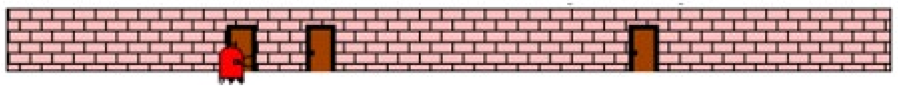
Particle filter for 1D localization

Lab -- ENGR 509

Problem Setup

Problem Setup



- Robot localization
  - Given the map, where am I (on the map)?

$$Bel(x_t) = \eta\, p(z_t \mid x_t) \int p(x_t \mid x_{t-1}, u_t)\, Bel(x_{t-1})\, dx_{t-1}$$

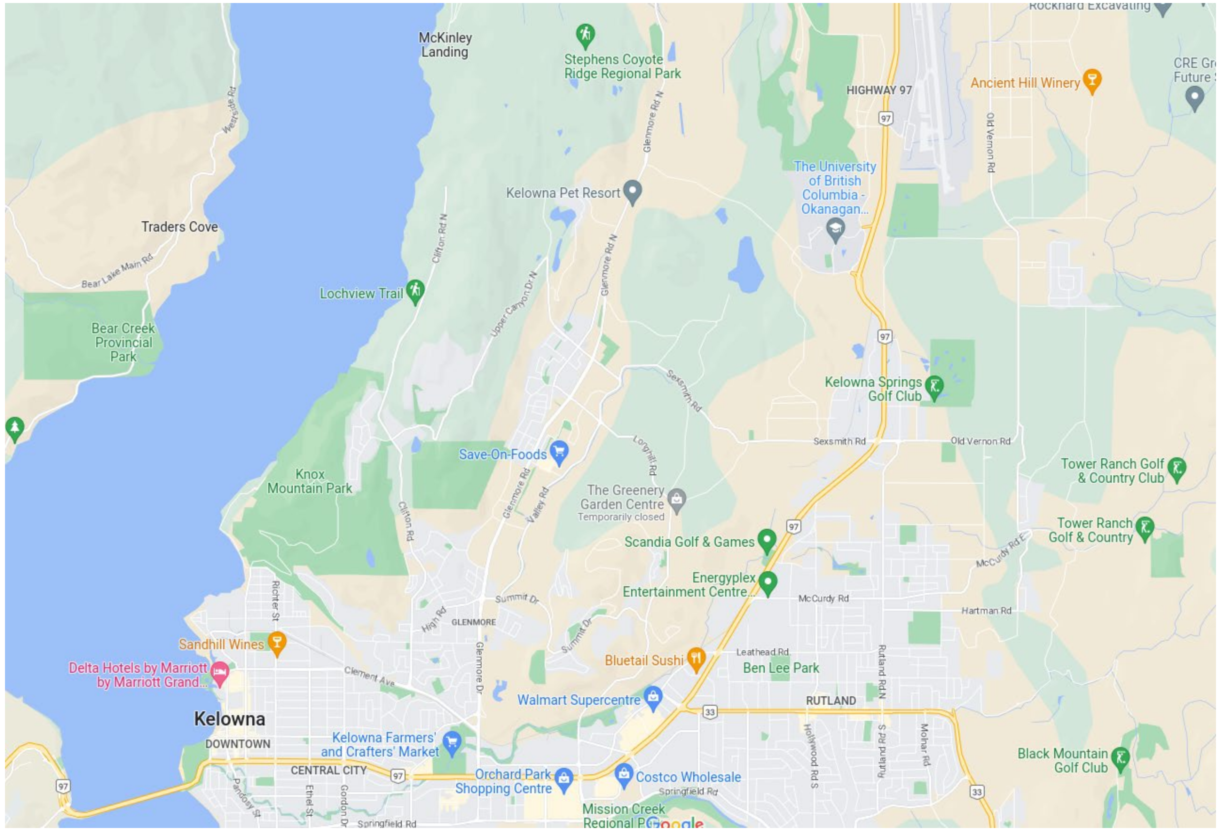
current location estimate

observation model

motion model

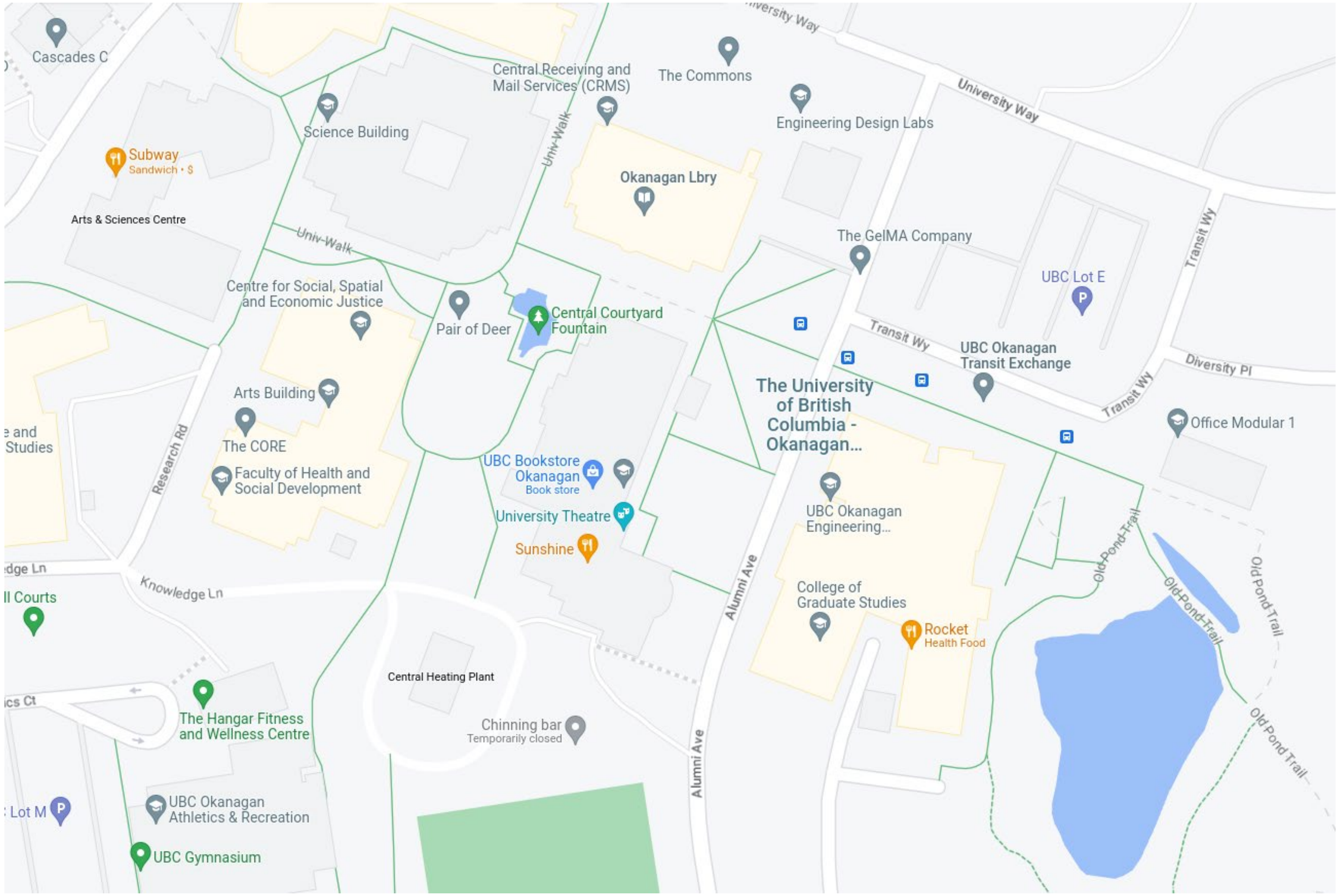
previous location estimate

Examples


McKinley Landing
Stephens Coyote Ridge Regional Park
HIGHWAY 97
Ancient Hill Winery
Kelowna Pet Resort
The University of British Columbia - Okanagan...
Traders Cove
Lochview Trail
Bear Creek Provincial Park
Kelowna Springs Golf Club
Save-On-Foods
Knox Mountain Park
The Greenery Garden Centre
Temporarily closed
Tower Ranch Golf & Country Club
Tower Ranch Golf & Country
Scandia Golf & Games
Energyplex Entertainment Centre...
Sexsmith Rd
Old Vernon Rd
McCurdy Rd
Hartman Rd
GLENMORE
Sandhill Wines
Delta Hotels by Marriott by Marriott Grand...
Bluetail Sushi
Ben Lee Park
Leathead Rd
Walmart Supercentre
RUTLAND
Kelowna
DOWNTOWN
Kelowna Farmers' and Crafters' Market
CENTRAL CITY
Orchard Park Shopping Centre
Costco Wholesale
Springfield Rd
Mission Creek Regional P
Black Mountain Golf Club
Google


Living Room
Kitchen
Entryway
Dining Room

Map



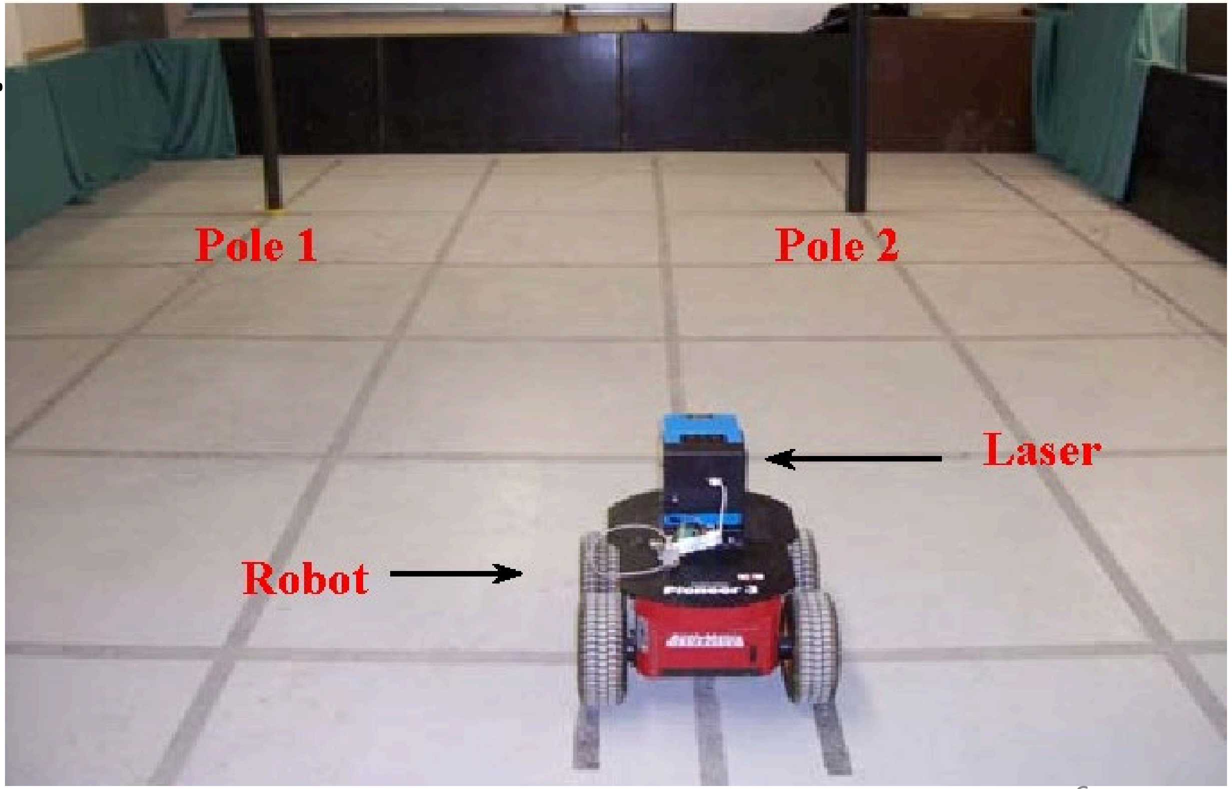
What are the information on the map that we need to do localization?

- locations of landmarks

As long as we have enough landmarks, and relative location to the landmarks, we can localize yourself.

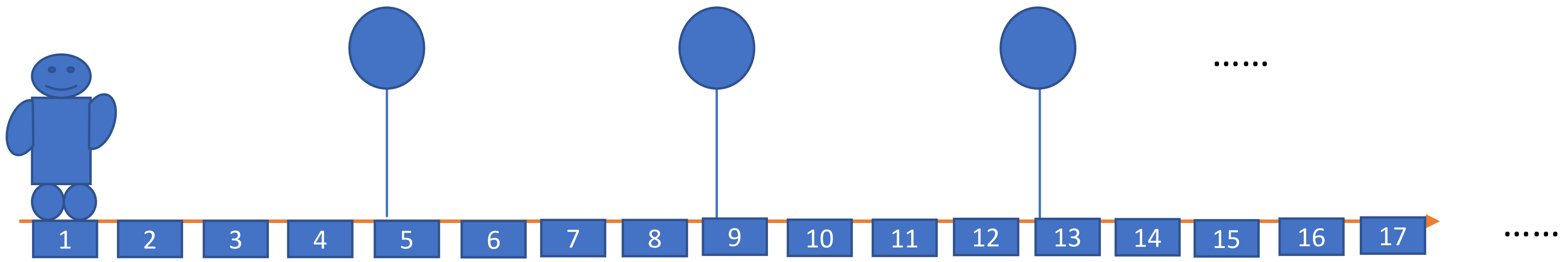
Our implementation

- landmark: poles
- A range sensor tells how far to a pole.
- Remember: robot knows the map.



Particle Filter on 1D localization

Particle filter for 1D localization



Now, we change the rule:

- The robot movement is not perfect. Although the control command is moving forward by 1.0 unit, the robot can move 1.0 unit plus some errors.
- The robot measurements are not perfect. For example, the measurement is 2.5 units to a pole, meaning distance could be 2.5 units plus some errors.
- For simplicity, we model the errors follow zero-mean Normal distributions.

Particle filter for 1D localization

$$\boxed{Bel(x_t) = \eta\, p(z_t \mid x_t) \int p(x_t \mid x_{t-1}, u_t)\, Bel(x_{t-1})\, dx_{t-1}}$$

draw x^i_{t-1} from $Bel(x_{t-1})$

draw x^i_t from $p(x_t \mid x^i_{t-1}, u_t)$

Importance factor for x^i_t:

$$w^i_t = \frac{\text{target distribution}}{\text{proposal distribution}} = \frac{\eta\, p(z_t \mid x_t)\, p(x_t \mid x_{t-1}, u_t)\, Bel\,(x_{t-1})}{p(x_t \mid x_{t-1}, u_t)\, Bel\,(x_{t-1})} \propto p(z_t \mid x_t)$$

Particle filter for 1D localization

- Sample the next generation for particles using the proposal distribution
- Compute the importance weights :

  weight = target distribution / proposal distribution
- Resampling: "Replace unlikely samples by more likely ones"

Algorithm Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):

$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$

for $m = 1$ to M do

  sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$

  $w_t^{[m]} = p(z_t \mid x_t^{[m]})$

  $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$

endfor

for $m = 1$ to M do

  draw i with probability $\propto w_t^{[i]}$

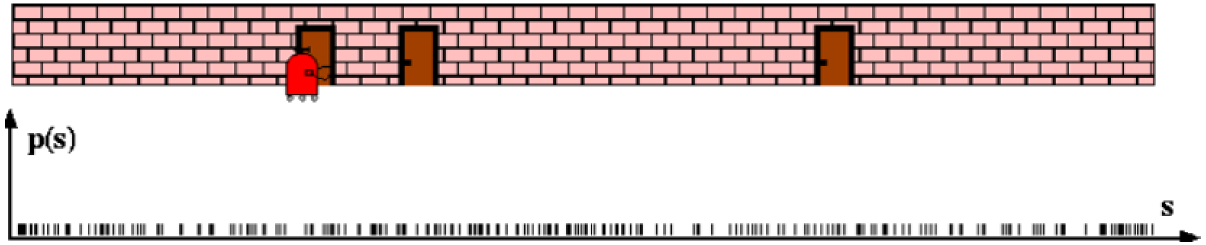
  add $x_t^{[i]}$ to $\mathcal{X}_t$

endfor

return $\mathcal{X}_t$

Particle filter for 1D localization

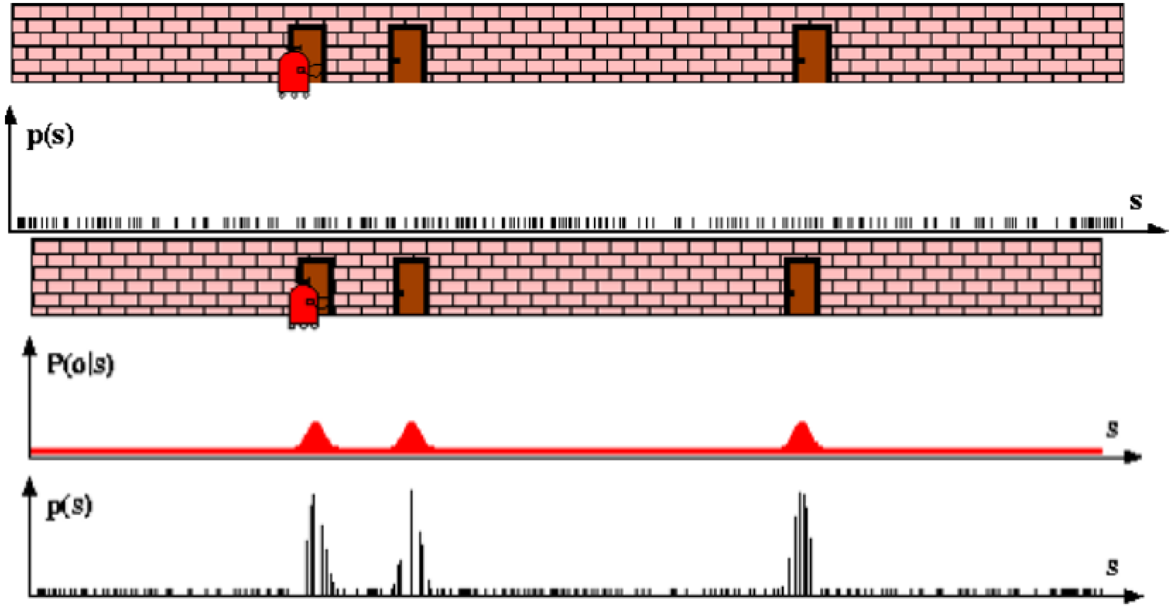
Prior:



Particle filter for 1D localization

Prior:

Update:

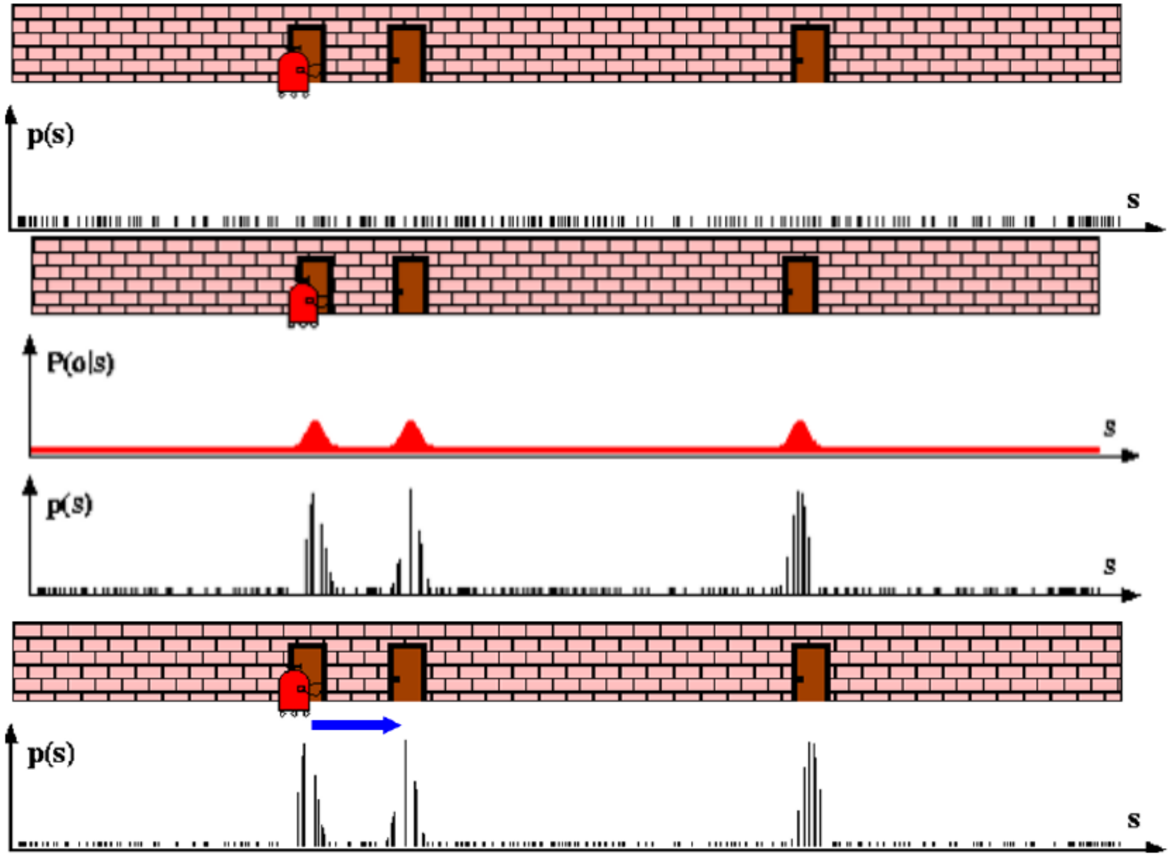


Particle filter for 1D localization

Prior:

Update:

Predict:



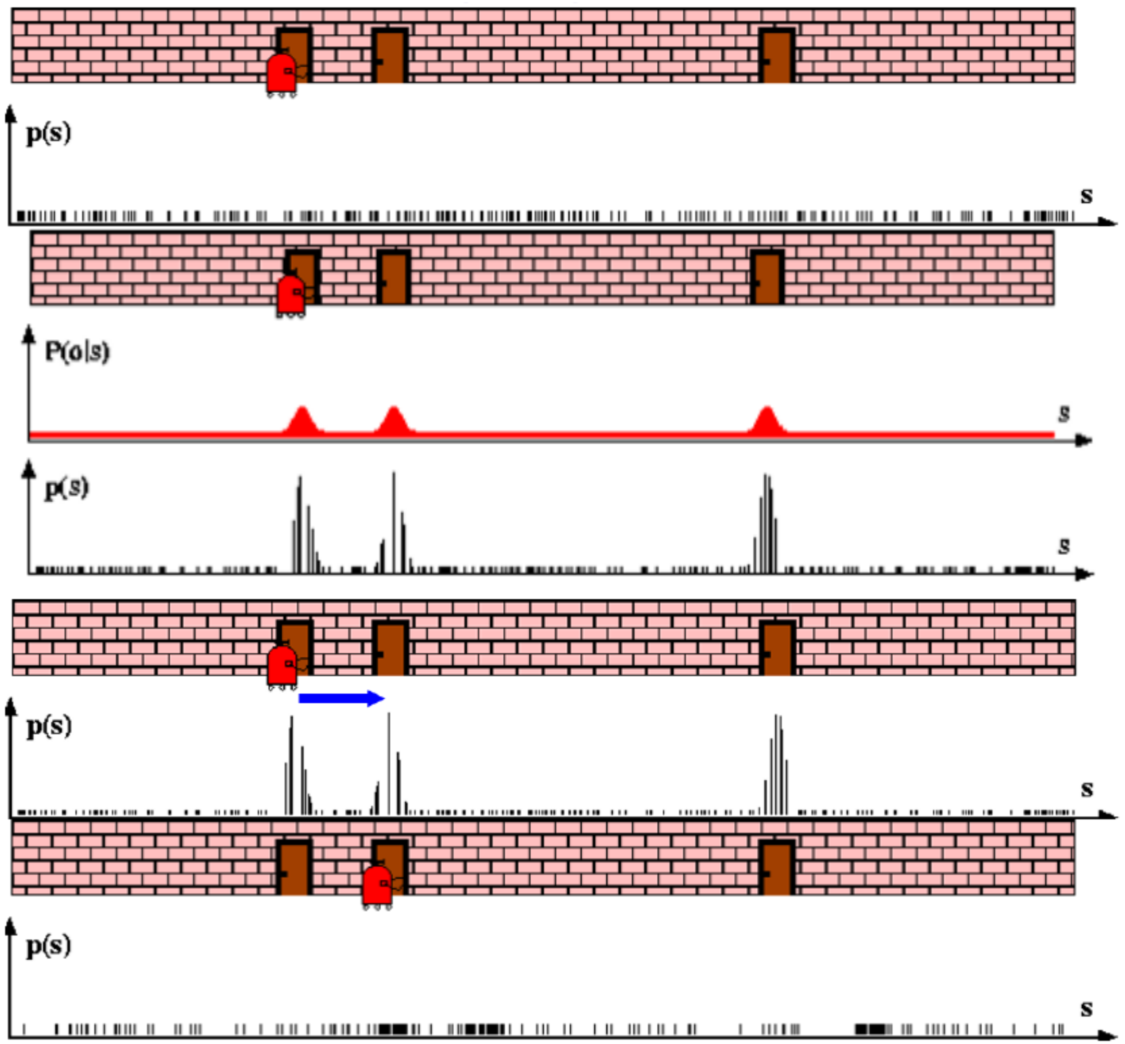
Particle filter for 1D localization

Prior:

Update:

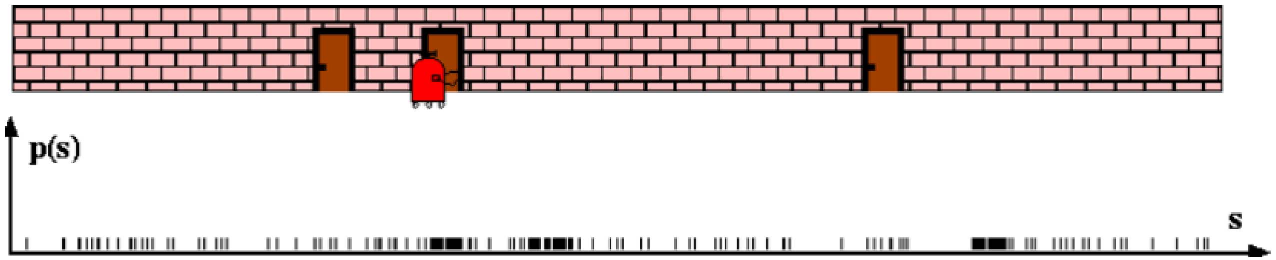
Predict:

Resample:



Particle filter for 1D localization

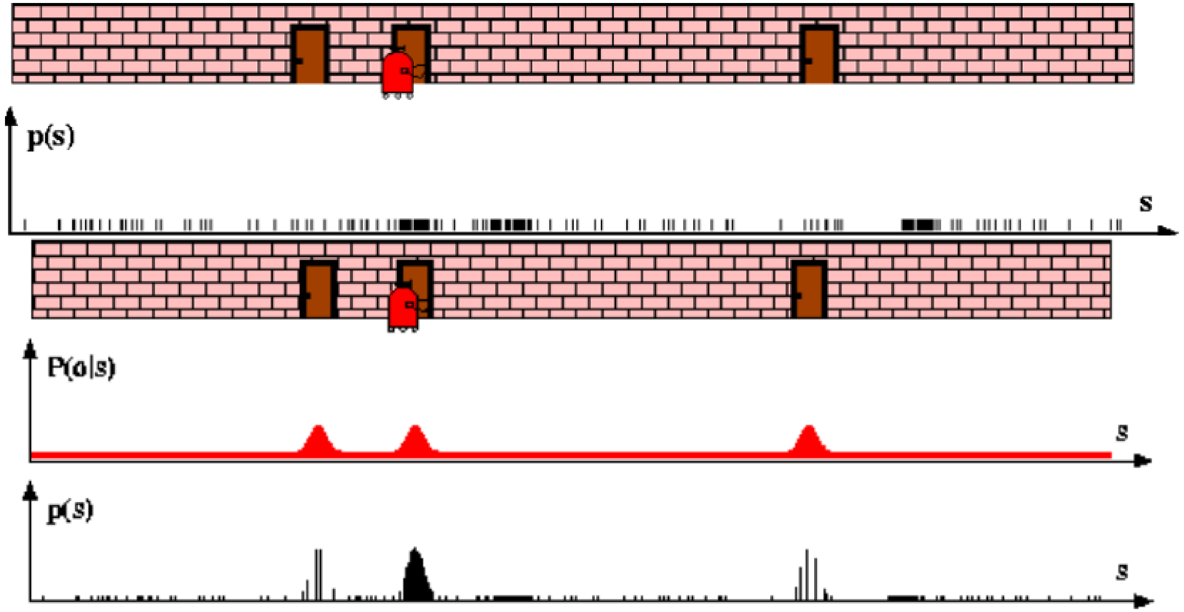
Prior:



Particle filter for 1D localization

Prior:

Update:

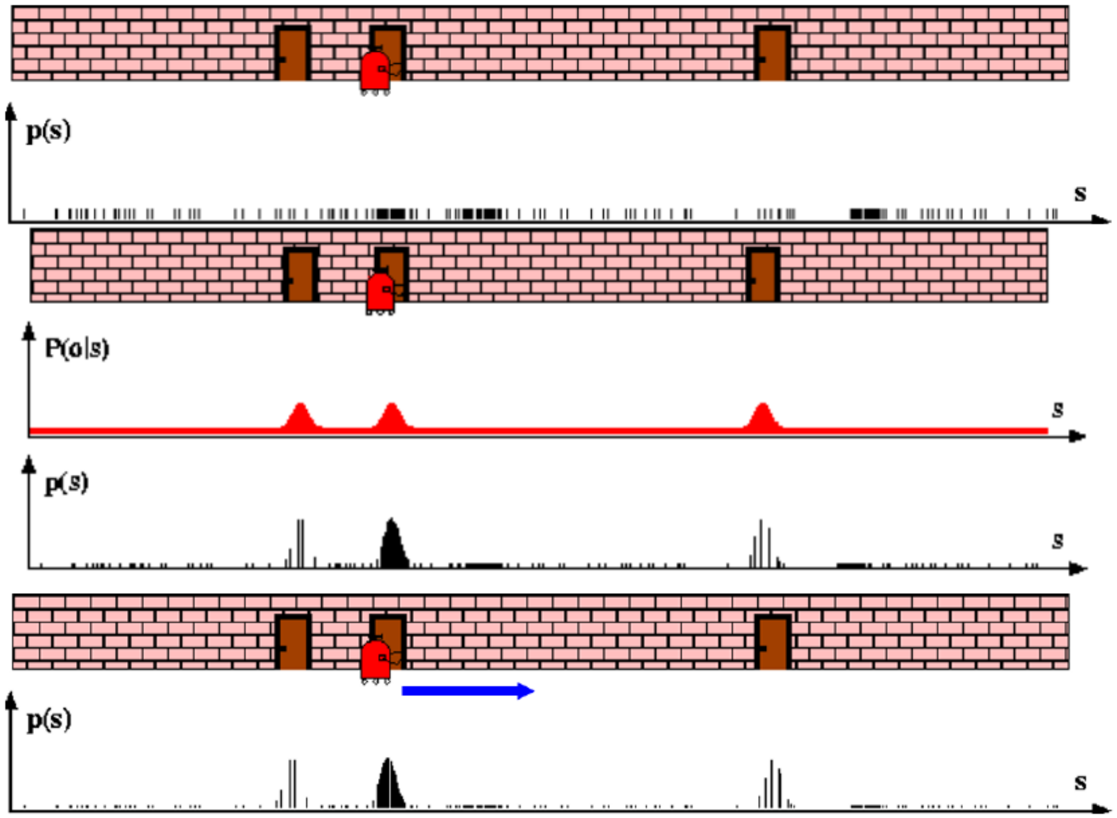


Particle filter for 1D localization

Prior:

Update:

Predict:



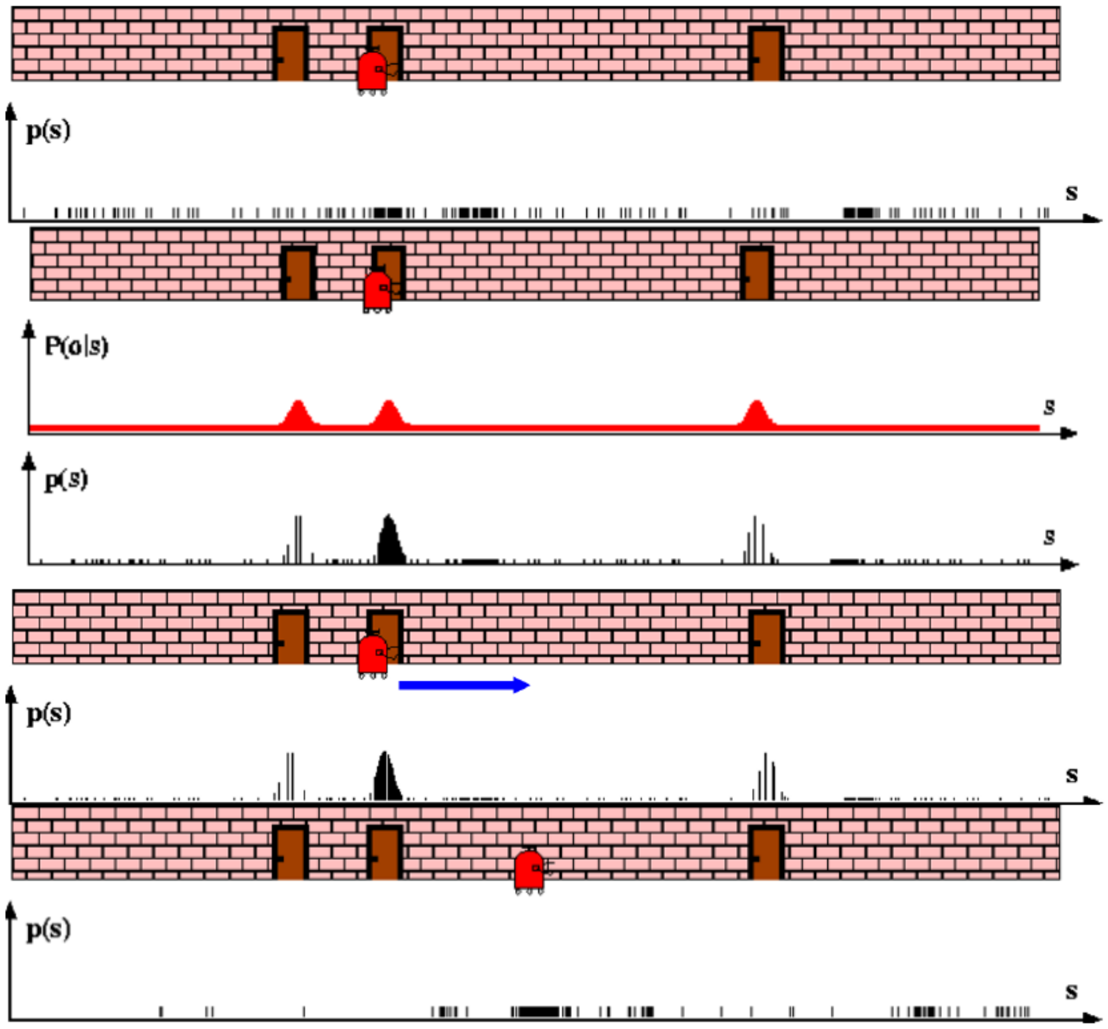
Particle filter for 1D localization

Prior:

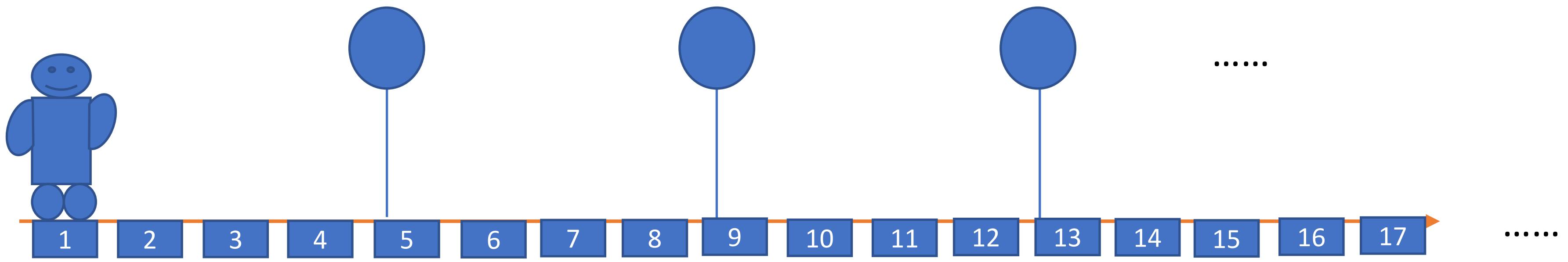
Update:

Predict:

Resample:



Step 1: Generate particles



Step 1: Generate particles based on our prior

- uniformly prior distributed (each spot has one particle)
- each particle moves following with robot moving, no uncertainties in their movements.
- at this step, we assume each particle holds their beliefs only to be true (belief=1) of false (belief=0)

Step 2: Movement uncertainty

Step 2: Add uncertainties in particles' movements

- movement errors follow zero-mean Gaussian distribution with a predefined standard deviation
- create one particle, move 10 times, print and observe the measurements
- uncomment # quit() to see how distribution converges with more samples.

Step 3: Sensor model

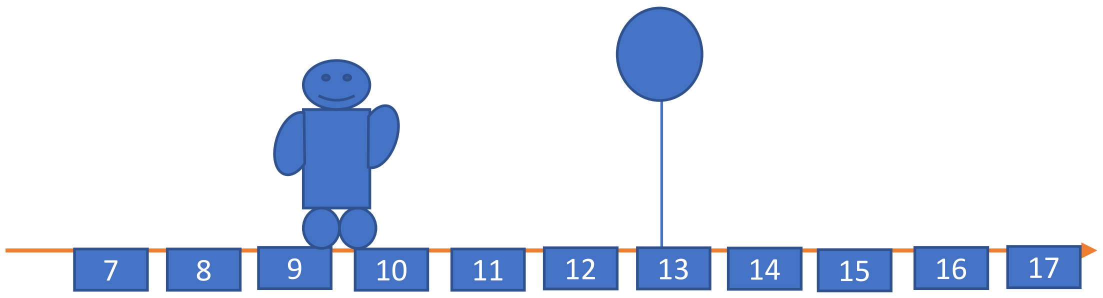
Step 3: More realistic sensor model:

Previous measurement

```python
def detect_pole(self, poles):
    if self.pos + 1 in poles:
        self.pole_detected = True
    else:
        self.pole_detected = False
```

More realistic measurement considering sensor specifications:

- Maximum measurement range: 3 units
- If object within the detection range, report the distance to the closest object
- If no object detected, output -1000



Step 3

Your practice:

Complete the *step-3-st.py*, fill in the measure function so that

"measurement should be XXX"

matches

"you measured: XXX"

Step 4: Update weight

- Step 4: Update weights for each particle while taking into account measurement uncertainty in the sensor model.

- For simplicity, we assume the sensor model: errors follow Gaussian distribution with a predefined standard deviation.
- Qs: if robot got measurement 3.0 units to a pole, particle A is at the location distancing 2.0 units to the pole, particle B at the location distancing 3.0 units. How would you assign the weights?
- Complete the step-4.py: update weights by setting the particle.weight value based on the given Gaussian probability density function.

Step 5: Resampling

Algorithm Low_variance_sampler($\mathcal{X}_t, \mathcal{W}_t$):

$\bar{\mathcal{X}}_t = \emptyset$

$r = \text{rand}(0; M^{-1})$

$c = w_t^{[1]}$

$i = 1$

for $m = 1$ to M do

  $U = r + (m-1) \cdot M^{-1}$

  while $U > c$

    $i = i + 1$

    $c = c + w_t^{[i]}$

  endwhile

  add $x_t^{[i]}$ to $\bar{\mathcal{X}}_t$

endfor

return $\bar{\mathcal{X}}_t$

- Step 5: Resample

  Complete the *resample_particles* function

  - Implementing the low variance sampling algorithm

How the algorithm works:

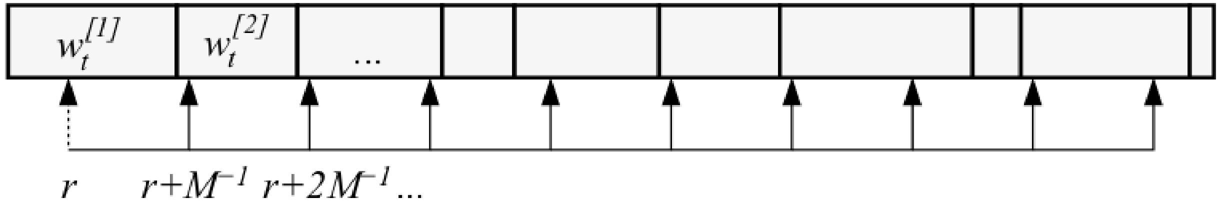


Figure 4.7 Principle of the low variance resampling procedure. We choose a random number r and then select those particles that correspond to $u = r + (m-1) \cdot M^{-1}$ where $m = 1, \ldots, M$.

Step 6: Put it together

- Step 6: Put it together

  - You can reuse all the steps we implemented before

  - Put them together to fill the *step-6-st.py*, which will implement a complete particle filter for 1D localization

Any problems found?
How do you fix it?