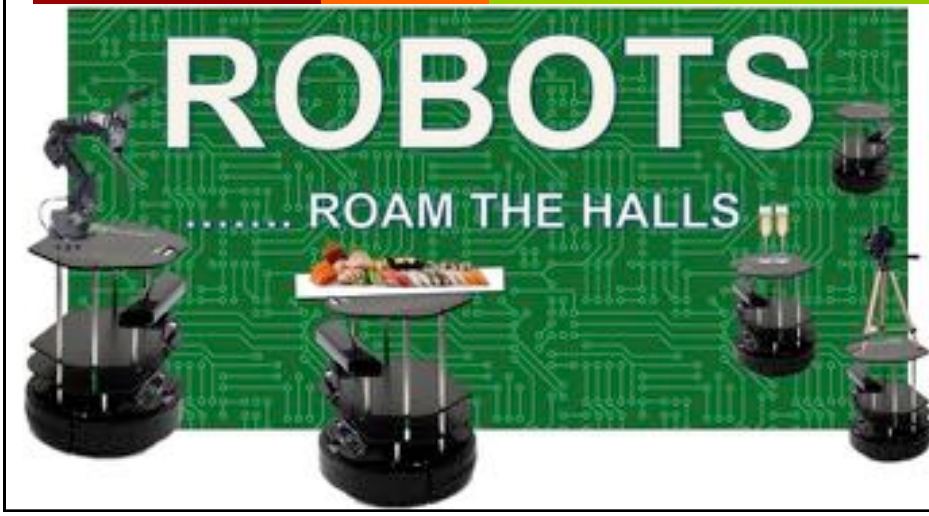
CS 189: Autonomous Robot Systems

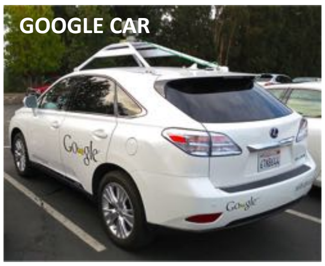
Spring 2019



Agenda

- Lecture: Robot Navigation -> Localization
- Demo Time: Lab 4 (Kalman Filters)
- Important: MUST DO LAB SAFETY TRAINING!
- Upcoming:
  - Have a Great Spring Break!
  - Pset 4 will be released Friday lecture after spring break.
- References:
  - Kalman Filter Notes, from "Computational Principles of Mobile Robotics", Dudek and Jenkin, 2000; posted on piazza resources.
  - Also "Introduction to AI Robotics", chapter 11, Robin Murphy, 2000 and "Introduction to AI", chapters 15 and 25, Russell and Norvig, 2009.

Today: Robots Navigating the World



Scenarios

- *Hospital Helper (e.g. Diligent, Tugs)*
- *Office security or mail-delivery (e.g. Cobal, Savioke)*
- *Tour Guide robot in a museum (Minerva)*
- *Autonomous Car with GPS and Nav system*

Biological analogies*:*
Humans, bees and ants, migrating birds, herds

Today: Robots Navigating the World

Second Part of CS189: High-level reasoning

From finite state machines to complex representation and memory

- Path Planning: *How to I get to my Goal?*
- Localization: *Where am I?*
- Mapping: *Where have I been?*
- Exploration: *Where haven't I been?*

Localization

- **Simple Question:** *Where am I?*
- **Not a simple answer:**
  - Do you have a map?
    - Yes => a global position in the world
    - No => position in reference to other objects? Or your own past?
  - What can you sense?
    - Can you sense and record your own self-movement?
    - Can you sense external things like landmarks?
    - How certain are you about what you sense?
- **Localization is a "collection of algorithms"**

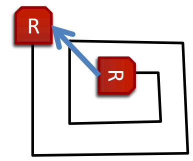
Today's Localization Techniques

- **Dead-reckoning (motion)**
  - Keep track of where you are without a map,
    by recording the series of actions that you made,
    using internal proprioceptive sensors. (also called Odometry, Path Integration)
- **Landmarks (sensing)**
  - Triangulate your position geometrically,
    by measuring distance to one or more known landmarks
    E.g. Visual beacons or features, Radio/Cell towers and signal strength, GPS!
- **State Estimation (uncertainty in motion & sensing)**
  Probabilistic Reasoning
  - **Kalman Filters** (combine both motion and sensing)
  - Particle Filters (also known as Monte Carlo Localization)
- **Who are the world's best localizers?**

Take two steps forward,
Take two steps back,
Are you back where you started?

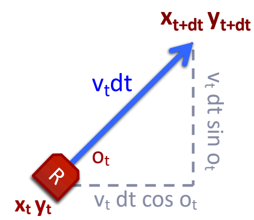
Dead-Reckoning

- **FORWARD KINEMATICS repeated**
  - Keep track of initial position and the series of movements/actions that you made.
  - Method: Take a "step", compute new position.
  - Also called odometry or path integration.



- **Our Motion Model**
  - Position at time t = (x_t, y_t, o_t)
  - Linear velocity = $\mathbf{v_t}$; Angular velocity = $\mathbf{w_t}$
  - Then for a *small time step dt*,
    we can compute the new position

$$x_{t+dt} = x_t + v_t \text{ dt} \cos o_t$$
$$y_{t+dt} = y_t + v_t \text{ dt} \sin o_t$$
$$o_{t+dt} = o_t + w_t \text{ dt}$$

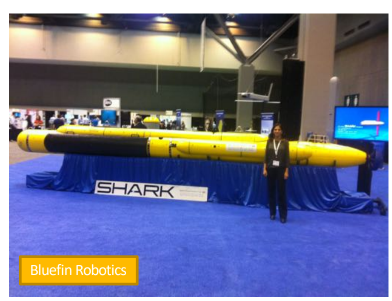


Dead-reckoning is even easier to calculate if you only Move or Turn at one time.

Example: INS

Inertial navigation systems (INS)

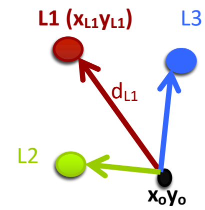
- Complex motion (momentum, external effects)
- Include **accelerometers** and **gyroscopes** to provide better measurements of instantaneous velocity.
- Expensive systems very good
  - satellites, submarines
- But, low-cost IMUs increasingly available



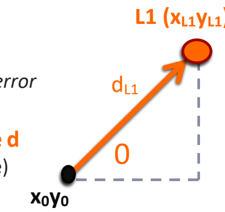
Landmarks

I can see the CITGO sign to my southeast, 15 miles away
Where am I?

- **How it works**
  - *Opposite of dead-reckoning!*
  - Use measurements to external landmarks of known position
  - Examples: visual landmarks, radio towers, GPS!



- **Example 1: 3 Landmarks + distance only (e.g. Radio towers)**
  - **Landmark positions: (x_{L1}, y_{L1}) (x_{L2}, y_{L2}) (x_{L3}, y_{L3})**
  - If you have three non-colinear landmarks,
    then you lie at the intersection of **three circles**! [triangulation]
  - Three equations of the form:
    square(d_{L1}) = square($x_{L1} – x_0$) + square($y_{L1} – y_0$) (Landmark L1)
  - **Solve for (x_0, y_0)**
    Or if they don't intersect exactly (noise), *minimize sum-of-squared-error*



- **Example 2: Single Landmark but known orientation O and distance d**
  - E.g. Facing the office label MD235 (can't see it from inside the office)
    cosO = (x_1-x_0)/d_L sinO = ($y_L – y_0$)/ d_L

Example: GPS

- GPS Satellites are your "landmarks"
  - Continually transmits a message
  - Message includes both time of transmission, and satellite position
- GPS Receiver
  - Compute distance by measuring signal transmission time (speed of light)
  - 3D: Lie on the intersection of 4 spheres!
- What are some limitation of GPS?

Today's Localization Techniques

- Dead-reckoning (motion)
  - Keep track of where you are without a map,
    by recording the series of actions that you made,
    using internal proprioceptive sensors. (also called Odometry, Path Integration)
- Landmarks (sensing)
  - Triangulate your position geometrically,
    by measuring distance to one or more known landmarks
    E.g. Visual beacons or features, Radio/Cell towers and signal strength, GPS!
- **State Estimation (uncertainty in motion & sensing)**
  Probabilistic Reasoning
  - **Kalman Filters** (combine both motion and sensing)
  - Particle Filters (also known as Monte Carlo Localization)
- **Who are the world's best localizers?**

Two Techniques

- **Key Idea: Combine Motion and Sensing**
  - **(Dead-reckoning + uncertainty)** + **(Landmarks + uncertainty)**
  - Each has error, but the error can be complementary
- **Kalman Filters**
  - Take advantage of mathematics of Gaussians to model uncertainty
  - General method for state estimation (not just localization)
  - Applications: Car + GPS, Lawnmower + beacons, warehouse robots
- **Particle Filters (Monte Carlo Localization)**
  - Use a discrete distribution of "Particles" to represent uncertainty (think of sampling or histograms)
  - Useful when environment is complex and ambiguous
  - Application: A robot wandering in a building with a map

Dead-reckoning + uncertainty
Landmarks + uncertainty

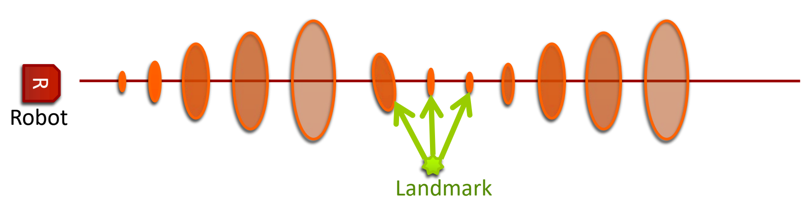
Kalman Filters

- **How it works**
  - Take a motion step: use dead-reckoning to get position (mean) but also keep track of uncertainty in movement
  - Take a sensing step: use landmarks to triangulate position, then combine with previous estimate based on relative confidence.
- **Technique and Limitations**
  - Uses Gaussians (bell curves) to capture uncertainty

Dead-reckoning + uncertainty
Landmarks + uncertainty

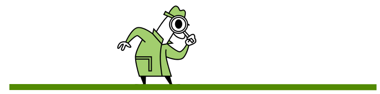
Kalman Filters

- **How it works**
  - Take a motion step: use dead-reckoning to get position (mean) but also keep track of uncertainty in movement
  - Take a sensing step: use landmarks to triangulate position, then combine with previous estimate based on relative confidence.
- **Technique and Limitations**
  - Uses Gaussians (bell curves) to capture uncertainty

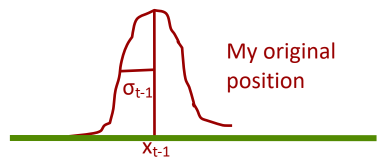


1D Kalman Filter Example

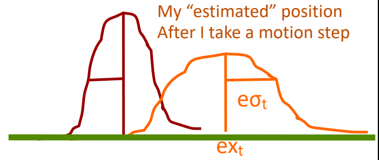
- **"Belief" of my current state**
  - x_{t-1} with variance σ_{t-1}



- **"Model" of how I work**
  - Control u_t and its variance r
  - Measurement z_t and its variance q
    - We are assuming that we can model ***noise as a Gaussian,*** *with a mean and variance* (experimentally determined)

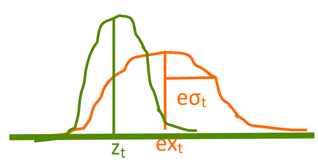


- **Step 1: Take a step, calculate new belief**
  - $ex_t = x_{t-1} + u_t$
  - $e\sigma_t = \sigma_{t-1} + r$
  - Note that my uncertainty has *increased* due to the noise in my control.



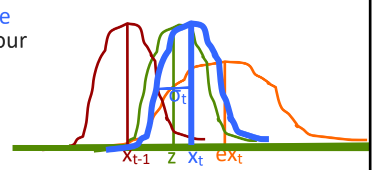
1D Kalman Filter Example

- **Step 2: Take a measurement z_t**
  Combine to create a calculate new belief
  - Simplest idea? take the average $x_t = (ex_t + z_t)/2$



  - **Better Idea!** New estimate is a weighted combination of our old estimate and measurement
    - $x_t = a*ex_t + (1-a)\ z_t$
    - $\sigma_t = (1/e\sigma_t + 1/q)^{-1}$
  - The Kalman Gain "a" is determined by our relative confidence in our belief about our old state and our confidence in the current measurement.
    - $a = q / (q + e\sigma_t)$

      Consider case where q=0
      then we will go with our noise free landmark measurement
      Consider case where $e\sigma_t=0$
      then we will ignore our measurements and go with prev position



1D Kalman Filter Example

- Final Form 1D example
  - $ex_t = x_{t-1} + u_t$
  - $e\sigma_t = \sigma_{t-1} + r$
  - $x_t = \sigma_t \, (ex_t/e\sigma_t + z_t/q)$
  - $\sigma_t = (1/e\sigma_t + 1/q)^{-1}$

Step 1: Motion
Adds uncertainty

Step 2: Measurement
Reduces uncertainty

And Repeat!

- Caveats
  - We assumed that ut and zt were in the same state space as xt (position), often not true.
  - Also still 1D.....

Kalman Filter

- Final Form 1D example
  - $ex_t = x_{t-1} + u_t$
  - $e\sigma_t = \sigma_{t-1} + r$
  - $x_t = \sigma_t \, (ex_t/e\sigma_t + z_t/q)$
  - $\sigma_t = (1/e\sigma_t + 1/q)^{-1}$
- Final Form 3D
  - $ex_t = Ax_{t-1} + Bu_t$
  - $e\sigma_t = A\sigma_{t-1}A^T + R$
  - $x_t = \sigma_t \, (ex_t/e\sigma_t + C^T Q^{-1} z_t)$
  - $\sigma_t = (1/e\sigma_t + C^T Q^{-1} C)^{-1}$

Position x = [x, y, theta]

A and B and C are matrices that convert old position, control input, and observation into the correct state space (note, A is often identity matrix)

R is a Co-variance Matrix
Q is a Co-variance Matrix
σ is a Co-Variance Matrix
The uncertainty in [x, y, theta] is not all independent of each other.
(you supply R and Q)

Kalman Filter

- Final Form 1D example
  - $ex_t = x_{t-1} + u_t$
  - $e\sigma_t = \sigma_{t-1} + r$
  - $x_t = \sigma_t (ex_t/e\sigma_t + z_t/q)$
  - $\sigma_t = (1/e\sigma_t + 1/q)^{-1}$
- Final Form 3D
  - $ex_t = Ax_{t-1} + Bu_t$
  - $e\sigma_t = A\sigma_{t-1}A^T + R$
  - $x_t = \sigma_t (ex_t/e\sigma_t + C^T Q^{-1} z_t)$
  - $\sigma_t = (1/e\sigma_t + C^T Q^{-1} C)^{-1}$

Extended Kalman Filter

Lets say that $u_t = [D, w]$ (distance, rotation)

$x_{t-1} = [x', y', w']$
$ex_t = [x' + D\cos w', y' + D\sin w', w'+w]$

Unfortunately, this is non-linear!
(can't express as $ex_t = Ax_{t-1} + Bu_t$)

In EKF, the system is "linearized"
by computing the Jacobian
of the motion model
and the measurement model.

See Dudek and Jenkins notes for more details

Extensions of the basic idea

- **Multiple sensors! (sensor fusion)**
  - Just repeat step 2 (sensing) multiple times
  - This is especially useful if you have "occasional" sensors (e.g. landmarks)
- **When is a Kalman Filter good to use?**
  - When *control* and *sensor noise* are well approximated by a Gaussian
    - (e.g. GPS and car/robot controls are usually decently approximated this way)
  - When *estimated state (x)* can be represented by just a Gaussian.
    - Classic bad case: car and two neighboring lanes;
      = expected location is best approximated by two Gaussians
- **Many Applications of Kalman Filters!**
  - Object tracking in a video! (opposite of "self" localization)

I could be TWO PLACES at once!!

I could be TWO PLACES at once!!

Particle Filters

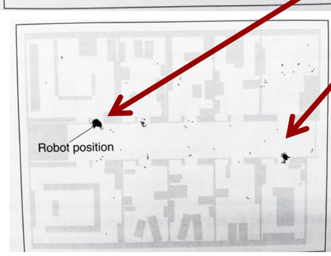
- What if you are in a building with a map.
  - But you have no idea where you are? (ambiguity)
    - You are definitely in a bathroom, but don't know 1st or 2nd floor
  - Problem: Gaussians are not the right model of uncertainty
- Instead
  - Represent our estimated position and uncertainty (our "belief") using a constant set of "particles"
    - Think of this as a "sampling" from a probability distribution
    - That is why it is called Monte Carlo Localization

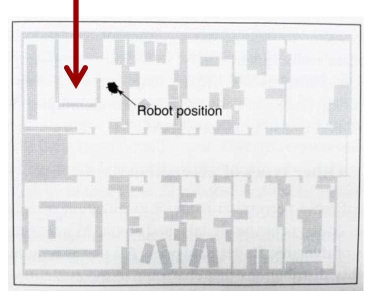
What it looks Like

Particle "density" = Probability
Initially particles everywhere

Observation of corridors narrows the possibilities (bimodal distribution)

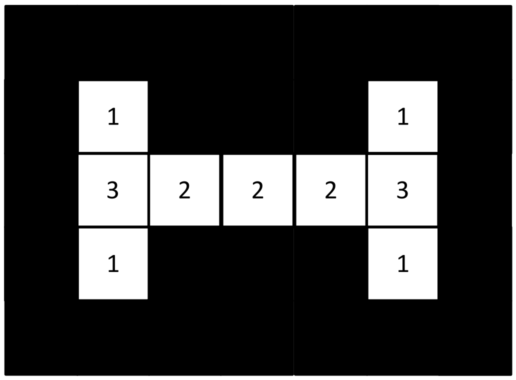
More movement results in disambiguating the two cases (now more Gaussian-like)



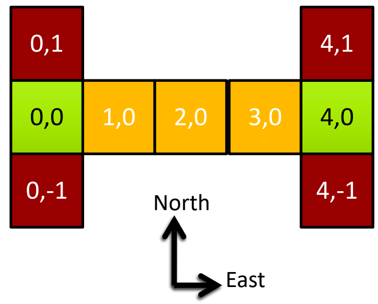


Lets do an Example

Occupancy Matrix Map



My world consists of hallways, corridor ends And 4 unique offices

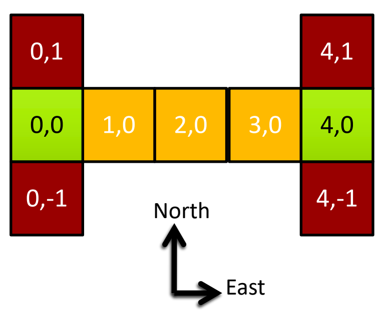


Topological Map

Lets do an Example

- **Sensor Model**
  Pr(zt | xt)
  - Depends on where you are standing
    And your error in feature sensing
  - Pr (hallway detection | (1,0)) = 0.8
  - Pr (end detection | (1,0)) = 0.2 *(error!)*
    - There is a small chance that you may think you are at the end instead of a hallway….
- **Motion Model**
  Pr (xt+1 | xt, action_t)
  - Extremely simple model
  - Move using a Compass (N,S,E,W)
  - Pr(stay) = 0.1 (fail to move); Pr(succeed) = 0.9
  - Pr (also depends on position)
    - E.g. if obstacle (like a wall) then Pr(stay) = 1

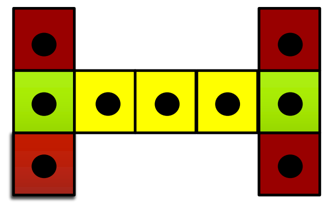
My world consists of hallways, corridor ends And 4 unique offices



**I am making lots of simplifications here that you wouldn't do in a real system

Lets do an Example

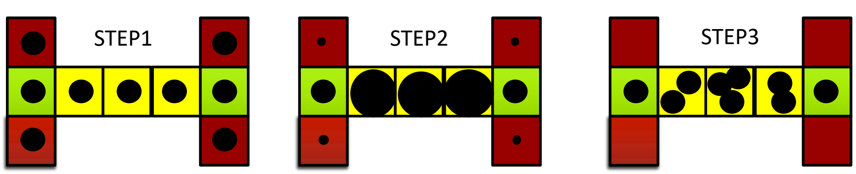
- Basic Question: Where am I?
  - Instead of a Gaussian we will represent position by a fixed number of particles distributed over space
  - But basic ideas same as Kalman filter!



- At the beginning of time
  - I could be anywhere
    - With equal likelihood
    - N particles, then avg d/N particles in each of the d locations.

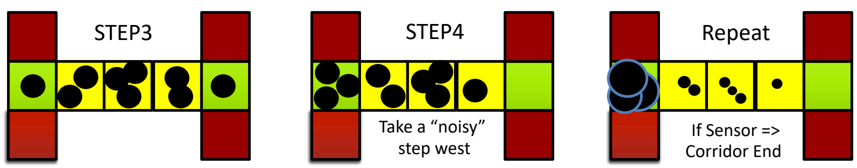
Take a Sensing Step

- STEP1: Take a sensor reading and get "evidence"
  - Lets say the Sensor => in a hallway
- STEP2: Weight each location's particles by likelihood of that reading
  - Pr (xt | given that you sensed a hallway)
- STEP3: Resample N particles but from the distribution of weights
  - Create a *new particle distribution* that represents your believed location



Take a Motion Step

- Take a motion step
  - Lets say you move west 1 spot
- STEP4: Use your motion model to predict what will happen
  - E.g. If at (1,0) and take a step west, 90% chance you succeed (0,0)
    But there's a 10% chance you will not move and end up still in (1,0)
  - Roll the dice for each particle and move.
- STEP5: Loop to STEP 1
  - *Take a Sensor Reading and reduce your uncertainty!*



More Sophisticated Version PseudoCode

```
function MONTE-CARLO-LOCALIZATION(a, z, N, P(X'|X, v, ω), P(z|z*), m) returns
a set of samples for the next time step
  inputs: a, robot velocities v and ω
          z, range scan z_1, ..., z_M
          P(X'|X, v, ω), motion model
          P(z|z*), range sensor noise model
          m, 2D map of the environment
  persistent: S, a vector of samples of size N
  local variables: W, a vector of weights of size N
                   S', a temporary vector of particles of size N
                   W', a vector of weights of size N

  if S is empty then        /* initialization phase */
     for i = 1 to N do
        S[i] ← sample from P(X_0)
  for i = 1 to N do     /* update cycle */
     S'[i] ← sample from P(X'|X = S[i], v, ω)
     W'[i] ← 1
     for j = 1 to M do
        z* ← RAYCAST(j, X = S'[i], m)
        W'[i] ← W'[i] · P(z_j | z*)
  S ← WEIGHTED-SAMPLE-WITH-REPLACEMENT(N, S', W')
  return S
```

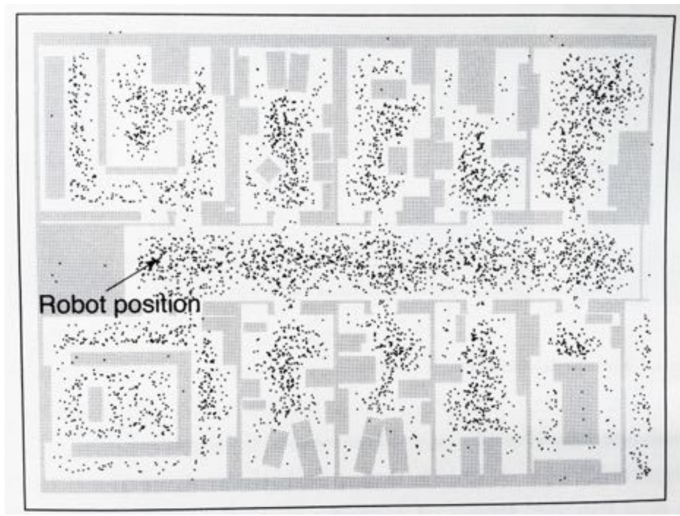
Figure 25.9 A Monte Carlo localization algorithm using a range-scan sensor model with independent noise.

Key Differences:

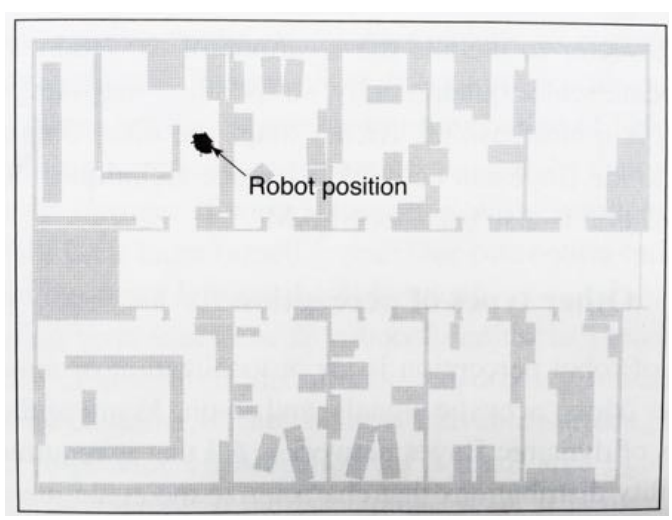
1. N Positions particles are in *continuous space*
1. Sensing is a laser scan comparison P(z|z*)
2. You have a map (m) that lets you "estimate" what a laserscan should return ("Raycast") and compared to what you actually sensed ("z")

From Russell and Norvig, Chapter 25

What it looks Like



What it looks Like

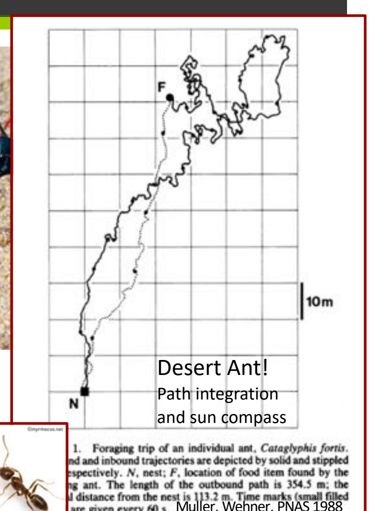


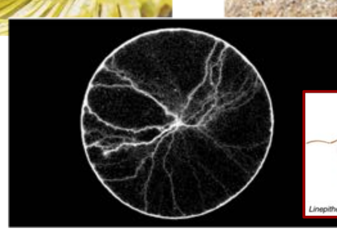
Today's Localization Techniques

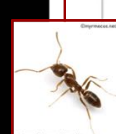
- **Dead-reckoning (motion)**
- **Landmarks (sensing)**
- **State Estimation (uncertainty motion & sensing)**
  - Kalman Filters
  - Particle Filters
- **Who are the world's best localizers?**

Some TINY but GREAT Localizers

Honey Bees!
Optical flow and
sun compass







Argentine Ant!
Pheromone Trails
(aka bread crumbs)