

EECE 5550 Mobile Robotics

Lecture 10: Localization

Derya Aksaray

Assistant Professor

Department of Electrical and Computer Engineering

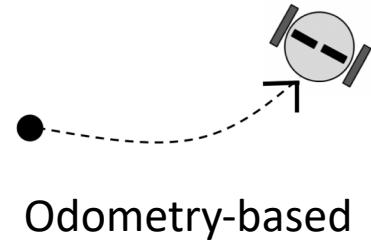


Northeastern
University

Taxonomy of localization problems

Dead-reckoning: initial position known

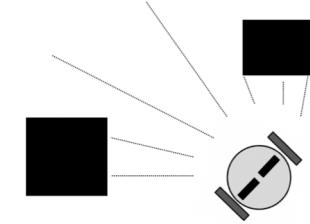
- Blindly update pose based on differential movements



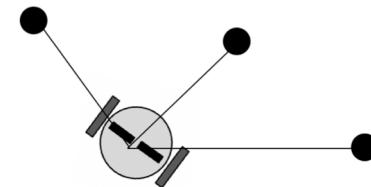
Odometry-based

Global localization: initial position can be unknown

- Map-based (with landmarks)
 - Sensors: laser, camera, etc.
 - Method: map-matching techniques (various)
- Beacon-based (with active infrastructure)
 - Bluetooth, WiFi, GPS (outdoor only), etc.
 - Method: trilateration, fingerprinting, etc.



Map-based



Beacon-based

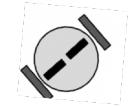
Challenges in localization problems

Dead-reckoning

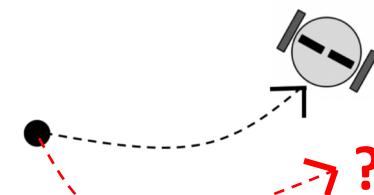
- Wheel slip, slack in actuation mechanism
- runaway (accumulative errors)

Global localization

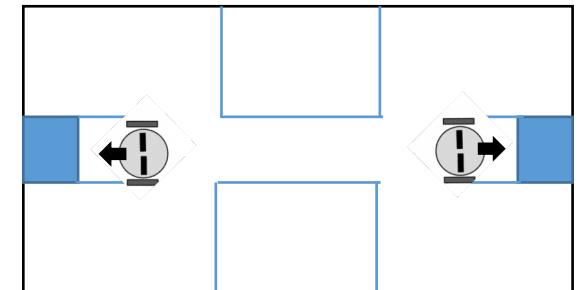
- Random errors and failures
- Non-Gaussian sensor noise
- Unavailability of sensor (e.g., GPS denial)
- Map ambiguity
- Dynamic environments
- Kidnapped robot problem



Kidnapped robot



Dead-reckoning error



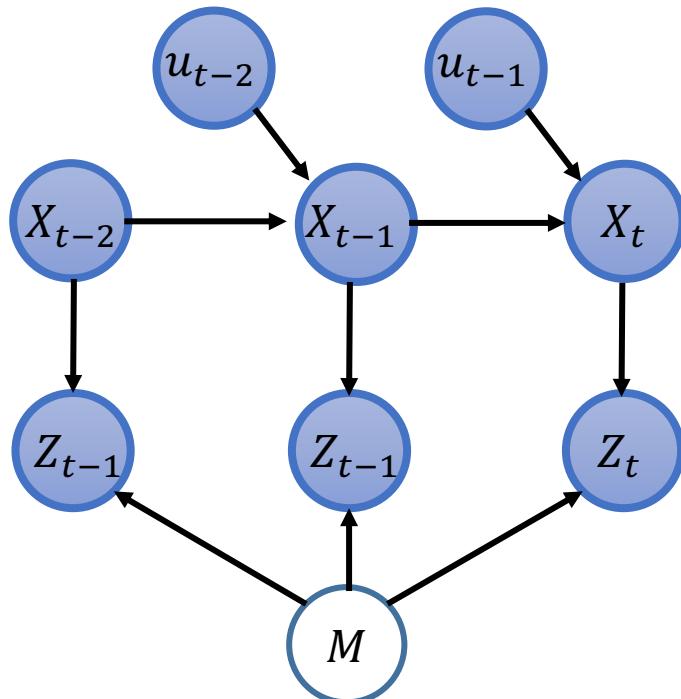
Map ambiguity

Probabilistic approaches

Last time: mapping

Given: Robot poses $x_{0:t}$, measurements $z_{1:t}$

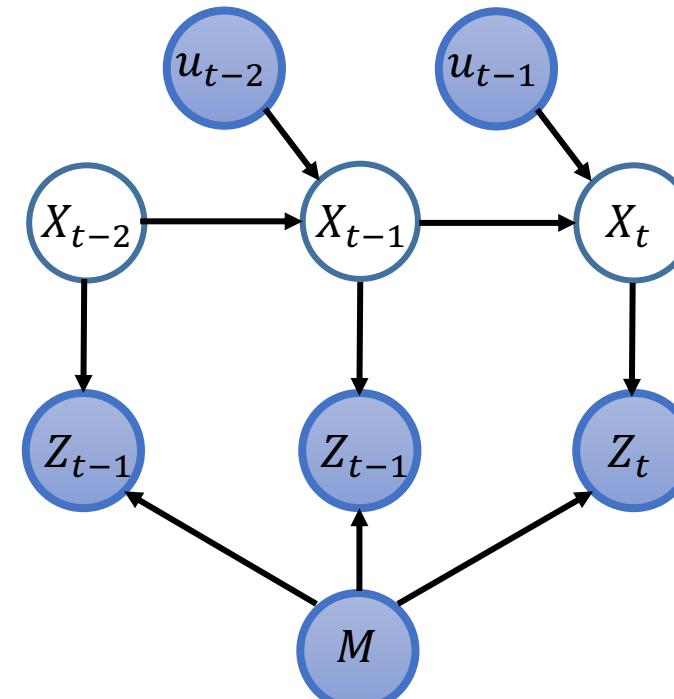
Estimate: Belief $p(m|x_{0:t}, z_{1:t})$ over the map M



This time: localization

Given: Prior $p(x_0)$, map m , controls $u_{0:t}$, obs. $z_{1:t}$

Estimate: Belief $p(x_t|m, z_{1:t})$ over the robot pose

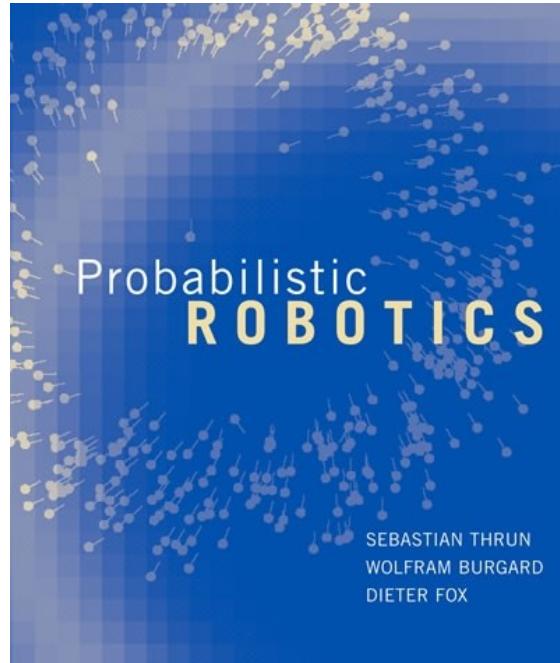


Plan of the day

- Monte Carlo localization
- Models:
 - Probabilistic motion models
 - Forward beam sensor models
- Practicalities:
 - Particle diversity / depletion
 - Tracking localization performance
 - Adaptive sample sets: KLD-sampling



References



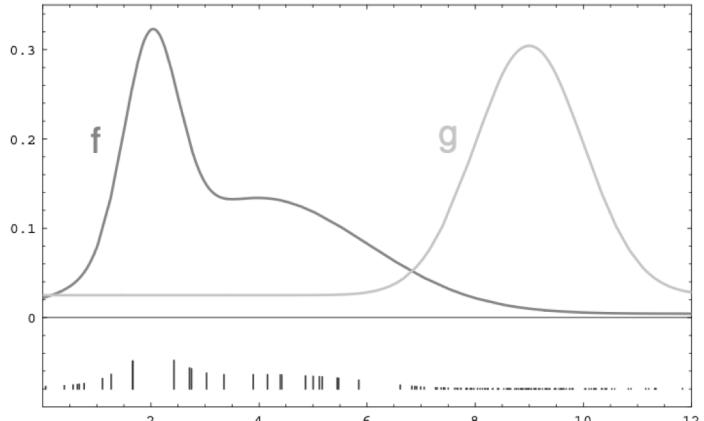
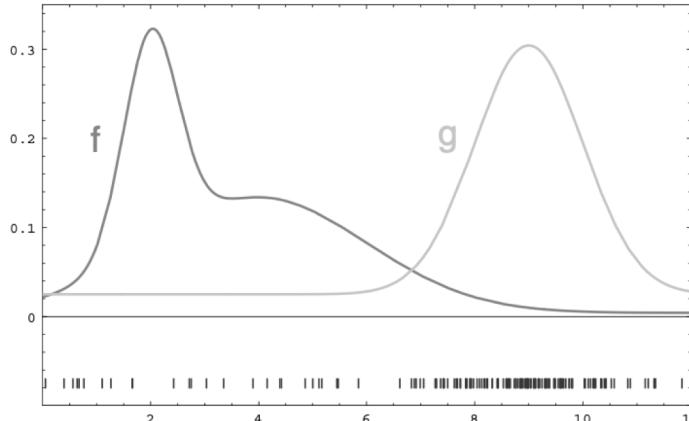
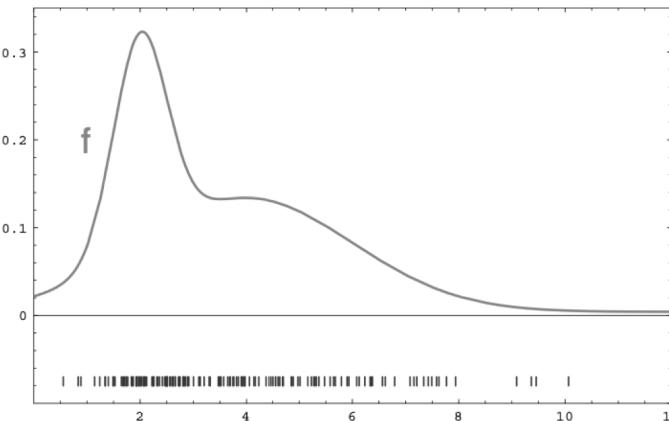
Chapters 5, 6, and 8

Monte Carlo Localization

Main idea: Monte Carlo localization (MCL) is simply localization using the Particle Filter.

Three keys steps in particle filter:

1. Sample particles from the proposal distribution
2. Compute the importance factors (weights)
3. Re-sample the particles: replace unlikely samples with more likely ones.



Monte Carlo Localization

Main idea: Monte Carlo localization (MCL) is simply localization using the Particle Filter.

Three keys steps in particle filter:

1. Sample particles from the proposal distribution
2. Compute the importance factors (weights)
3. Re-sample the particles: replace unlikely samples with more likely ones.

Particle filter applied to localization:

1. Proposal distribution is given by the motion model: $x_t^m \sim p(x_t | x_{t-1}, u_t)$
2. Particles are weighted by the measurement model:

$$w_t^m \sim p(z_t | x_t) \propto \frac{\text{target}}{\text{proposal}}$$

3. Particles are resampled

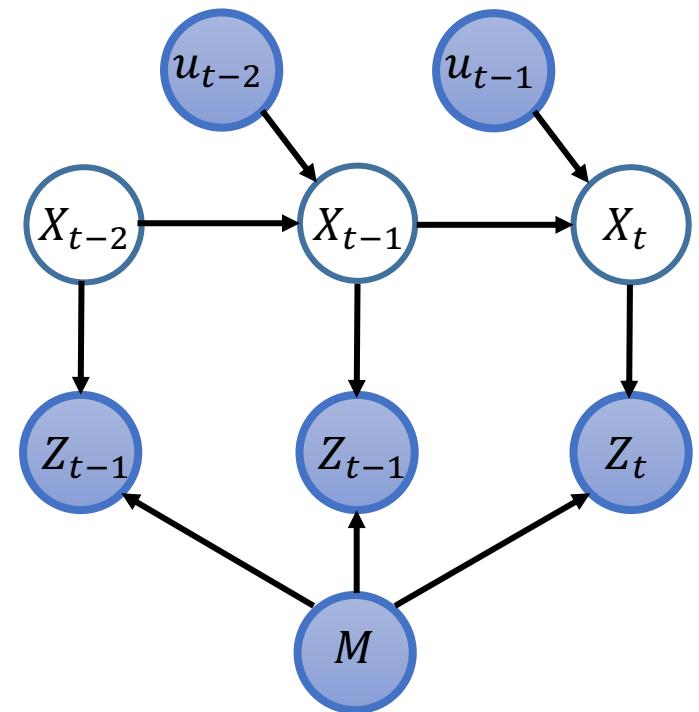
Monte Carlo Localization

Main idea: Monte Carlo localization (MCL) is simply localization using the Particle Filter.

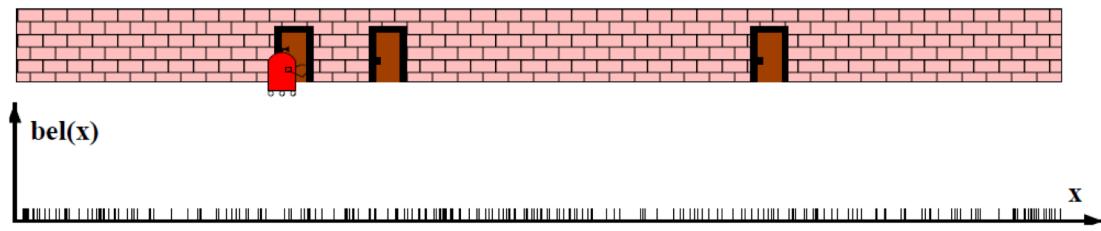
Given: Prior $p(x_0)$, map m , controls $u_{0:t}$, observations $z_{1:t}$

Estimate: Belief $p(x_t | m, z_{1:t})$ over the robot pose

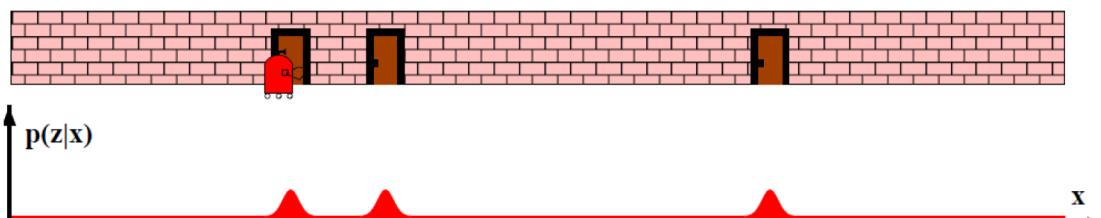
```
1:   Algorithm Particle_filter( $\mathcal{X}_{t-1}, u_t, z_t$ ):
2:      $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
3:     for  $m = 1$  to  $M$  do
4:       sample  $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$ 
5:        $w_t^{[m]} = p(z_t | x_t^{[m]})$ 
6:        $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
7:     endfor
8:     for  $m = 1$  to  $M$  do
9:       draw  $i$  with probability  $\propto w_t^{[i]}$ 
10:      add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
11:    endfor
12:  return  $\mathcal{X}_t$ 
```



Example application



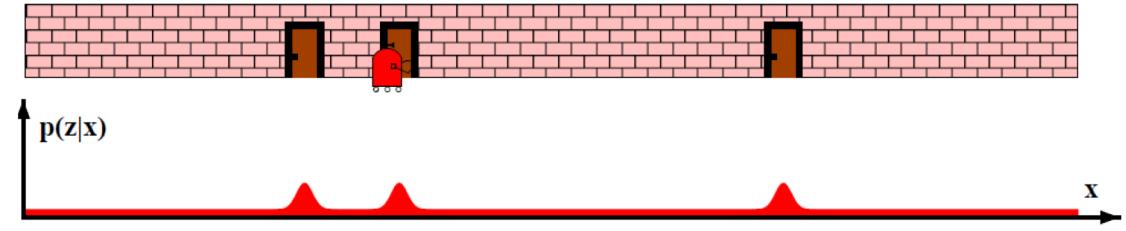
Initial belief



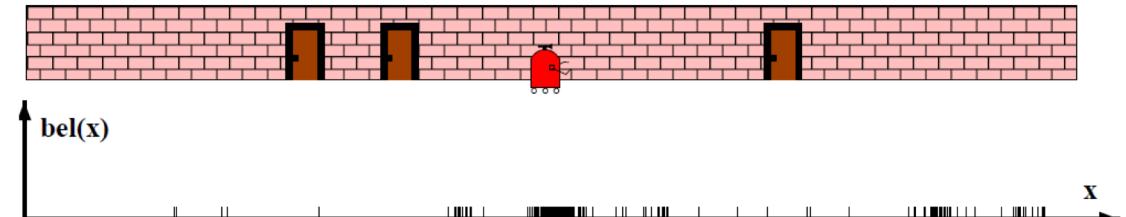
Measurement



Move right



Measurement



Move right

Monte Carlo Localization

Main idea: Monte Carlo localization (MCL) is simply localization using the Particle Filter.

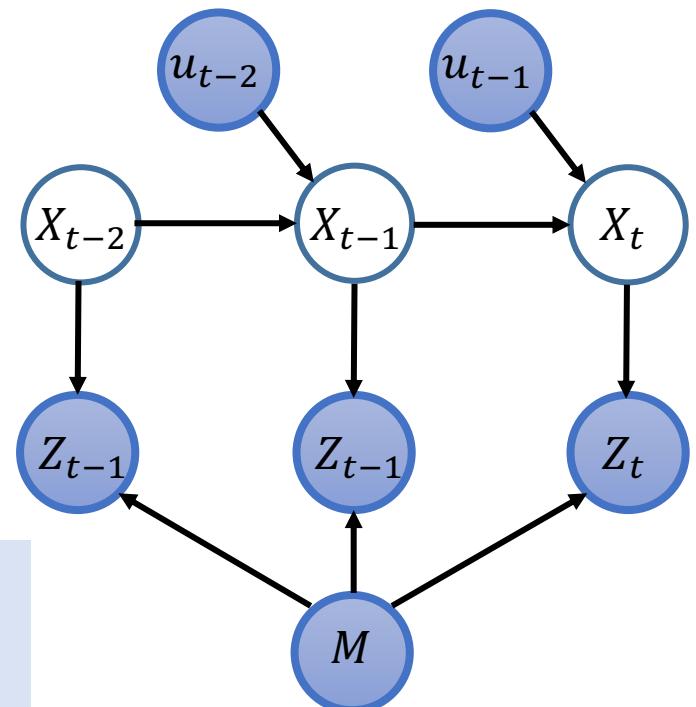
Given: Prior $p(x_0)$, map m , controls $u_{0:t}$, observations $z_{1:t}$

Estimate: Belief $p(x_t | m, z_{1:t})$ over the robot pose

```
1: Algorithm Particle_filter( $\mathcal{X}_{t-1}, u_t, z_t$ ):
2:    $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
3:   for  $m = 1$  to  $M$  do
4:     sample  $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$  ← Motion model
5:      $w_t^{[m]} = p(z_t | x_t^{[m]})$  ← Sensing model
6:      $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
7:   endfor
8:   for  $m = 1$  to  $M$  do
9:     draw  $i$  with probability  $\propto w_t^{[i]}$ 
10:    add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
11:   endfor
12:   return  $\mathcal{X}_t$ 
```

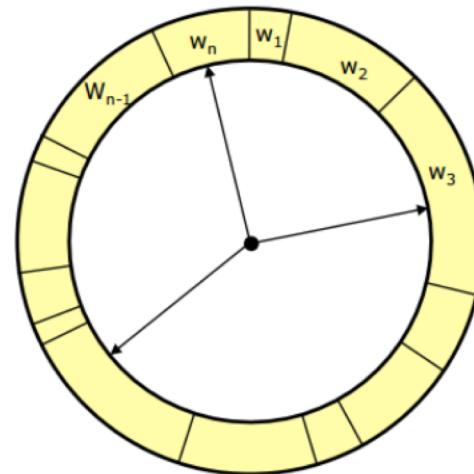
Requires:

- A *sampler* for the motion model
- *Likelihood function* for the sensor model

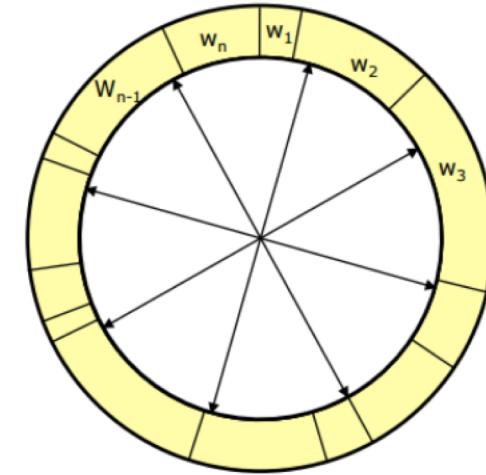


Resampling

- Sampling with replacement
- Normalize weights



- Roulette wheel
- Binary search
- $O(J \log J)$



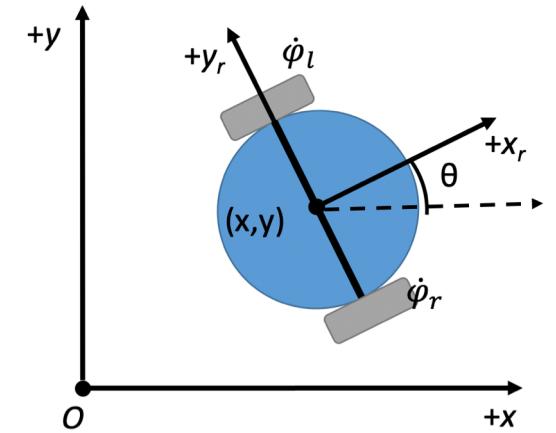
- Stochastic universal sampling
- Low variance
- $O(J)$

J : the total # of samples

Robot motion models

We will consider two primary types of robot motion models:

- Velocity-based
- Odometry-based



Velocity-based motion models

Goal: We need a procedure for **drawing samples** $x_{t+1} \sim p(x_{t+1}|x_t, u_t)$ from the robot's motion model.

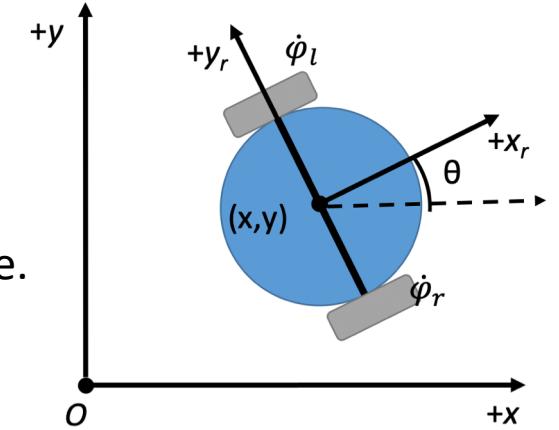
Recall: We derived the kinematic equations for a differential-drive robot. These give the robot's body-centric **velocity** $\dot{v}_r \triangleq (\dot{x}_r, \dot{y}_r, \dot{\theta}_r)$ as a function of its wheel speeds $(\dot{\phi}_l, \dot{\phi}_r)$.

In the **noiseless** case, we can integrate \dot{v}_r to calculate the next pose x_{t+1} given the current pose x_t .

We can model **noisy motion** by supposing that the robot's velocity \dot{v}_r is subject to noise. This leads to a very natural **sampling algorithm**:

sample_velocity_motion_model($x_t, \dot{\phi}_l, \dot{\phi}_r$):

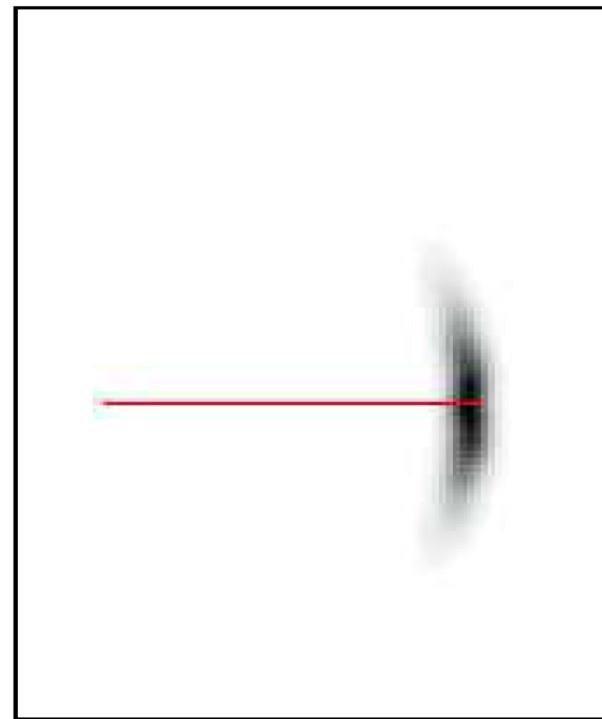
1. Calculate body-centric velocity \dot{v}_r from $(\dot{\phi}_l, \dot{\phi}_r)$ using kinematic model
2. **Add noise:** $\tilde{v}_r \triangleq \dot{v}_r + \Delta v_r$, where Δv_r is sampled from a noise distribution (e.g. Gaussian)
3. Calculate next pose x_{t+1} by integrating **noisy** velocity \tilde{v}_r



$$\begin{pmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta}_r \end{pmatrix} = \begin{pmatrix} \frac{r}{2}(\dot{\phi}_r + \dot{\phi}_l) \\ 0 \\ \frac{r}{w}(\dot{\phi}_r - \dot{\phi}_l) \end{pmatrix}$$

The Banana distribution

This velocity-based motion model induces a “banana”-shaped marginal distribution over position



In robotics, this is often referred to (appropriately enough) as the “banana distribution”

Odometry-based motion models

Velocity-based motion models provide a simple means of simulating noisy robot motion.

But: They are often grossly oversimplified (\Rightarrow not very accurate)

Alternative: Treat an **odometry measurement** z_t between pose x_{t-1} and x_t *as if* it was a **motion command** u_{t-1} . That is, consider:

$$x_t \sim p(x_t | x_{t-1}, z_t)$$

Payoffs:

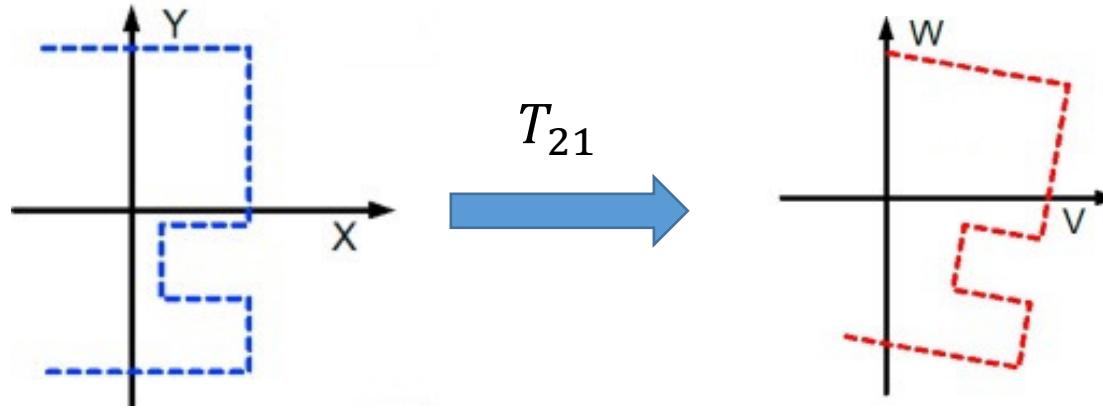
- Often significantly more accurate than velocity models
- Odometry can be gotten from several sources:
 - Wheel encoders
 - Inertial measurement units
 - **Scan matching** or visual odometry

Con: Only available *post hoc* – can't use for planning



Estimating odometry via scan matching

Suppose I have two scans of environment taken from nearby poses (so that they partially overlap)



Question:

What is the coordinate transformation T_{21} that relates their overlap?

Recall: If T_{W1} and T_{W2} are poses of the scanner in the world frame W , and p is a point, then:

$$\begin{aligned} T_{W1}p_1 &= p_W = T_{W2}p_2 \\ \Rightarrow p_2 &= T_{W2}^{-1}T_{W1}p_1 \\ &\quad \underbrace{\phantom{T_{W2}^{-1}T_{W1}}}_{=} T_{21} \end{aligned}$$

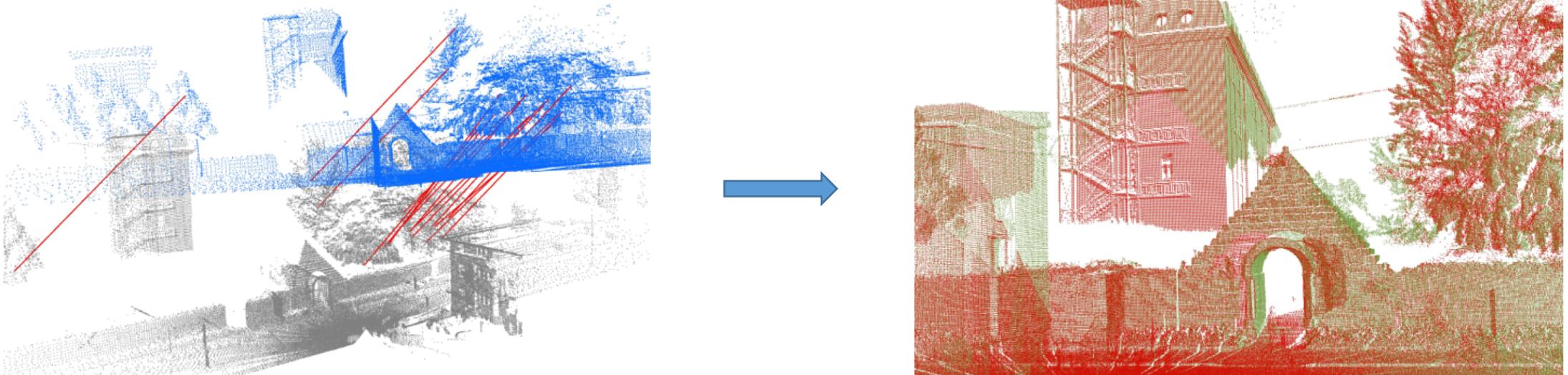
Punchline: Scan matching provides a means of *directly measuring odometry*

But: How can we actually *find* T_{21} given only the two scans?

Pointcloud registration

Problem: Given point sets $X = \{x_i\}_{i=1}^n$ and $Y = \{y_i\}_{i=1}^n$ in \mathbb{R}^d and their correspondences, find the transformation $T = (t, R) \in SE(d)$ that *optimally aligns* them:

$$(t, R) = \operatorname{argmin}_{t,R} \sum_{i=1}^n \|y_i - (Rx_i + t)\|^2$$



Pointcloud registration

Given: particle sets $X = \{x_i\}_{i=1}^n$ and
 $Y = \{y_i\}_{i=1}^n$ in \mathbb{R}^d

Horn's method:

1. Calculate centroids:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i,$$

2. Center the pointclouds: $x'_i \triangleq x_i - \bar{x}$, $y'_i \triangleq y_i - \bar{y}$
3. Construct outer product matrix W :

$$W \triangleq \sum_i y'_i x'^T_i$$

4. Recover optimal rotation R :

$$R = U \cdot \text{diag}(1, \dots, 1, \det(UV^T)) \cdot V^T$$

where $W = U\Sigma V^T$ is the singular value decomposition of W

5. Recover optimal translation t : $t = \bar{y} - R\bar{x}$

Iterative Closest Point

Catch: Horn's method assumes that we know the *point correspondences* $x_i \leftrightarrow y_i$. What if we don't have these?

One approach: Estimate these together with the registration $T = (t, R)$!

Iterative closest point (ICP): Given $X = \{x_i\}_{i=1}^n$ and $Y = \{y_j\}_{j=1}^m$, initial guess $T = (t_0, R_0)$, maximum association distance $d > 0$:

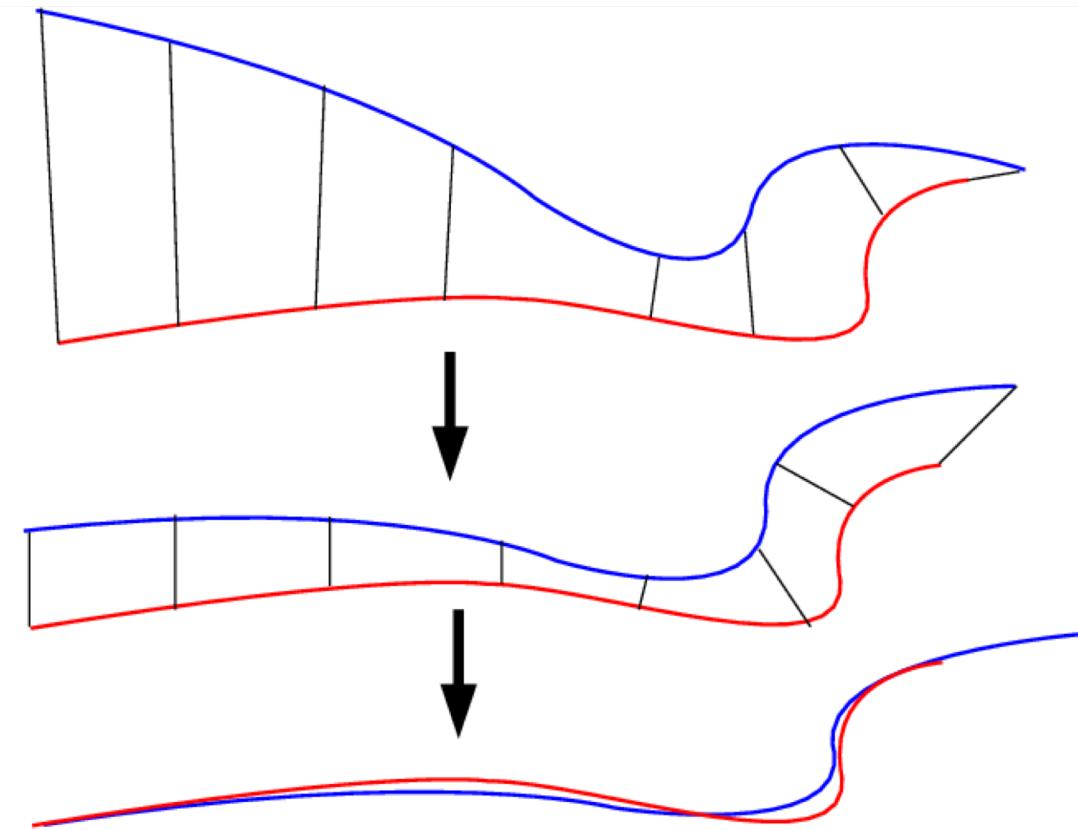
For $k = 0, \dots$ until convergence:

1. **Estimate matches:** Let $y_{l_i} = \operatorname{argmin}_j \|y_j - (R_k x_i + t_k)\|$ be the closest point in Y to the image of x_i under the current registration estimate $T_k = (t_k, R_k)$.

If $\|y_{l_i} - (R_k x_i + t_k)\| \leq d$, accept $x_i \leftrightarrow y_{l_i}$ as a match

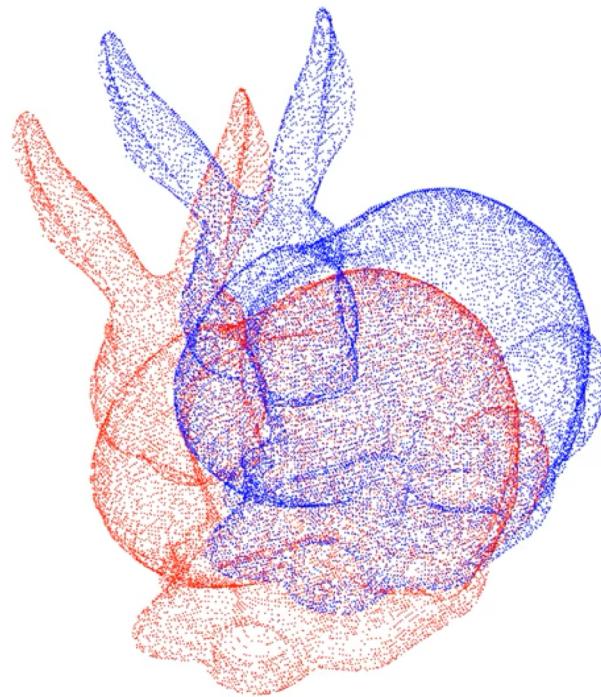
2. **Estimate registration:** Compute next registration estimate $T_{k+1} = (t_{k+1}, R_{k+1})$ by applying Horn's method to matches $\{x_i \leftrightarrow y_{l_i}\}$.

ICP illustrated

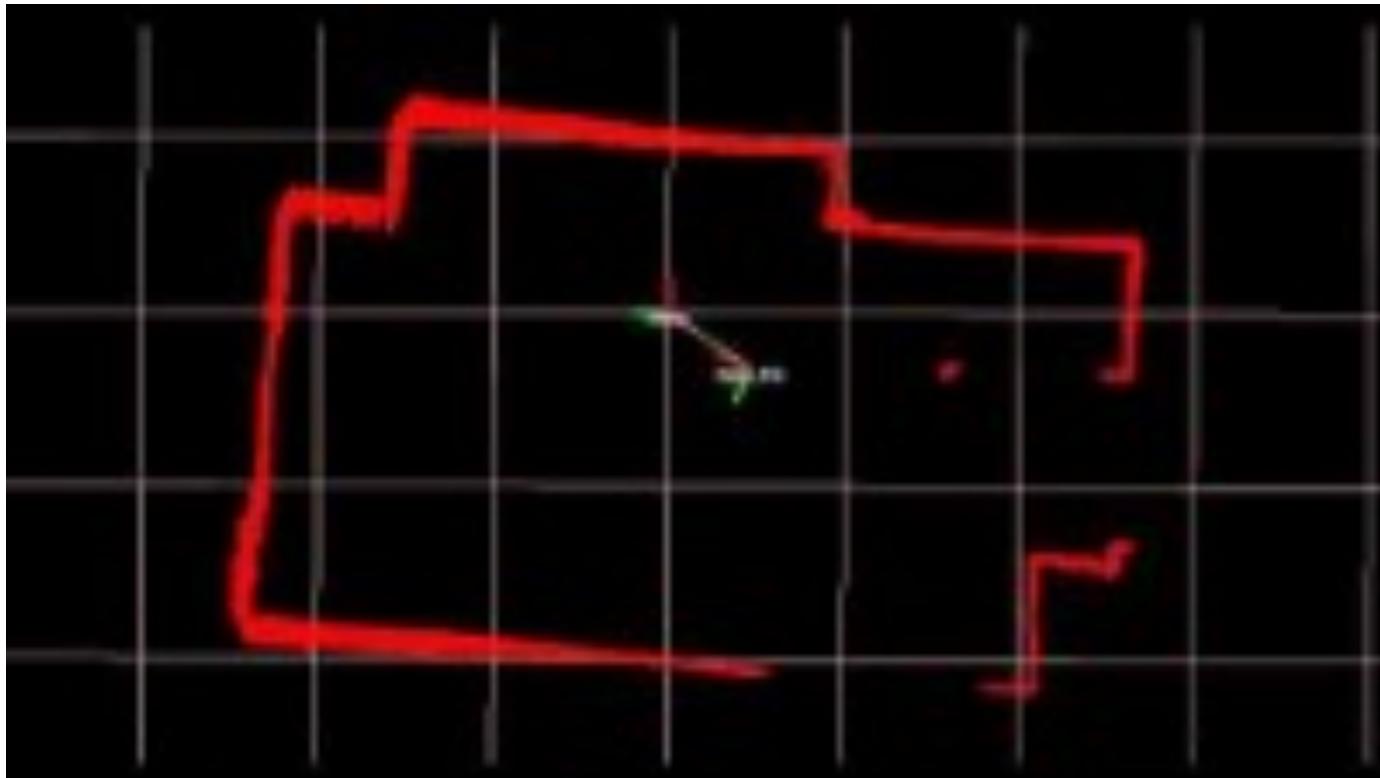


Iterative closest point

Iteration 0



Odometry estimation using ROS's Canonical Scan Matcher



Monte Carlo Localization

Main idea: Monte Carlo localization (MCL) is simply localization using the Particle Filter.

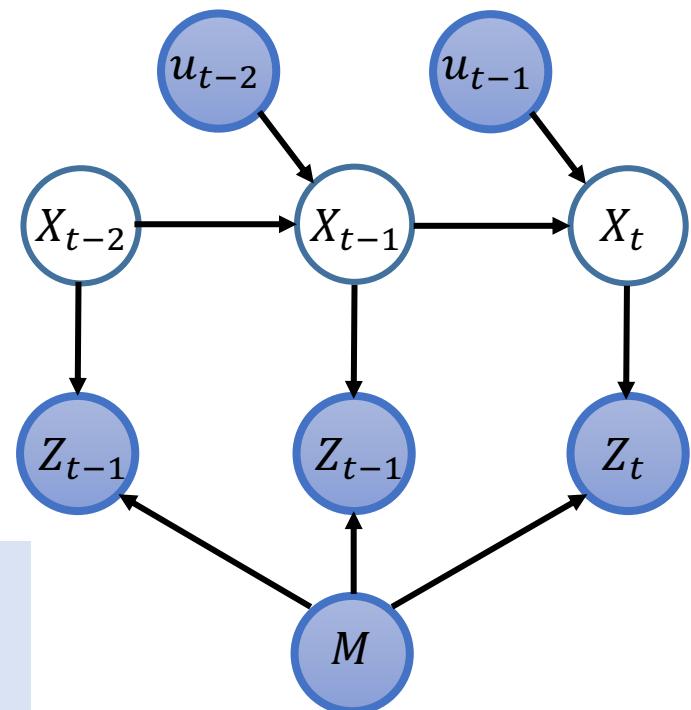
Given: Prior $p(x_0)$, map m , controls $u_{0:t}$, observations $z_{1:t}$

Estimate: Belief $p(x_t | m, z_{1:t})$ over the robot pose

```
1: Algorithm Particle_filter( $\mathcal{X}_{t-1}, u_t, z_t$ ):
2:    $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
3:   for  $m = 1$  to  $M$  do
4:     sample  $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$  ← Motion model
5:      $w_t^{[m]} = p(z_t | x_t^{[m]})$  ← Sensing model
6:      $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
7:   endfor
8:   for  $m = 1$  to  $M$  do
9:     draw  $i$  with probability  $\propto w_t^{[i]}$ 
10:    add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
11:   endfor
12:   return  $\mathcal{X}_t$ 
```

Requires:

- A *sampler* for the motion model
- *Likelihood function* for the sensor model



Beam sensor likelihood model

Last time: (Approximate) *inverse* sensor model: $p(m_i|x_t, z_t)$

⇒ Approximate likelihood of *map* given *pose and measurement*

Today: *Forward* sensor model $p(z|x_t, m)$

⇒ Likelihood of *measurement* given *pose and map*

Many common sensors in robotics are beam-based: Sonar, Radar, LIDAR, etc.

- interpreting a reflected signal to indicate distance from the sensor to the nearest solid object

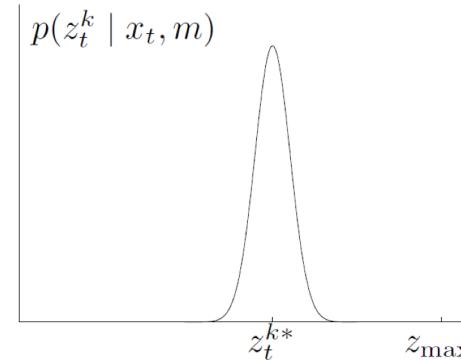
Typical sources of error in range measurements:

- Beams reflected by small obstacles
- Beams reflected by people or moving objects
- Multipath interference
- Maximum range measurements
- Random noise

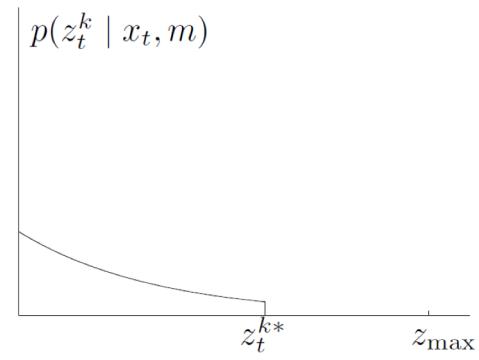
Beam sensor likelihood model

We consider a more physically-faithful sensor model that accounts for **4 distinct sources** of error

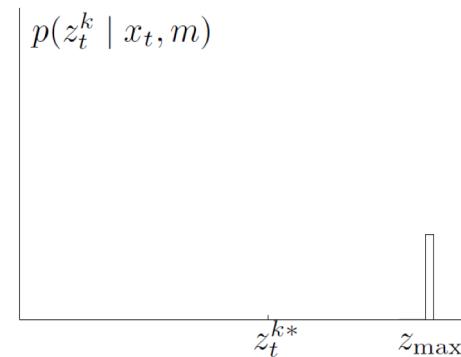
- **Correct return w/ small measurement noise**
 - Nominal case for operation
 - Modeled as a **Gaussian** p_{hit} centered on true range
- **Unexpected / transient objects**
 - Corresponds to e.g. people moving in the scene
 - Causes a ***short return***
 - Modeled as an **exponential** distribution p_{short} (one can justify this mathematically by appealing to e.g. survival analysis).
- **Failed detection**
 - Sensor does not detect a reflected beam – can be caused by e.g. specular reflection or absorption of the beam on target
 - Appears as a ***maximum range*** return
 - Can be modeled as a ***point mass*** (or very narrowly peaked distribution) p_{max} at the sensor's maximum range
- **Otherwise unexplained**
 - Catch-all category for other “general weirdness”
 - Modeled as a ***uniform distribution*** p_{rand} over the sensor's range



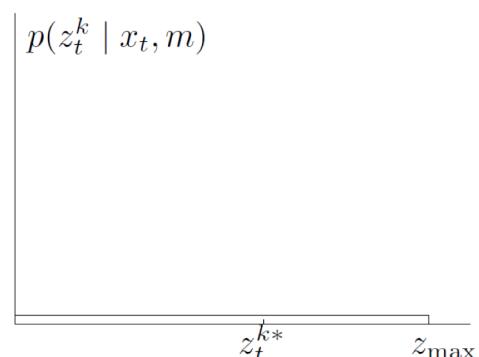
p_{hit}



p_{short}



p_{max}



p_{rand}

Beam sensor likelihood model

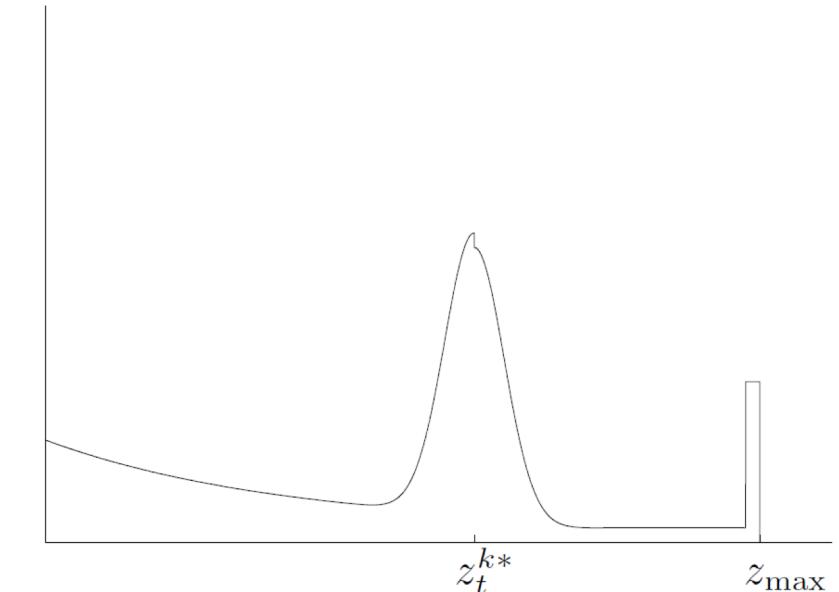
Our overall beam sensor model $p(z|x_t, m)$ is a *mixture* of these 4 components:

$$\begin{aligned} p(z|x_t, m) \\ = w_{hit} p_{hit} + w_{short} p_{short} + w_{max} p_{max} + w_{rand} p_{rand} \end{aligned}$$

where $w_{hit}, w_{short}, w_{max}, w_{rand}$ are the *mixture weights* (nonnegative and sum to 1).

We also assume that each beam is sampled *independently*, so the joint probability for an entire (n -beam) scan is:

$$p(z_t|x_t, m) = \prod_{k=1}^n p(z_t^k|x_t, m)$$



Complete (mixture) beam sensor model $p(z|x_t, m)$

Monte Carlo Localization

Main idea: Monte Carlo localization (MCL) is simply localization using the Particle Filter.

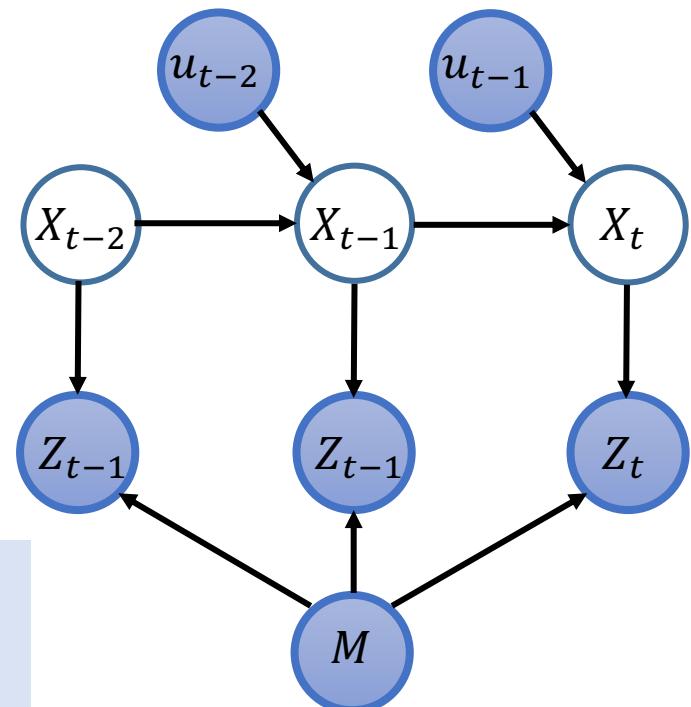
Given: Prior $p(x_0)$, map m , controls $u_{0:t}$, observations $z_{1:t}$

Estimate: Belief $p(x_t | m, z_{1:t})$ over the robot pose

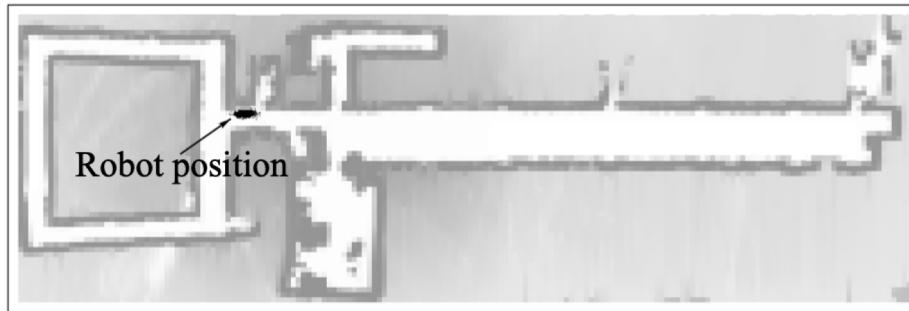
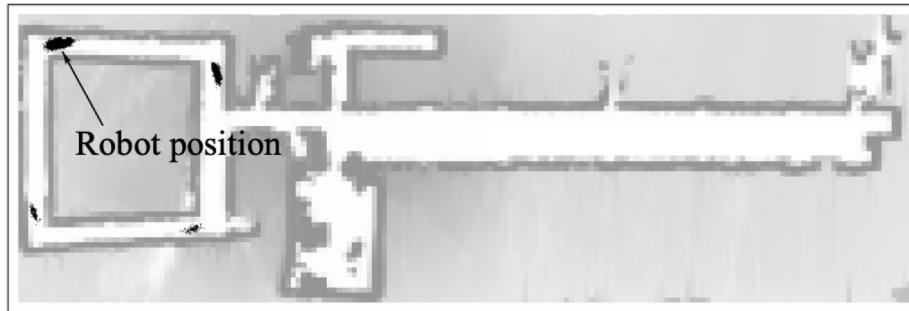
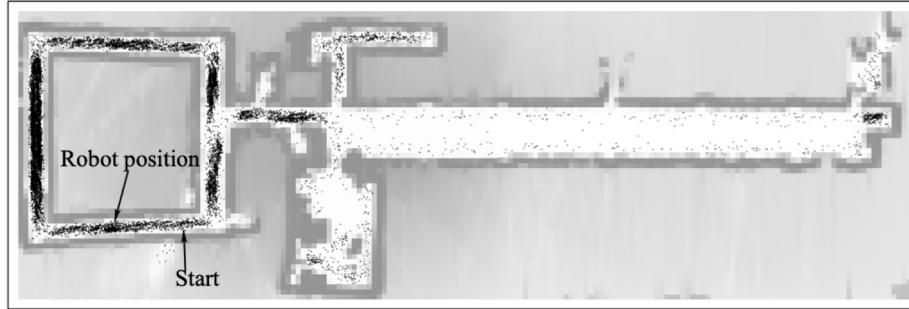
```
1: Algorithm Particle_filter( $\mathcal{X}_{t-1}, u_t, z_t$ ):
2:    $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
3:   for  $m = 1$  to  $M$  do
4:     sample  $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$  ← Motion model
5:      $w_t^{[m]} = p(z_t | x_t^{[m]})$  ← Sensing model
6:      $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
7:   endfor
8:   for  $m = 1$  to  $M$  do
9:     draw  $i$  with probability  $\propto w_t^{[i]}$ 
10:    add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
11:   endfor
12:   return  $\mathcal{X}_t$ 
```

Requires:

- A *sampler* for the motion model
- *Likelihood function* for the sensor model



Monte Carlo localization



Monte Carlo localization on a RACECAR

Real-Time Particle Filter Localization Demo

Stata Basement Loop

Instructor Solution by Corey Walsh

6.141 Spring 2017

Lab 5



Practicalities

Recall two major issues with particle filters:

- **Particle depletion:** Need to preserve diversity in the particle set
⇒**Augmented MCL:** Method for increasing particle diversity
- **Computational cost:** Might need a **lot** of particles to model complex beliefs
⇒**Adaptive sampling:** *Dynamically* adjust particle set size

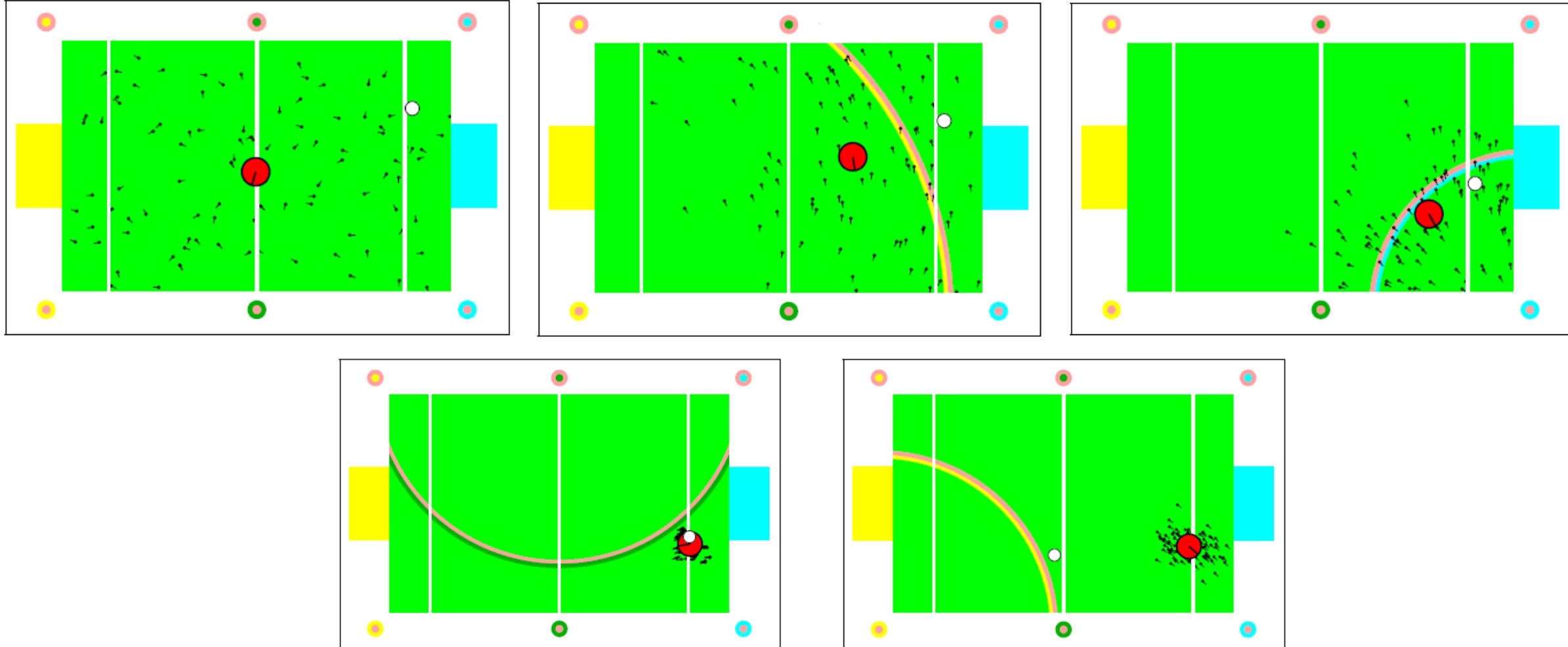
The “kidnapped robot” problem

Def: A scenario in which a robot must relocalize itself after being (instantly) moved to an *arbitrary* location in a map

Ex: Repositioning a robot that is powered off
(Moving a Roomba up / down a floor)



Particle depletion and the kidnapped robots

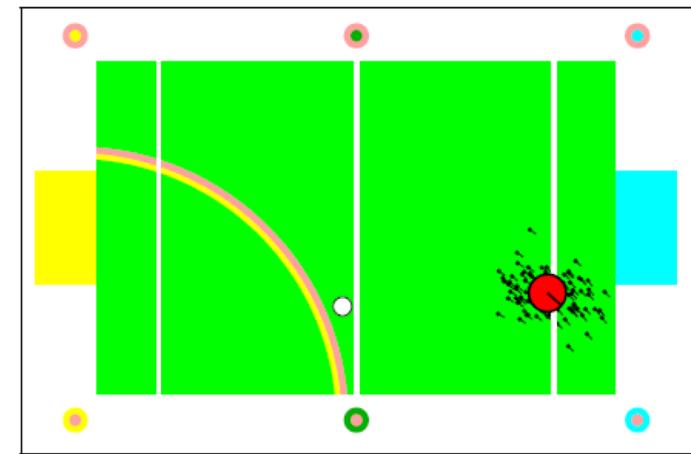


Preserving particle diversity: Augmented MCL

Problem: We need to preserve *diversity* in the filter's particle set

⇒ Need to have particles near the true state to recover from localization failures

Idea: We can *increase diversity* by resampling a small fraction of (*purely*) *random* particles



1. How many particles should be added at each iteration?
 2. From which distribution should we generate these particles?
-
- A fixed number of random particles at each iteration
 - Add particles based on some estimate of the localization performance

Preserving particle diversity: Augmented MCL

Idea: We can **increase diversity** by resampling a small fraction of (**purely**) *random* particles

Approach: Use *measurement likelihood* (weights) as a measure of localization quality

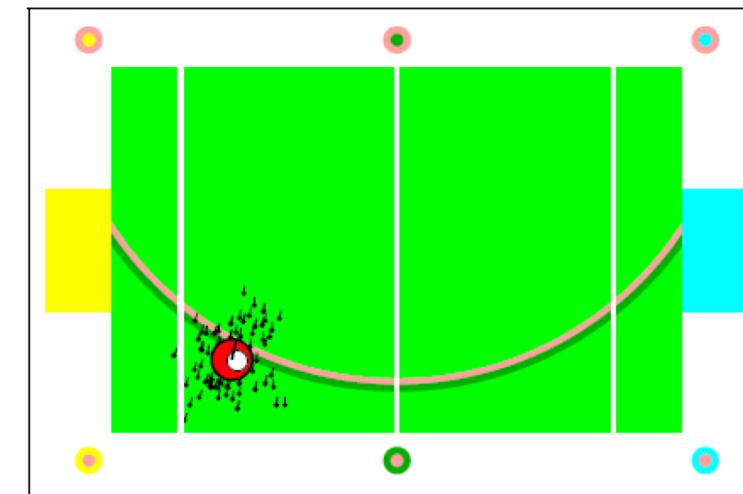
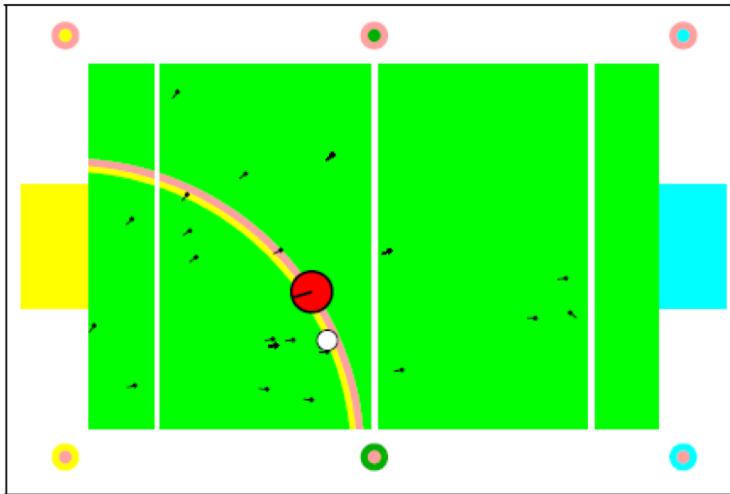
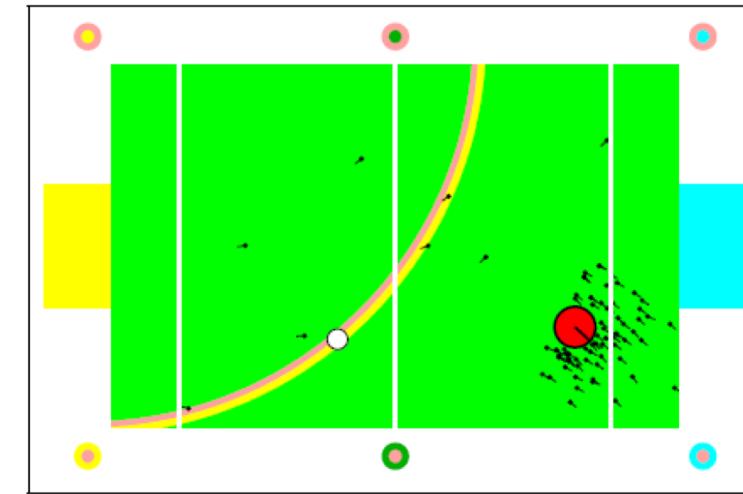
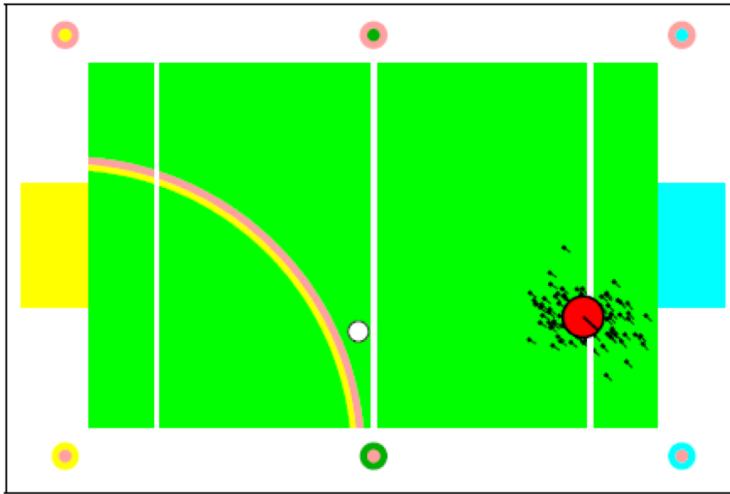
⇒ Sudden decrease in avg particle likelihood vs. historical avg may indicate localization failure

⇒ In that case, we should increase diversity of particle set

```
1: Algorithm Augmented_MCL( $\mathcal{X}_{t-1}, u_t, z_t, m$ ):
2:   static  $w_{\text{slow}}, w_{\text{fast}}$ 
3:    $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
4:   for  $m = 1$  to  $M$  do
5:      $x_t^{[m]} = \text{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$ 
6:      $w_t^{[m]} = \text{measurement\_model}(z_t, x_t^{[m]}, m)$ 
7:      $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
8:      $w_{\text{avg}} = w_{\text{avg}} + \frac{1}{M} w_t^{[m]}$ 
9:   endfor
10:   $w_{\text{slow}} = w_{\text{slow}} + \alpha_{\text{slow}}(w_{\text{avg}} - w_{\text{slow}})$ 
11:   $w_{\text{fast}} = w_{\text{fast}} + \alpha_{\text{fast}}(w_{\text{avg}} - w_{\text{fast}})$ 
12:  for  $m = 1$  to  $M$  do
13:    with probability  $\max(0.0, 1.0 - w_{\text{fast}}/w_{\text{slow}})$  do
14:      add random pose to  $\mathcal{X}_t$ 
15:    else
16:      draw  $i \in \{1, \dots, N\}$  with probability  $\propto w_i^{[i]}$ 
17:      add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
18:    endwith
19:  endfor
20:  return  $\mathcal{X}_t$ 
```

Table 8.3 An adaptive variant of MCL that adds random samples. The number of random samples is determined by comparing the short-term with the long-term likelihood of sensor measurements.

Example: Kidnapped robot revisited



KLD-sampling

KLD-sampling is a method for *dynamically adjusting* the number of particles that we maintain in the sample set.

Main idea: The number of particles that we need is related to the *uncertainty* in our current belief:
More uncertainty \Rightarrow belief is more “spread out” \Rightarrow need more particles to model

Approach: *Dynamically adjust* the number of particles based upon an estimate of the *Kullback-Leibler divergence* between particle set and true posterior

Kullback-Leibler divergence: a measure of difference between two probability distributions.

<https://www.robots.ox.ac.uk/~cvrg/hilary2005/adaptive.pdf>

