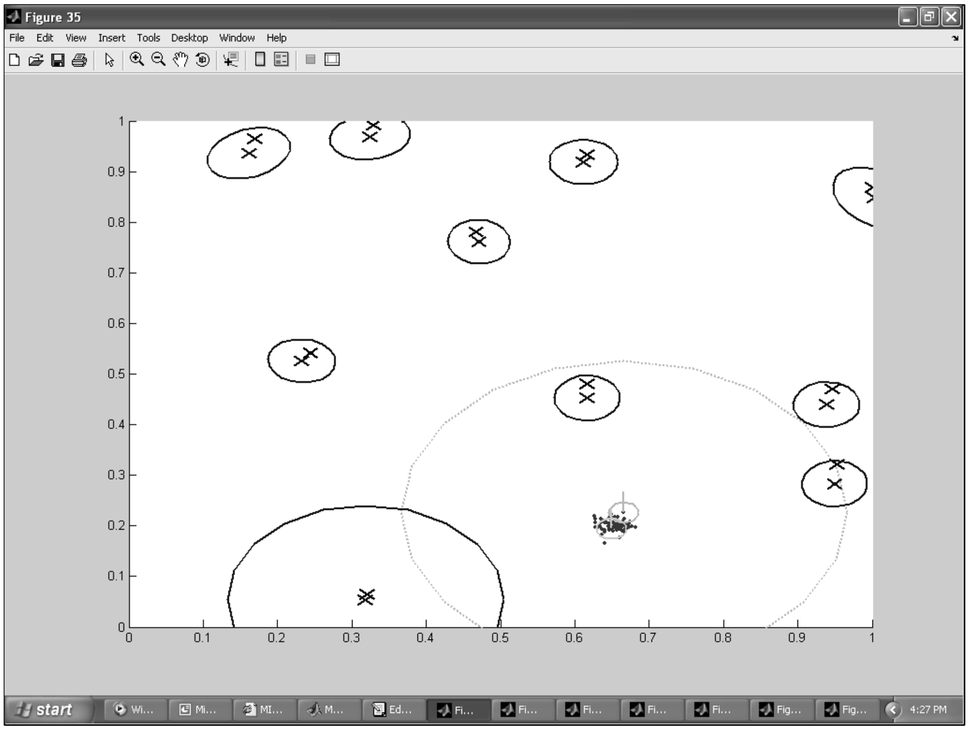


Figure 35
File Edit View Insert Tools Desktop Window Help
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
Wi...
Mi...
MI...
M...
Ed...
Fi...
Fi...
Fi...
Fi...
Fi...
Fig...
Fig...
4:27 PM

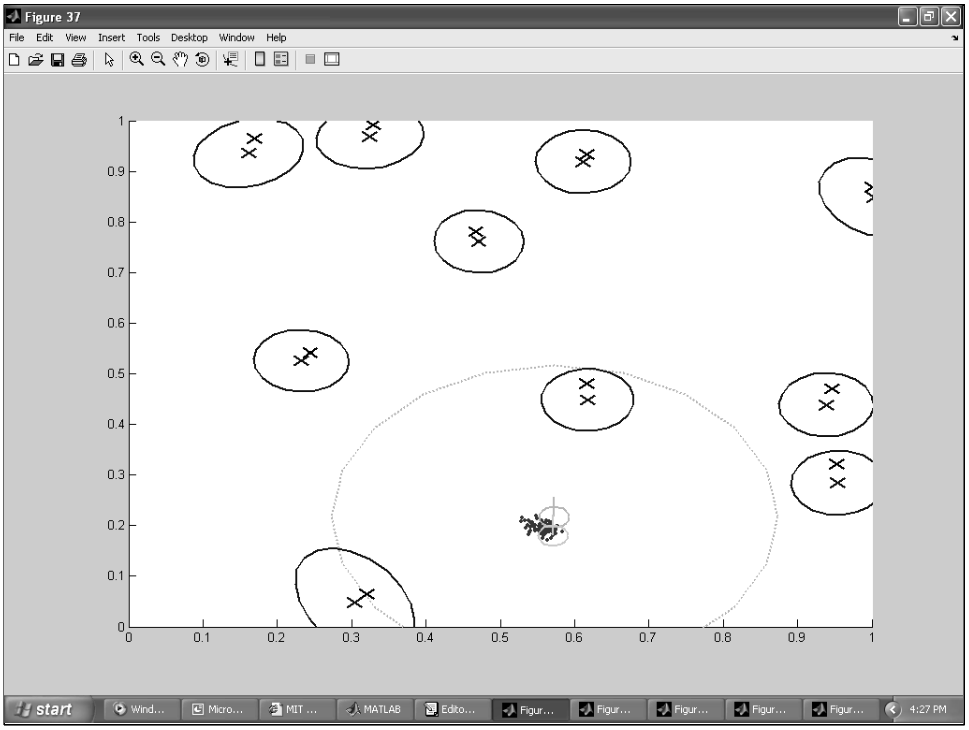


Figure 37
File
Edit
View
Insert
Tools
Desktop
Window
Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
Wind...
Micro...
MIT ...
MATLAB
Edito...
Figur...
Figur...
Figur...
Figur...
Figur...
4:27 PM

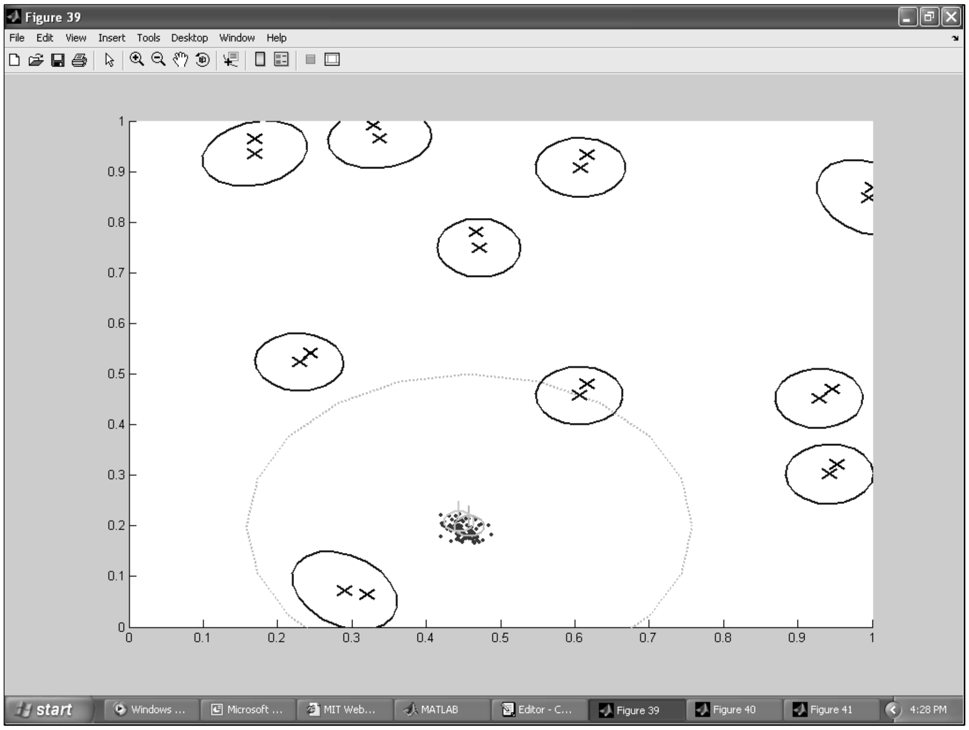
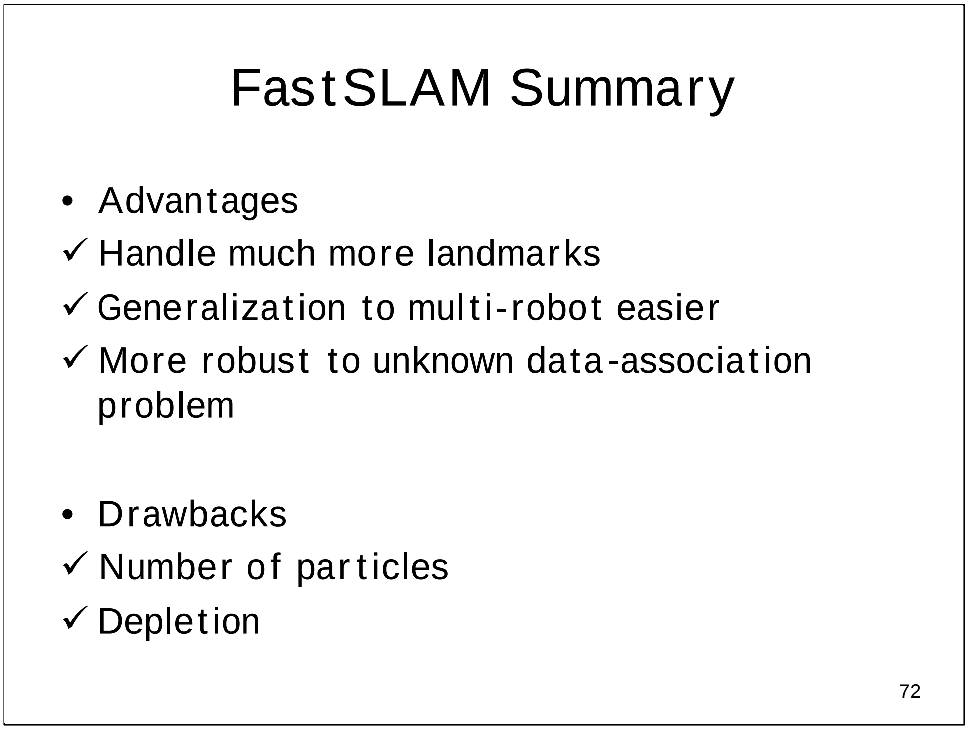


Figure 39
File Edit View Insert Tools Desktop Window Help
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
start
Windows ...
Microsoft ...
MIT Web...
MATLAB
Editor - C...
Figure 39
Figure 40
Figure 41
4:28 PM

Last slide of the demo: the robot's position is more accurately estimated, as we see from the spreading of the particles. The landmarks' positions are also much more accurately estimated, as we see from the size of the blue circles around them.



FastSLAM Summary

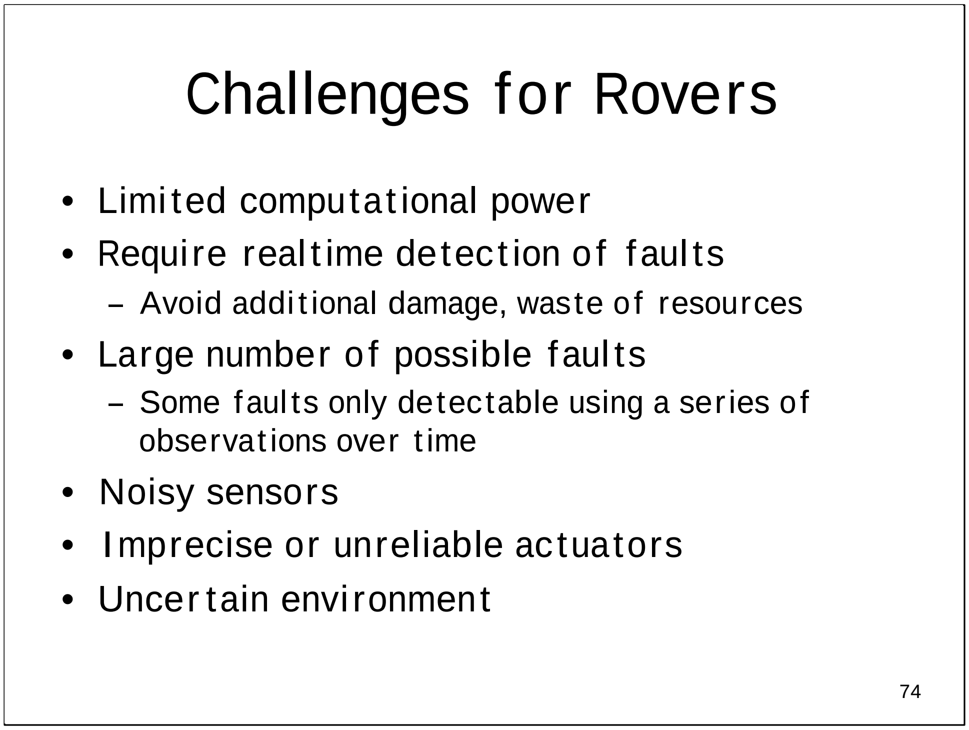
- Advantages
- ✓ Handle much more landmarks
- ✓ Generalization to multi-robot easier
- ✓ More robust to unknown data-association problem

- Drawbacks
- ✓ Number of particles
- ✓ Depletion

This slide recaps the pros and cons of the particle filtering applied to the SLAM problem. When presenting this slide, I mention the so-called ‘depletion’ problem, which consists of the number of particles going to zero through the resampling stage.

Outline

- Introduction to Particle Filters
- Particle Filters in SLAM: FastSLAM
- Particle Filters in Rover Fault Diagnosis
  - Challenges for rover fault diagnosis
  - Fault diagnosis as state estimation
  - Approximating a solution using particle filtering
  - Particle filter enhancements
    - Risk Sensitive Particle Filter
    - Variable Resolution Particle Filter



Challenges for Rovers

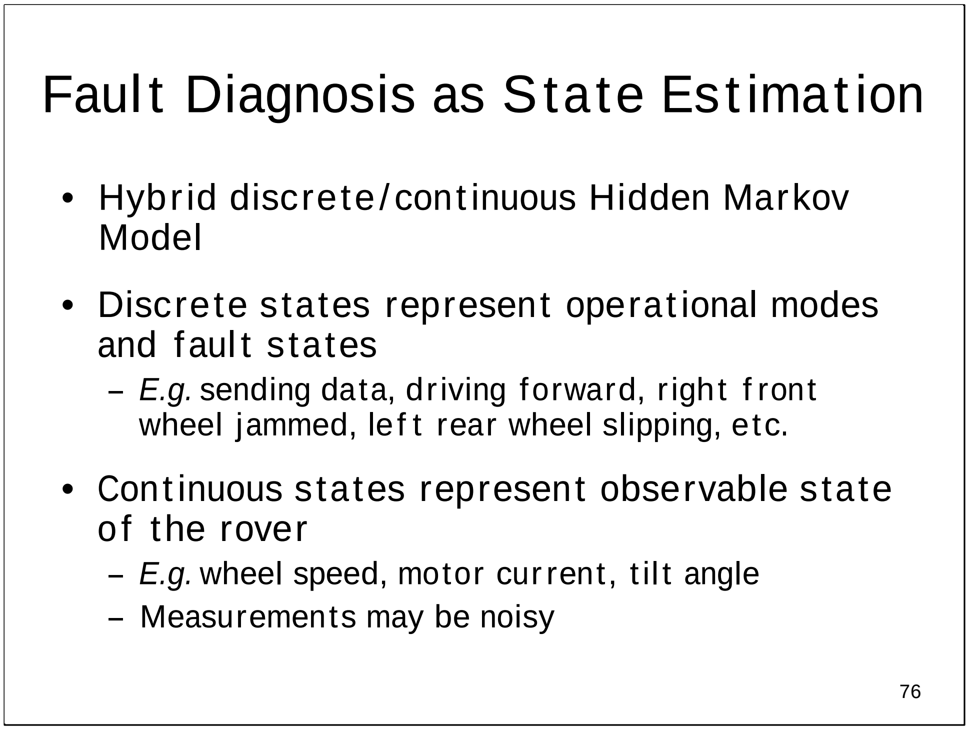
- Limited computational power
- Require realtime detection of faults
  - Avoid additional damage, waste of resources
- Large number of possible faults
  - Some faults only detectable using a series of observations over time
- Noisy sensors
- Imprecise or unreliable actuators
- Uncertain environment

(This slide is mostly self explanatory. I added the part below when giving the talk.)

Some faults cannot be detected using a single set of observations. For example, it may be normal for a wheel to slip occasionally but repeated slipping could indicate a problem.

Outline

- Introduction to Particle Filters
- Particle Filters in SLAM: FastSLAM
- Particle Filters in Rover Fault Diagnosis
  - Challenges for rover fault diagnosis
  - Fault diagnosis as state estimation
  - Approximating a solution using particle filtering
  - Particle filter enhancements
    - Risk Sensitive Particle Filter
    - Variable Resolution Particle Filter

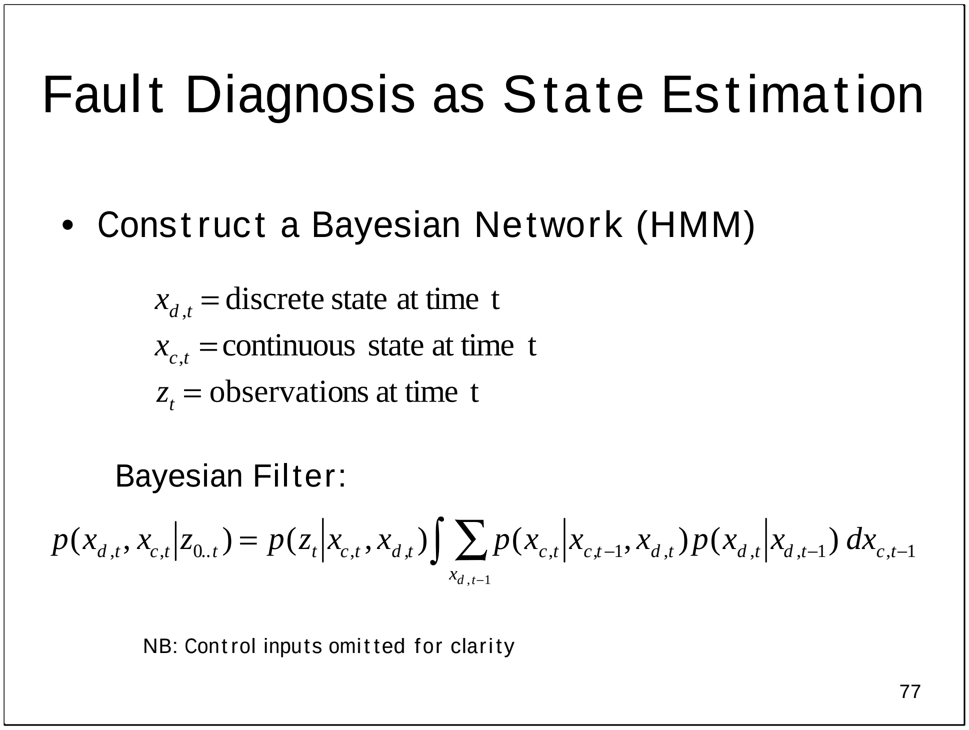


Fault Diagnosis as State Estimation

- Hybrid discrete/continuous Hidden Markov Model
- Discrete states represent operational modes and fault states
  - *E.g.* sending data, driving forward, right front wheel jammed, left rear wheel slipping, etc.
- Continuous states represent observable state of the rover
  - *E.g.* wheel speed, motor current, tilt angle
  - Measurements may be noisy

Since we have so many sources of uncertainty, it's natural for us to represent the problem using a probabilistic model. In this case, we choose a hybrid Hidden Markov Model where the discrete states correspond to functional modes and fault conditions in the rover. These are the hidden states in the model since we cannot measure them directly.

The continuous states are the things that we *can* measure such as wheel speed or motor current. These states are needed because we assume that the sensors are noisy and therefore the observations are only approximations of this continuous state.



Fault Diagnosis as State Estimation

- Construct a Bayesian Network (HMM)

$x_{d,t}$ = discrete state at time t
$x_{c,t}$ = continuous state at time t
z_t = observations at time t

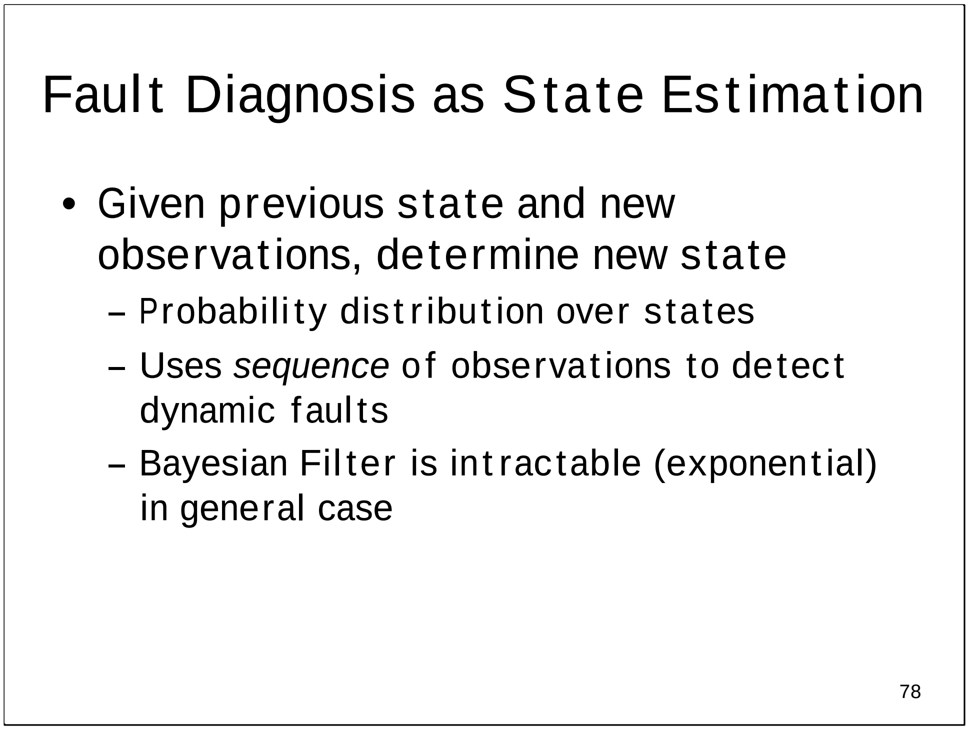
Bayesian Filter:

$$p(x_{d,t}, x_{c,t} \mid z_{0..t}) = p(z_t \mid x_{c,t}, x_{d,t}) \int \sum_{x_{d,t-1}} p(x_{c,t} \mid x_{c,t-1}, x_{d,t})\, p(x_{d,t} \mid x_{d,t-1})\, dx_{c,t-1}$$

NB: Control inputs omitted for clarity

77

Now we can construct a Bayesian network using these states and the probabilities of transitioning between them. The equations are all similar to what was presented in the first part of the talk except that our state has both discrete and continuous components. The Bayesian filter equation changes slightly because the new discrete state depends only on the previous discrete state while the new continuous state depends on the previous continuous state and the new discrete state. Essentially, the discrete state changes independently while the continuous state depends on the discrete state. Note that I have omitted the control inputs from my equations for clarity but they can be added in just as they were in the earlier equations.



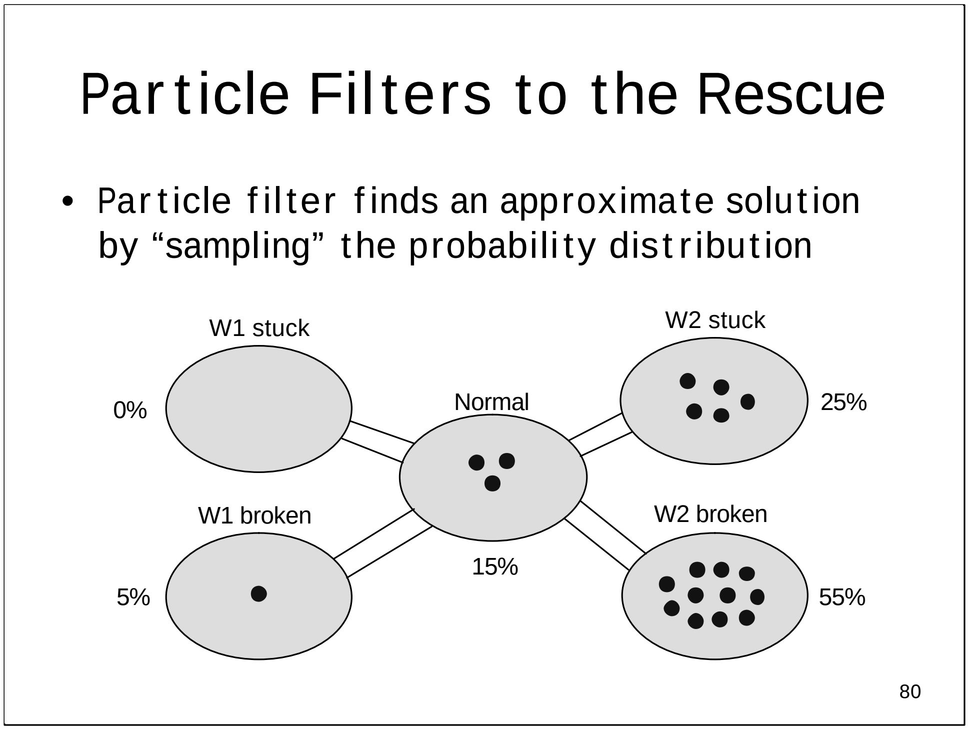
Fault Diagnosis as State Estimation

- Given previous state and new observations, determine new state
  - Probability distribution over states
  - Uses *sequence* of observations to detect dynamic faults
  - Bayesian Filter is intractable (exponential) in general case

Using this type of model, we are able to track the current state and thereby watch for faults. At each time step, we calculate the new state based on the previous state and a new set of observations. However, by doing fault detection using a Hidden Markov Model, we actually get something better. We get a probability distribution across all the possible states. This is convenient since it feeds directly into certain types of planners such as the POMDP planners discussed in an earlier lecture. The Hidden Markov Model also incorporates a series of observations into its estimate allowing us to detect those dynamic faults I mentioned. However, the problem is that there is no closed-form solution to the Bayesian filter equation when using arbitrary probability distributions.

Outline

- Introduction to Particle Filters
- Particle Filters in SLAM: FastSLAM
- Particle Filters in Rover Fault Diagnosis
  - Challenges for rover fault diagnosis
  - Fault diagnosis as state estimation
  - Approximating a solution using particle filtering
  - Particle filter enhancements
    - Risk Sensitive Particle Filter
    - Variable Resolution Particle Filter

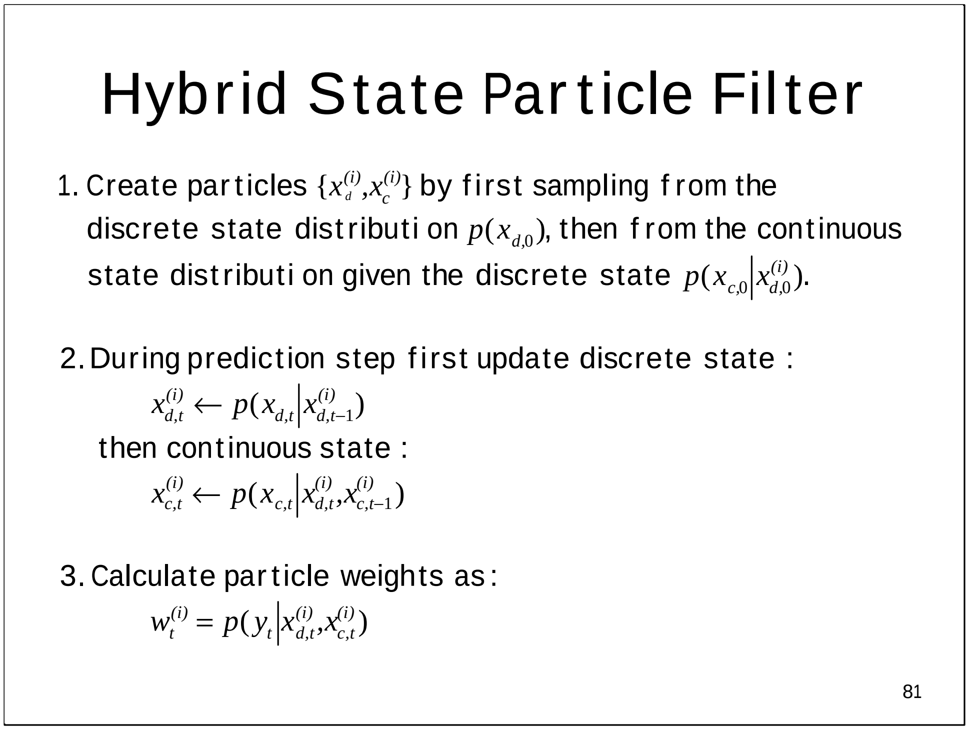


Particle Filters to the Rescue

- Particle filter finds an approximate solution by “sampling” the probability distribution

W1 stuck
0%
W2 stuck
25%
Normal
15%
W1 broken
5%
W2 broken
55%

So, naturally, we’ll use particle filters to find an approximate solution. Here is a simple example model where we have one “normal” state and four “fault” states. To find the probability that we’re in a particular state, we simply count the particles in that state and divide by the total number of particles in the system.



Hybrid State Particle Filter

1. Create particles $\{x_d^{(i)}, x_c^{(i)}\}$ by first sampling from the discrete state distribution $p(x_{d,0})$, then from the continuous state distribution given the discrete state $p(x_{c,0} | x_{d,0}^{(i)})$.

2. During prediction step first update discrete state:

   $$x_{d,t}^{(i)} \leftarrow p(x_{d,t} | x_{d,t-1}^{(i)})$$

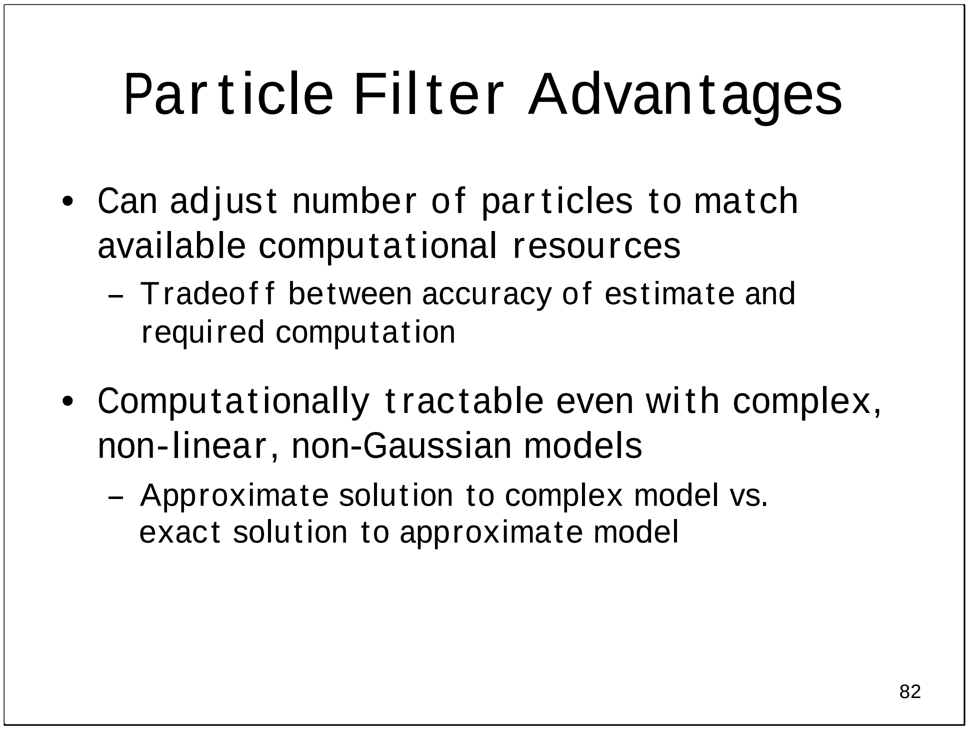
   then continuous state:

   $$x_{c,t}^{(i)} \leftarrow p(x_{c,t} | x_{d,t}^{(i)}, x_{c,t-1}^{(i)})$$

3. Calculate particle weights as:

   $$w_t^{(i)} = p(y_t | x_{d,t}^{(i)}, x_{c,t}^{(i)})$$

The calculations for the particle filter are the same as those presented earlier except for the way the two different types of state are handled. Again, the continuous states are dependent on the discrete state. Therefore, when we create the initial particles, we sample from the discrete state prior distribution first and then sample from the continuous state distribution given the discrete state that we selected. Similarly, during the prediction step of the algorithm, we update the discrete state first and then update the continuous state based on the new discrete state. The importance weights are calculated using the entire state, as before.



Particle Filter Advantages

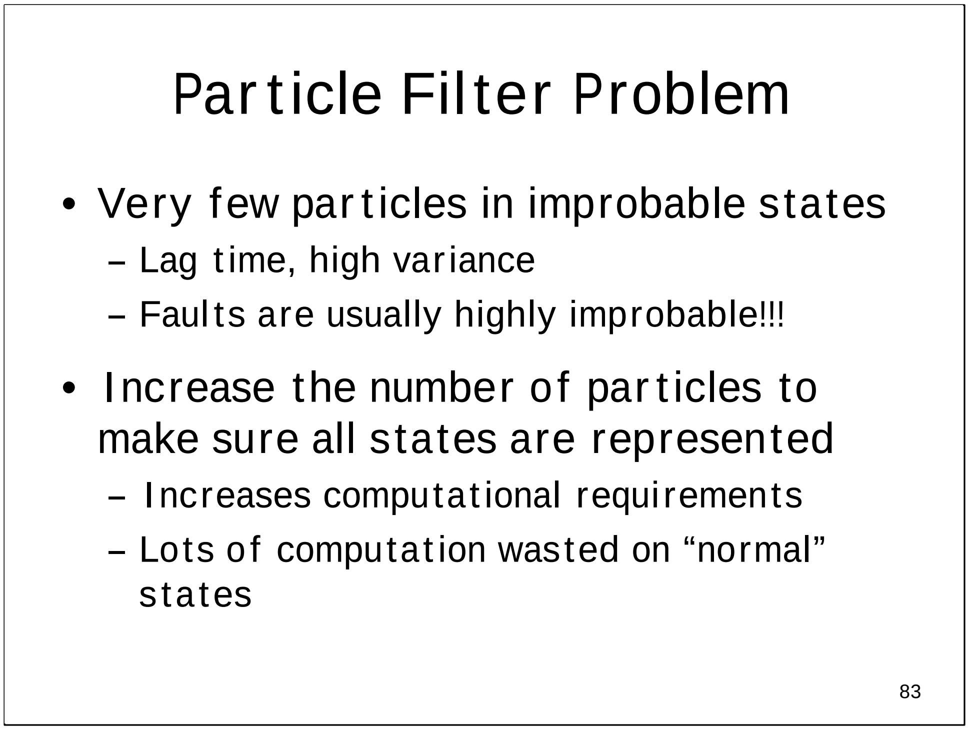
- Can adjust number of particles to match available computational resources
  - Tradeoff between accuracy of estimate and required computation
- Computationally tractable even with complex, non-linear, non-Gaussian models
  - Approximate solution to complex model vs. exact solution to approximate model

What do we gain by using a particle filter approach to fault diagnosis?

First, we can easily adjust the amount of computation that we need to do. By increasing or decreasing the number of particles in the system, we can trade off the accuracy of our estimates for faster results. If we reduce the number of particles, we have less work to do but our approximation will not be as close to the correct answer. In fact, we can even adjust the number of particles on-the-fly as conditions permit. For example, a rover may be able to change the frequency of its processor depending on the amount of power that’s available. Then, when a lot of light is falling on its solar panels, it can use a lot of particles and get an accurate approximation and when it’s cloudy it can scale back to fewer particles and reduce its accuracy.

This works because we’ve retained the complex model of the system and are just approximating a solution with varying degrees of accuracy. This is in contrast to approaches where the model is simplified (e.g. by assuming Gaussian distributions) to deal with the intractability. In that case you’re stuck with whatever approximations you’ve made and can’t improve your accuracy on-demand.

For these reasons, particle filtering seems to be a good fit for rover applications. The complex model can be created offline, before the rover is sent out, and the rover need only perform the relatively simple particle update calculations in the field. Once it is deployed, the rover can tailor its accuracy



Particle Filter Problem

- Very few particles in improbable states
  - Lag time, high variance
  - Faults are usually highly improbable!!!
- Increase the number of particles to make sure all states are represented
  - Increases computational requirements
  - Lots of computation wasted on “normal” states

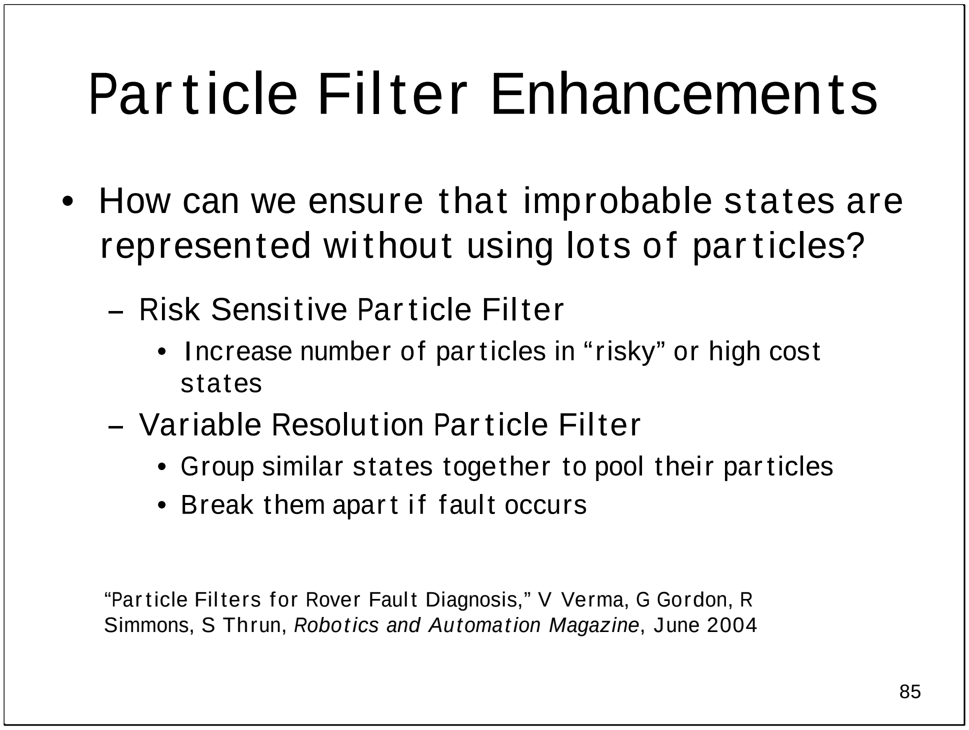
However, there’s one problem with using particle filters and that is that improbable states have very few or no particles. This can create two problems. First, there can be a delay between when an event occurs and when the corresponding state becomes likely. Normally, when something happens, our observations will cause the particles in a particular state to be weighted more heavily and they will multiply. However, if that state has no particles in it, we have to wait for at least one particle to randomly transition into it before it can start multiplying. This can take a long time if that particular transitional probability is very low. Second, we can get a high variance in the estimate for a state because the probability represented by a single particle is larger that the probability we’re trying to estimate. Therefore, it will tend to flip back and forth between zero and one particles rather than settling on a consistent intermediate value. Of course, the reason these are problems is that faults are usually improbable.

The obvious solution is to increase the number of particles so that each particle represents a smaller probability. The problem with this is that you may need an enormous number of particles to represent very small probabilities. This will drastically increase the amount of computation you have to do. Plus, all of the states will get more particles so most of the additional computation will be wasted updating particles in the states that already had lots of particles.

(Note: I considered adding another slide to better explain the lag time problem but everyone that I asked about it said that they got it just fine from what I said.)

Outline

- Introduction to Particle Filters
- Particle Filters in SLAM: FastSLAM
- Particle Filters in Rover Fault Diagnosis
  - Challenges for rover fault diagnosis
  - Fault diagnosis as state estimation
  - Approximating a solution using particle filtering
  - Particle filter enhancements
    - Risk Sensitive Particle Filter
    - Variable Resolution Particle Filter



Particle Filter Enhancements

- How can we ensure that improbable states are represented without using lots of particles?
  - – Risk Sensitive Particle Filter
    - Increase number of particles in “risky” or high cost states
  - – Variable Resolution Particle Filter
    - Group similar states together to pool their particles
    - Break them apart if fault occurs

“Particle Filters for Rover Fault Diagnosis,” V Verma, G Gordon, R Simmons, S Thrun, *Robotics and Automation Magazine*, June 2004

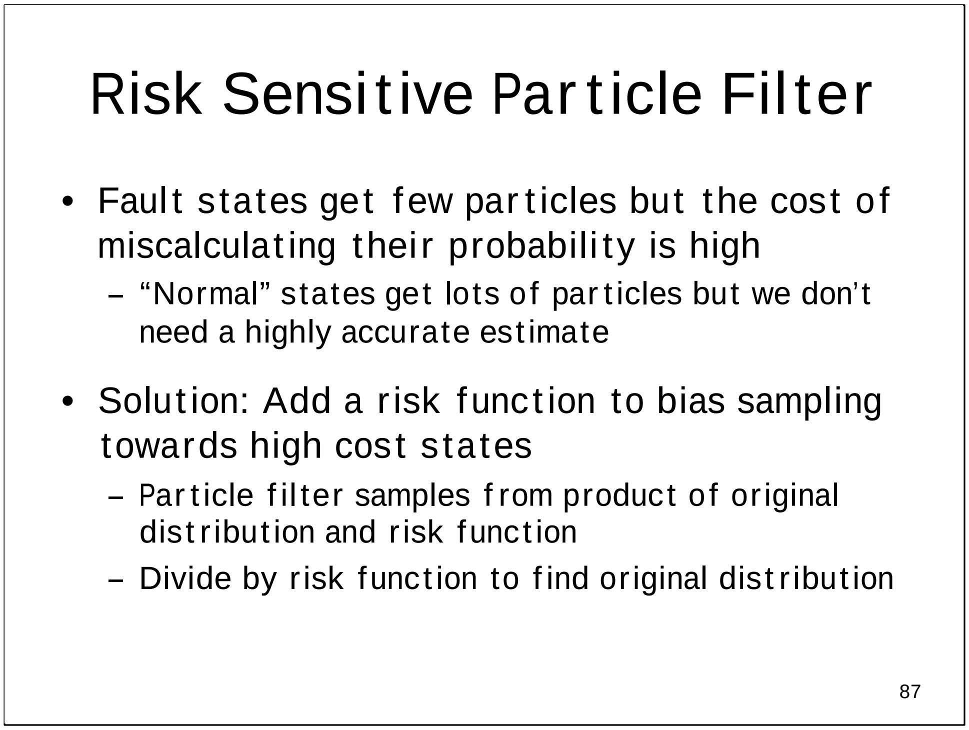
There are various approaches to addressing this problem in the literature. I’m going to show you two approaches presented by Verma et al. in the paper reference here.

The goal of these enhancements is to make sure that all the states in a system are represented by at least a few particles while still keeping the total number of particles small.

(The rest of the slide is pretty self-explanatory.)

Outline

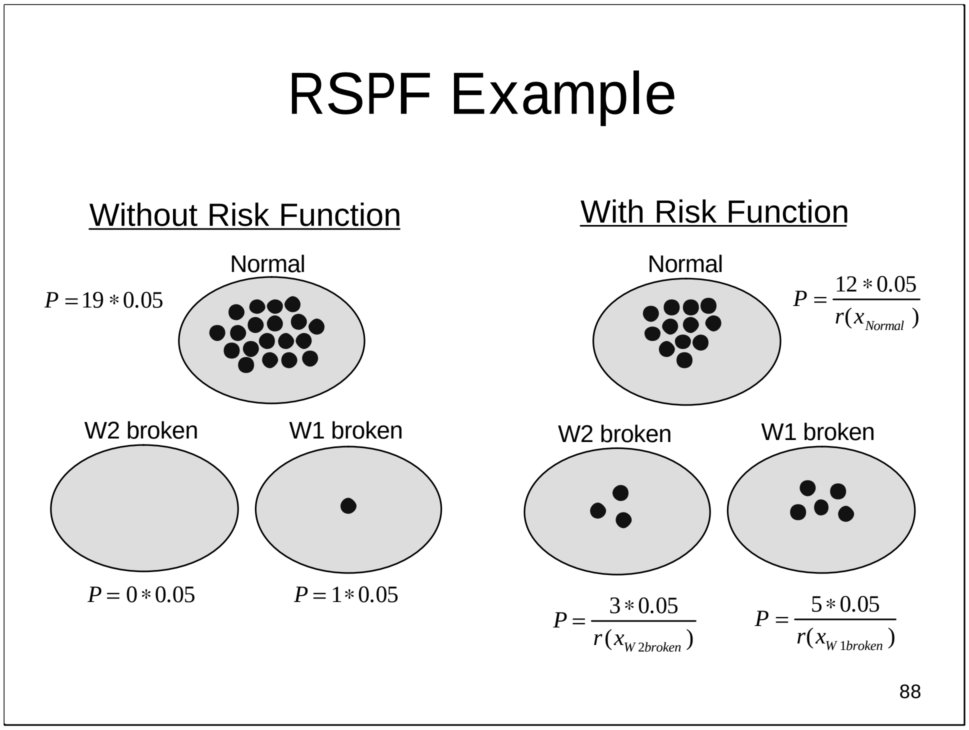
- Introduction to Particle Filters
- Particle Filters in SLAM: FastSLAM
- Particle Filters in Rover Fault Diagnosis
  - Challenges for rover fault diagnosis
  - Fault diagnosis as state estimation
  - Approximating a solution using particle filtering
  - Particle filter enhancements
    - Risk Sensitive Particle Filter
    - Variable Resolution Particle Filter



Risk Sensitive Particle Filter

- Fault states get few particles but the cost of miscalculating their probability is high
  - “Normal” states get lots of particles but we don’t need a highly accurate estimate
- Solution: Add a risk function to bias sampling towards high cost states
  - Particle filter samples from product of original distribution and risk function
  - Divide by risk function to find original distribution

(Self-explanatory)



RSPF Example

Without Risk Function

Normal

$P = 19 * 0.05$

W2 broken

W1 broken

$P = 0 * 0.05$

$P = 1 * 0.05$

With Risk Function

Normal

$P = \frac{12 * 0.05}{r(x_{Normal})}$

W2 broken

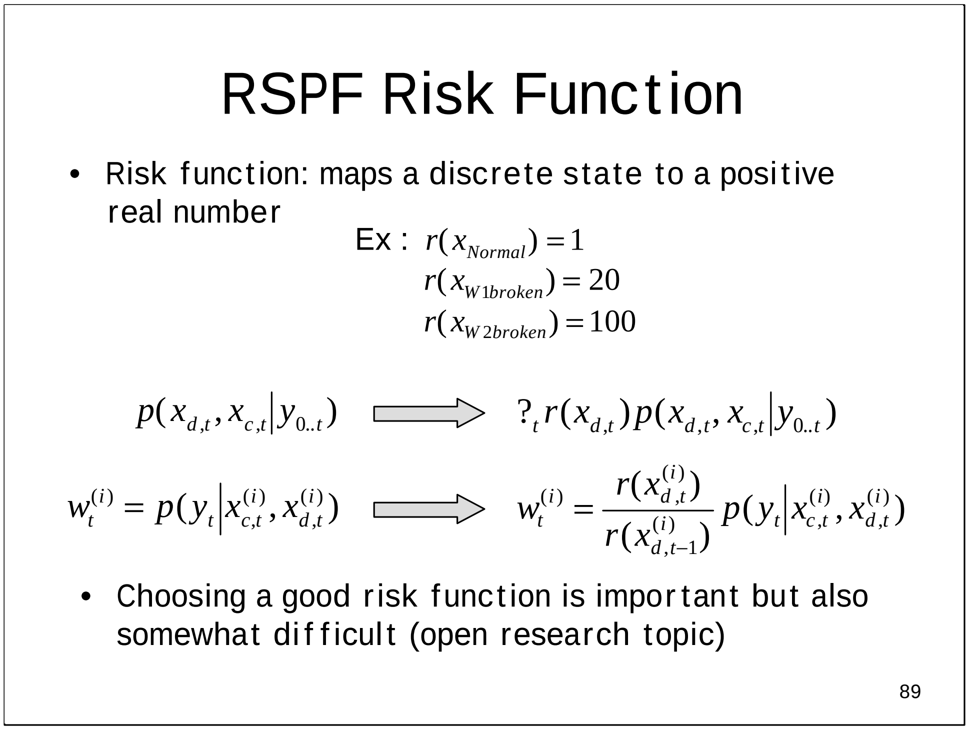
W1 broken

$P = \frac{3 * 0.05}{r(x_{W2broken})}$

$P = \frac{5 * 0.05}{r(x_{W1broken})}$

Here’s a simple example of the effect of the risk function. Without the risk function, the normal state gets almost all of the particles while the fault states are poorly represented. The estimated probability of each state can be calculated directly from the number of particles in each state.

With the risk function, the sampling is biased so that the fault states get more particles. We then need to divide by the bias in order to get the actual probability distribution that we are trying to estimate.



RSPF Risk Function

- Risk function: maps a discrete state to a positive real number

Ex : $r(x_{Normal}) = 1$
$r(x_{W1broken}) = 20$
$r(x_{W2broken}) = 100$

$$p(x_{d,t}, x_{c,t} | y_{0..t}) \Longrightarrow ?_t r(x_{d,t}) p(x_{d,t}, x_{c,t} | y_{0..t})$$

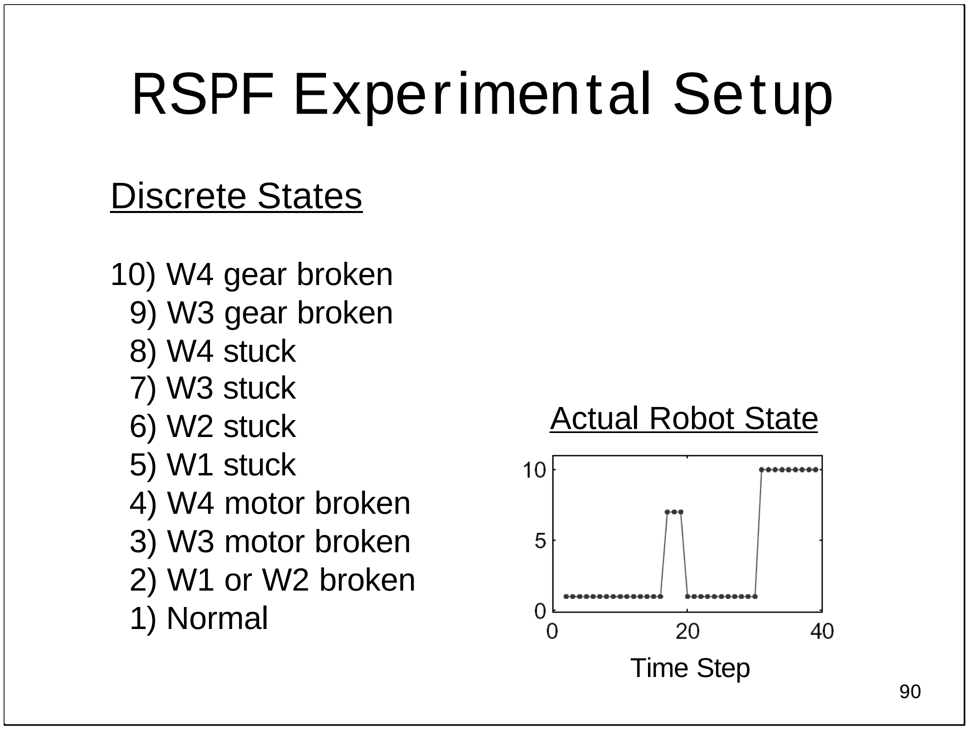
$$w_t^{(i)} = p(y_t | x_{c,t}^{(i)}, x_{d,t}^{(i)}) \Longrightarrow w_t^{(i)} = \frac{r(x_{d,t}^{(i)})}{r(x_{d,t-1}^{(i)})} p(y_t | x_{c,t}^{(i)}, x_{d,t}^{(i)})$$

- Choosing a good risk function is important but also somewhat difficult (open research topic)

The risk function simply maps a discrete state to a positive number. As an example, we might assign a value of “1” to the normal state. Then we assign larger values to the fault states to help ensure that they get enough particles. Let’s say the W1 is a drive wheel and W2 is a steering wheel. If W1 were to break (and we didn’t notice) we would probably just stop. Whereas if W2 were to break and we didn’t notice, we might veer off in a random direction and drive off a cliff. Therefore, we assign a higher value to W2 because it could have a higher cost if we miss it.

The top arrow shows the distribution that we were originally trying to estimate on the left and the new distribution that we want to estimate on the right. Gamma is a normalization constant which ensures that the expression on the right is a probability distribution. To switch from the first distribution to the second, we only need to draw our original samples from the product of the risk function and the initial distribution and then modify the importance weight calculation as shown by the second arrow.

Clearly, the key to making this work well is finding a good risk function. Unfortunately, this is still a topic of active research so there is currently no accepted way to do this. An expert may be able to guess fairly decent risk function. The authors also referred to another paper which used a POMDP planner to create a risk function based on the potential future cost of inaccurately tracking each state.



RSPF Experimental Setup

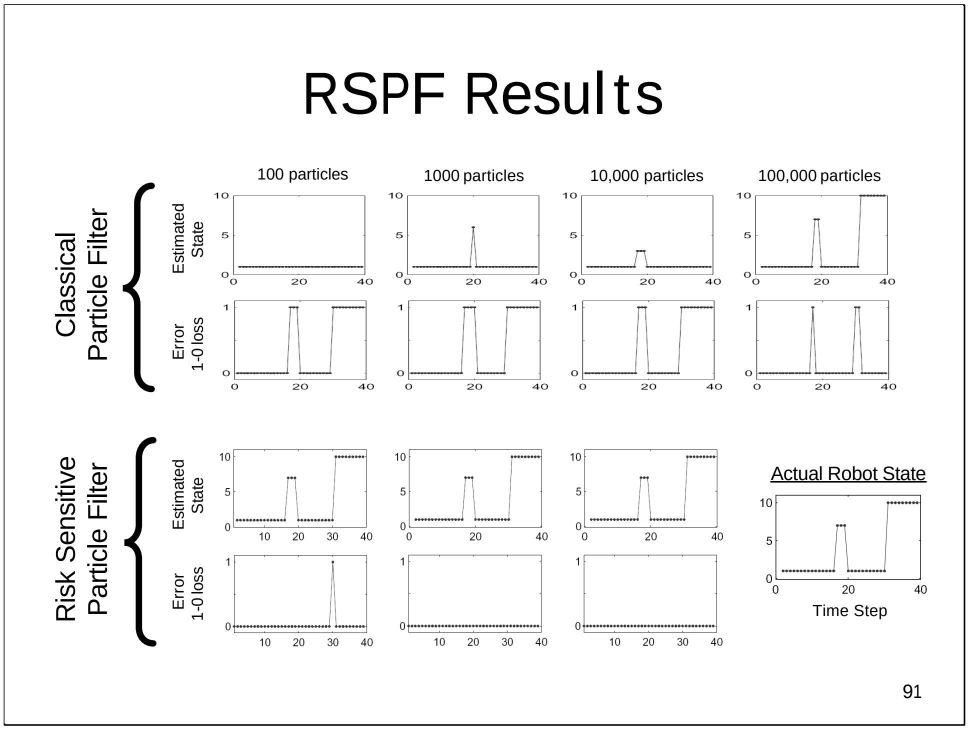
Discrete States

10) W4 gear broken
9) W3 gear broken
8) W4 stuck
7) W3 stuck
6) W2 stuck
5) W1 stuck
4) W4 motor broken
3) W3 motor broken
2) W1 or W2 broken
1) Normal

Actual Robot State

10
5
0
0
20
40
Time Step

Now, I 'll talk about an experiment from the paper where they compared a risk sensitive particle filter to a classical one. In this case, they were modeling a four wheeled rover with 10 discrete states; one normal state and nine fault states. The states are arbitrarily numbered. The graph shows the actual state of the robot as it changes during the simulation. It starts in the normal state and then, at time step 17, wheel 3 becomes stuck against a rock. They then tell the robot to back up, wheel 3 becomes unstuck and the rover operates normally until time step 30 where the gear on wheel 4 breaks. It then remains broken for the rest of the simulation.



RSPF Results

91

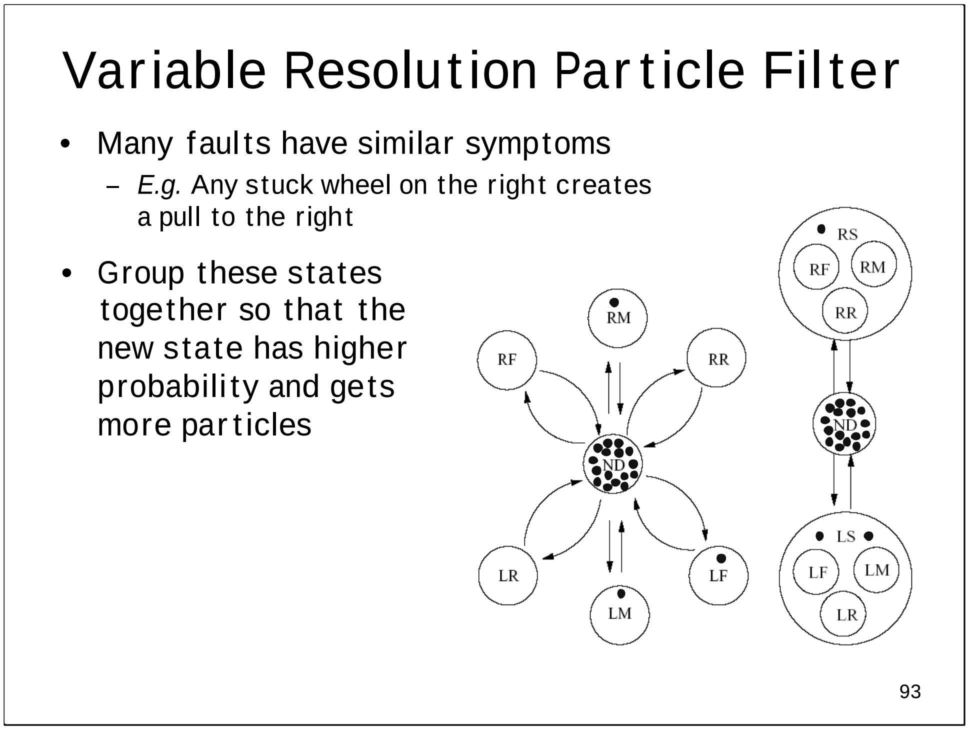
The top row of each set shows the estimated state of the robot during the simulation. Here we see only the most likely state, not the full distribution. The bottom row shows the error in the estimate; “0” if the estimate is correct and “1” if it is wrong.

The classical particle filter does very poorly with anything less than 100,000 particles, either missing faults completely or detecting the wrong fault. With 100,000 particles, the estimate is close but there is still a lag between the fault occurring and the rover detecting it.

With the risk sensitive particle filter, the estimate is already very good with only 100 particles and is perfect with 1000 or more particles. The authors did show the actual risk function used for this experiment. They only say that it was derived “heuristically.”

Outline

- Introduction to Particle Filters
- Particle Filters in SLAM: FastSLAM
- Particle Filters in Rover Fault Diagnosis
  - Challenges for rover fault diagnosis
  - Fault diagnosis as state estimation
  - Approximating a solution using particle filtering
  - Particle filter enhancements
    - Risk Sensitive Particle Filter
    - Variable Resolution Particle Filter



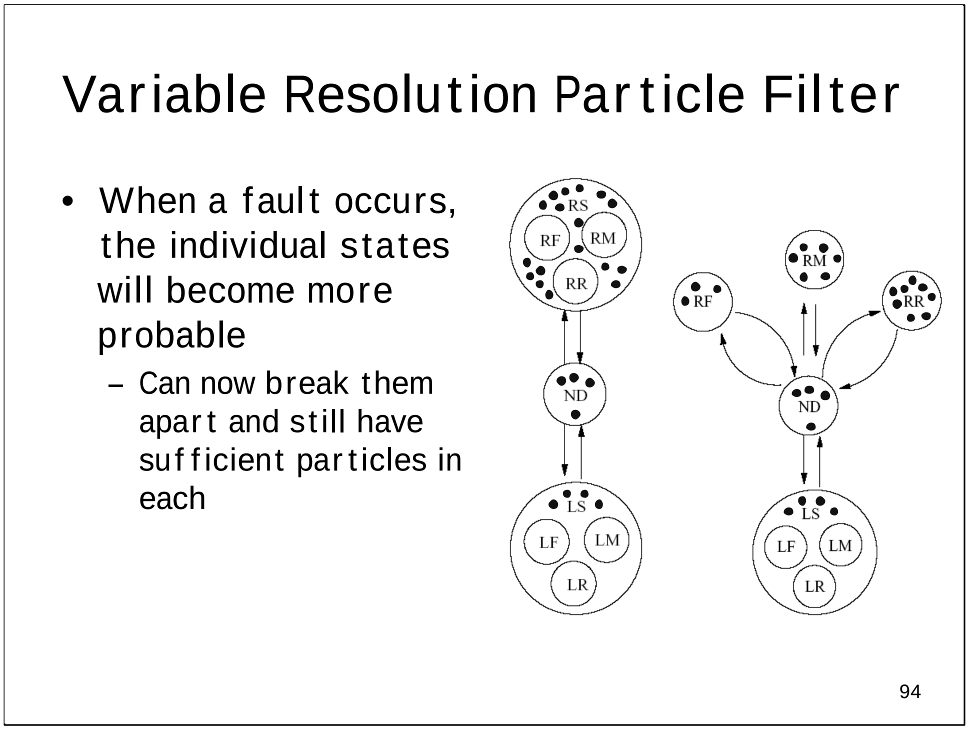
Variable Resolution Particle Filter

- Many faults have similar symptoms
  - *E.g.* Any stuck wheel on the right creates a pull to the right
- Group these states together so that the new state has higher probability and gets more particles

This enhancement is based on the observation that some faults have similar symptoms. In this case, we can group those faults together into one abstract state that contains the individual fault states. This hierarchical model is constructed ahead of time by the rover designer and therefore relies on an expert to appropriate group states together. In this example, we see a simple model for a six-wheeled rover. There is one normal state (ND) and one fault state for each of the six wheels being stuck. The states are labeled with an “L” or “R” for “left” or “right” and an “F”, “M” or “R” for “front”, “middle” or “rear.” In the higher-level model, all the “R” states are combined to form an abstract “RS” (for “right side”) state. The same is done of the left side.

Without using the variable resolution particle filter, the fault states are likely to have few or no particles. In this situation, when a fault does occur, the rover will likely pick the wrong fault. For example, if a fault occurred in one of the right wheels while there was only a particle in RM (as in the diagram on the left), the RM particle would multiply and the rover would decide that the fault was in RM. This happens because the symptoms for all the right side wheels are similar.

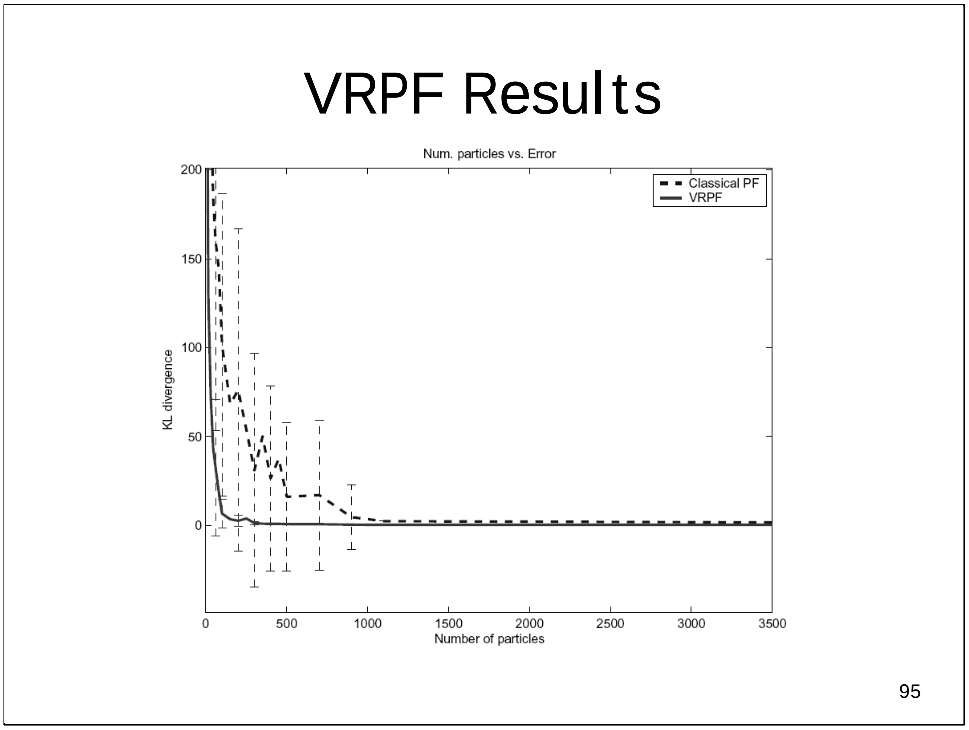
When the states for the left and right sides are grouped together, it effectively pools the particles from the individual states. This makes it more likely that the abstract state will be well-represented. It also means that the rover does not immediately try to pick a specific fault when something happens. Instead it detects that *some* fault has occurred but delays the determination of the specific fault.



Variable Resolution Particle Filter

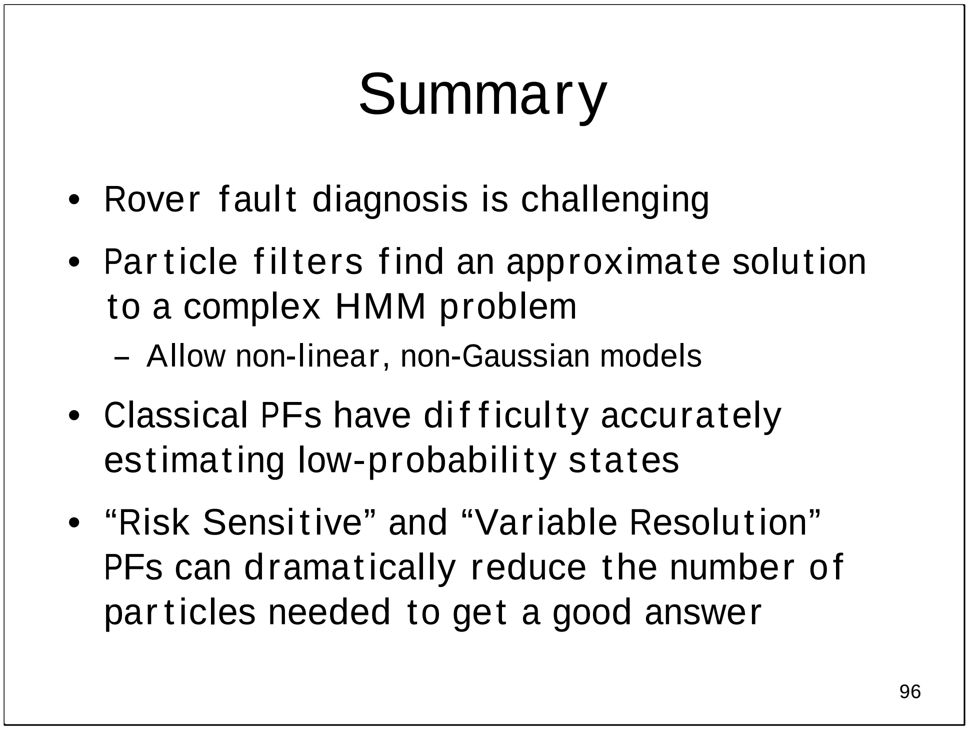
- When a fault occurs, the individual states will become more probable
  - Can now break them apart and still have sufficient particles in each

Once the rover has determined that a fault has occurred in one of the abstract states, it can switch to a more refined model for that state to determine exactly where the fault occurred. The particles from the large abstract state are distributed to the individual states according to their prior probabilities and the algorithm continues. In this example, since sufficient particles have accumulated in RS it is decomposed into RF, RM and RR and its particles are distributed among them.



VRPF Results

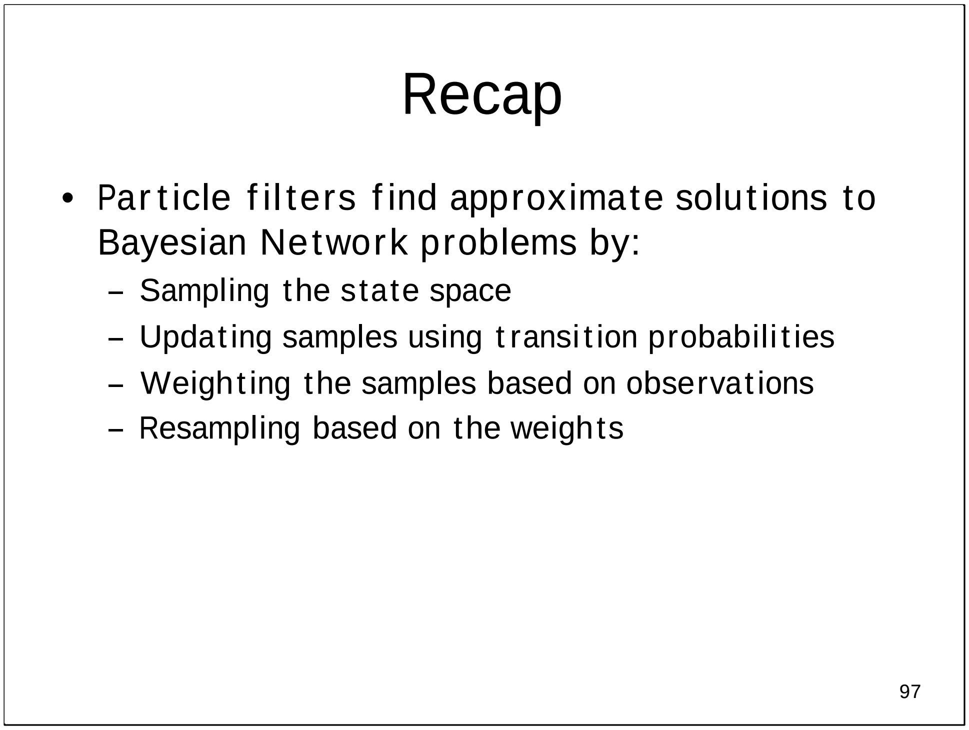
These results were collected using a model similar to the one in the previous slides. Many simulations were run with varying numbers of particles. For each experiment the KL divergence between the estimated probability distribution and the actual probability distribution was computed. (A particle filter with 1,000,000 particles was used as the actual distribution.) For large numbers of particles, both the classical and variable resolution particle filters performed comparably. This is because there were sufficient particles to populate the faults states, even without using the variable resolution enhancement. However, for small numbers of particles, the variable resolution particle filter had much lower divergence as well as lower variance (as indicated by the error bars).



Summary

- Rover fault diagnosis is challenging
- Particle filters find an approximate solution to a complex HMM problem
  - Allow non-linear, non-Gaussian models
- Classical PFs have difficulty accurately estimating low-probability states
- “Risk Sensitive” and “Variable Resolution” PFs can dramatically reduce the number of particles needed to get a good answer

In summary, fault diagnosis is difficult for rovers. The models can be very complex and the computational resources are very limited. Particle filters allow us to find an approximate solution with whatever resources are available. However, naively applying particle filtering to fault diagnosis can lead to poor tracking of the most important states. Risk sensitive and variable resolution particle filters are two techniques that can dramatically improve fault detection while keeping the computational complexity low.

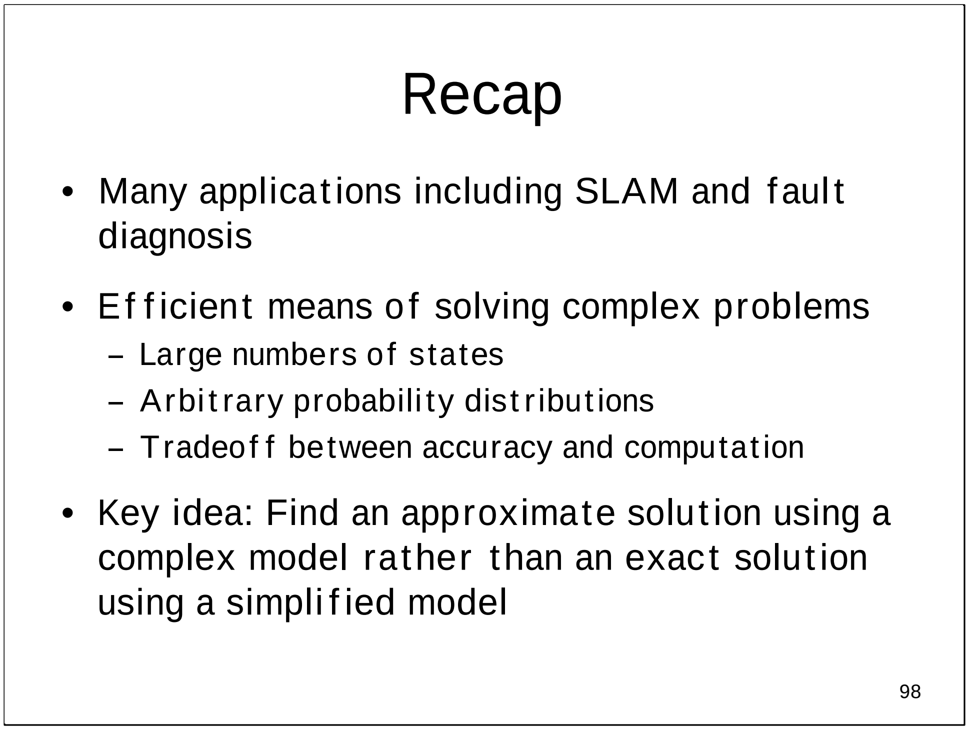


Recap

- Particle filters find approximate solutions to Bayesian Network problems by:
  - Sampling the state space
  - Updating samples using transition probabilities
  - Weighting the samples based on observations
  - Resampling based on the weights

Hopefully in this lecture you’ve gotten an idea of how particle filters work and the types of problems they can solve efficiently.

(Reiterate each step.)



Recap

- Many applications including SLAM and fault diagnosis
- Efficient means of solving complex problems
  - Large numbers of states
  - Arbitrary probability distributions
  - Tradeoff between accuracy and computation
- Key idea: Find an approximate solution using a complex model rather than an exact solution using a simplified model

We’ve shown you SLAM and fault diagnosis but there are also many other applications of particle filters.

They are particularly useful when your (Bayesian network) problem has a large number of states or requires arbitrary probability distributions either in the results or in the model. They can also be useful when you want to be able to tradeoff (possibly on-the-fly) between accuracy and computational complexity.

Bibliography

- Particle Filters
  - Rekleitis, Ioannis, "A Particle Filter Tutorial for Mobile Robot Localization," International Conference on Robotics and Automation (ICRA) 2003.
- Fast SLAM
  - Montemerlo, Thrun, Koller, Wegbreit, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," Proceedings of the AAAI National Conference on Artificial Intelligence, 2002.
- Particle Filters in Robot Fault Diagnosis
  - V Verma, G Gordon, R Simmons, S Thrun, "Particle Filters for Rover Fault Diagnosis," *Robotics and Automation Magazine*, June 2004