Key for the corrections

Particle filter Algorithm

1) **Initialize**: Randomly place particles across the map.

2) **Predict**: Move each particle based on control input and motion noise.

3) **Update**: Weigh each particle by how well it matches sensor data.

4) **Resample**: Select particles based on their weights to focus on high-probability areas.

5) **Repeat**: Iterate with new data for refined location tracking.

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 $\propto w_t^{[i]}$

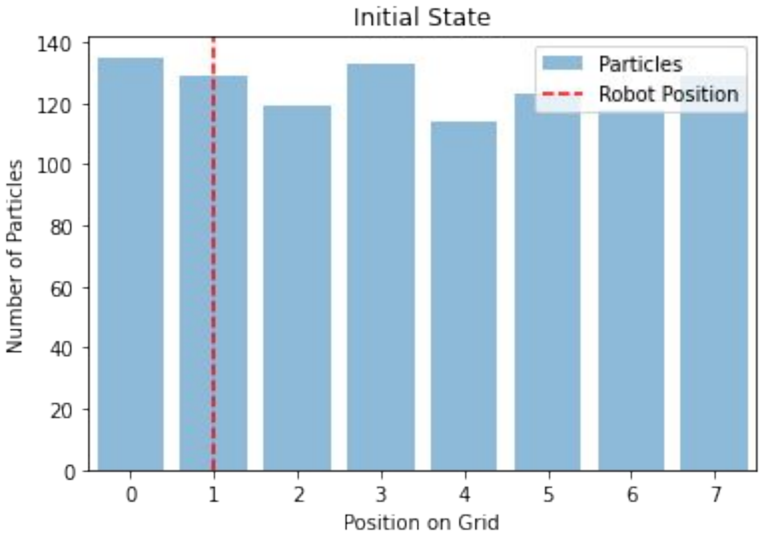
  add $x_t^{[i]}$ to $\mathcal{X}_t$

endfor

return $\mathcal{X}_t$

Initialization

The grid is **initialized** along with N=1000 particles randomly distributed. The robot's true position is also set.



Motion Update

Move particles similarly to the robot, with some noise

```
positions = []
robot_positions = []

for t in range(num_iterations):
    # Move the robot one step to the right
    robot_pos = (robot_pos + 1) % grid_length
    # Move particles similarly to the robot, with some noise
    movement_noise = np.random.choice([-1, 0, 1], N, p=[0.1, 0.8, 0.1])
    particles = (particles + 1 + movement_noise) % grid_length
```

Update weights

Update weights based on sensor reading

```
# Measurement model probabilities
prob_correct = 0.9
prob_incorrect = 0.1
```

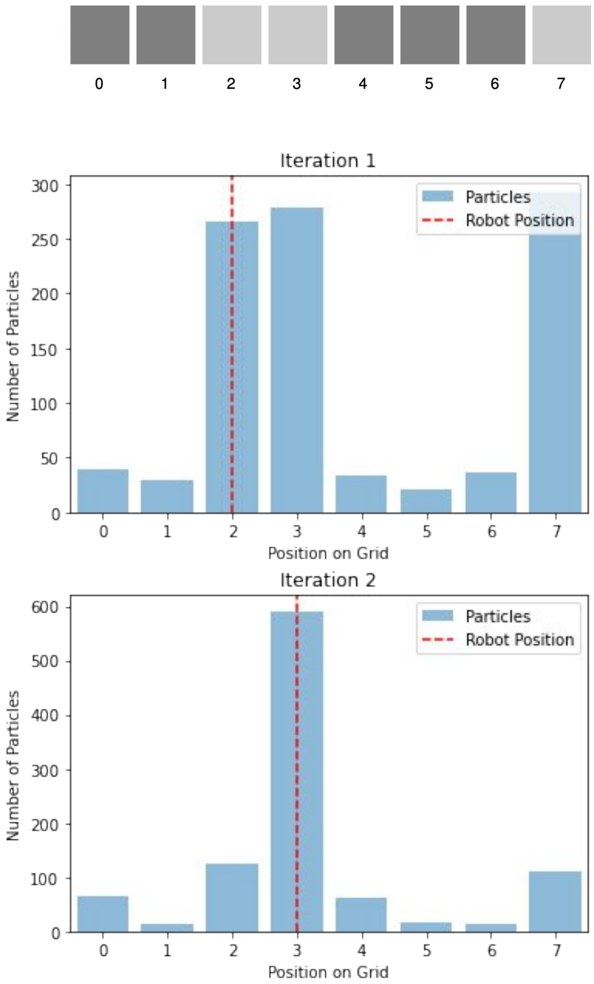
```
# Sensor reading at the robot's true position
sensor_reading = grid[robot_pos]

# Update weights based on sensor reading
weights *= np.where(grid[particles] == sensor_reading, prob_correct, prob_incorrect)

# Normalize weights
weights += 1.e-300      # Avoid division by zero
weights /= weights.sum()
```

Resampling

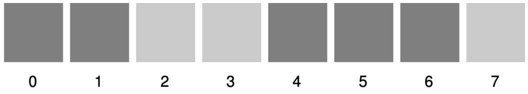
Particles are **resampled** according to their weights, focusing more on likely positions.

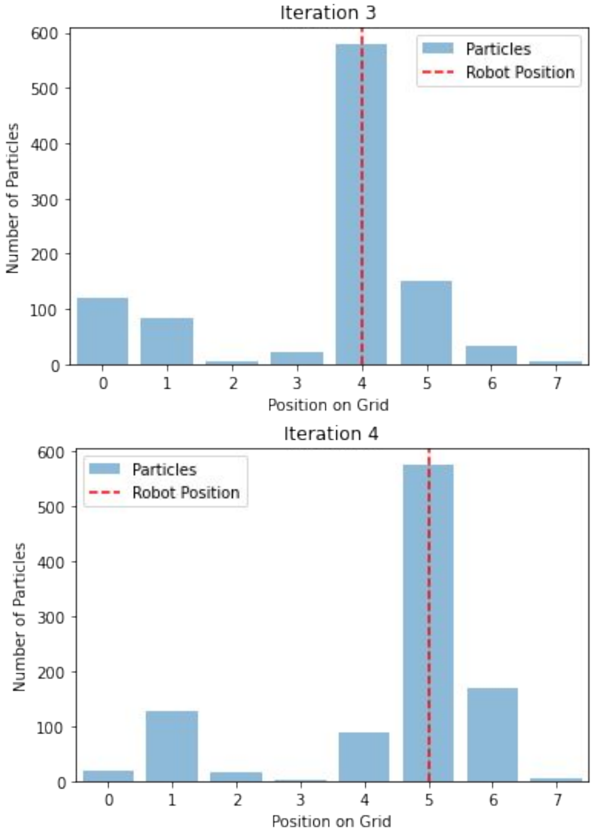
```
# Resample particles based on weights
indices = np.random.choice(N, N, p=weights)
particles = particles[indices]
```

Iterate


0
1
2
3
4
5
6
7
Iteration 1
Particles
Robot Position
Number of Particles
Position on Grid
Iteration 2
Particles
Robot Position
Number of Particles
Position on Grid

Iterate


0
1
2
3
4
5
6
7


Iteration 3
Particles
Robot Position
Number of Particles
Position on Grid
Iteration 4
Particles
Robot Position
Number of Particles
Position on Grid

Correction

Why is 'Resampling' crucial in a particle filter?

- ☐ To distribute particles uniformly
- ☐ To focus particles around the right position
- ☐ To distribute particles in Gaussian distribution
- ☐ To distribute particles in linear distribution

Correction

Why is 'Resampling' crucial in a particle filter?

- [ ] To distribute particles uniformly
- [x] To focus particles around the right position ←
- [ ] To distribute particles in Gaussian distribution
- [ ] To distribute particles in linear distribution

Correction

The value of the weights is related to:

- ☐ Size of the sample
- ☐ Fit to measurement
- ☐ Amplitude of movement

Correction

The value of the weights is related to:

- [ ] Size of the sample
- [x] Fit to measurement
- [ ] Amplitude of movement

Correction

How would adding more noise in the 'Motion Update' reflect the hardware features ?

- [ ] it correspond to less precise motor power control
- [ ] it correspond to less precise ground color sensor
- [ ] it correspond to less precise wheel motion encoder
- [ ] it correspond to more precise ground color sensor

Correction

How would adding more noise in the 'Motion Update' reflect the hardware features ?

- [ ] it correspond to less precise motor power control
- [ ] it correspond to less precise ground color sensor
- [x] **it correspond to less precise wheel motion encoder**
- [ ] it correspond to more precise ground color sensor

Correction

How does the memory requirement of a particle filter compare to a histogram-based approach (=last exercise session with the Thymio) as the environment size increases?

- ○ Particle filter memory increases slightly, while histogram-based increases exponentially.
- ○ Both increase linearly with environment size.
- ○ Particle filter memory decreases, while histogram-based increases.
- ○ Particle filter memory increases linearly, while histogram-based remains constant.

Correction

How does the memory requirement of a particle filter compare to a histogram-based approach (=last exercise session with the Thymio) as the environment size increases?

- ◉ Particle filter memory increases slightly, while histogram-based increases exponentially. ←
- ○ Both increase linearly with environment size.
- ○ Particle filter memory decreases, while histogram-based increases.
- ○ Particle filter memory increases linearly, while histogram-based remains constant.

Correction

Say we are resampling over and over again, without any motion update. What will happen in the limit?

- A) The filter will spread out the evenly across the grid
- B) The filter will maintain an equal distribution of particles across different states
- C) The filter will converge to a single position
- D) The filter will keep oscillating between different particles
- E) I don't know

Correction

Say we are resampling over and over again, without any motion update. What will happen in the limit?

- A) The filter will spread out the evenly across the grid
- B) The filter will maintain an equal distribution of particles across different states
- C) The filter will converge to a single position
- D) The filter will keep oscillating between different particles
- E) I don't know

Comparison with Histogram-based Method

Particle Filters offer flexibility and can handle **non-linear**, **non-Gaussian problems**, ideal for complex, real-world scenarios.

In contrast, **Histogram-based** methods are simpler but struggle with **high-dimensional spaces** and assume linear, Gaussian noise.