COLUMBIA UNIVERSITY EEME E6911 FALL '25

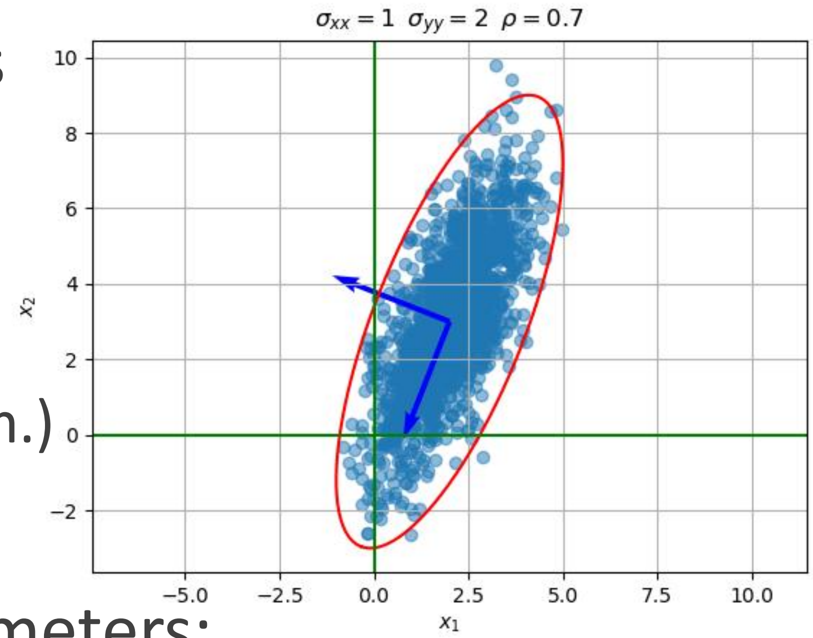
TOPICS IN CONTROL : PROBABILISTIC ROBOTICS

PARTICLE FILTER

Instructor: Ilija Hadzic

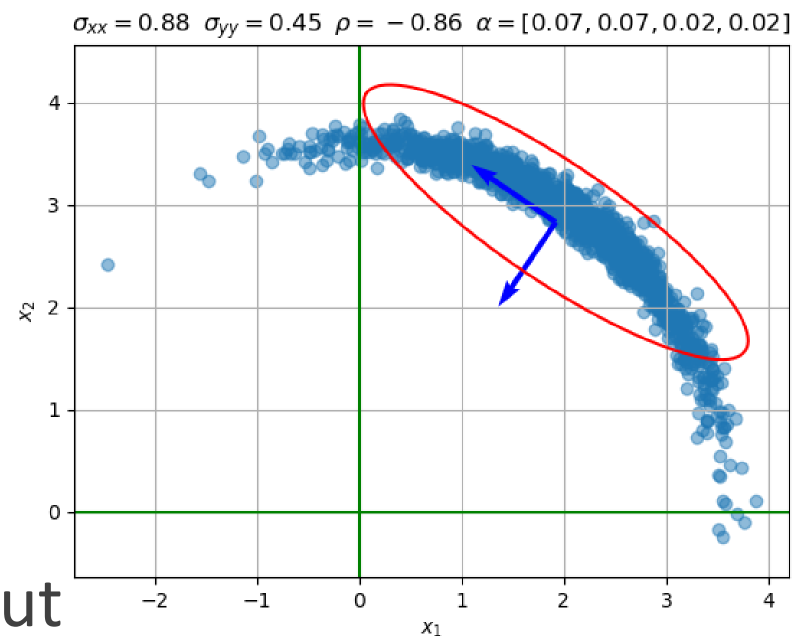
Recall Gaussian Distribution

- Draw and plot realizations
- What do we get?
  - Ellipse (2D)
  - Ellipsoid (3D)
  - Hyper-ellipsoid (higher dim.)
- Single mode
- Fully defined by two parameters:
  - Mean
  - Covariance



Recall Banana Distribution

- Draw and plot realizations
- Find best-fit ellipse
- Pick your favorite method
- Never a good match!
- EKF/KF **forces** Gaussian
- Introduces error at the input
- Can we track non-Gaussian distributions better?



Particle Filter – General Concept

- Represent distribution with N realizations
  - We call them particles
  - For $N \to \infty$ realizations represent true distributions
  - In practice N is finite (typically 500-1000)
- Prediction model:
  - Track/predict each particle state
  - Add randomness
- Measurement model:
  - Evaluate likelihood of each particle
- Resampling:
  - Kill or reproduce each particle proportional to the likelihood

Algorithm – Prediction + Scoring

- Thrun, p98, Table 4.3
- Prior (set of particles):
  - $\mathcal{X}[n-1] = \{x^{[m]}[n-1], m \in [0, N)\}$
- Sample from prediction model (new set) of particles
- Each particle has associated **likelihood** score
  - $\bar{\mathcal{X}}[n] = \left\{(\bar{x}^{(m)}[n], w^{[m]}[n]), m \in [0, N)\right\}$
  - $\bar{x}^{[m]}[n] \sim p\left(\bar{x}\middle|u[n], x^{[m]}[n-1]\right)$
  - $w^{[m]} = p(z[n]|\bar{x}^{[m]}[n])$
- This is our “bucket” of “eligible” particles

Algorithm – Resampling

- Start with the “bucket”
- Draw N instances of $x^{[m]}[n]$:
  - $\mathcal{X}[n] = \left\{x^{[m]}[n], m \in [0, N)\right\}$
  - $s[n] = \sum w^{i}[n]$
  - $p\left(x^{[m]}[n] = \bar{x}^{[i]}[\mathrm{n}]\right) = w^{[i]}[n]/s[n],\ (\bar{x}^{[i]}[\mathrm{n}], \mathrm{w}^{[i]}[n]) \in \bar{\mathcal{X}}[t]$
- Some particles may not be drawn at all (killed)
- Some particles may be drawn multiple times (regenerated)

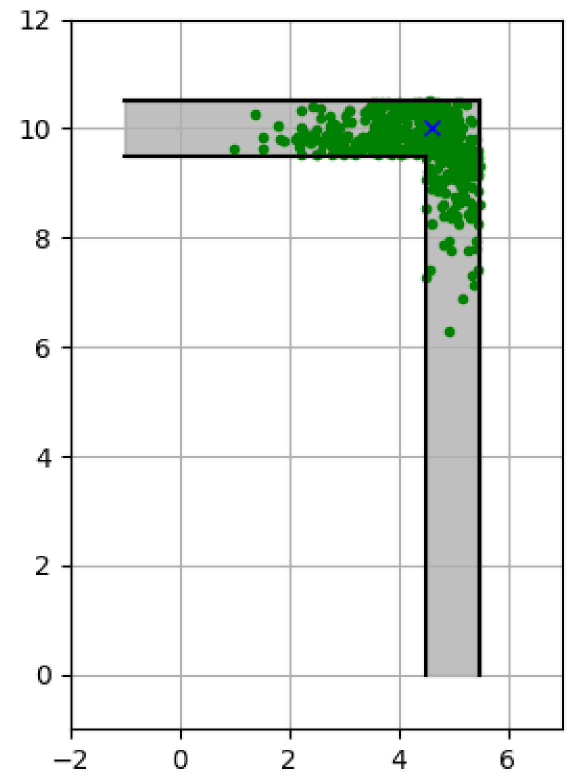
Algorithm – Resampling

Normalize before drawing, need legit probability!

- Start with the "bucket"
- Draw N instances of $x^{[m]}[n]$:
  - $\mathcal{X}[n] = \{x^{[m]}[n], m \in [0, N)\}$
  - $s[n] = \sum w^{i}[n]$
  - $p\left(x^{[m]}[n] = \bar{x}^{[i]}[\mathrm{n}]\right) = w^{[i]}[n]/s[n],\ (\bar{x}^{[i]}[\mathrm{n}], \mathrm{w}^{[i]}[n]) \in \bar{\mathcal{X}}[t]$
- Some particles may not be drawn at all (killed)
- Some particles may be drawn multiple times (regenerated)

Example: Fusing GPS and map

- Vehicle moves along the L-shaped road.
- Road map is known to the estimator.
- State is the particle cloud along the road
- GPS measurements follow Gaussian error model with 1m standard deviation



Motion Model

$$\boldsymbol{v}(x,y)+\boldsymbol{\epsilon}=\begin{cases}\begin{cases}-v\boldsymbol{e_x} & \text{random}=1\\ v\boldsymbol{e_y} & \text{random}=0\end{cases} & \text{intersection}\\ v\boldsymbol{e_y} & \text{vertical segment}\\ -v\boldsymbol{e_x} & \text{horizontal segment}\end{cases}$$

- Noise $\boldsymbol{\epsilon}$ properties:
  - Draw sample from Gaussian zero-mean $\mathcal{N}(\mathbf{0},\boldsymbol{\Sigma}_v)$.
  - If particle after motion ends up off-road, draw again.
  - Repeat until all particles have moved.

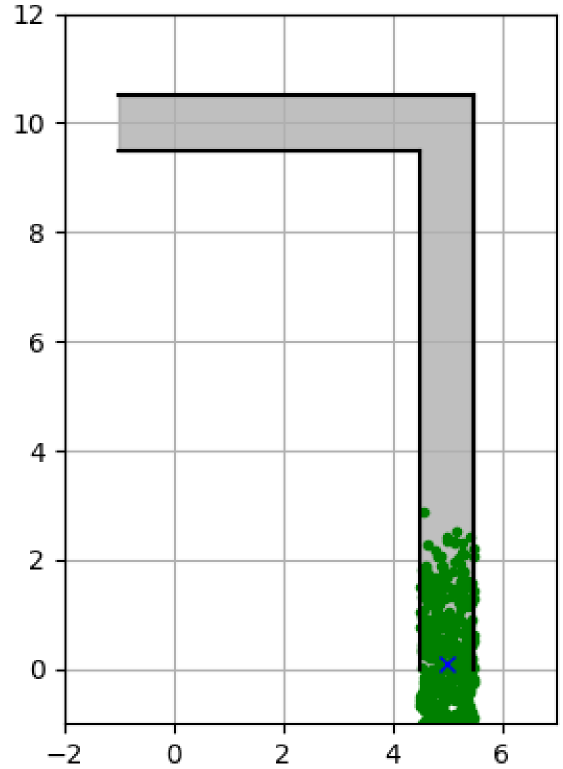
Motion Model – helper functions

```
def move_up(self, x, y, delta_t):
    return np.random.multivariate_normal(
        mean = [ x, y + self.velocity * delta_t],
        cov = [
            [ self.velocity_variance * delta_t * delta_t, 0 ],
            [ 0, self.velocity_variance * delta_t * delta_t ]
        ]
    )

def move_left(self, x, y, delta_t):
    return np.random.multivariate_normal(
        mean = [ x - self.velocity * delta_t, y],
        cov = [
            [ self.velocity_variance * delta_t * delta_t, 0 ],
            [ 0, self.velocity_variance * delta_t * delta_t ]
        ]
    )

def is_on_road(self, x, y):
    if x > self.x1 and x < self.x2 and y < self.y2:
        return True
    if y > self.y1 and y < self.y2 and x < self.x2:
        return True
    return False
```

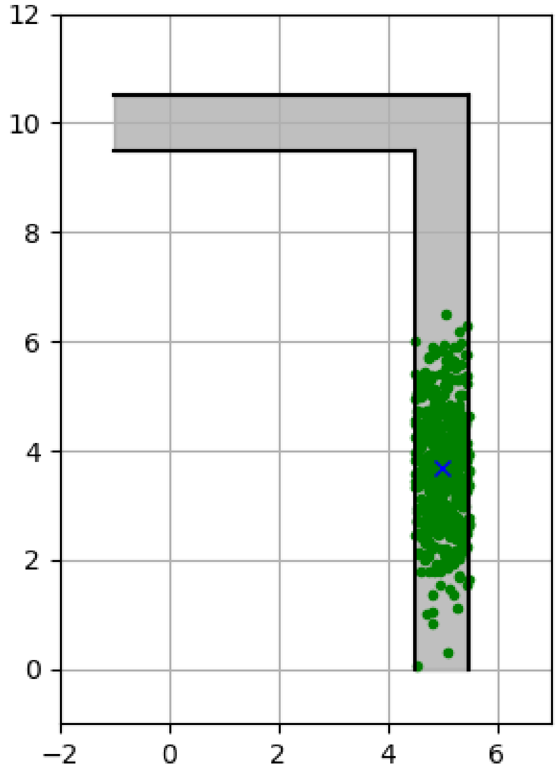
Motion Model – each particle

```
while True:
    if x > self.x1 and y > self.y1:
        # intersection, moving in either direction
        if random.choice([True, False]):
            x_new, y_new = move_up(x, y, delta_t)
        else:
            x_new, y_new = move_left(x, y, delta_t)
    elif x > self.x1:
        # vertical road segment, moving up
        x_new, y_new = move_up(x, y, delta_t)
    else:
        # horizontal road segment, moving left
        x_new, y_new = move_left(x, y, delta_t)
    if self.is_on_road(x_new, y_new):
        break
return x_new, y_new
```

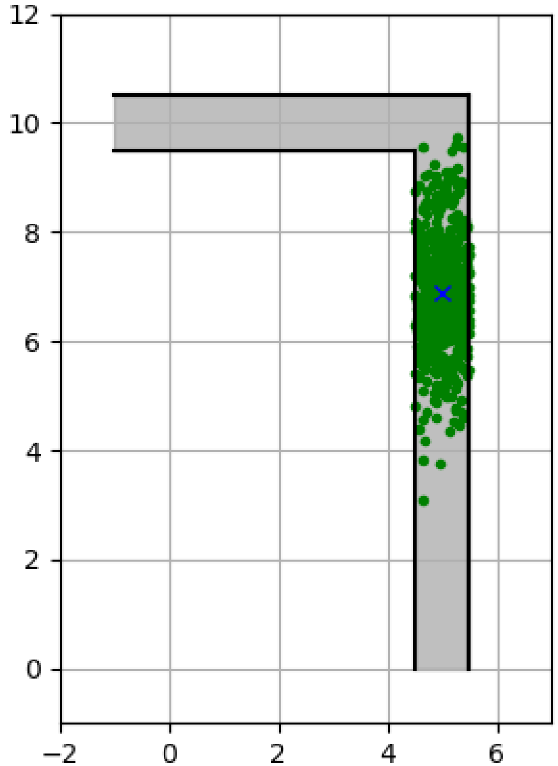
Motion Model – visualization



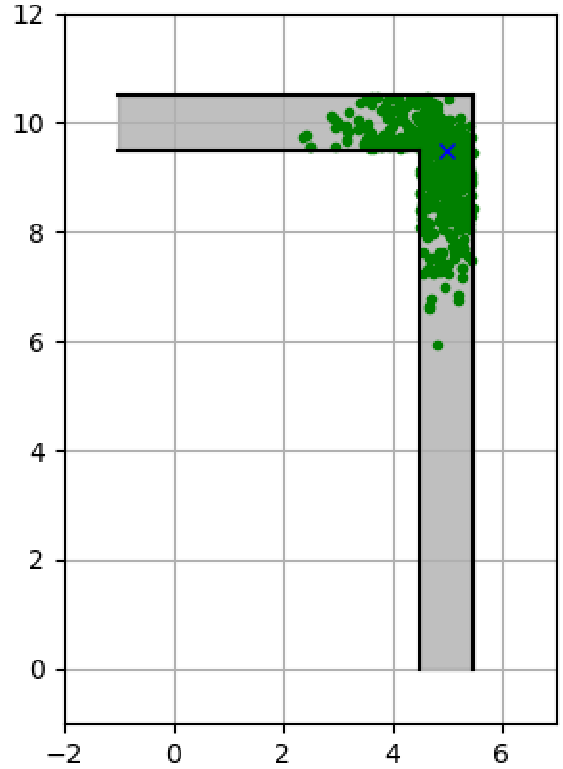
Motion Model – visualization



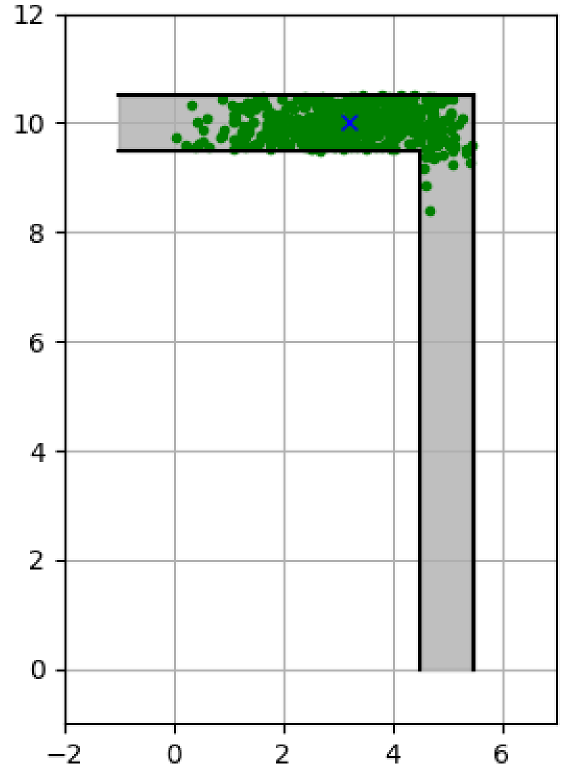
Motion Model – visualization



Motion Model – visualization



Motion Model – visualization



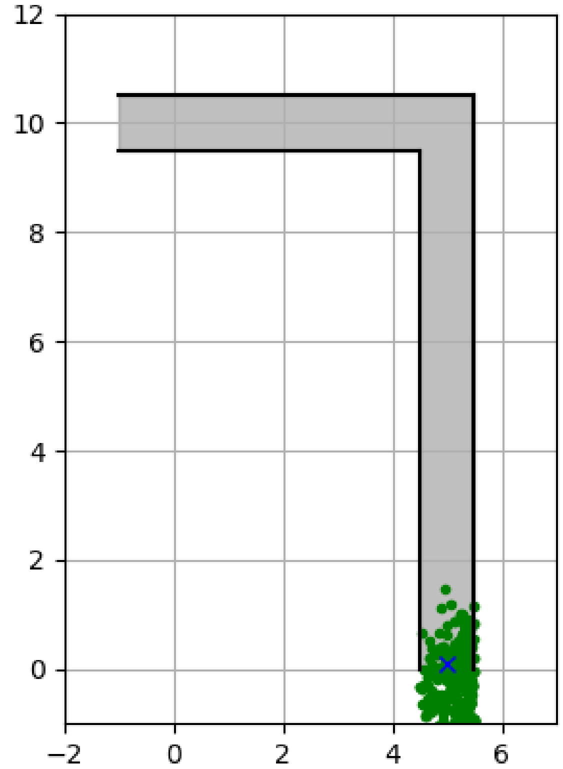
Measurement Model

- Gaussian model
- Calculate the Gaussian PDF for $(x_p - x_m, y_p - y_m)$
- x_p, y_p: particle position
- x_m, y_m: reported measurement
- Normalize

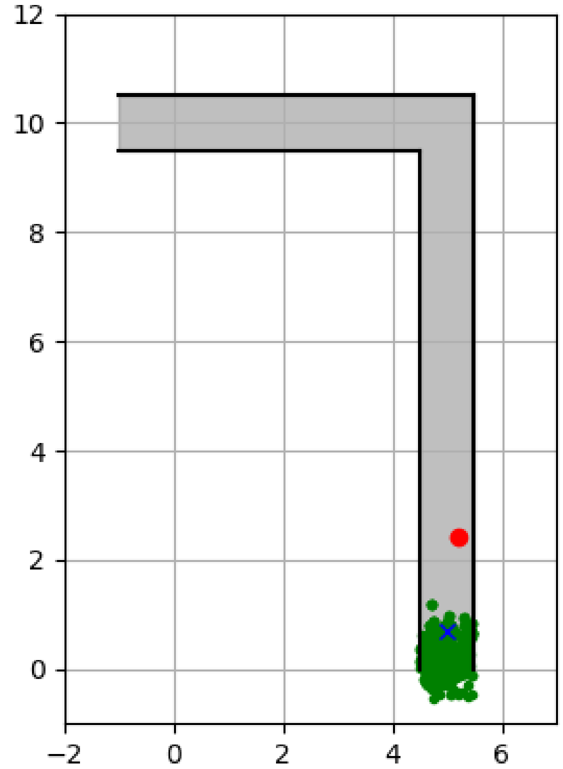
Measurement Model

```
def score_particles(self):
    particle_importance = [
        norm.pdf(np.linalg.norm([x - self.measurement_x,
                                 y - self.measurement_y]),
                                 scale = self.measurement_variance) \
        for x, y in self.predicted_particles
    ]
    self.particle_importance = [
        x / sum(particle_importance) for x in particle_importance ]


def resample(self):
    sample_indices = np.random.choice(
        np.arange(0, self.num_particles),
        size = self.num_particles,
        p = self.particle_importance
    )
    self.particles = [
        self.predicted_particles[i] for i in sample_indices
    ]
```

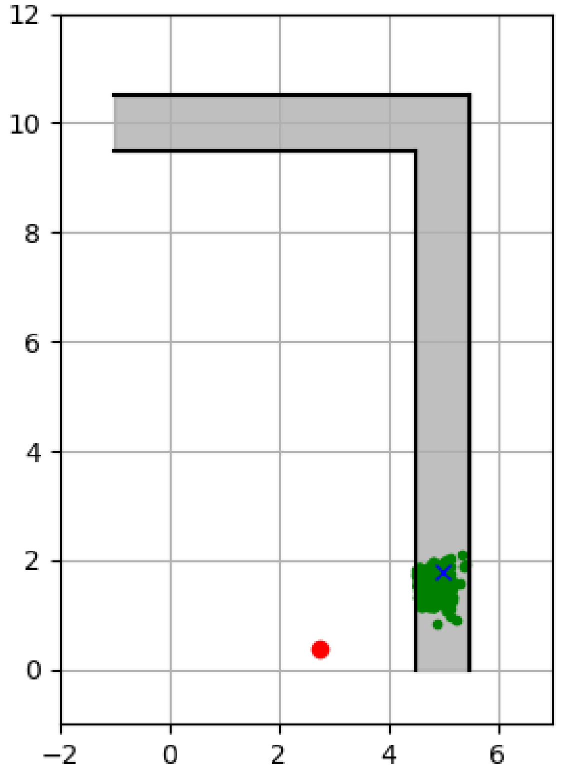
Filter Output – visualization



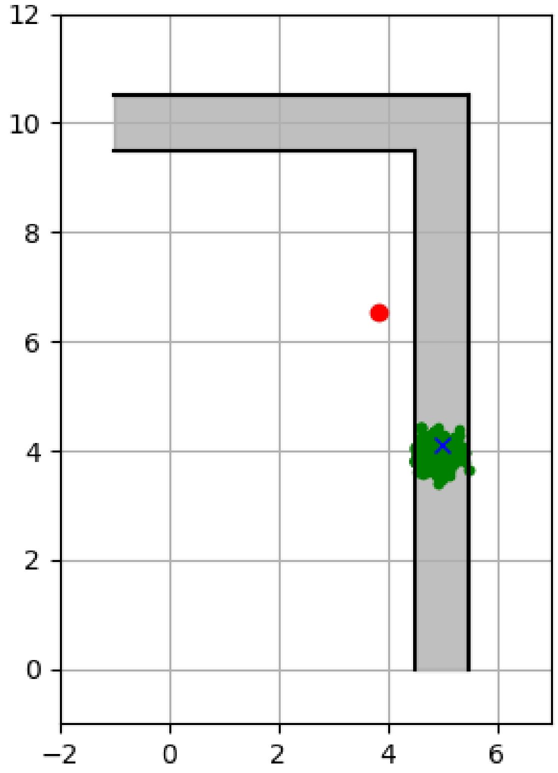
Filter Output – visualization



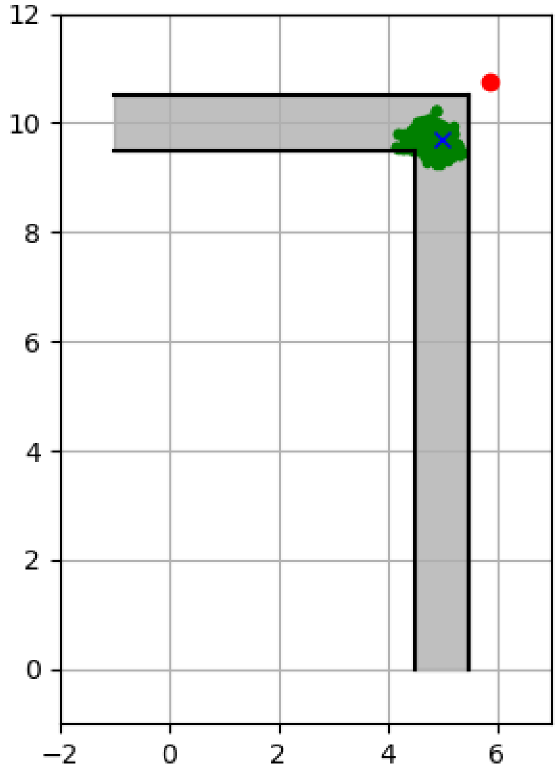
Filter Output – visualization



Filter Output – visualization



Filter Output – visualization



Exercises

- Full code is available in Canvas.
- Instrument the code to measure how much time the filter spends in each step.
- Vary the number of particles and evaluate how the runtime performance scales.

Practical Concerns

- PF handles multimodal and weird-shape distributions well at the expense of computational load.
- Alternative (multi-modal distribution): multi-state (Gaussian mixture) EKF.
- Particle Deprivation
- Sampling Bias
- Dynamic Particle Cloud Size
- Cloud Clustering