Project 3

CS 3630

Particle Filter Overview

Initialize particles (uniform distribution, equal weights)

Repeat:

Reset particle weights to $\frac{1}{n}$

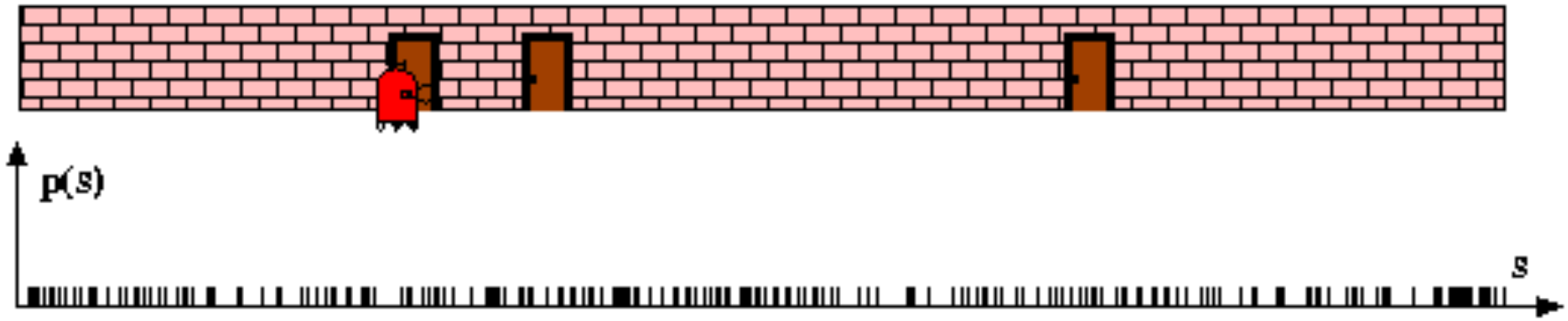
For each particle x:
    Perform motion update

For each particle x:
    Update particle weight using sensing update

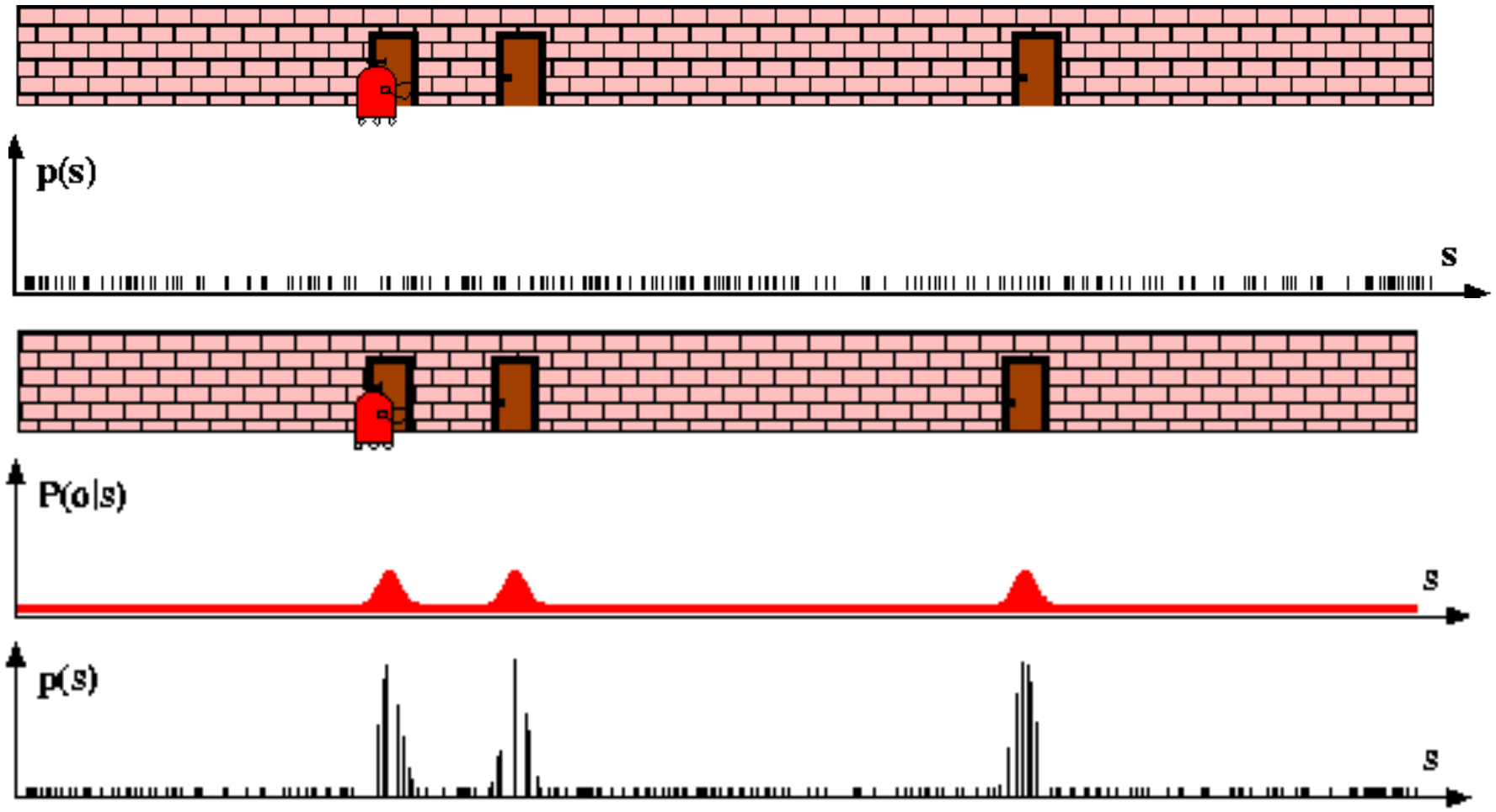
Normalize importance weights
Perform resampling

Estimate current robot state

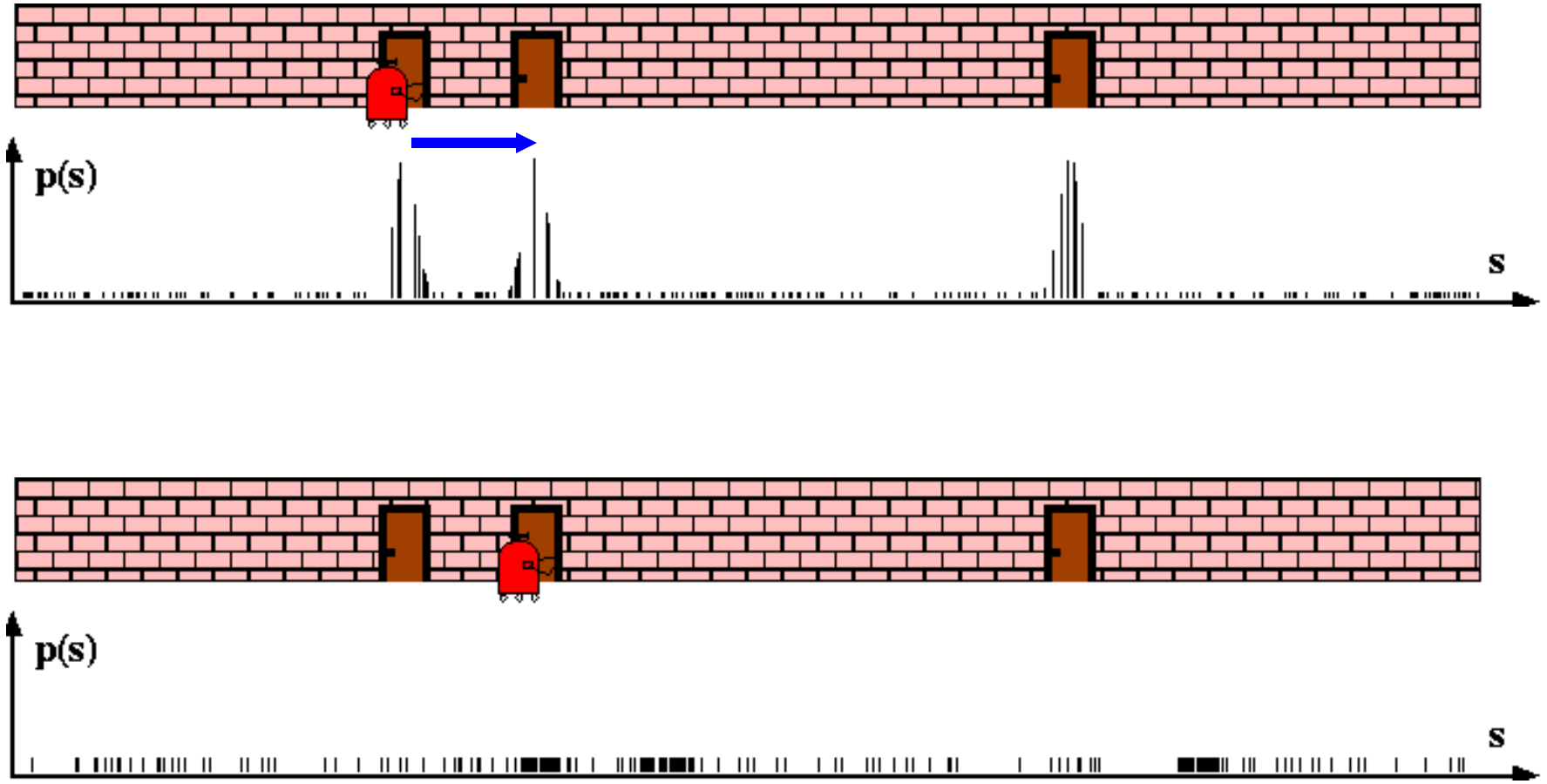
Particle Filters



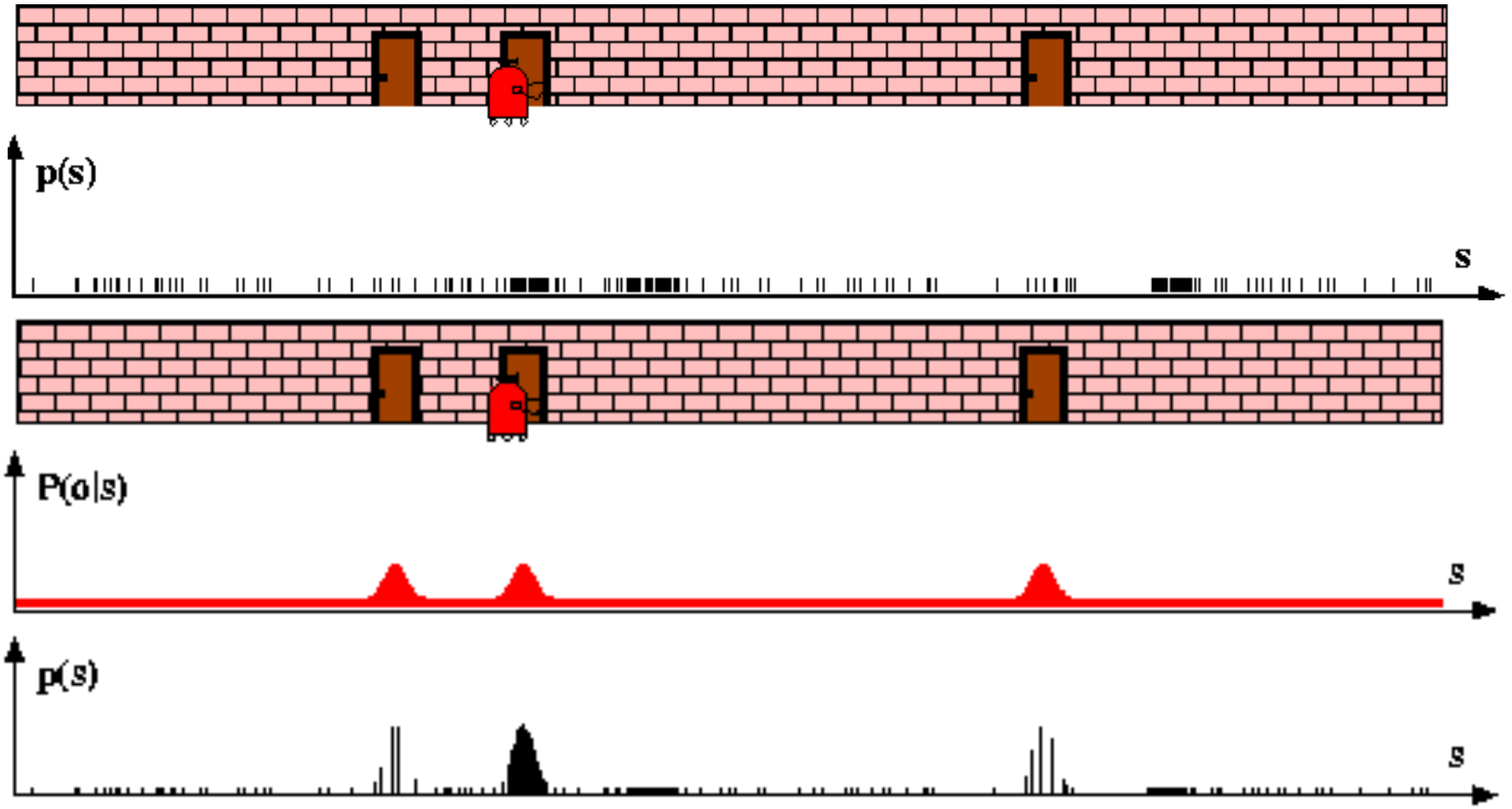
Sensor Information: Importance Sampling



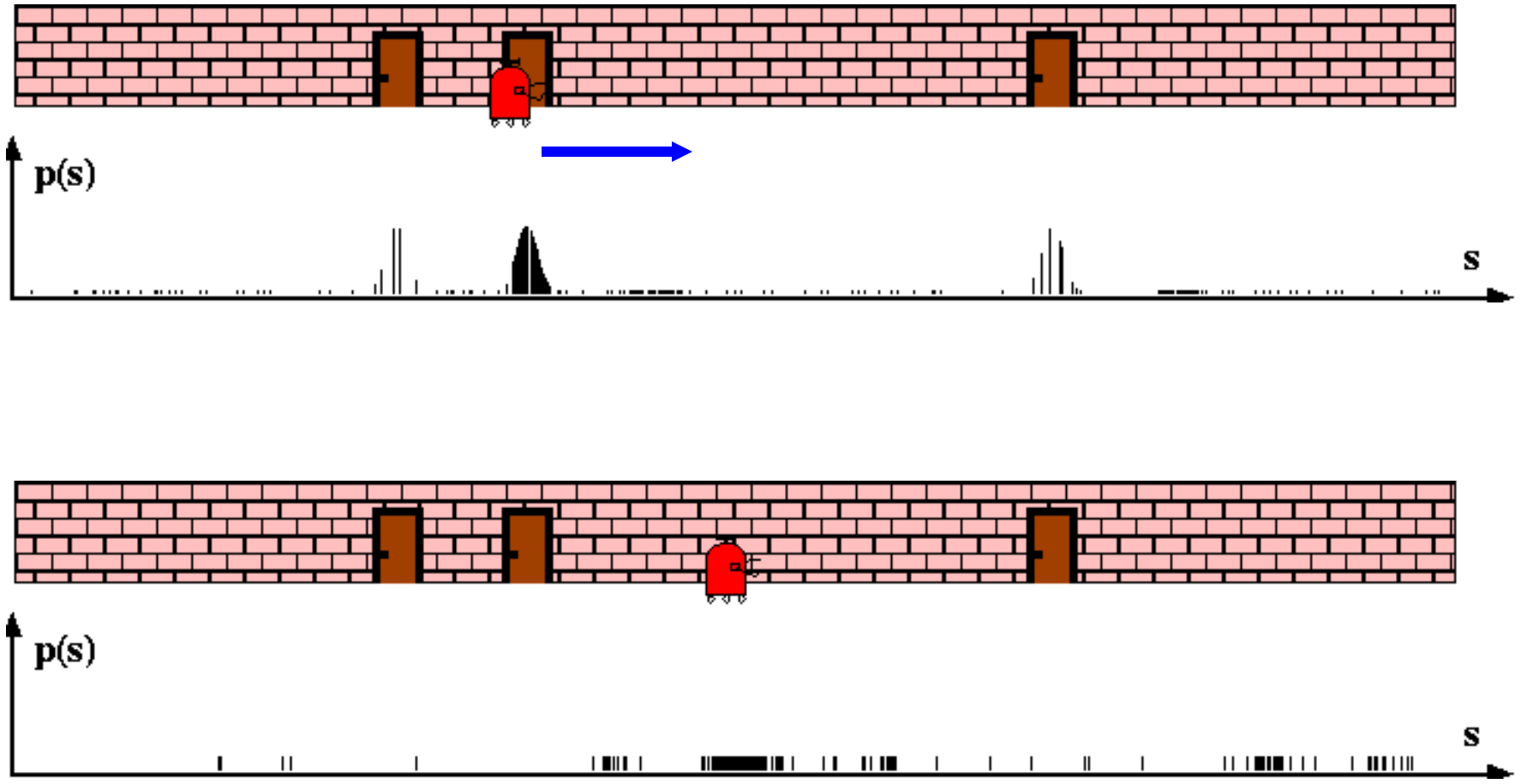
Robot Motion



Sensor Information: Importance Sampling



Robot Motion



How do we estimate pose of the robot from the set of particles?

A few options:

- Particle with **max weight**
- **Robust mean**: mean of particles located within some ϵ distance from the max particle (reduces discretization error compared to just using max weight particle)
- …

Question

- What happens if there are no particles near the correct robot location?

  We're going to get a really bad localization estimate!

- How do we detect this?

Coefficient of Variation: $cv_t = \sqrt{\sum_{i=1}^{N} (Nw_i - 1)^2}$

What will happen if all weights are equal?

Question

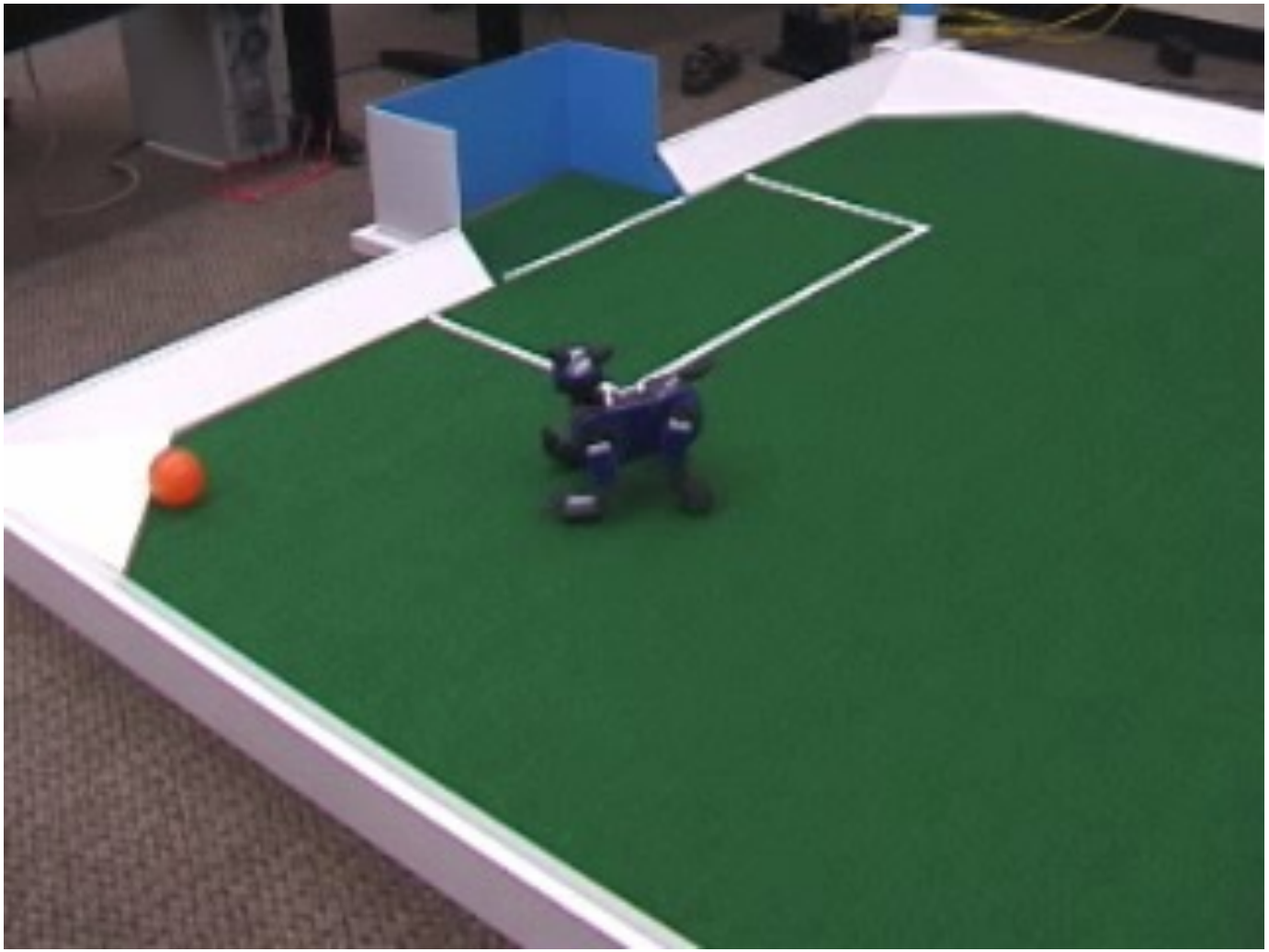
- How do we increase the likelihood of a particle being close to the robot location?

- Many possible variants:
  - Add m random points each cycle
  - Add m random points each cycle, where m is proportional to the average measurement error
  - …

Number of Samples

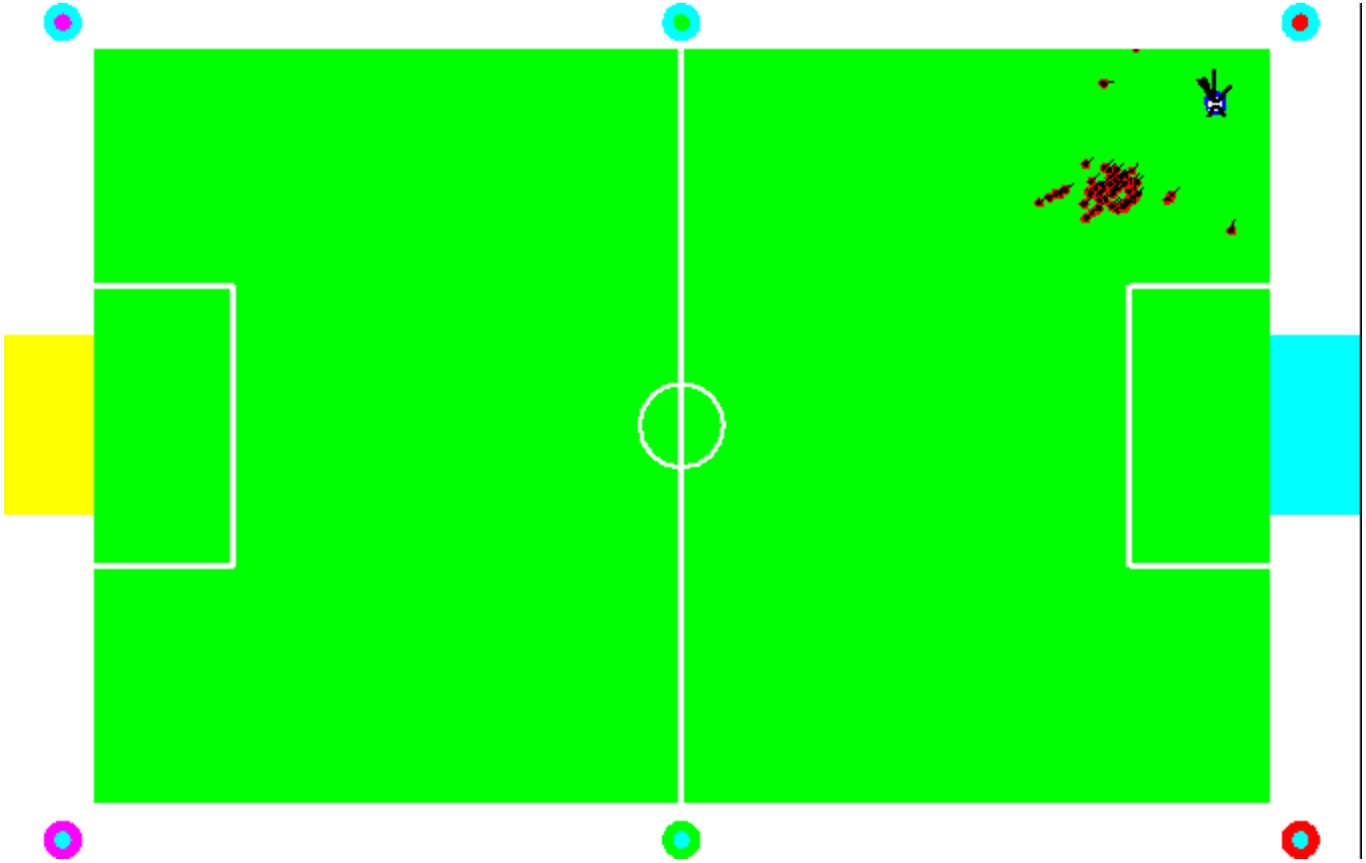
- Number of samples needed to achieve a desired level of accuracy varies dramatically depending on the situation
  - During global localization: robot is ignorant of where it is → need lots of samples
  - During position tracking: robot's uncertainty is small → don't need as many samples
- Some variants determine sample size "on the fly"

Modeling objects in the environment



http://www.cs.washington.edu/research/rse-lab/projects/mcl

Modeling objects in the environment



http://www.cs.washington.edu/research/rse-lab/projects/mcl

Project 3 Notes

Components

There are three parts of a particle filter implementation that you will be responsible for:

- Motion Update

- Sensing update
  - 2.1 Update weights
  - 2.2 Resample

Motion Update

Move each particle in accordance with the odometry estimate received

- i.e. for each particle, update its position and add some noise

Some things to keep in mind:

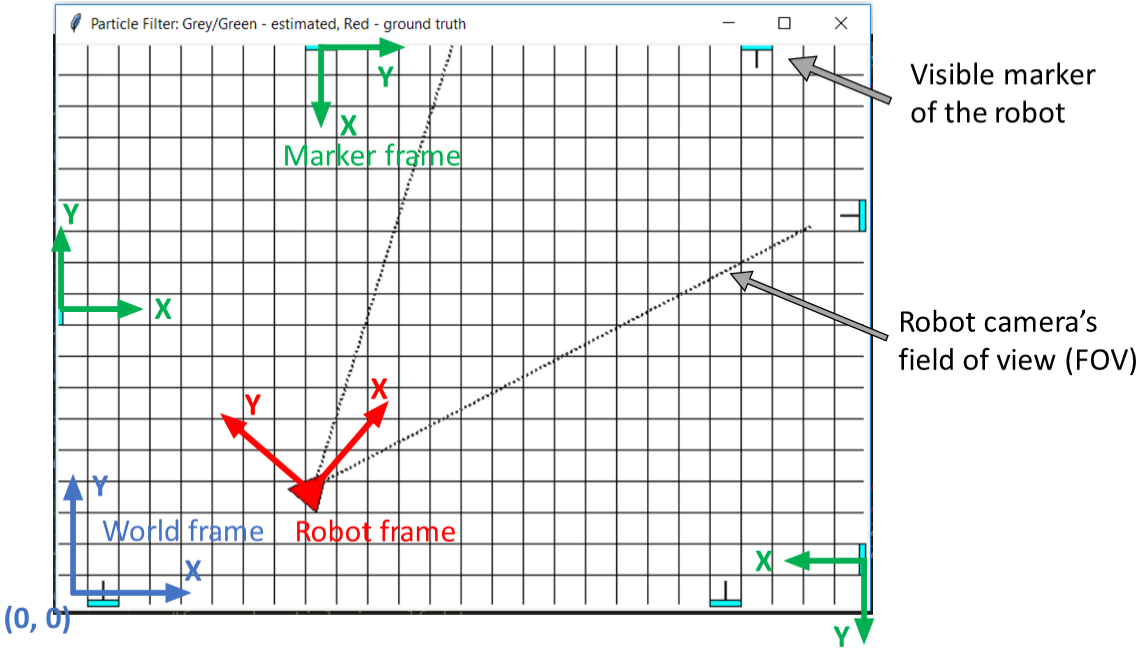
- if the robot doesn't move, the particles shouldn't move either
- remember to account for both rotation and translation movement
- add gaussian noise to each update to inject some randomness into the model. Each particle should have its own noise sample, so over time, if the robot does not see any landmarks, the particles should spread out.

Measurement/Sensing Update

1. Update the weight of each particle according to how well it matches the current sensor readings

2. Resample: Perform **importance sampling** where new particles are chosen based on the calculated weights

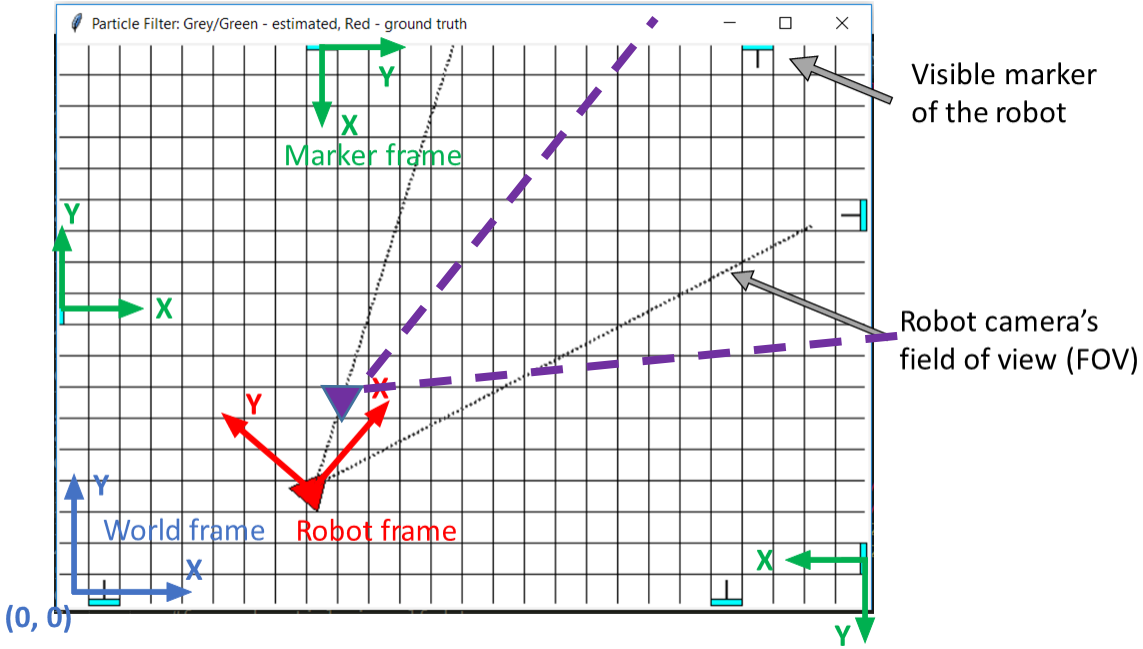
The robot's observations are the relative distance and angle of the markers/landmarks it sees in the world.



Update Weights

- Obtain real robot sensor readings, z_r
  - Note, the robot might be able to see more than one localization marker

- For each particle:
  - simulate the particle's field of view and obtain the list of localization markers the robot *would* see if it were really at the pose (position/orientation) specified by this particle, z_{p_i}
  - Next we want to compare z_{p_i} and z_r and assign a weight to the particle proportional to how well the two sets of observations match.



Note that in our domain, <u>all the localization markers look the same</u>, we have no way to distinguish one from another. As a result, the best we can do is come up with a (partial) matching of markers in z_r to markers in z_{p_i} based on distance/angle

Particle Weight

- Once you have identified the list of marker pairings, we can calculate the weight of the particle itself based on how well the markers seen by the robot match up with the markers "seen" by the particle.

$prob$ = 1.0;
for each landmark
  d = Euclidean distance to landmark
  $prob$ *= Gaussian probability of obtaining a reading at distance d for this landmark from this particle

return $prob$

The Gaussian probability density function is defined as:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

In our case μ would be a reading from the robot and x would be a simulated reading from a particle, thus we would obtain the probability of them matching.

However, in the case of the robot, we want to account for angle in addition to distance. Also, for simplification, we can ignore the first part of the equation because it remains the same for every particle. Therefore our probability update becomes:

$$P(x) = e^{-\left(\frac{(distBetweenMarkers)^2}{2\sigma^2} + \frac{(angleBetweenMarkers)^2}{2\sigma^2}\right)}$$

Summary of Weight Calculation

$prob$ = 1.0;
for each landmark/marker
  calculate $distBetweenMarkers$ and $angleBetweenMarkers$
  $prob \ *= e^{-\left(\frac{(distBetweenMarkers)^2}{2\sigma^2} + \frac{(angleBetweenMarkers)^2}{2\sigma^2}\right)}$
return $prob$

Other considerations

- Particles within an obstacle or outside the map should have a weight of 0

- If there are no new sensor readings, assign equal weight to all particles (there is no new sensor information, so there should be no bias in the resampling process)

Importance Sampling (Resampling)

- The final step of the pipeline is resampling, in which we will generate a new set of n particles

- First, now that we have the particle weights, remember we have to normalize them. Sum up all the weights and divide the weight of each particle by the sum.

- Next, we want to do probabilistic sampling with replacement to generate a new particle distribution.
  - Each particle should be resampled with a probability equal to the particle's normalized probability calculated in the previous step.

Additional implementation options

- Eliminate any particles with very low probabilities and replace them with random samples
- Always maintain some small percentage of random samples
- Throw away all particles and start again with a uniform distribution if all the particles are very unlikely (should not be needed in this lab)

Handling noisy sensor data

All simulated sensor data in Project 3 has some noise built in (i.e. distances/angles are not entirely accurate, similar to real robot perception)

Beyond that, the Project 3 simulator has parameters that allow you to change between:

1. **Deterministic perception**: if there's a marker in the robot's FOV, it will be reported as visible, and no extra markers will be reported
2. **Non-deterministic perception**: some markers that *should* have been visible, will not be; some markers that aren't really there will be "visible"

Look for DETECTION_FAILURE_RATE and SPURIOUS_DETECTION_RATE parameters under setting.py

External References

- Section 6.5 of the Corke book on Robotics, Vision and Control (available on Canvas)
- Chapter 5 of the Autonomous Mobile Robots book (less detailed)