UNIVERSITY OF SOUTH CAROLINA

CSCE 774 ROBOTIC SYSTEMS

Particle Filters

Ioannis Rekleitis

Bayesian Filter

- Estimate state ***x*** from data *Z*
  - *What is the probability of the robot being at x?*
- ***x*** could be robot location, map information, locations of targets, etc…
- *Z* could be sensor readings such as range, actions, odometry from encoders, etc…)
- This is a general formalism that does not depend on the particular probability representation
- Bayes filter **recursively** computes the posterior distribution:

$$Bel(x_T) = P(x_T \mid Z_T)$$

Iterating the Bayesian Filter

- Propagate the motion model:

$$Bel\_(x_t) = \int P(x_t \mid a_{t-1}, x_{t-1}) Bel(x_{t-1}) dx_{t-1}$$

Compute the current state estimate before taking a sensor reading by integrating over all possible previous state estimates and applying the motion model

- Update the sensor model:

$$Bel(x_t) = \eta P(o_t \mid x_t) Bel\_(x_t)$$

Compute the current state estimate by taking a sensor reading and multiplying by the current estimate based on the most recent motion history

Mobile Robot Localization

(Where Am I?)

- A mobile robot moves while collecting sensor measurements from the environment.
- Two steps, action and sensing:
  - Prediction/Propagation: what is the robots pose **x** after action **A**?
  - Update: Given measurement **z**, correct the pose **x'**
- What is the probability density function (*pdf*) that describes the uncertainty **P** of the poses **x** and **x'**?

(X,Y,θ)

State Estimation

- Propagation

$$P(x_{t+1}^{-} \mid x_t, \alpha)$$

- Update

$$P(x_{t+1}^{+} \mid x_{t+1}^{-}, z_{t+1})$$

Traditional Approach Kalman Filter

- Optimal for linear systems with Gaussian noise
- Extended Kalman filter:
  - Linearization
  - Gaussian noise models
- Fast!

Monte-Carlo State Estimation
(Particle Filtering)

- Employing a Bayesian Monte-Carlo simulation technique for pose estimation.
- A particle filter uses N samples as a discrete representation of the probability distribution function (*pdf*) of the variable of interest:

$$S = [\vec{\mathbf{x}}_i, w_i : i = 1 \cdots N]$$

where $\mathbf{x}_i$ is a copy of the variable of interest and w_i is a weight signifying the quality of that sample.

In our case, each particle can be regarded as an alternative hypothesis for the robot pose.

Particle Filter (cont.)

The particle filter operates in two stages:

- **Prediction:** After a motion (α) the set of particles S is modified according to the action model

$$S' = f(S, \alpha, \nu)$$

where (ν) is the added noise.

The resulting *pdf* is the prior estimate before collecting any additional sensory information.

Particle Filter (cont.)

- **Update:** When a sensor measurement (z) becomes available, the **weights** of the particles are updated based on the likelihood of (z) given the particle x_i

$$w_i' = P(z \mid \vec{\mathbf{x}}_i) w_i$$

The *updated particles* represent the posterior distribution of the moving robot.

Remarks:

- **In theory**, for an infinite number of particles, this method models the true *pdf*.
- **In practice**, there are always a finite number of particles.

Resampling

For finite particle populations, we must focus population mass where the *PDF* is substantive.

- Failure to do this correctly can lead to divergence.
- Resampling needlessly also has disadvantages.

One way is to estimate the need for resampling based on the variance of the particle weight distribution, in particular the coefficient of variance:

$$cv_t^2 = \frac{\mathrm{var}(w_t(i))}{E^2(w_t(i))} = \frac{1}{M}\sum_{i=1}^{M}(Mw_t(i)-1)^2$$

$$ESS_t = \frac{M}{1+cv_t^2}$$

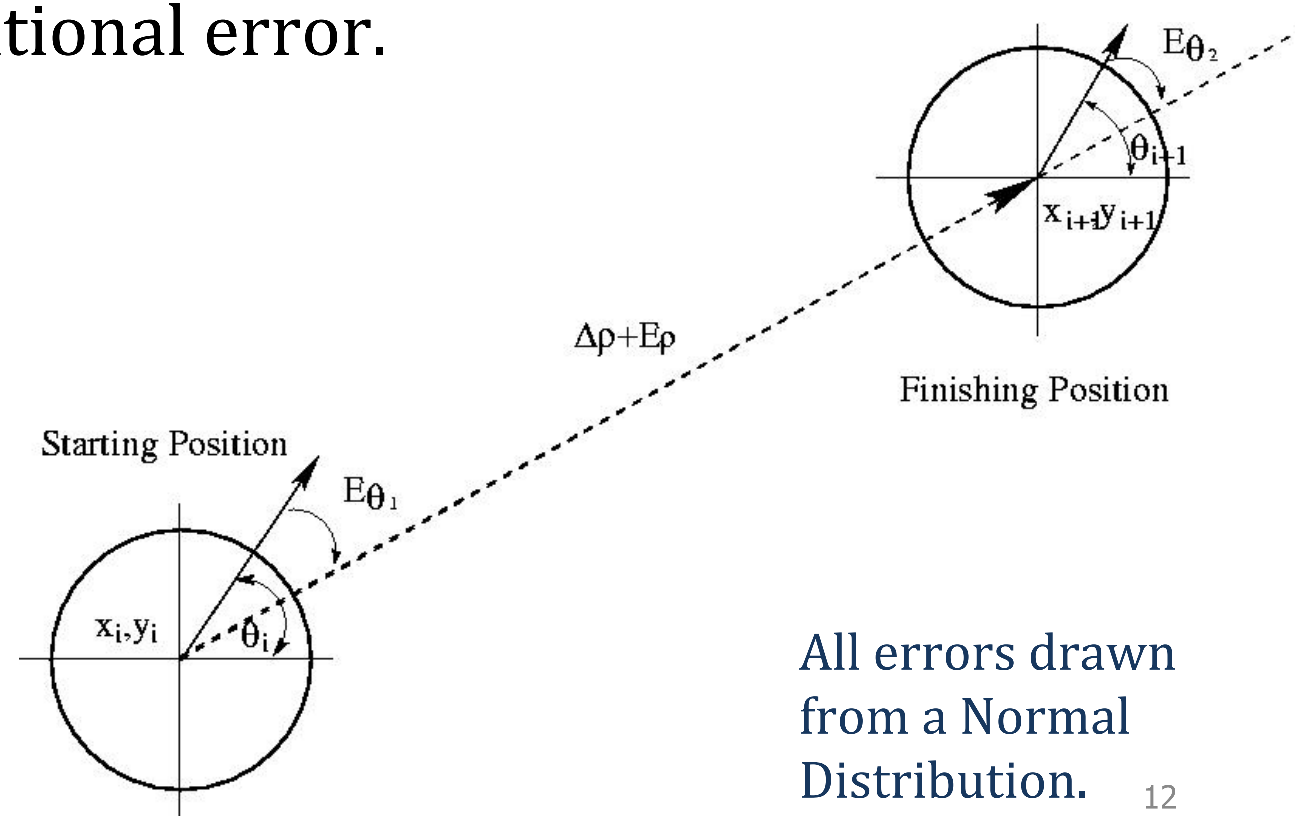
Prediction: Odometry Error Modeling

- **Piecewise linear motion: a simple example.**
- **Rotation**: Corrupted by Gaussian Noise.
- **Translation**: Simulated by multiple steps. Each step models translational and rotational error.

Single step:

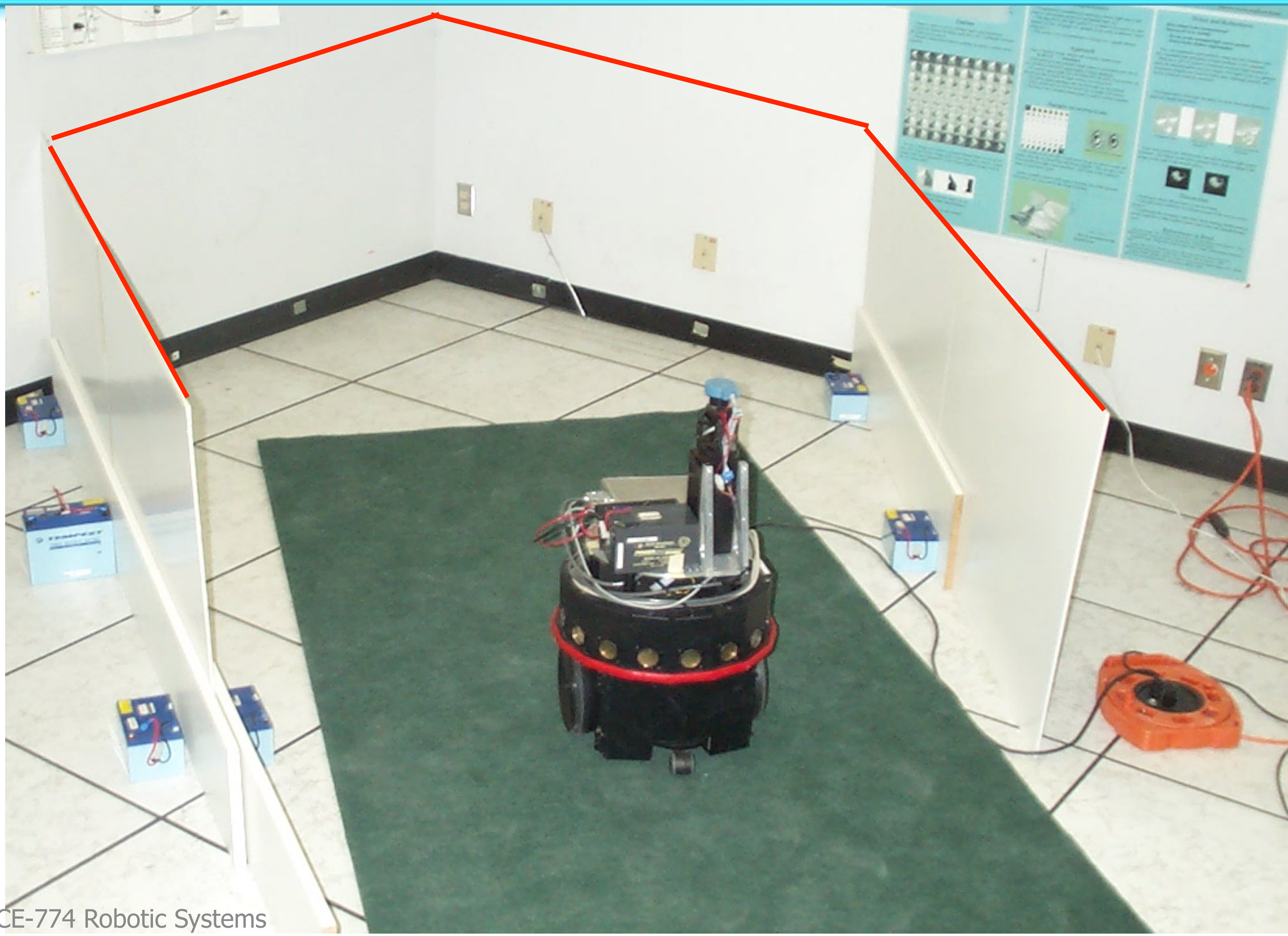
Small *rotational* error (drift) before and after the translation.

Translational error proportional to the distance traveled.

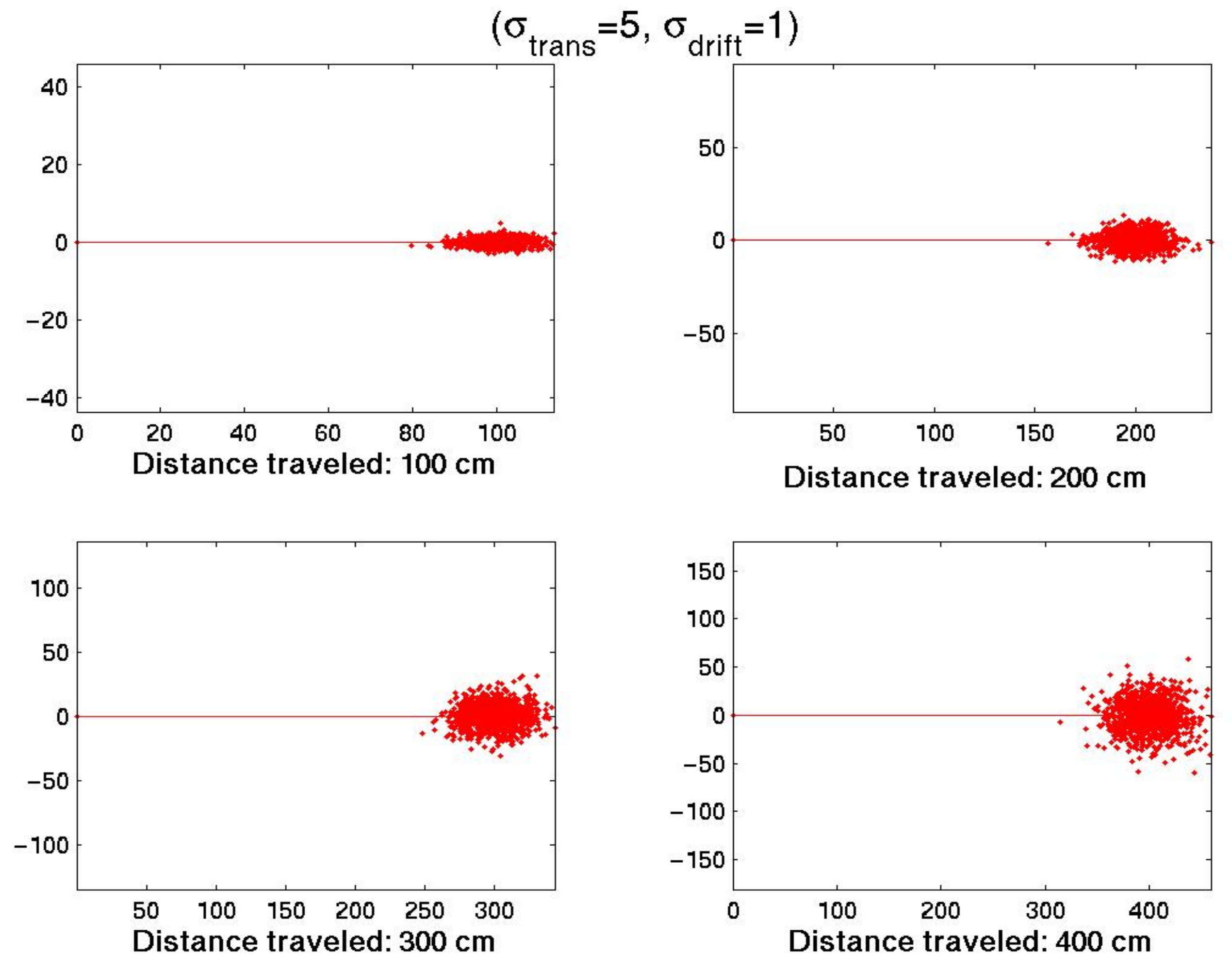


All errors drawn from a Normal Distribution.

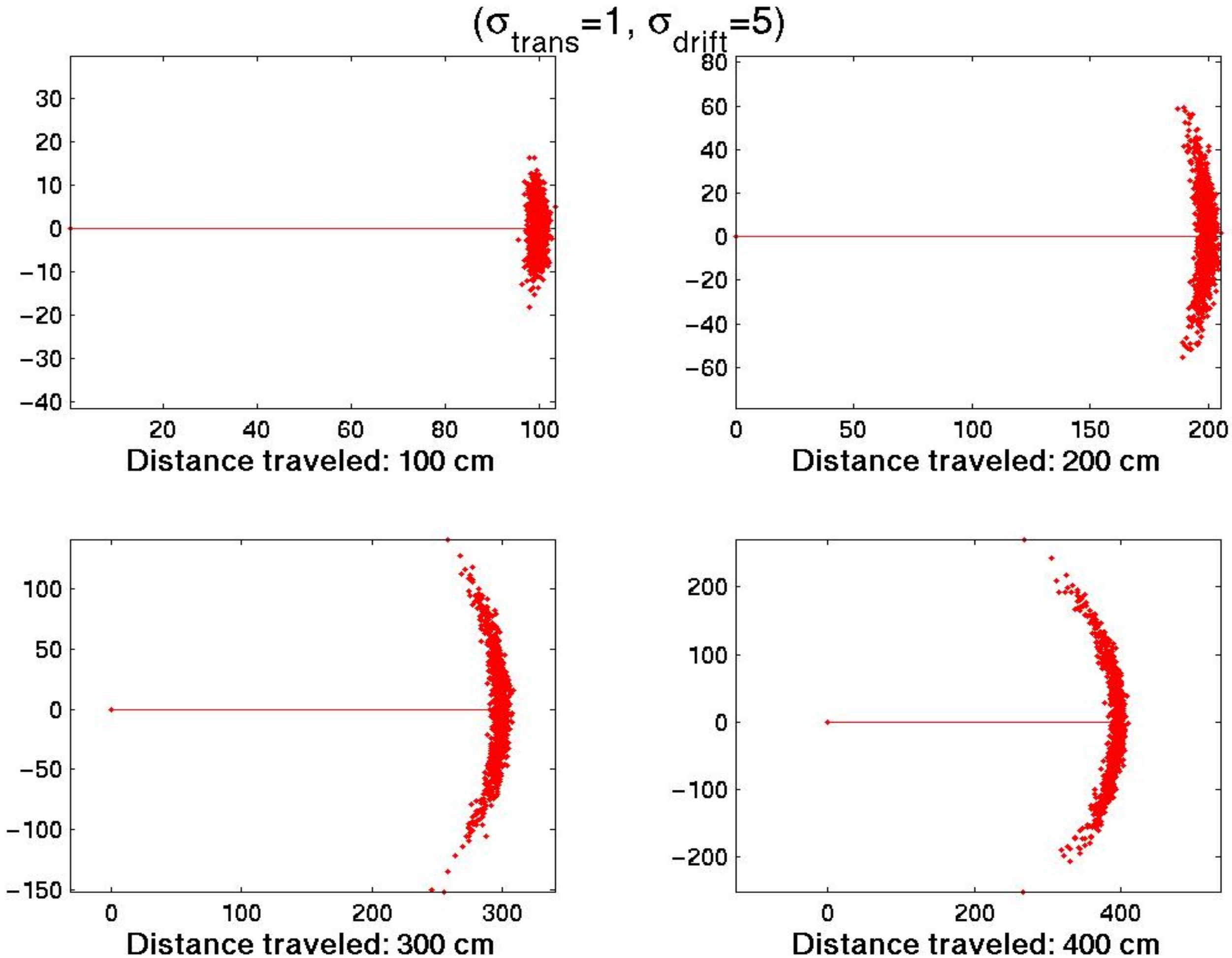
Odometry Error Modeling



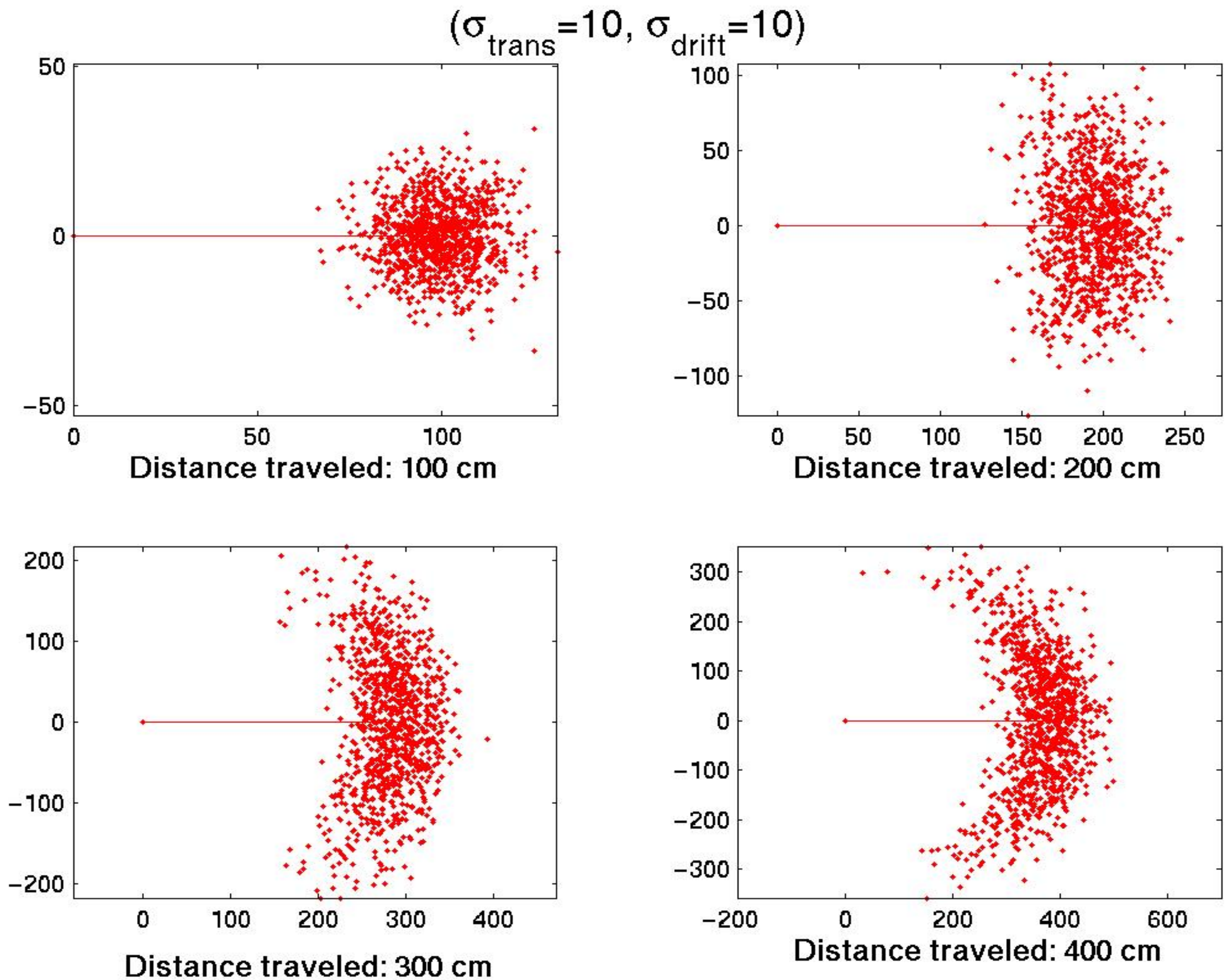
Odometry Error Modeling



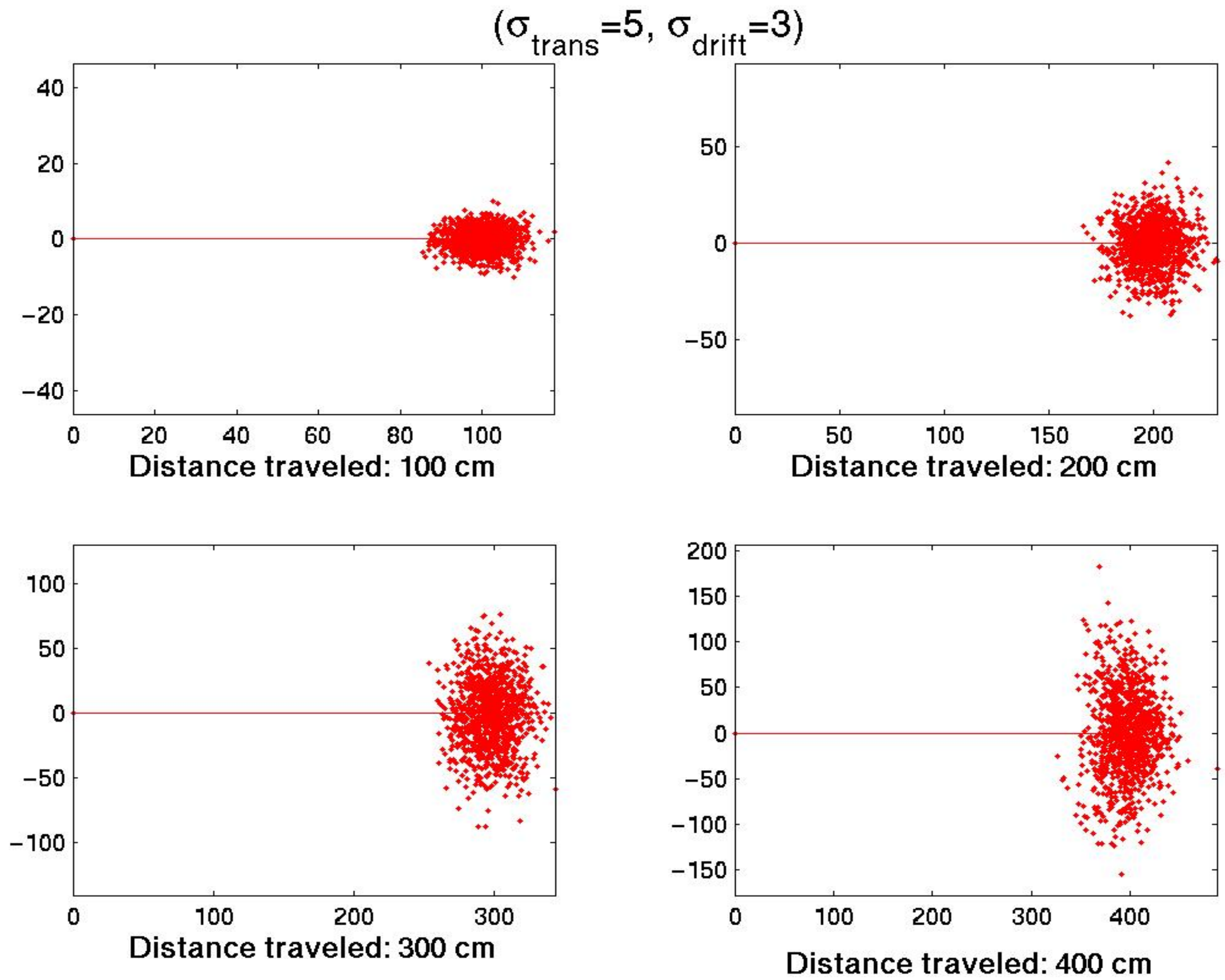
Odometry Error Modeling



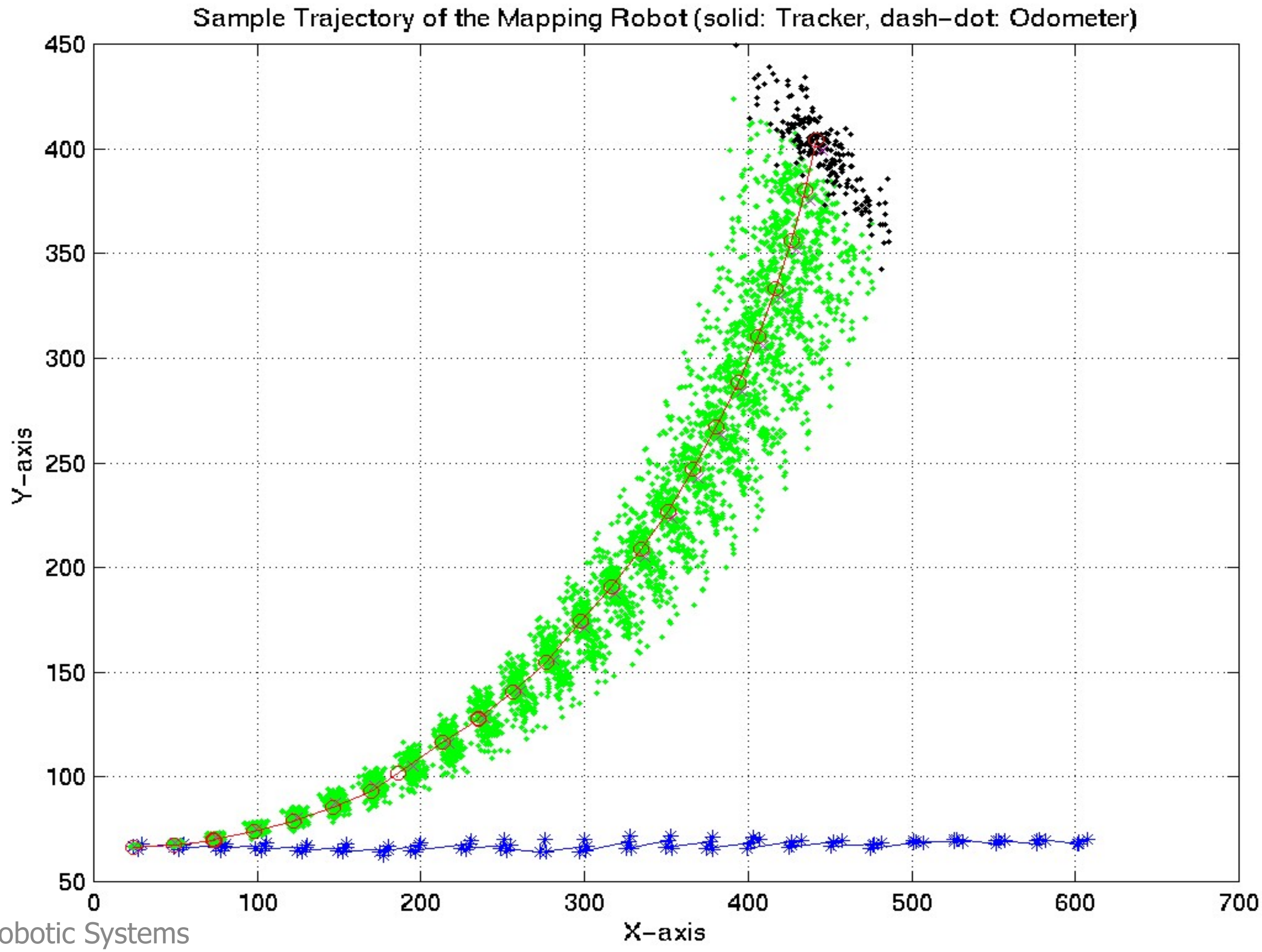
Odometry Error Modeling



Odometry Error Modeling



Prediction-Only Particle Distribution



Propagation of a discrete time system (δt=1 sec)

$$x_i^{t+1} = x_i^t + (v_t + w_{v_t})\delta t \cos\phi_i^t$$

$$y_i^{t+1} = y_i^t + (v_t + w_{v_t})\delta t \sin\phi_i^t$$

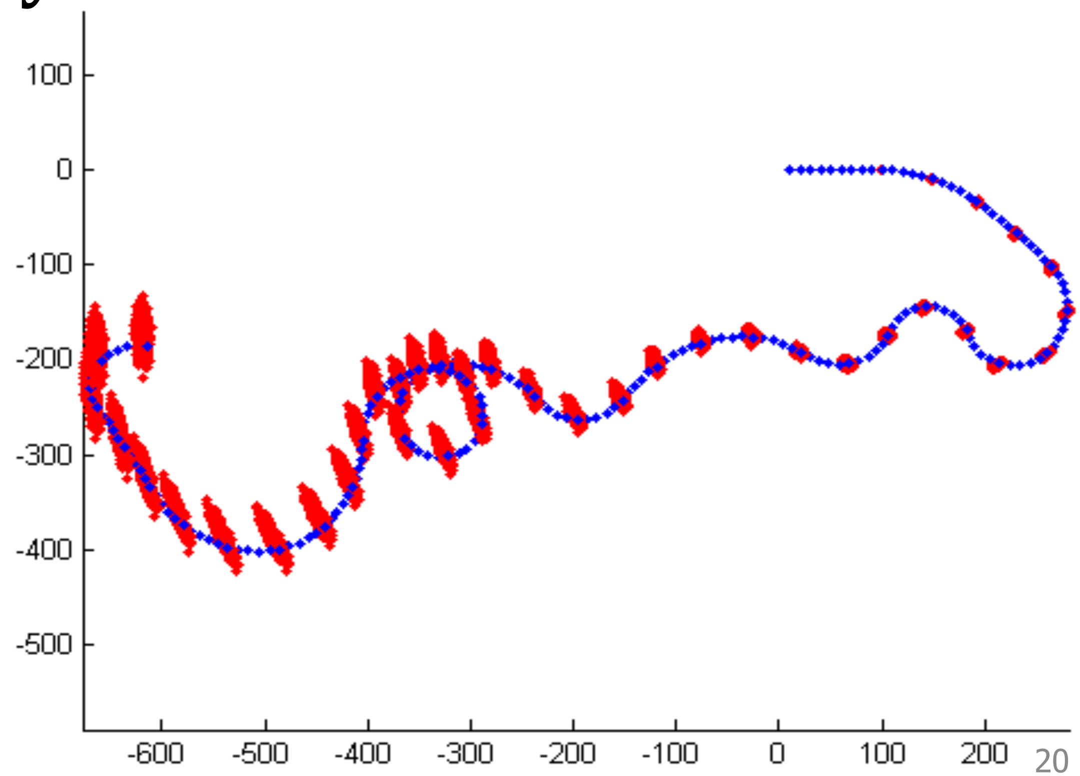
$$\phi_i^{t+1} = \phi_i^t + (\omega_t + w_{\omega_t})\delta t$$

Where w_{v_t} is the additive noise for the linear velocity, and

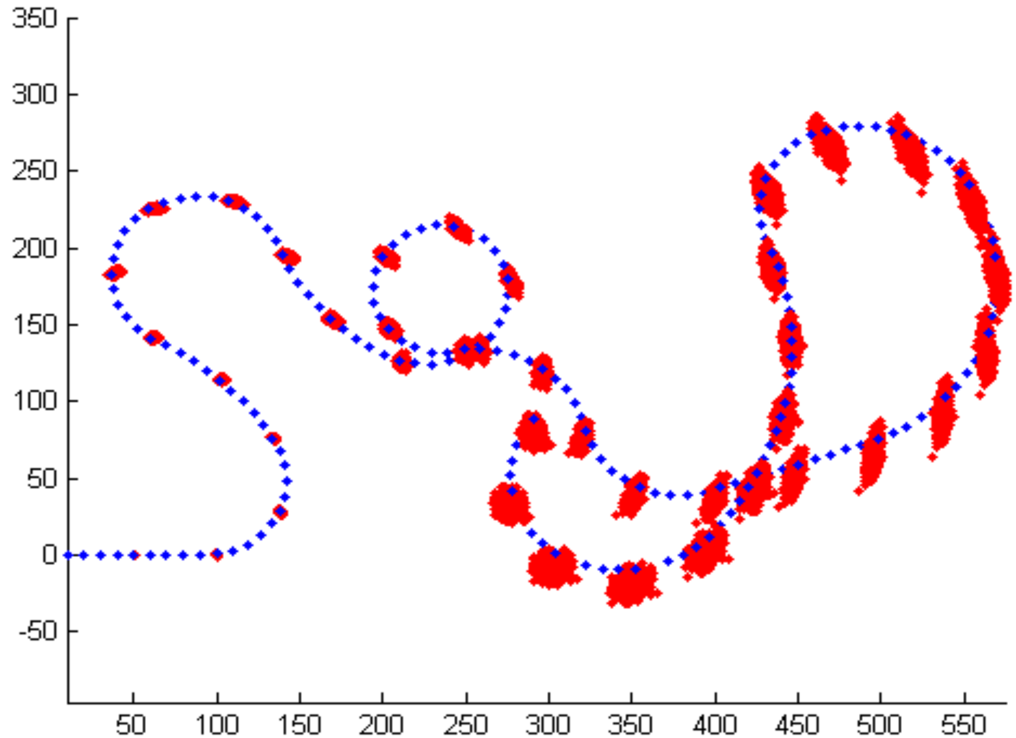
w_{ω_t} is the additive noise for the angular velocity

Continuous motion example

- Dt=1sec
- Plotting 1 sample/sec all the particles every 5 sec
- Constant linear velocity
- Angular velocity changes randomly every 10 sec



Continuous motion example



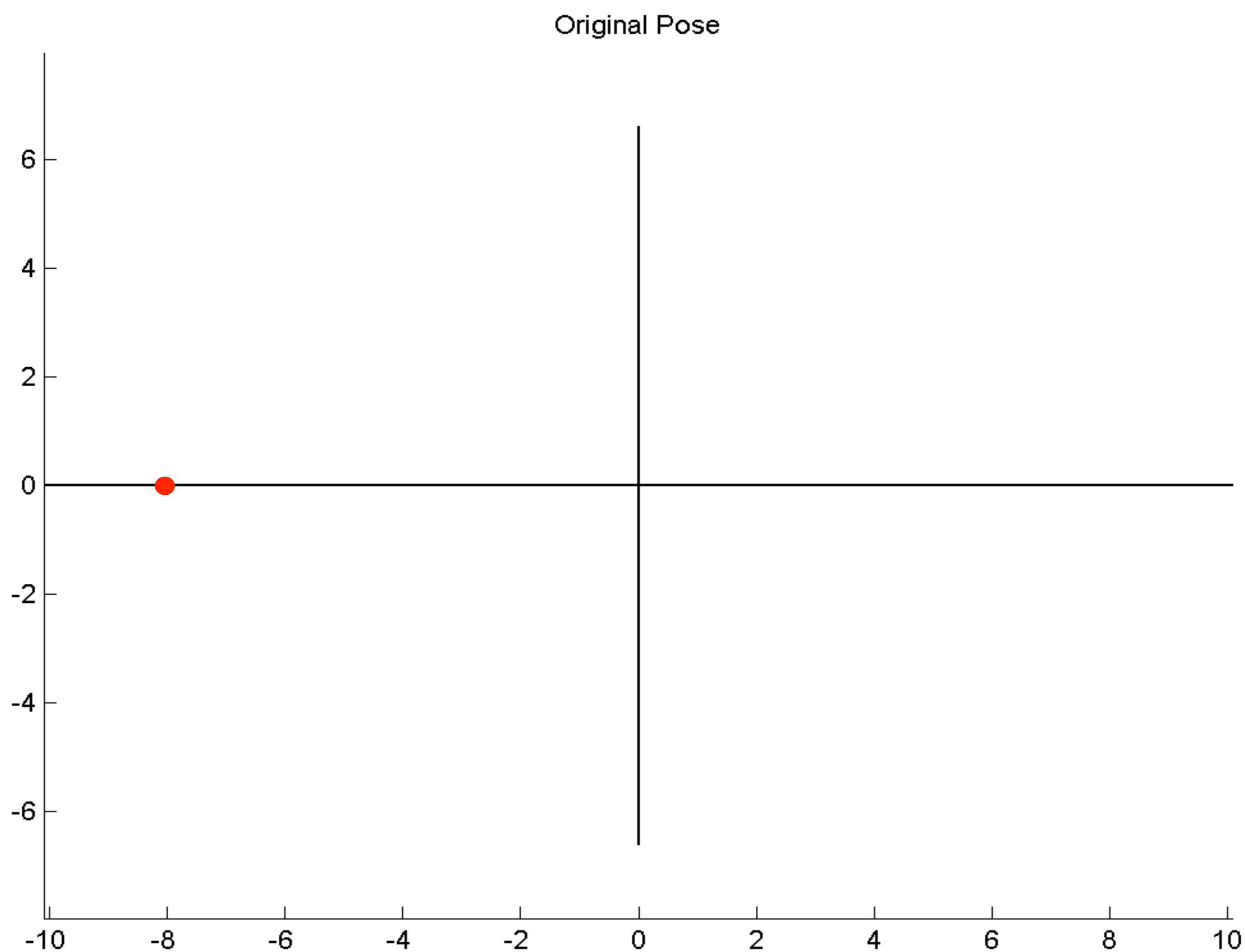
Prediction Examples Using a PF

Piecewise linear motion

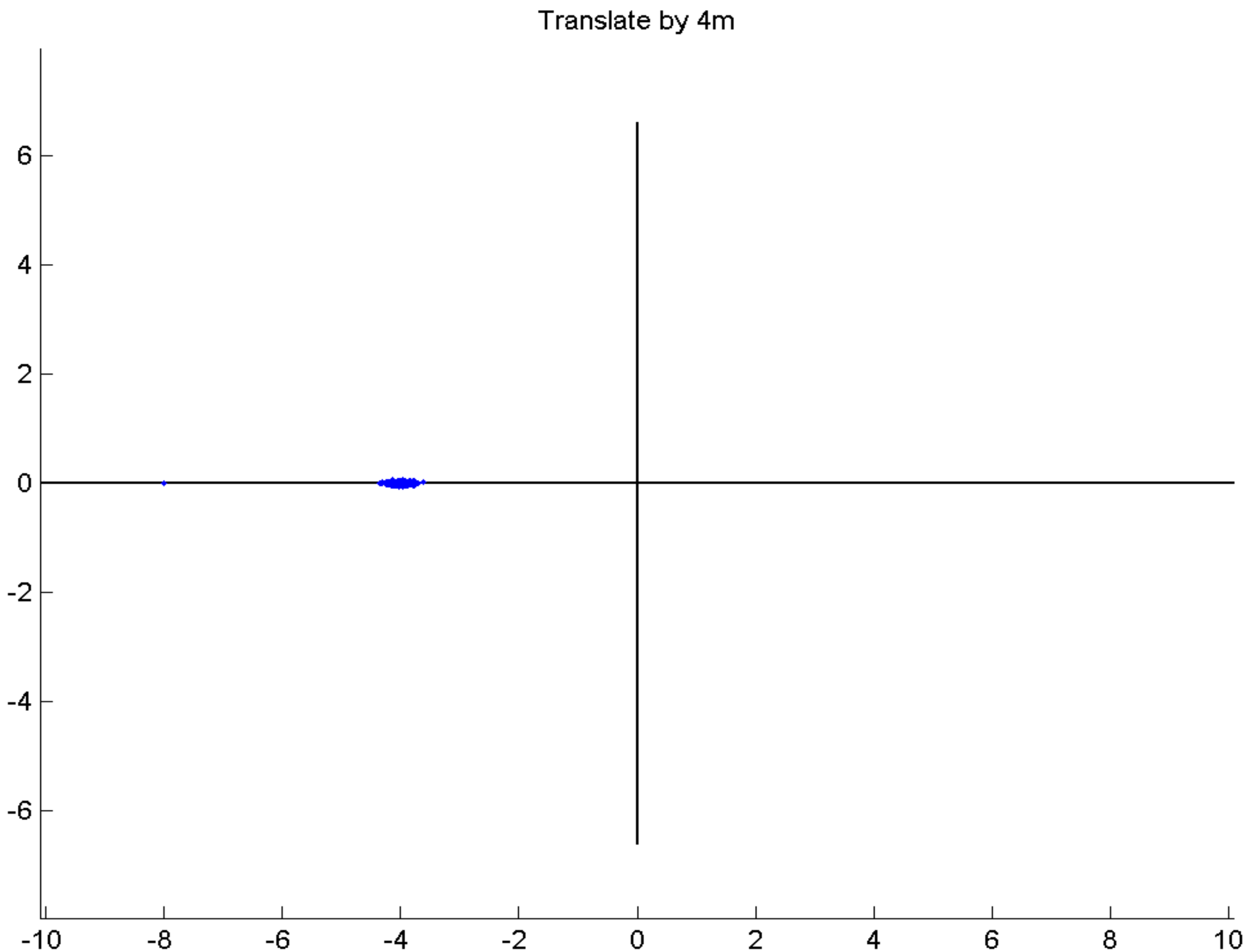
(Translation and Rotation)

- Command success 70%
- Start at [-8,0,0]
- Translate by 4m
- Rotate by 30^{o}
- Translate by 6m

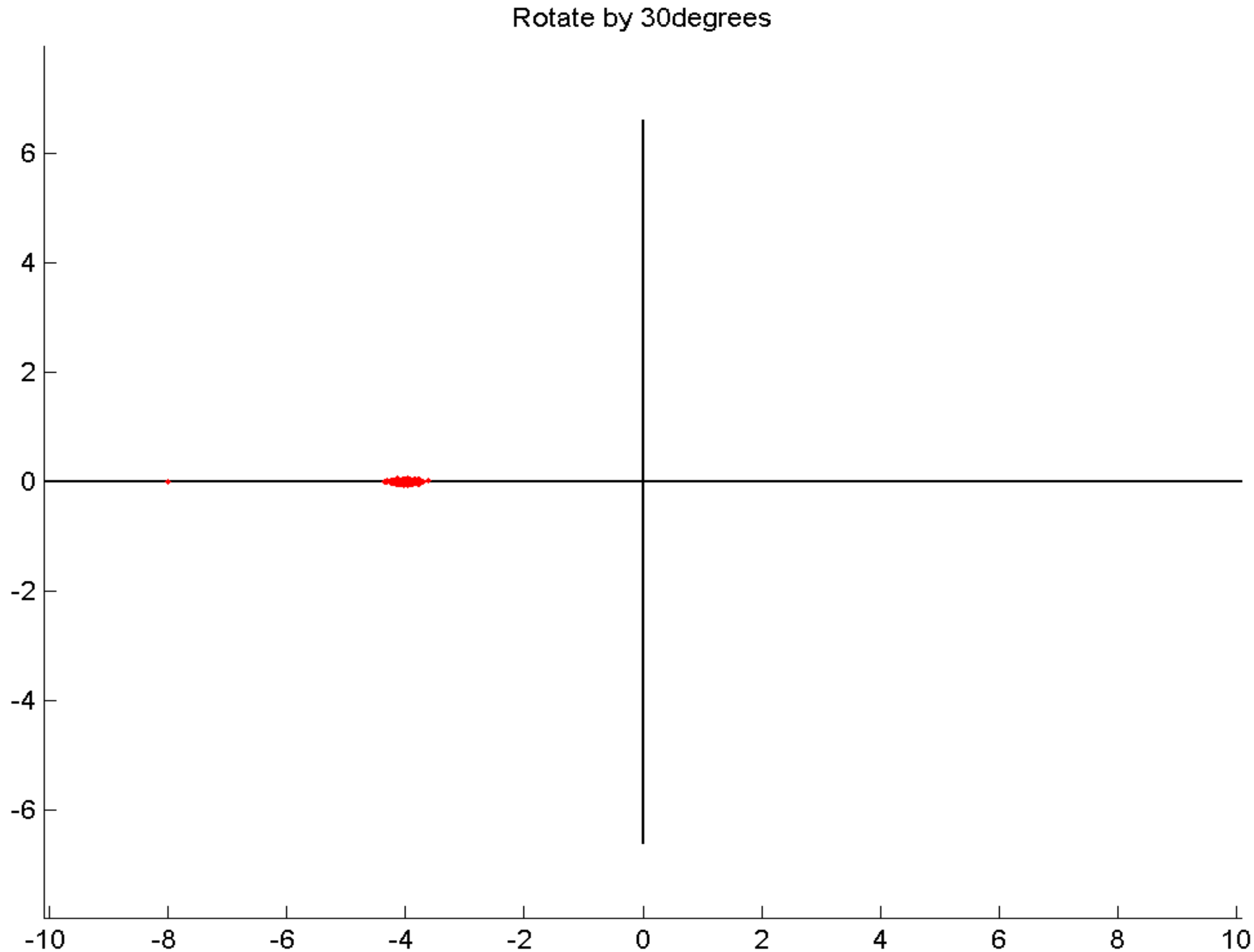
Start [-8,0,0º]



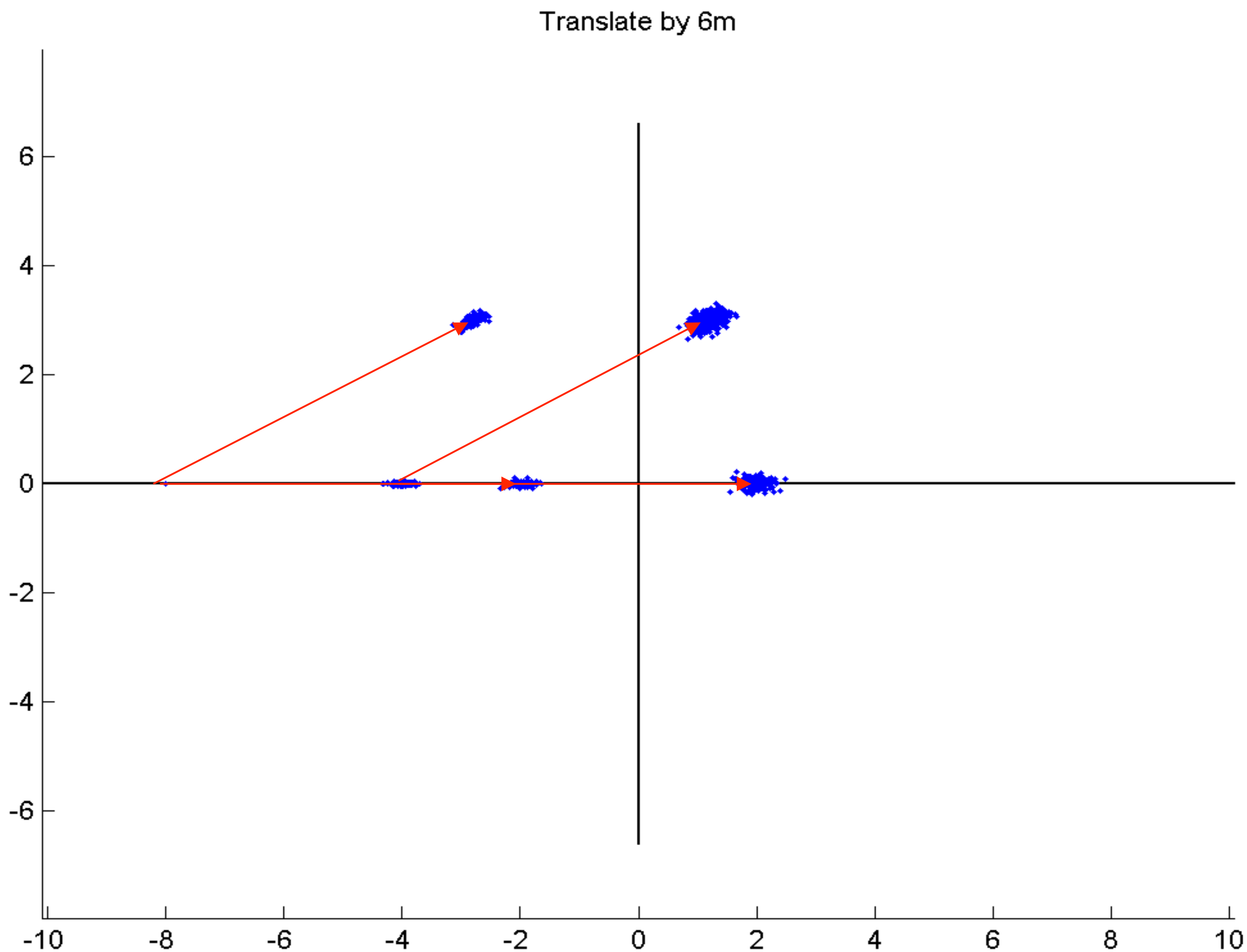
Translate by 4m



Rotate by 30°



Translate by 6m

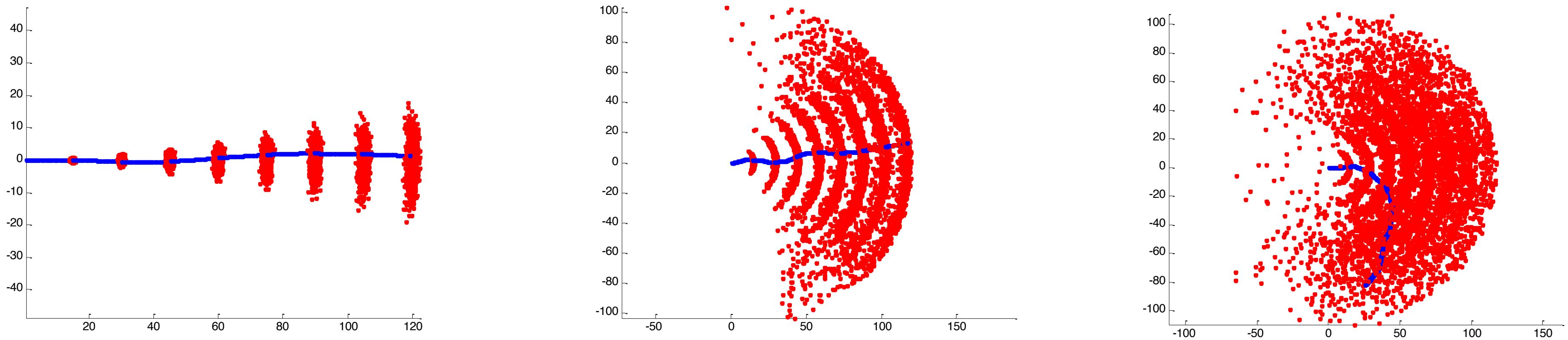


Propagation

- Known position, known orientation
- Bounded linear velocity [0.5 0.7] m/sec
- Bounded angular velocity
- Run 200 sec.
- Plotting every twenty fifth sec.

Bounded Velocities

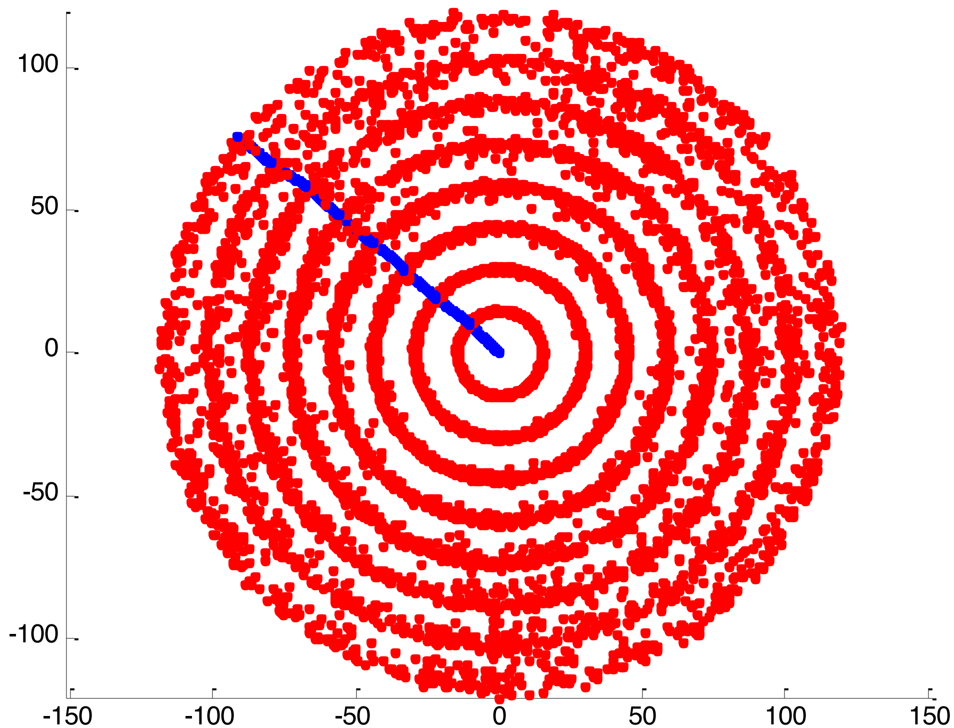
$\omega \in [-0.01 \quad 0.01]\, rad/\sec$ $\omega \in [-0.1 \quad 0.1]\, rad/\sec$ $\omega \in [-0.2 \quad 0.2]\, rad/\sec$



Propagation

- Known position, unknown orientation
- Bounded linear velocity [0.5 0.7] m/sec
- Bounded angular velocity [-0.1 0.1] rad/sec
- Run 200 sec.
- Plotting every twenty fifth sec.

Propagation

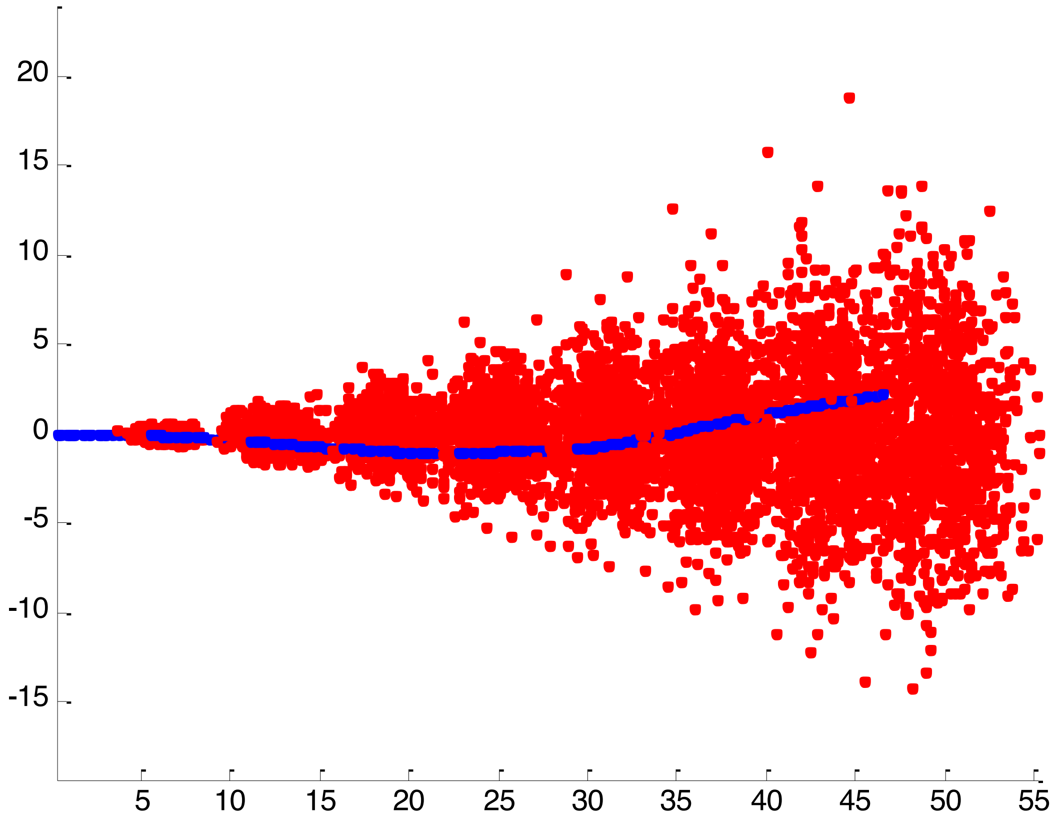


Propagation

- Known position, known orientation
- Bounded linear velocity [0.0 0.5] m/sec
- Bounded angular velocity [-0.01 0.01] rad/sec
- Run 200 sec.
- Plotting every twenty fifth sec.

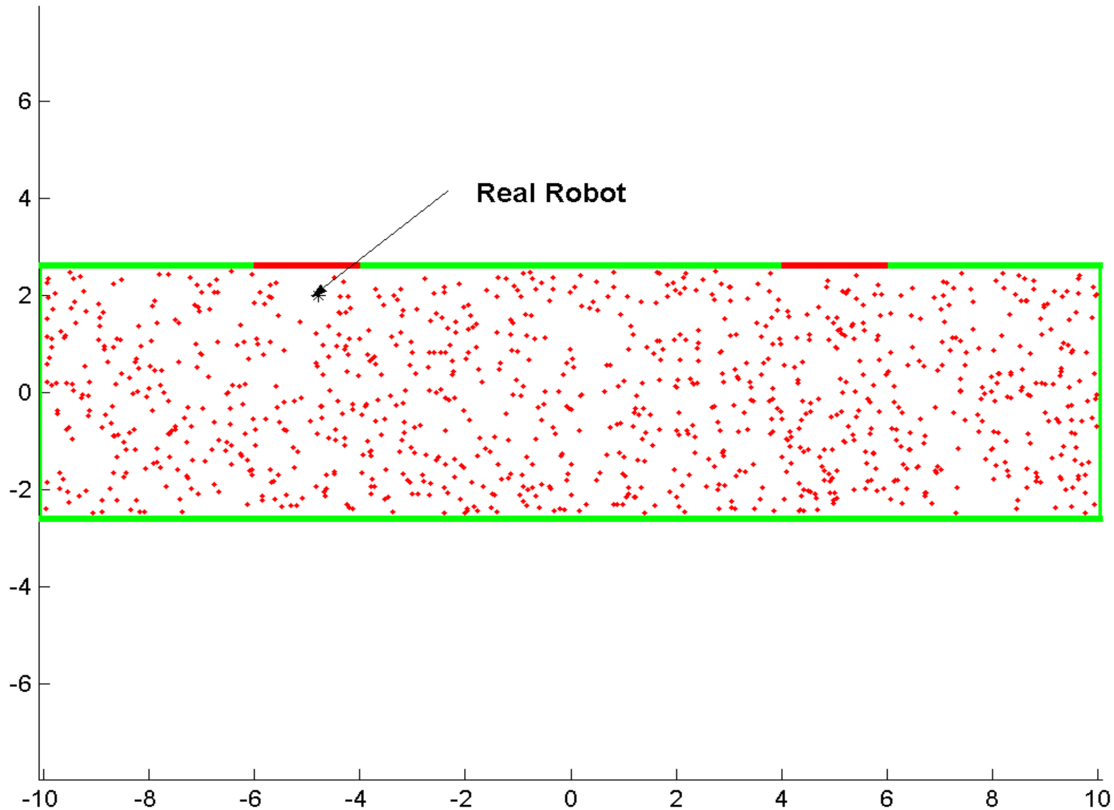
- For a particle to stay at the origin, it has to draw zero velocity 25 times in the row.

Bounded velocities

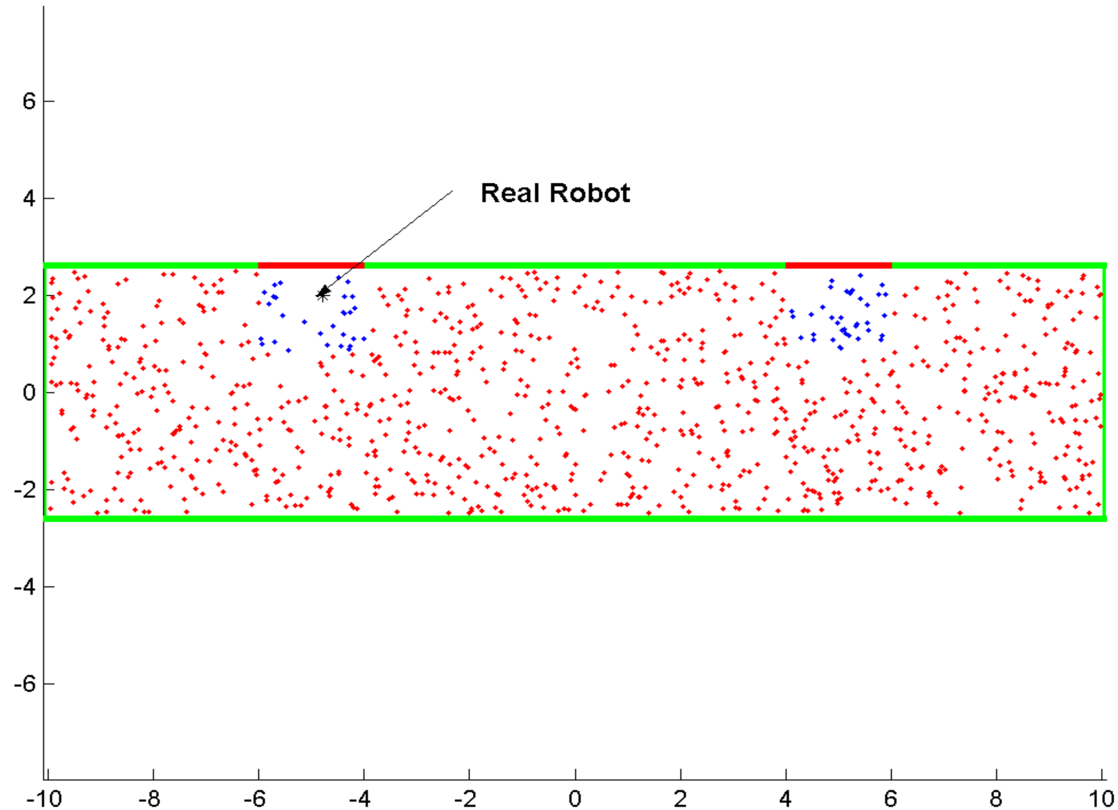


Update Examples Using a PF

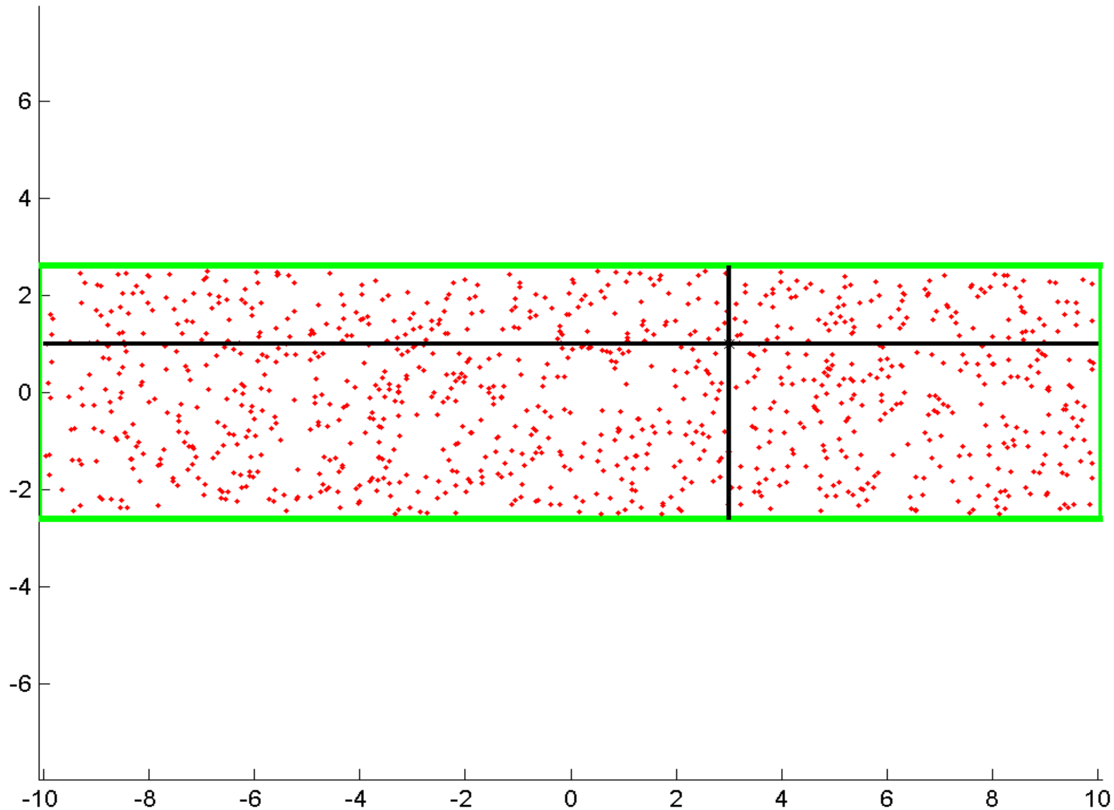
Environment with two red doors (uniform distribution)



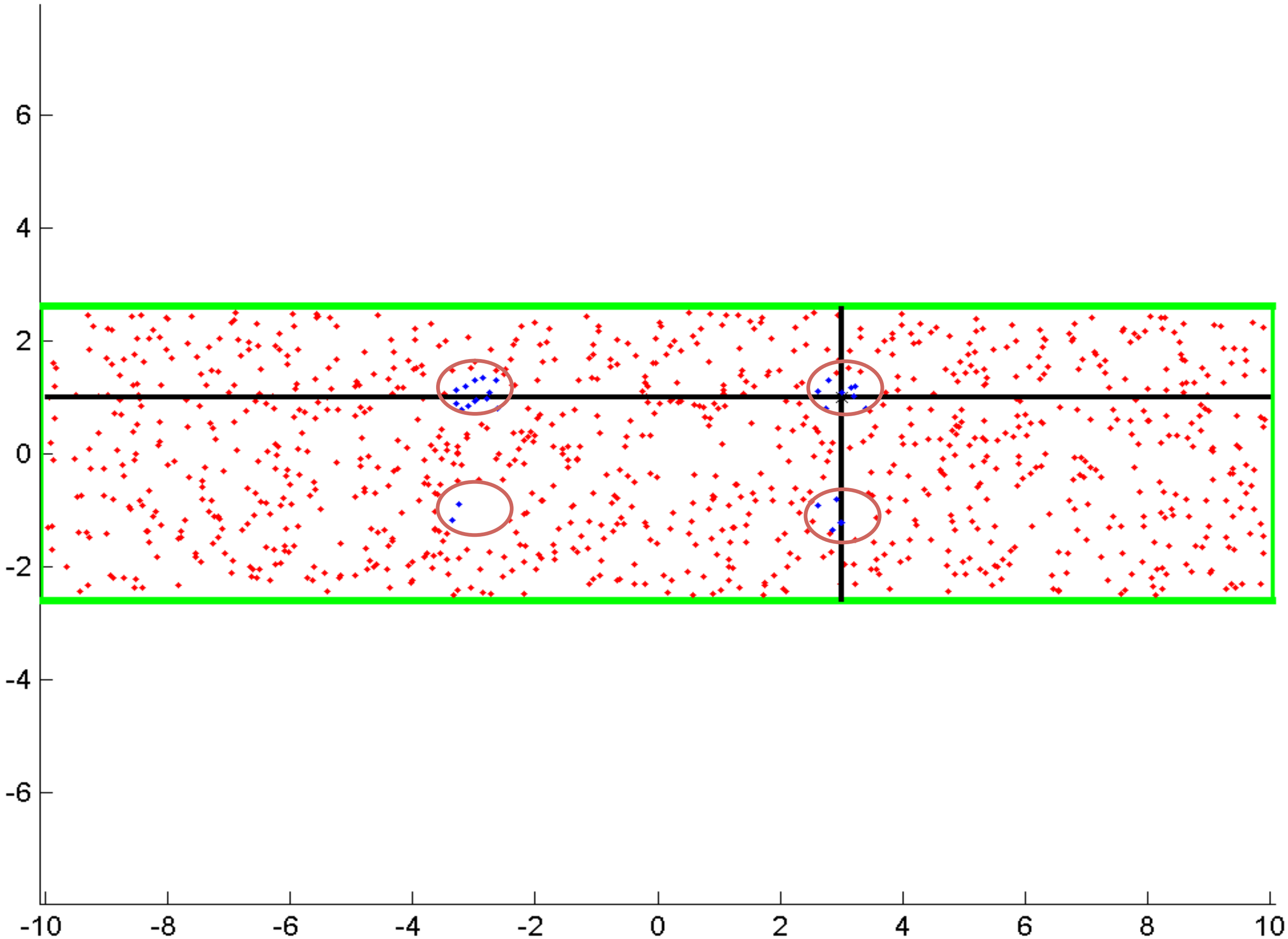
Environment with two red doors
(Sensing the red door)



Sensing four walls

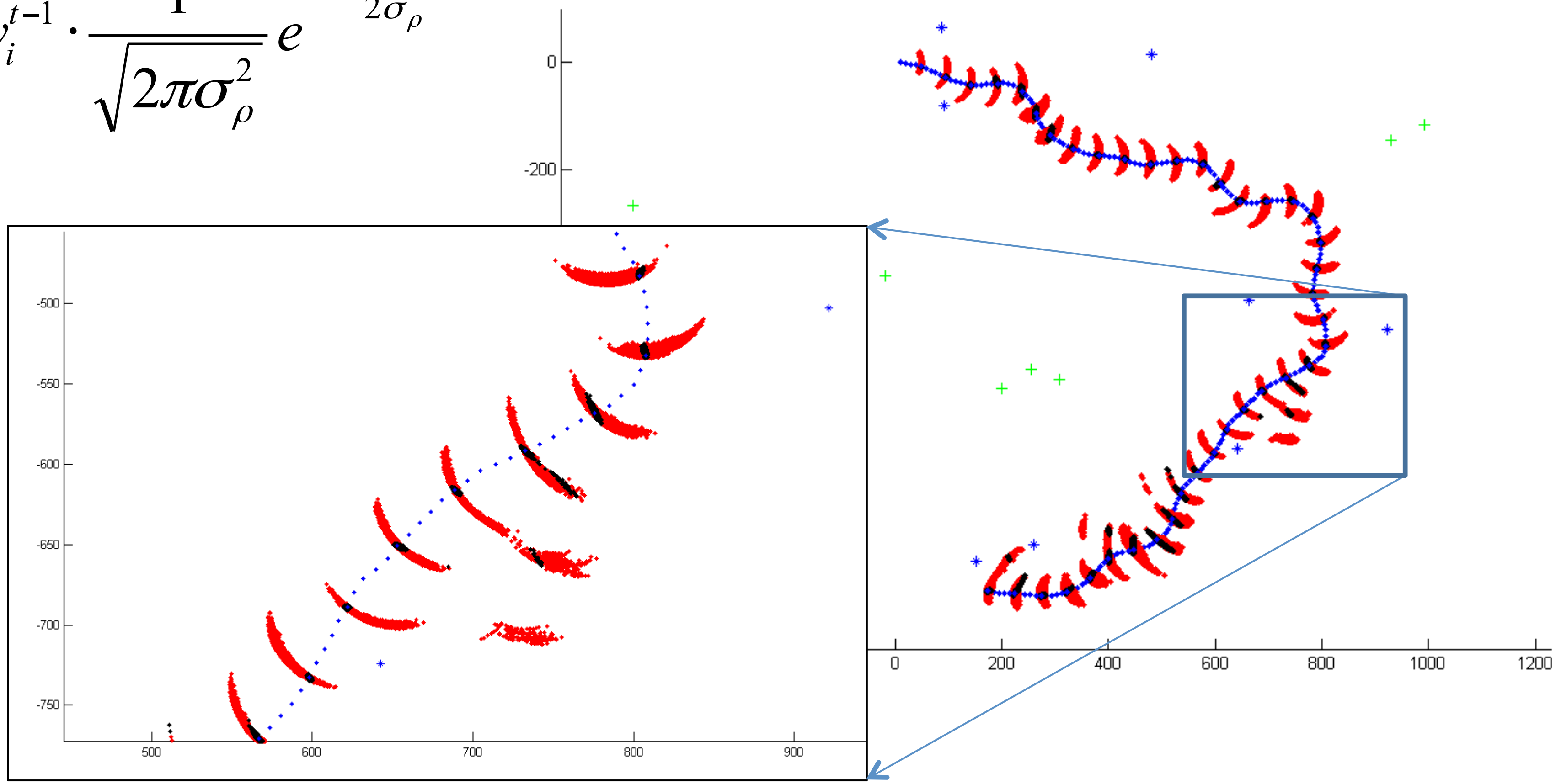


Four possible areas

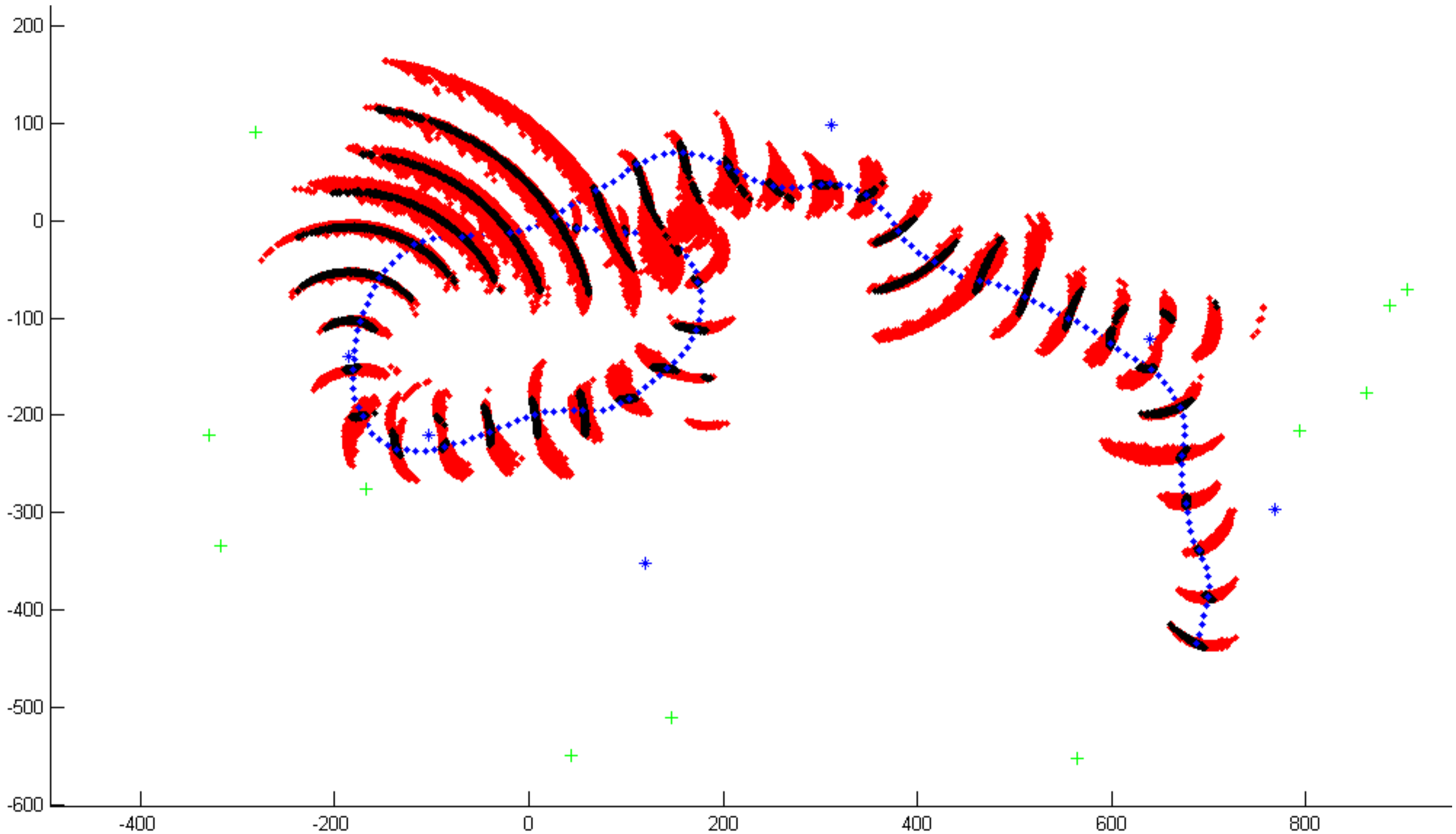


Update Range only

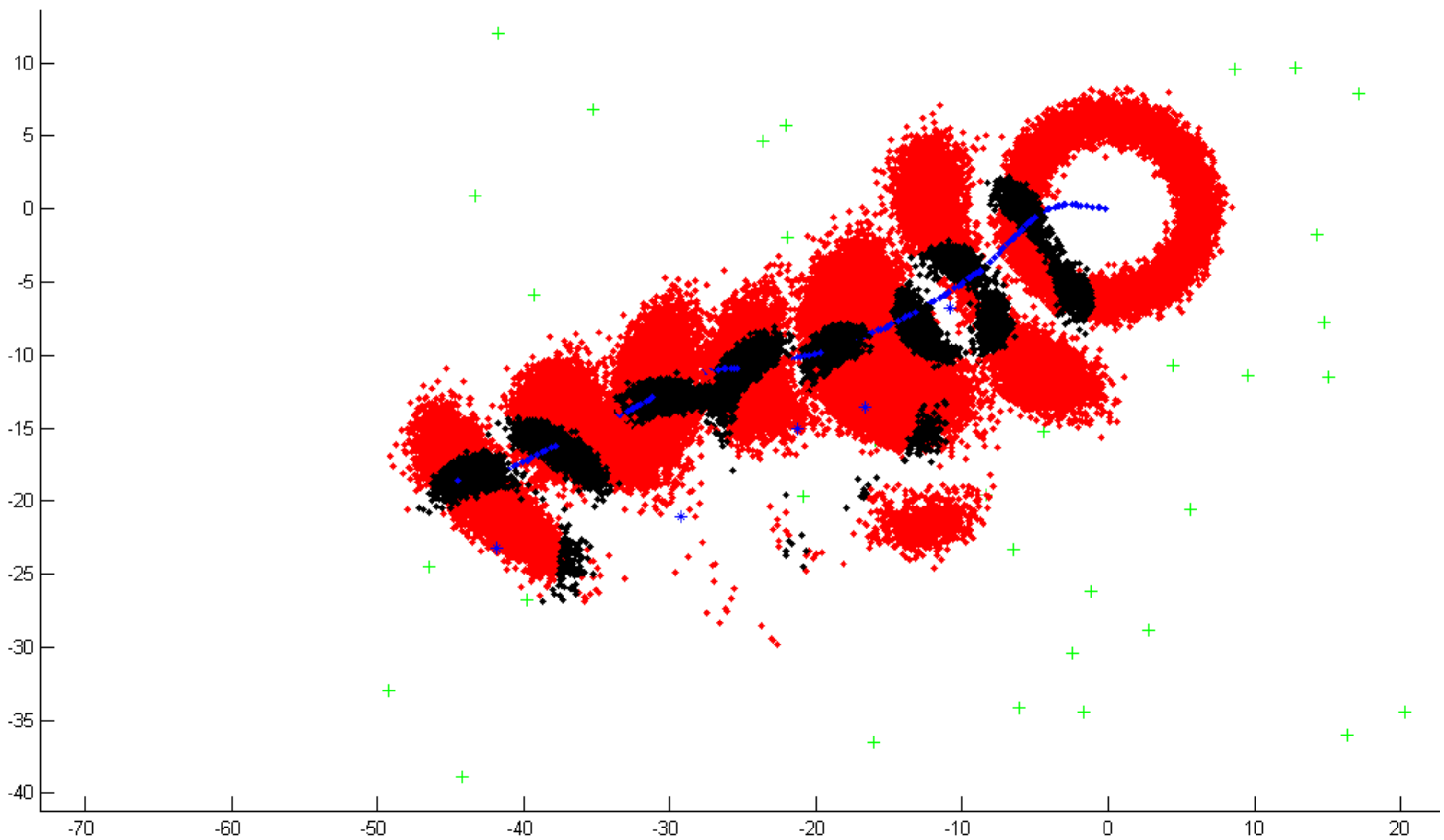
$$w_i^t = w_i^{t-1} \cdot \frac{1}{\sqrt{2\pi\sigma_\rho^2}} e^{-\frac{(\rho_i - \rho_r)^2}{2\sigma_\rho}}$$



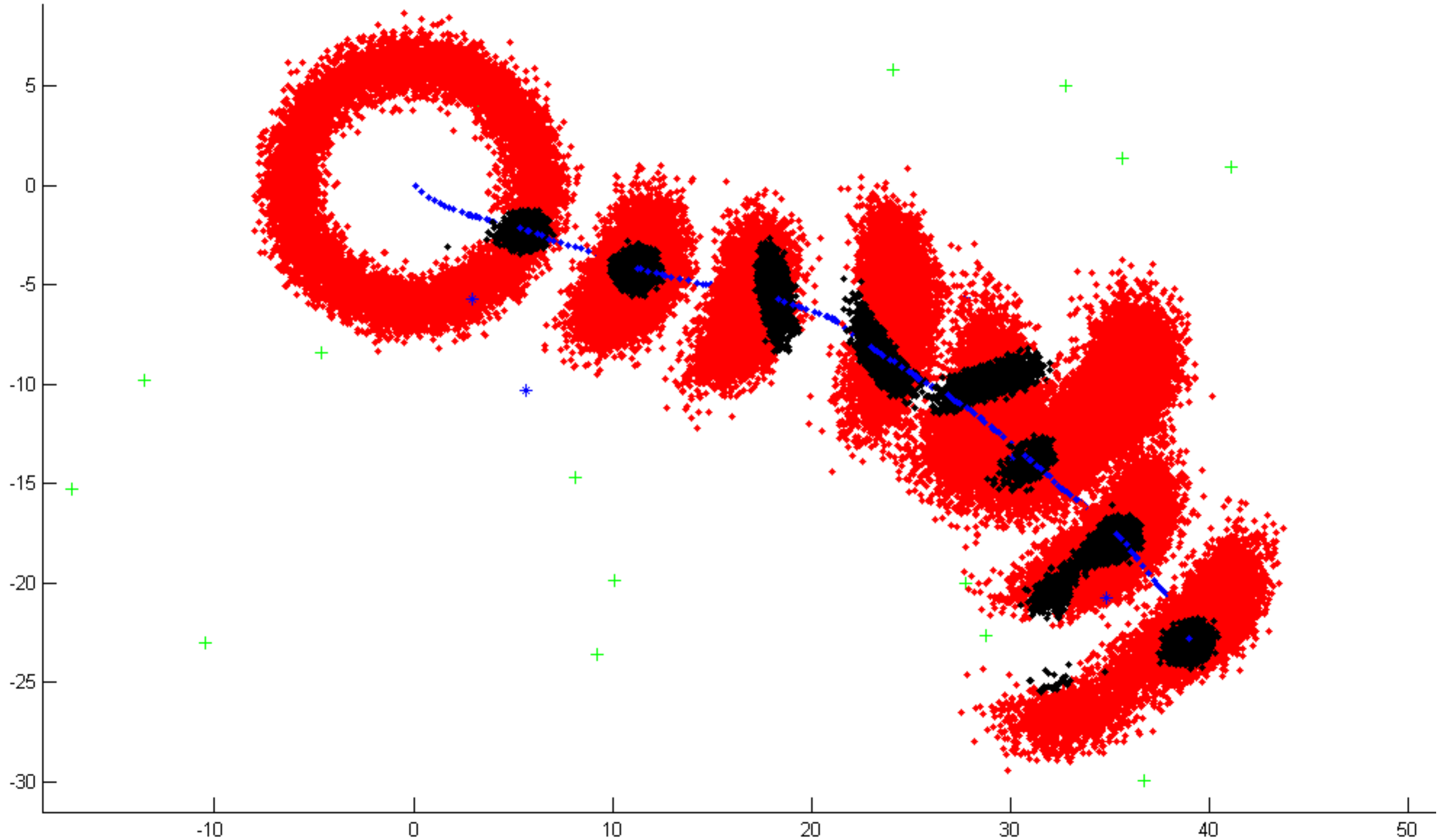
Update Range only



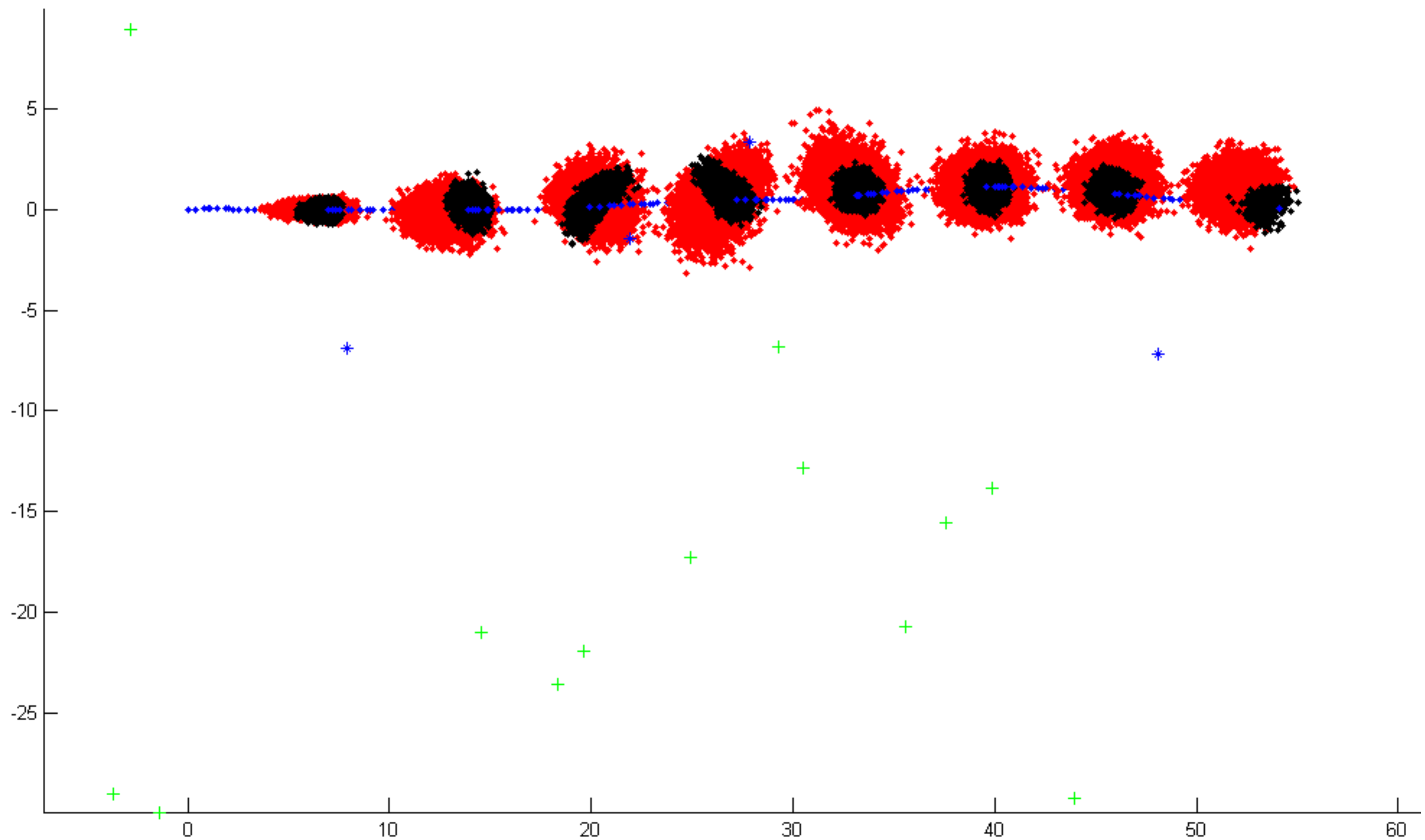
Update Range only



Update Range only

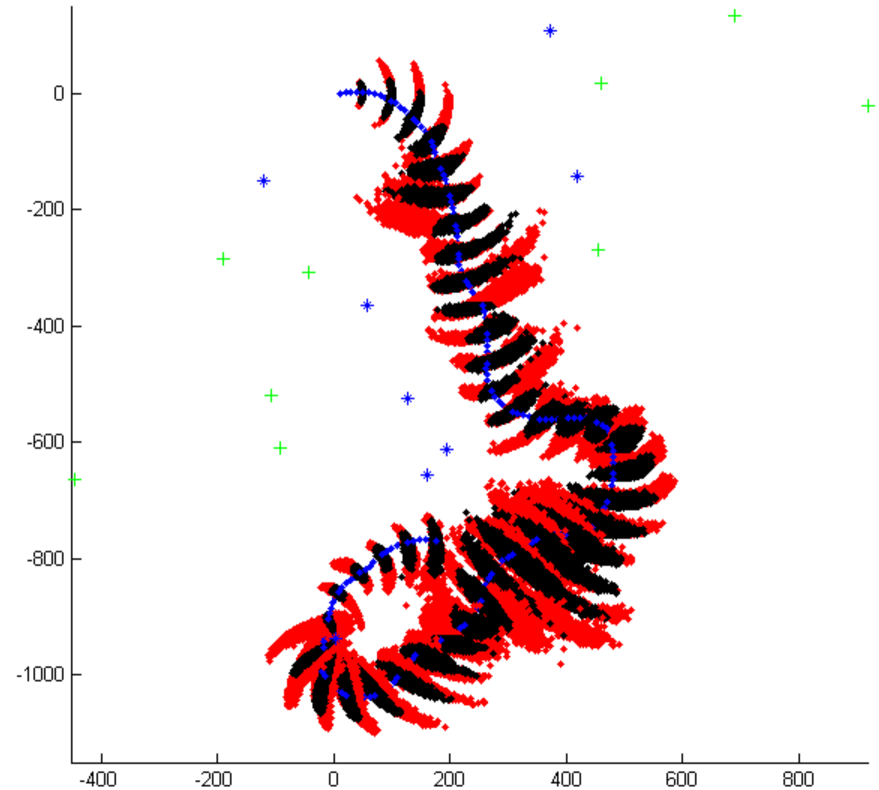


Update Range only

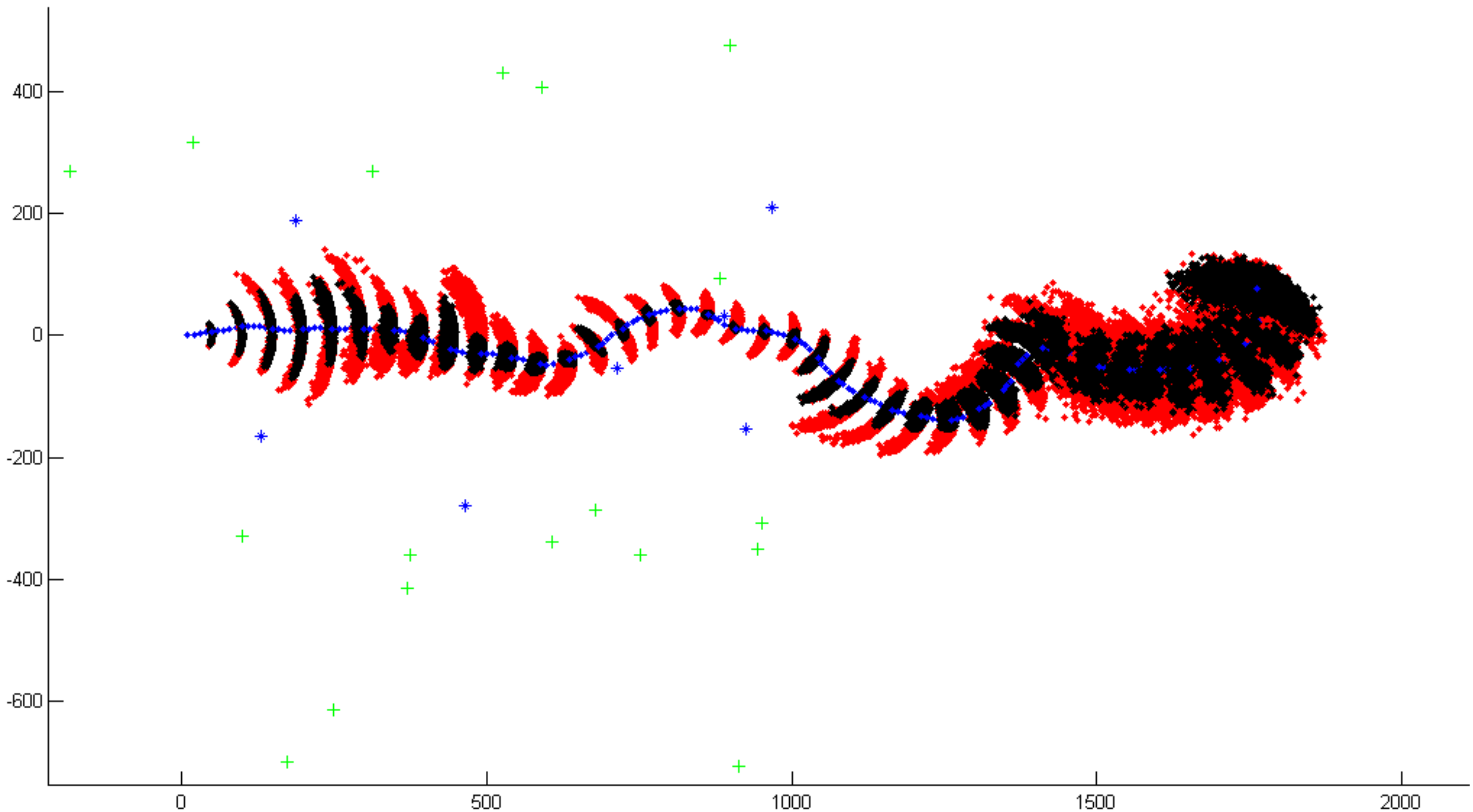


Update Bearing only

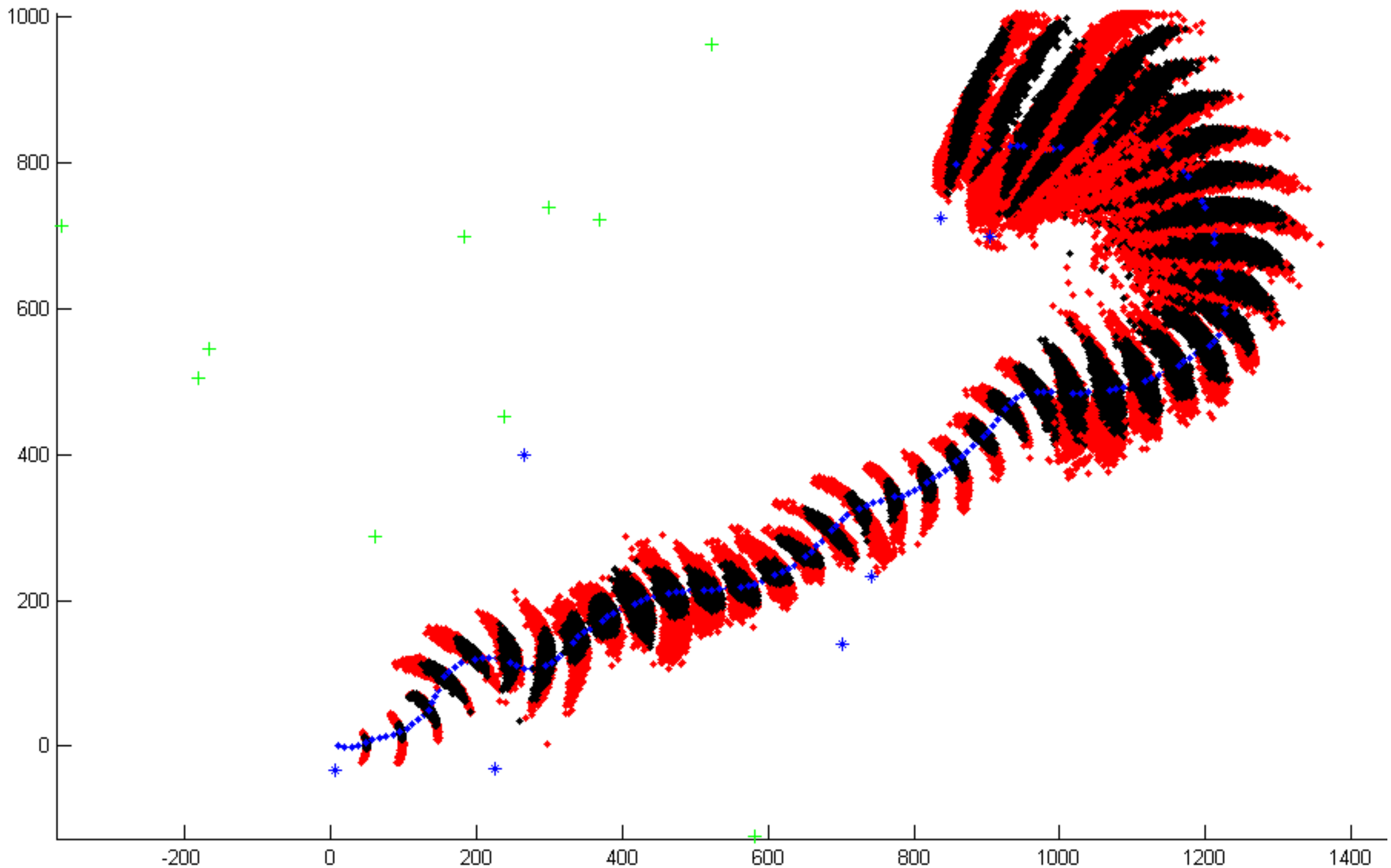
$$w_i^t = w_i^{t-1} \cdot \frac{1}{\sqrt{2\pi\sigma_\varphi^2}} e^{-\frac{(\varphi_i - \varphi_r)^2}{2\sigma_\varphi}}$$



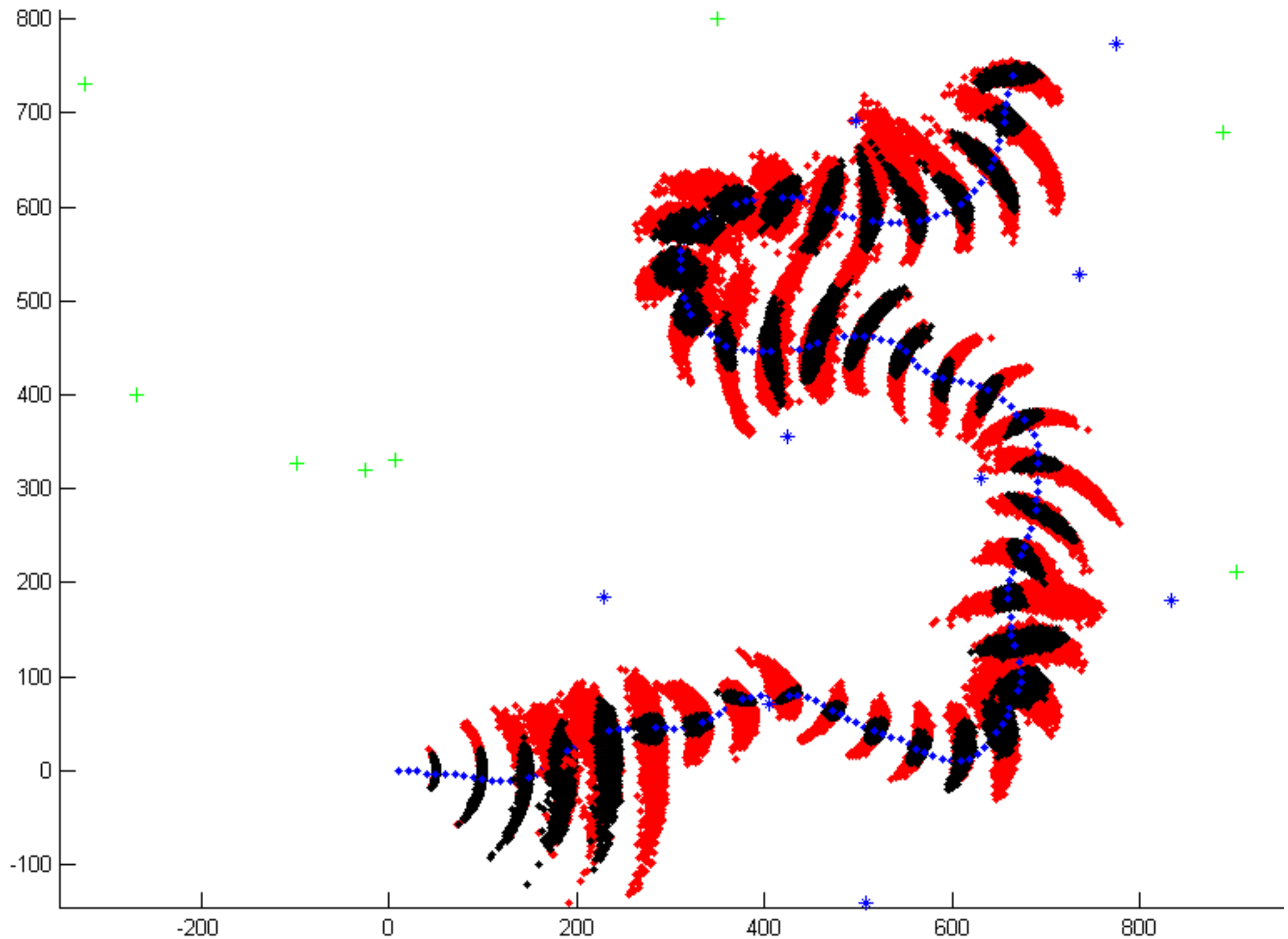
Update Bearing only



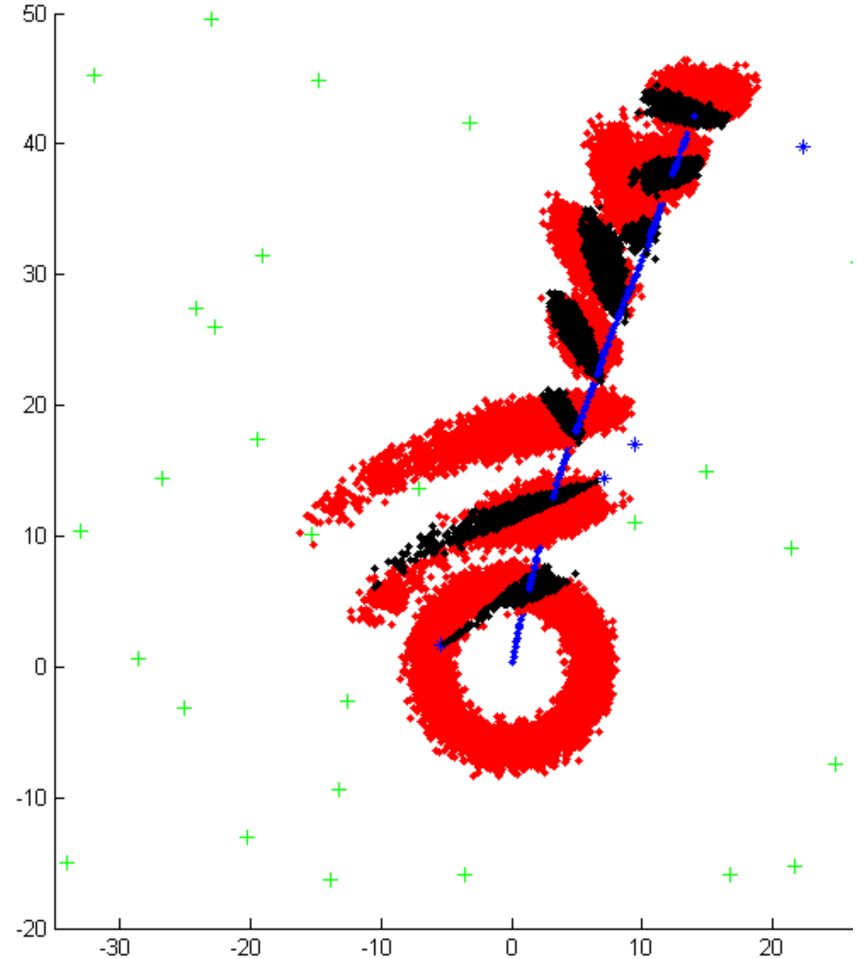
Update Bearing only



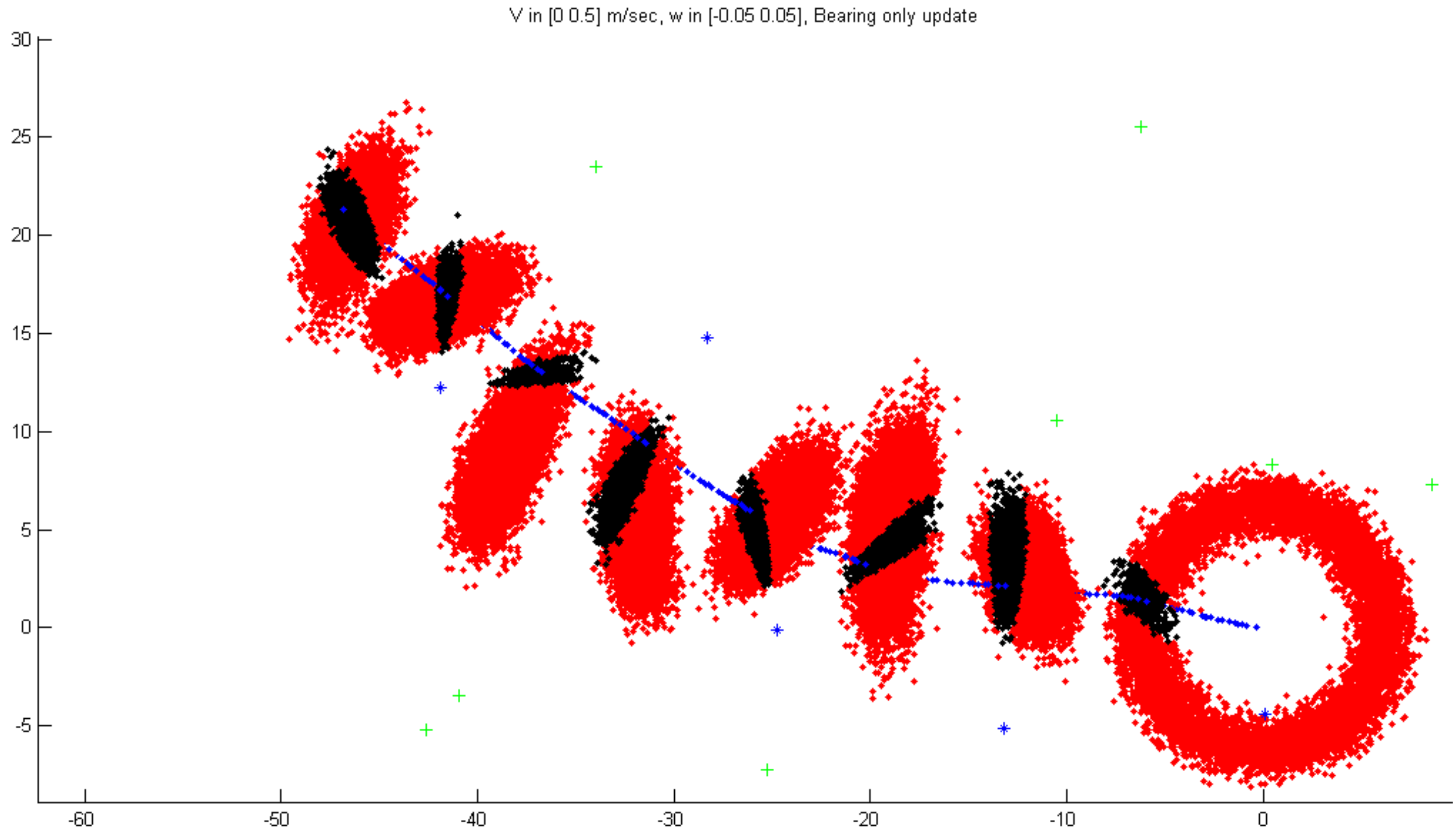
Update Bearing only



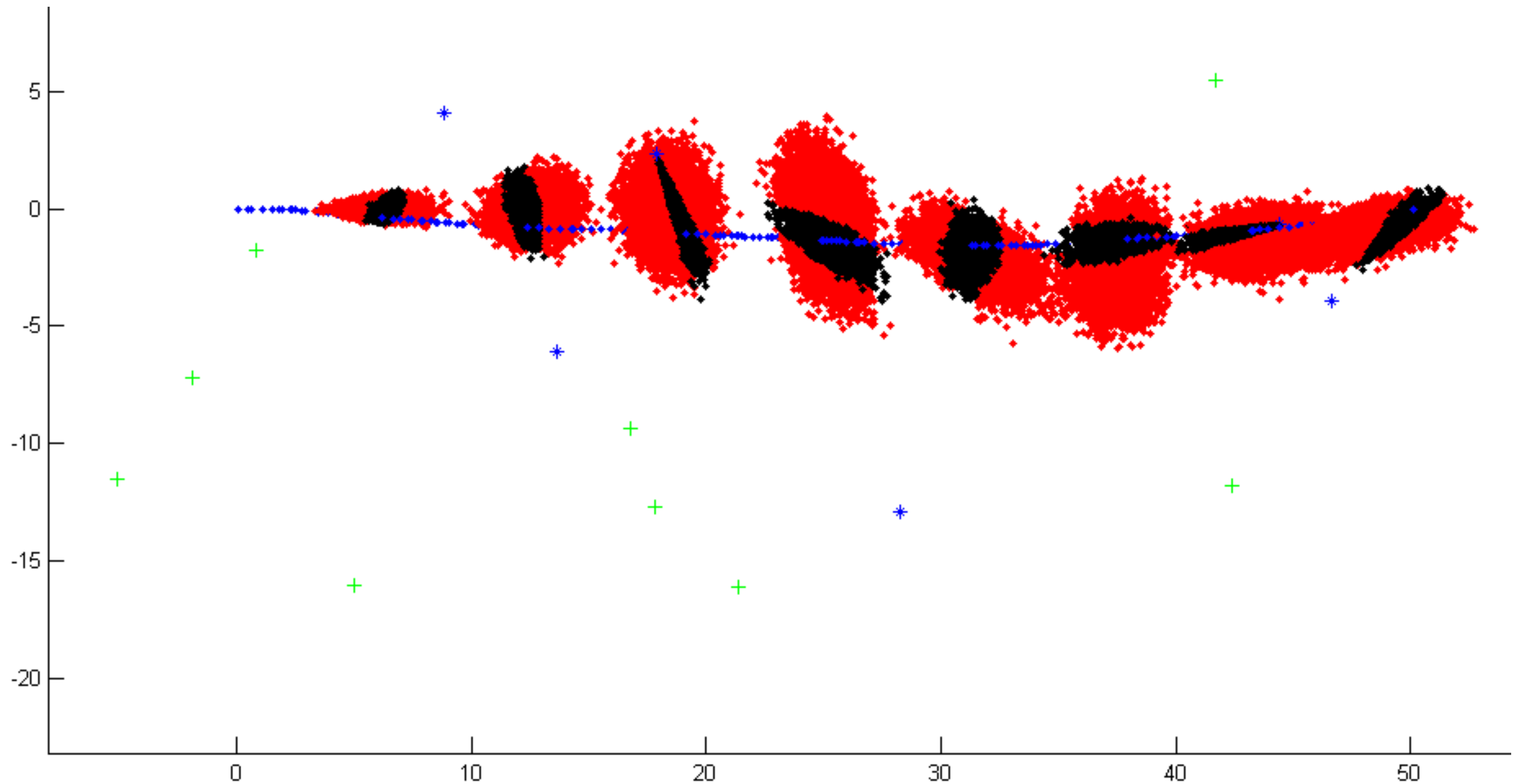
Update Bearing only



Update Bearing only

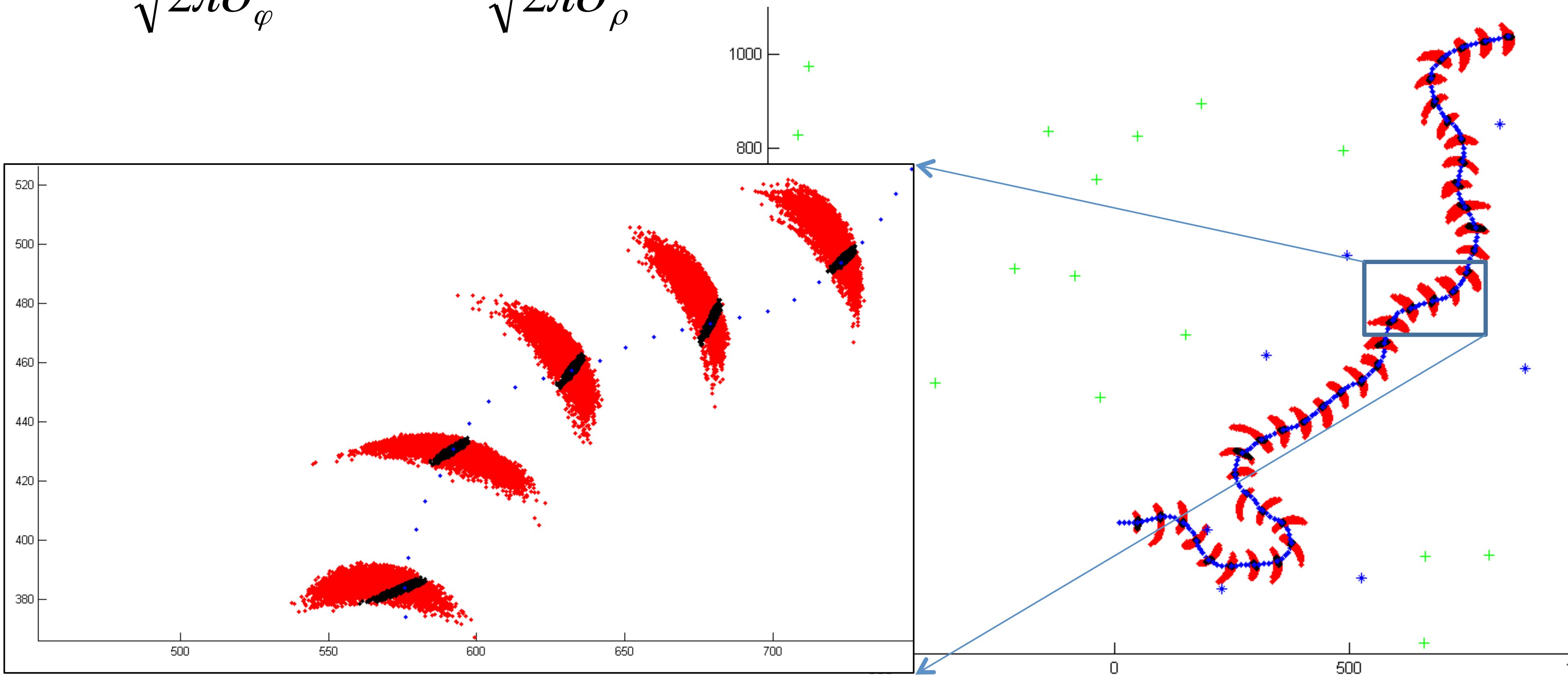


Update Bearing only



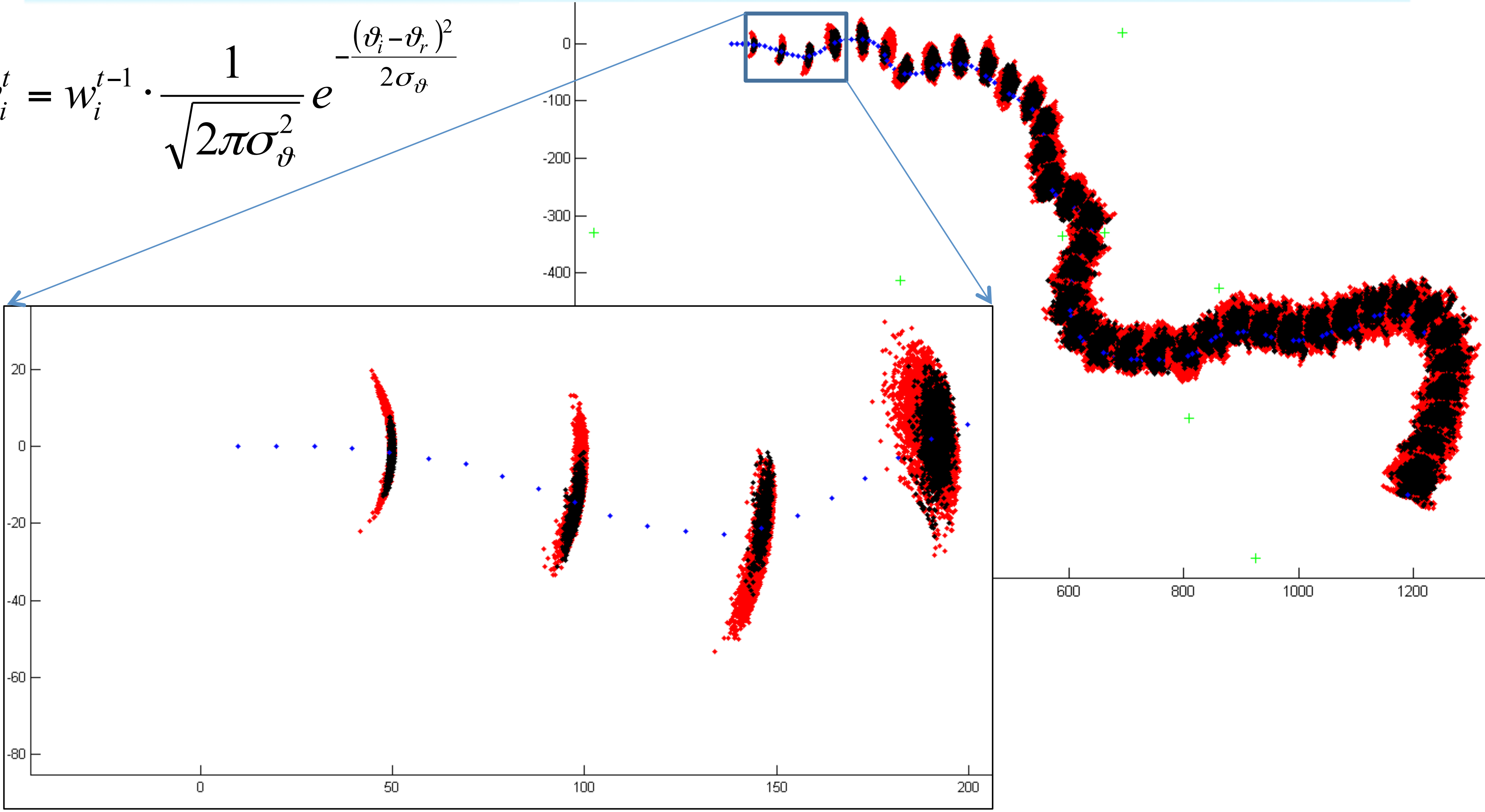
Update Range and Bearing

$$w_i^t = w_i^{t-1} \cdot \frac{1}{\sqrt{2\pi\sigma_\varphi^2}} e^{-\frac{(\varphi_i - \varphi_r)^2}{2\sigma_\varphi}} \cdot \frac{1}{\sqrt{2\pi\sigma_\rho^2}} e^{-\frac{(\rho_i - \rho_r)^2}{2\sigma_\rho}}$$

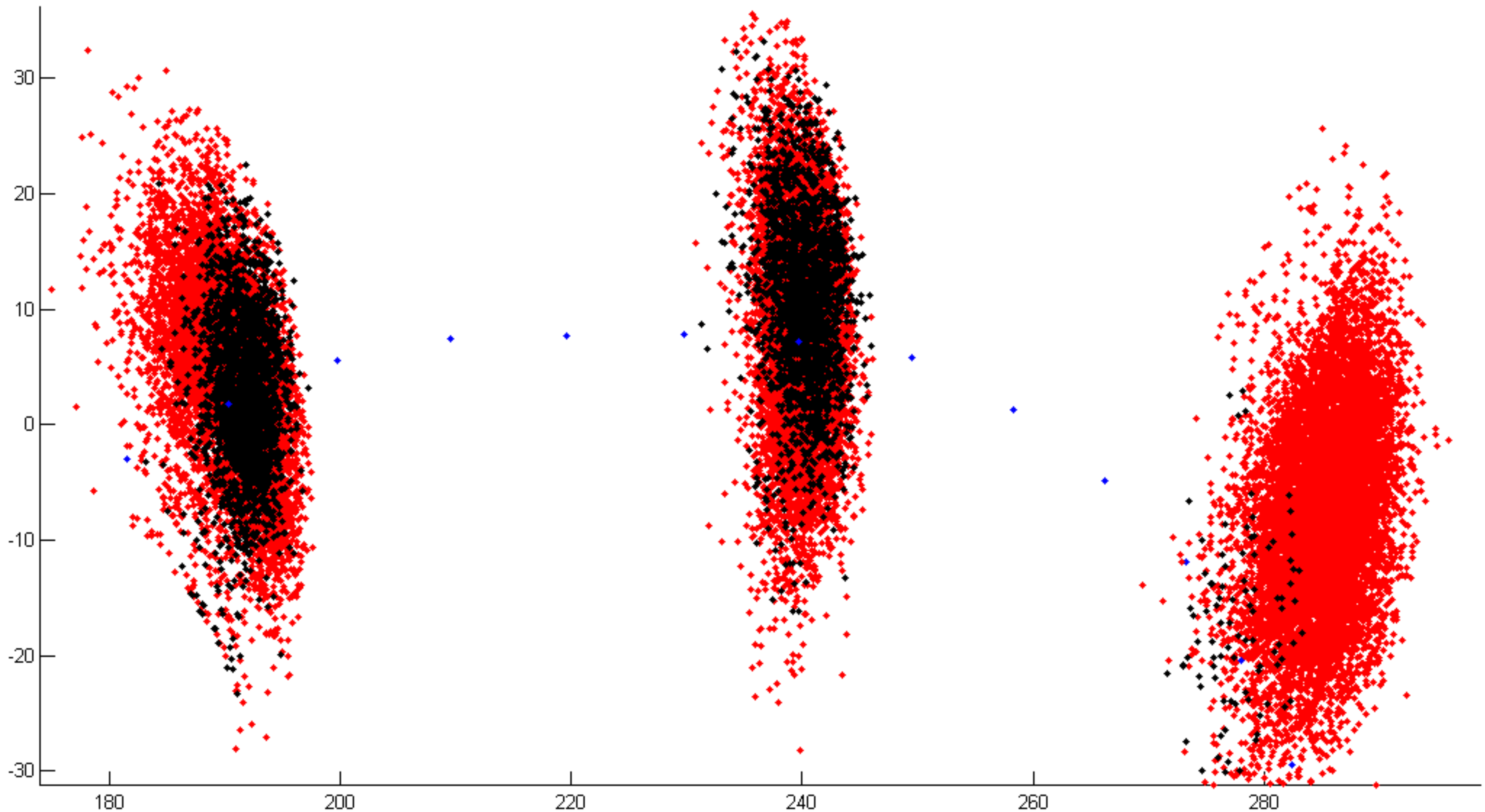


Update Compass only

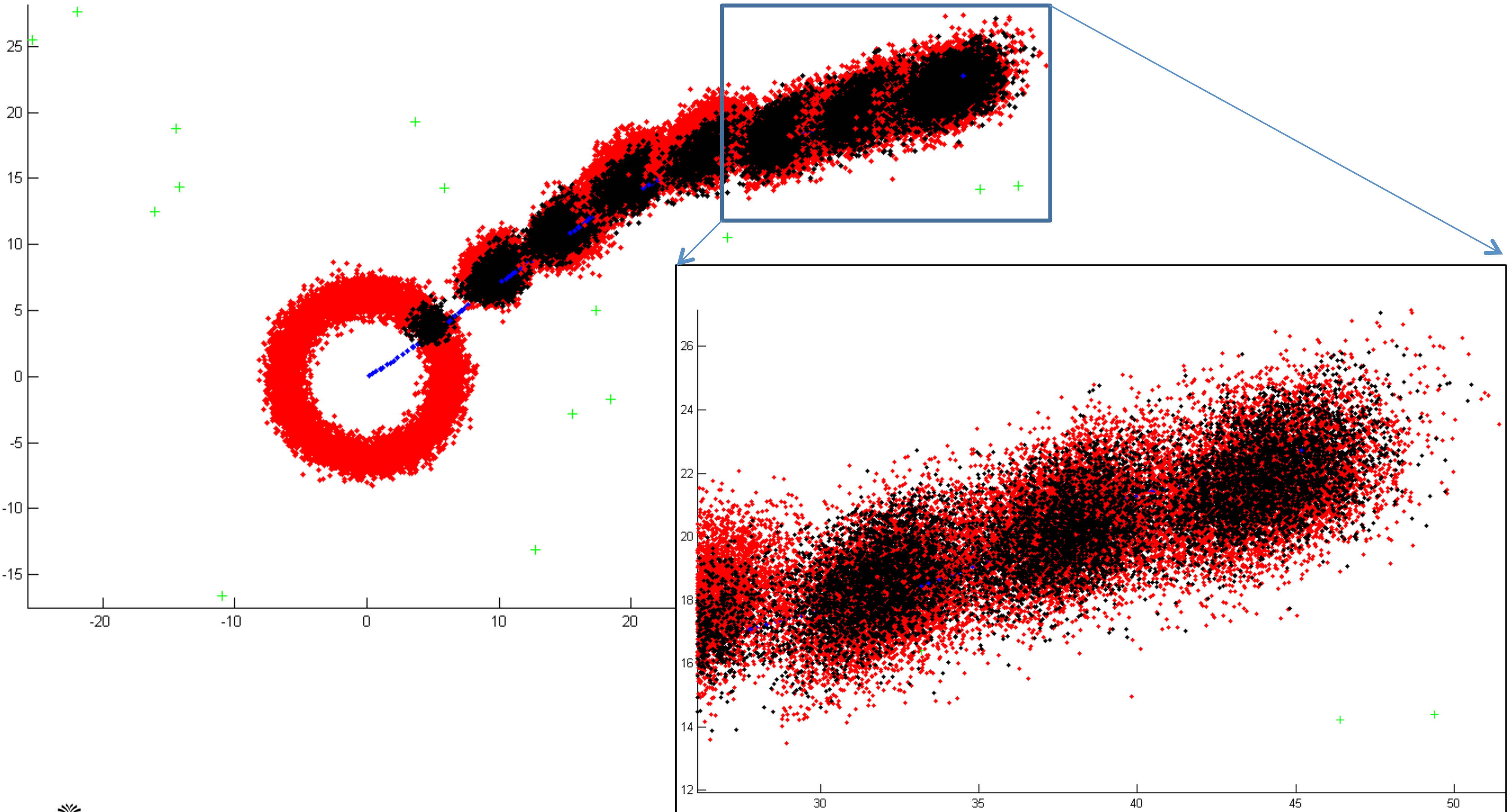
$$w_i^t = w_i^{t-1} \cdot \frac{1}{\sqrt{2\pi\sigma_\vartheta^2}} e^{-\frac{(\vartheta_i - \vartheta_r)^2}{2\sigma_\vartheta}}$$



Update Compass only

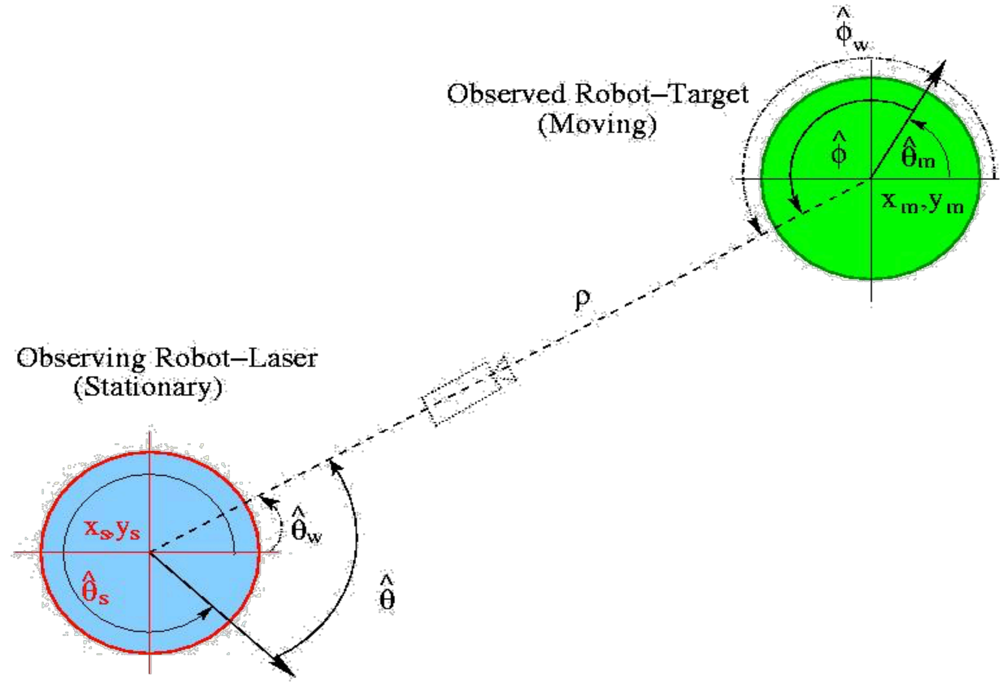


Update Compass only



Cooperative Localization

- Pose of the moving robot is estimated relative to the pose of the stationary robot. Stationary Robot observes the Moving Robot.

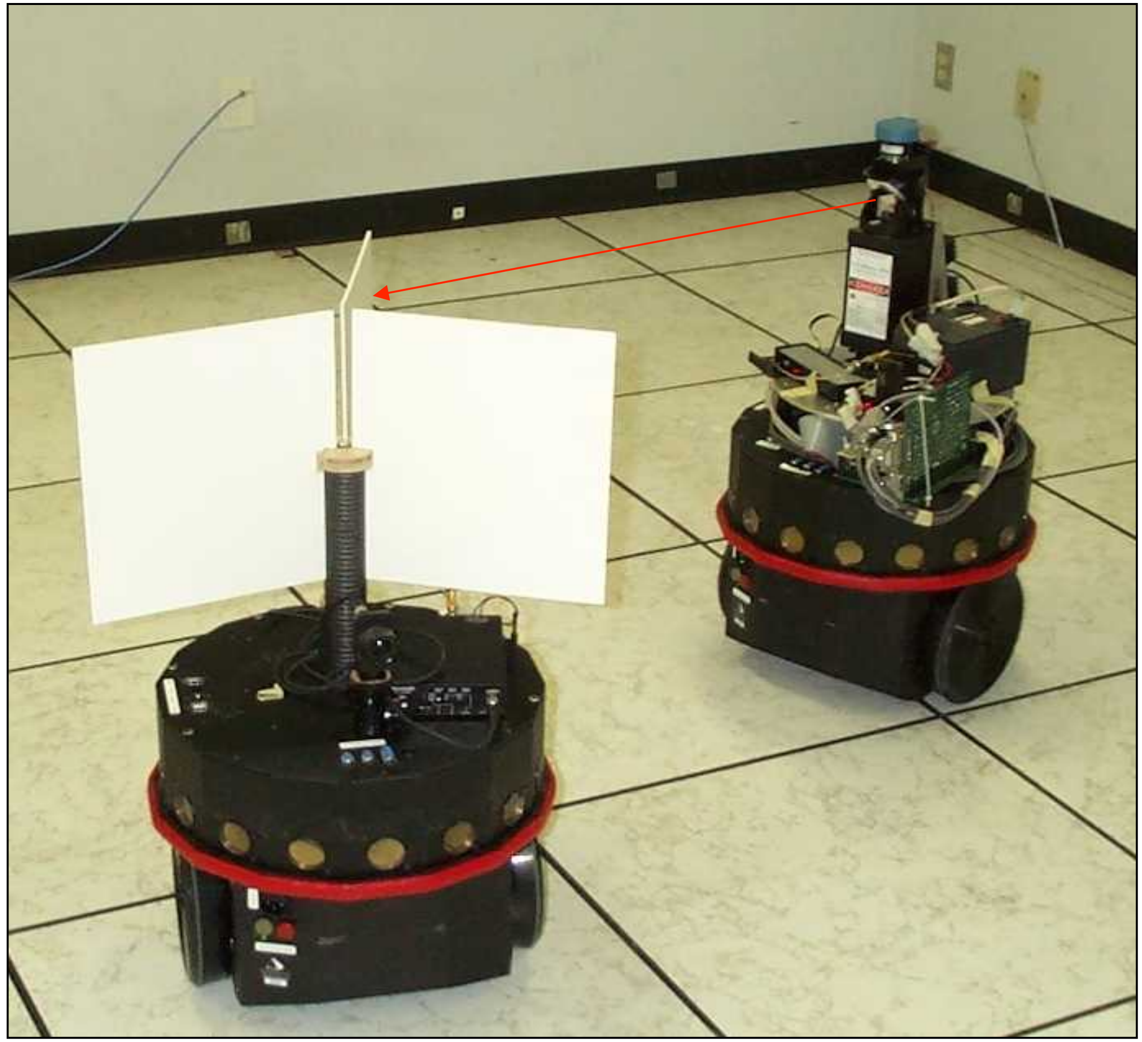


Robot Tracker Returns:

<ρ,θ,φ>

$$\mathbf{x}_{m_{est}}(k+1) = \begin{pmatrix} x_{m_{est}} \\ y_{m_{est}} \\ \theta_{m_{est}} \end{pmatrix} = \begin{pmatrix} x_s + \rho\cos(\theta + \theta_s) \\ y_s + \rho\sin(\theta + \theta_s) \\ \pi - (\phi - (\theta + \theta_s)) \end{pmatrix}$$

Laser-Based Robot Tracker

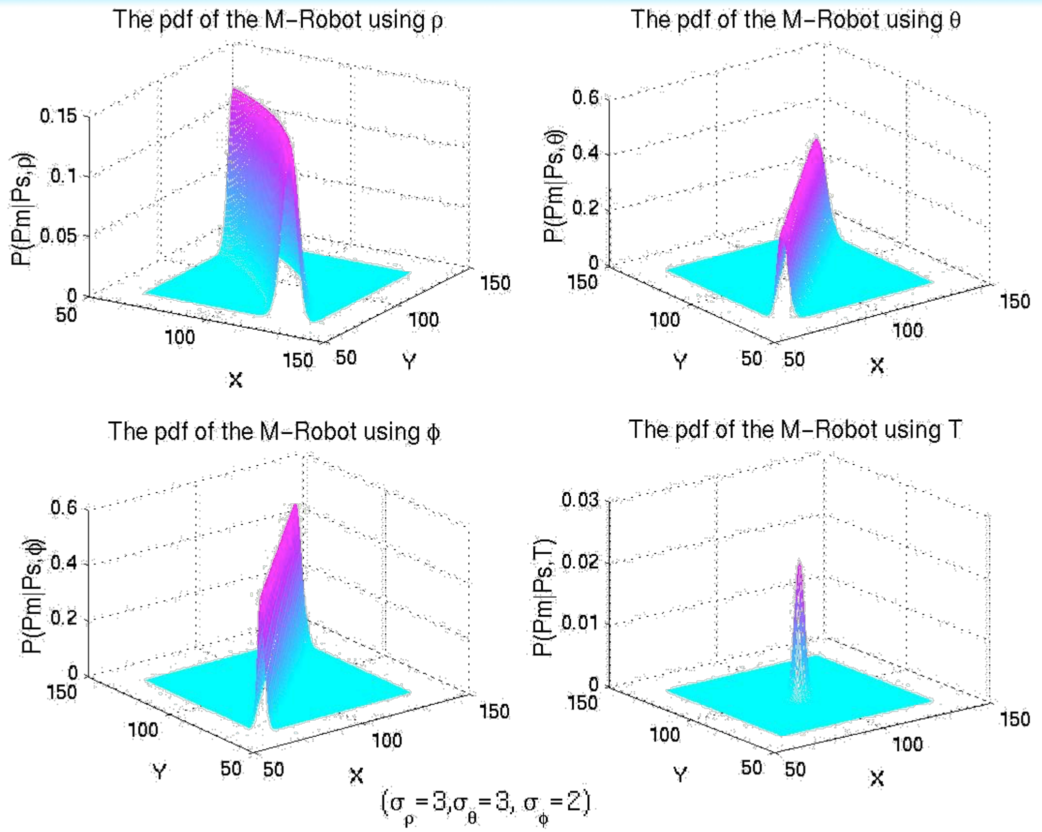


Robot Tracker Returns:

$<\rho,\theta,\phi>$

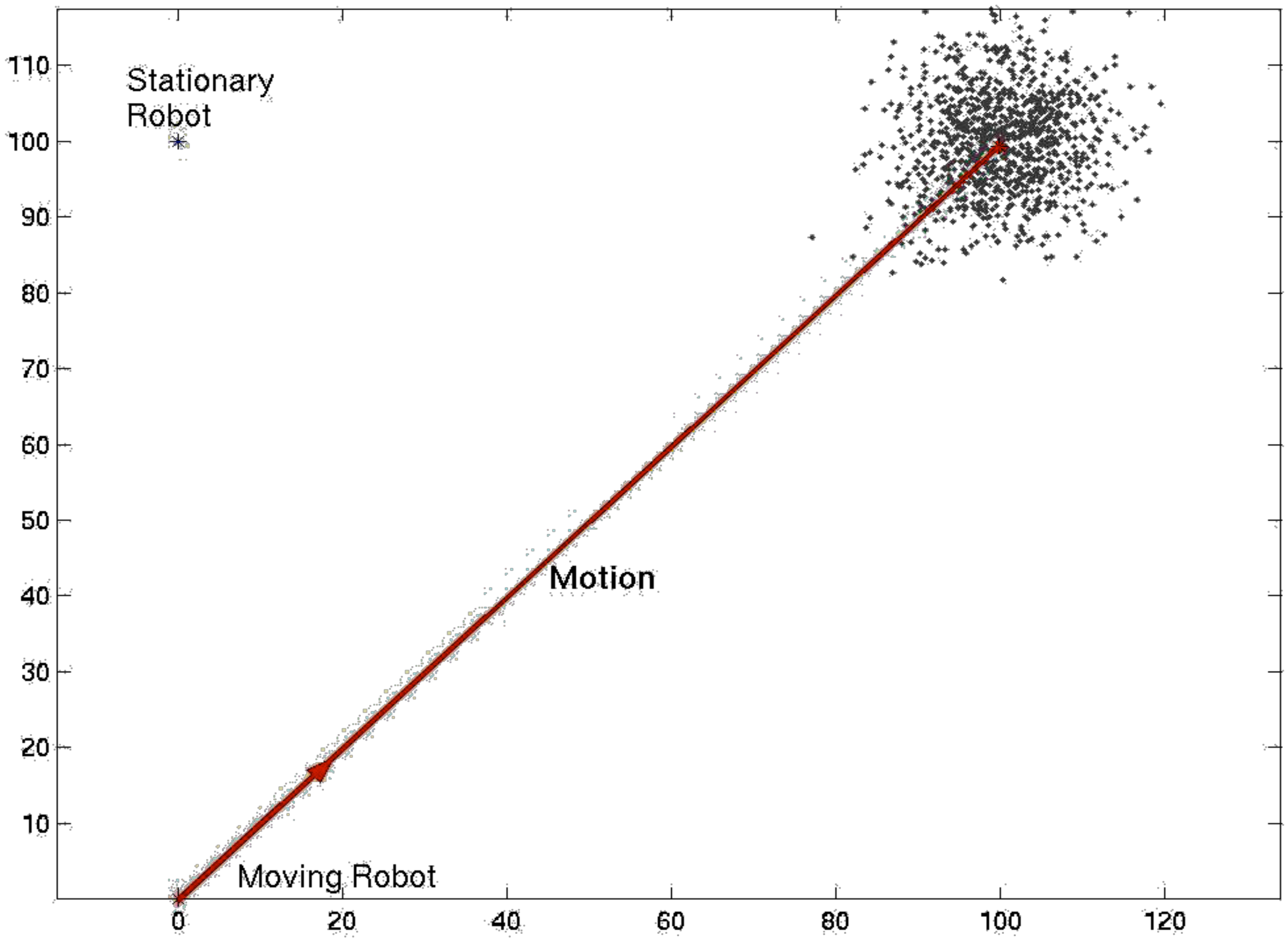
Tracker Weighting Function

U p d a t e

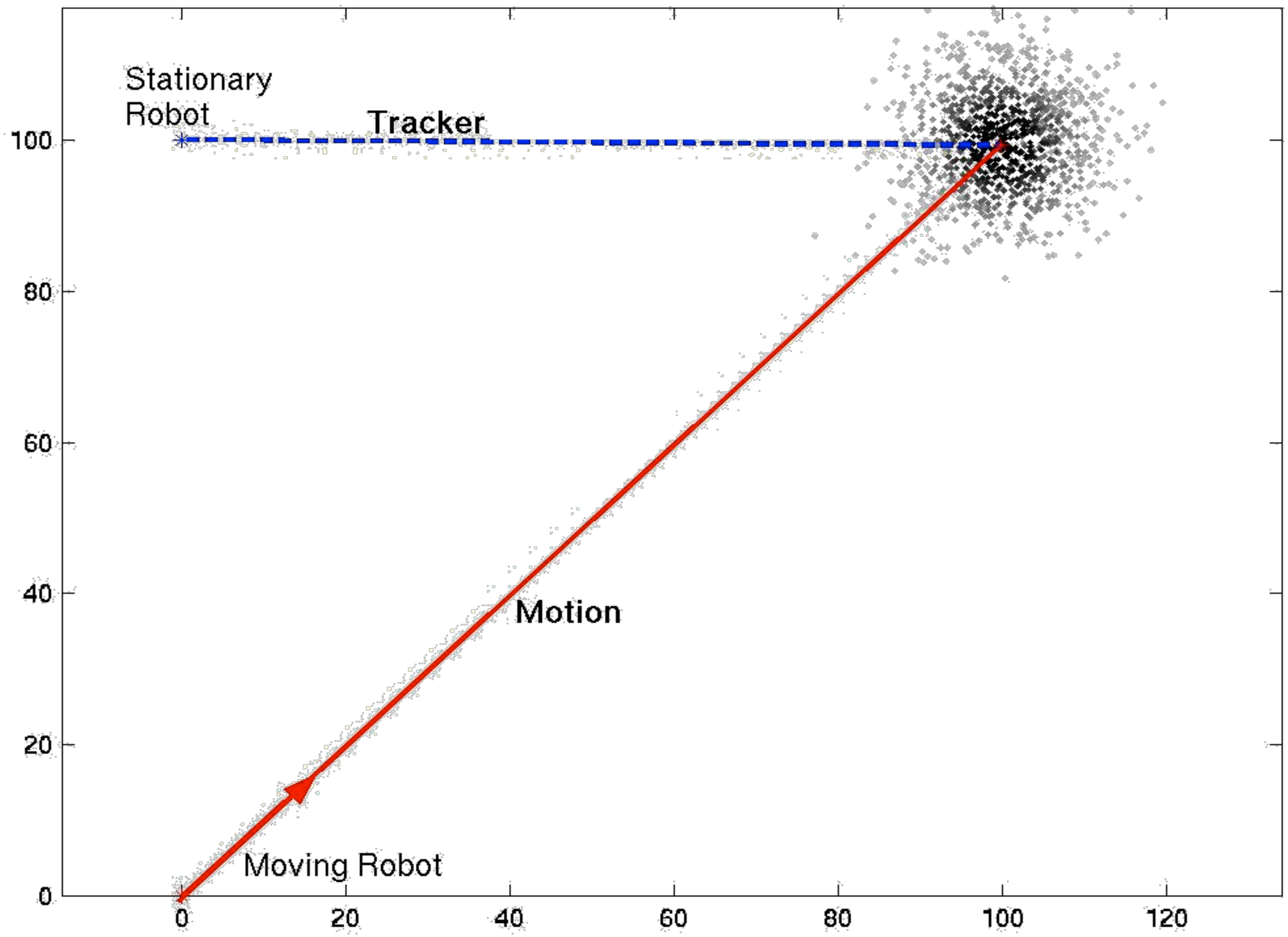


$$W = \frac{1}{\sqrt{2\pi\sigma_\rho^2}} e^{\frac{-(\rho-\rho_i)^2}{\sigma_\rho^2}} \frac{1}{\sqrt{2\pi\sigma_\theta^2}} e^{\frac{-(\theta-\theta_i)^2}{\sigma_\theta^2}} \frac{1}{\sqrt{2\pi\sigma_\phi^2}} e^{\frac{-(\phi-\phi_i)^2}{\sigma_\phi^2}}$$

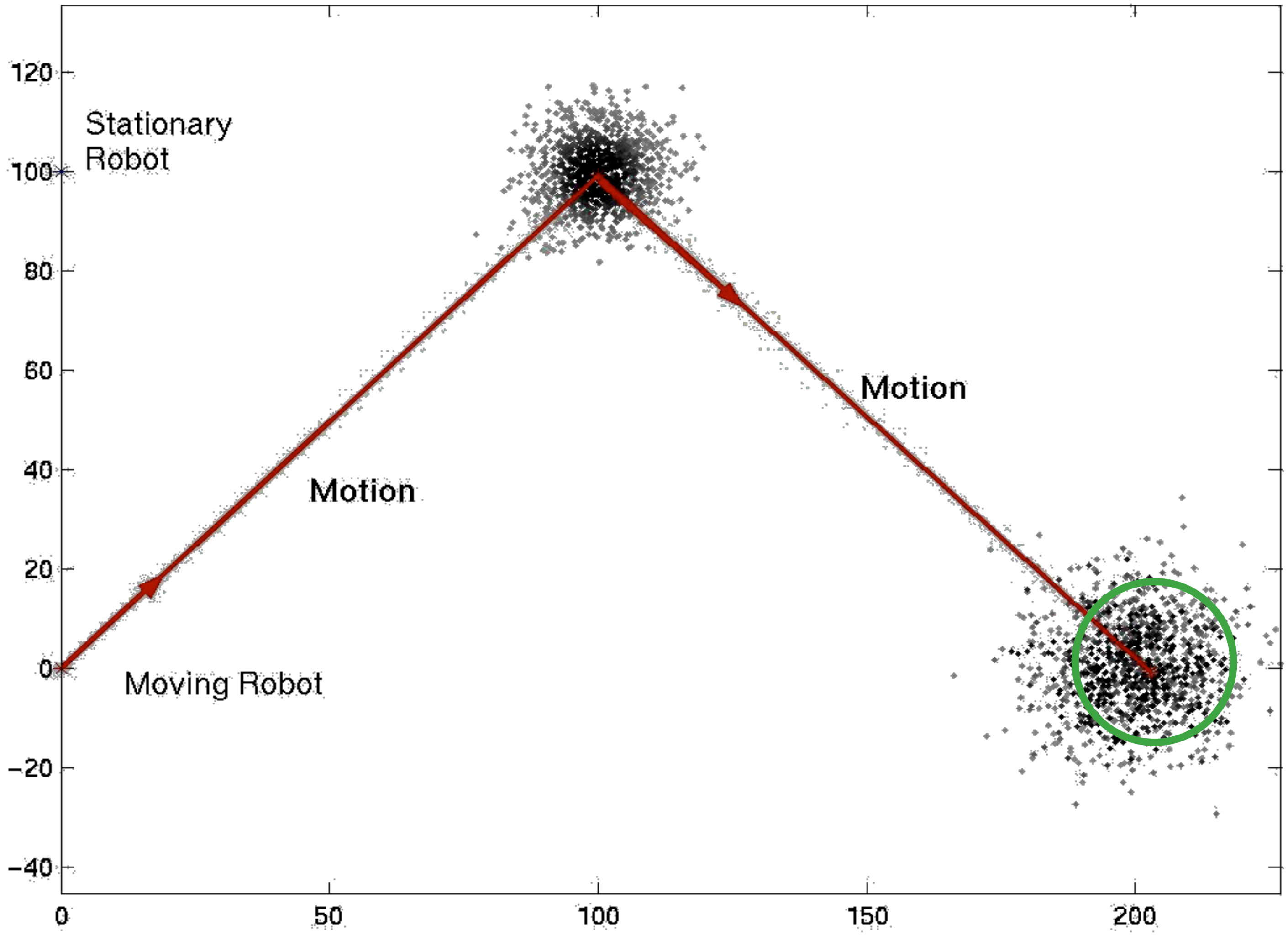
Example: Prediction



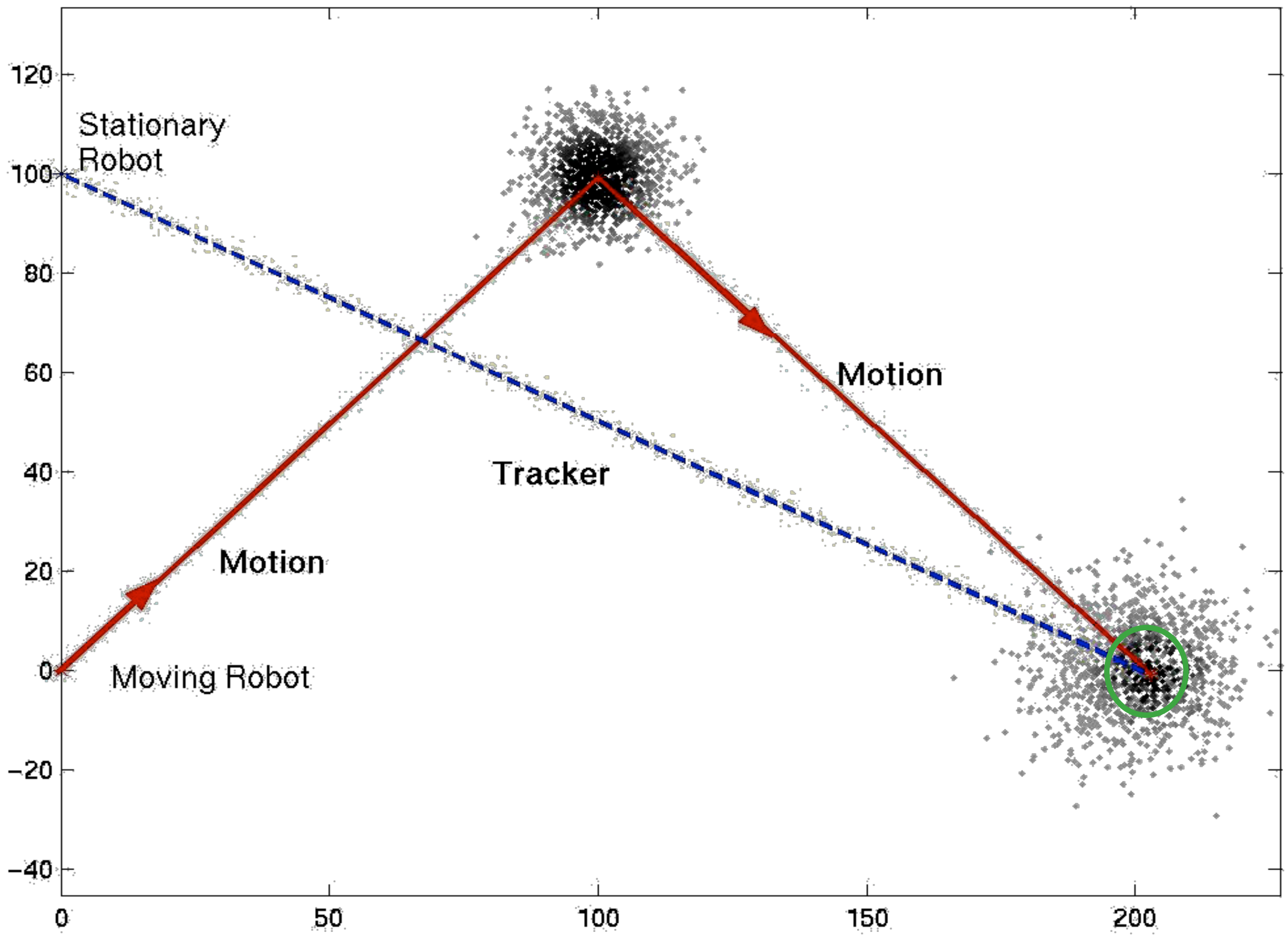
Example: Update



Example: Prediction



Example: Update

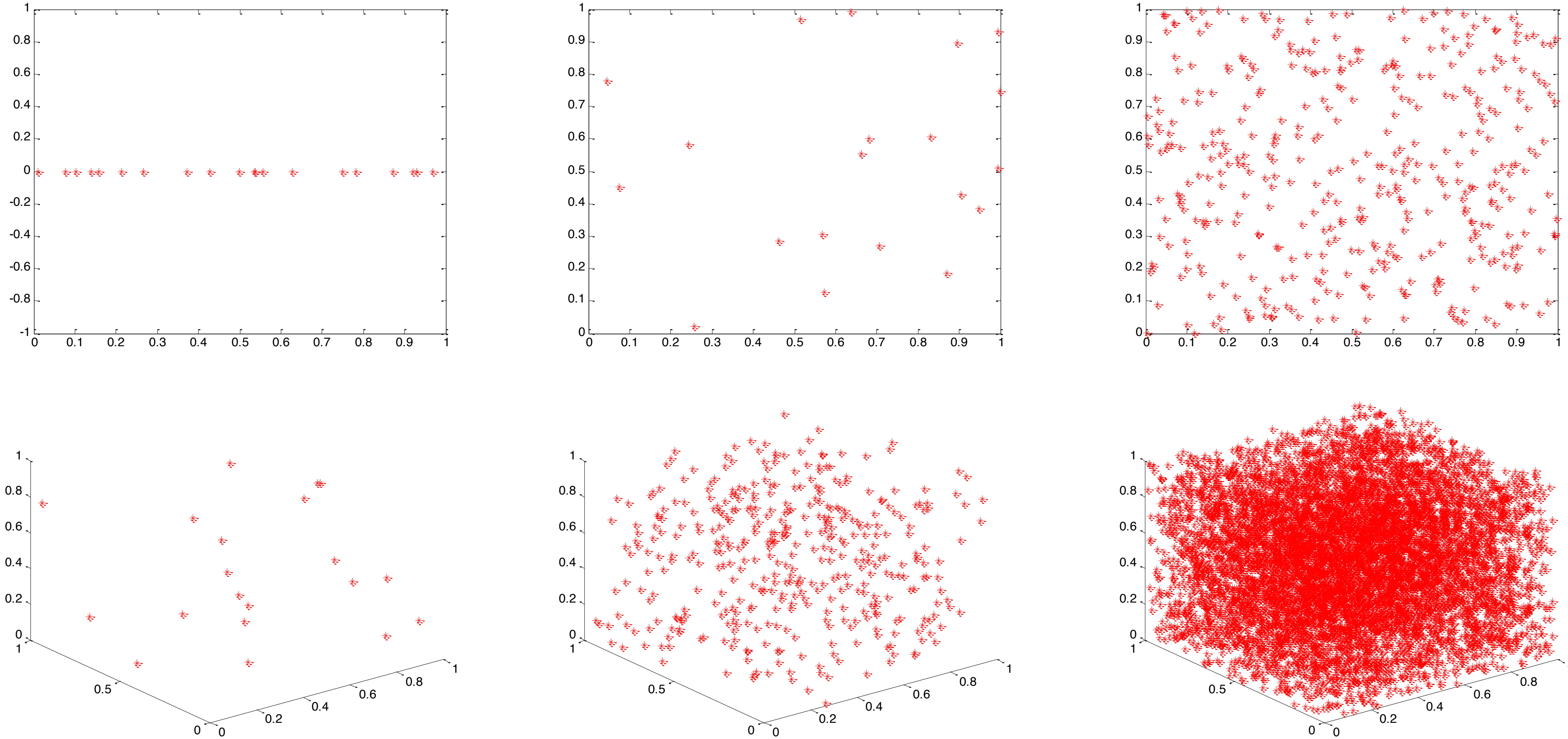


Variations on PF

- Add some particles uniformly
- Add some particles where the sensor indicates
- Add some jitter to the particles after propagation
- Combine EKFs to track landmarks

Keep in Mind:

- The number of particles increases with the dimension of the state space



Complexity results for SLAM

- n=number of map features
- Problem: naïve methods have high complexity
  - EKF models O(n^2) covariance matrix
  - PF requires prohibitively many particles to characterize complex, interdependent distribution
- Solution: exploit conditional independencies
  - Feature estimates are independent given robot's path

Generating Random Numbers

From a uniform RNG produce samples following the Normal distribution:

The most basic form of the transformation looks like:

y1 = sqrt(- 2 ln(x1)) cos(2 pi x2)

y2 = sqrt(- 2 ln(x1)) sin(2 pi x2)

The **polar form** of the Box-Muller transformation is both faster and more robust numerically. The algorithmic description of it is:

```
float x1, x2, w, y1, y2;
do {
    x1 = 2.0 * ranf() - 1.0; x2 = 2.0 * ranf() - 1.0;
    w = x1 * x1 + x2 * x2;
} while ( w >= 1.0 );
 w = sqrt( (-2.0 * ln( w ) ) / w );
y1 = x1 * w;
y2 = x2 * w;
```

See: http://www.taygeta.com/random/gaussian.html

Rao-Blackwellization

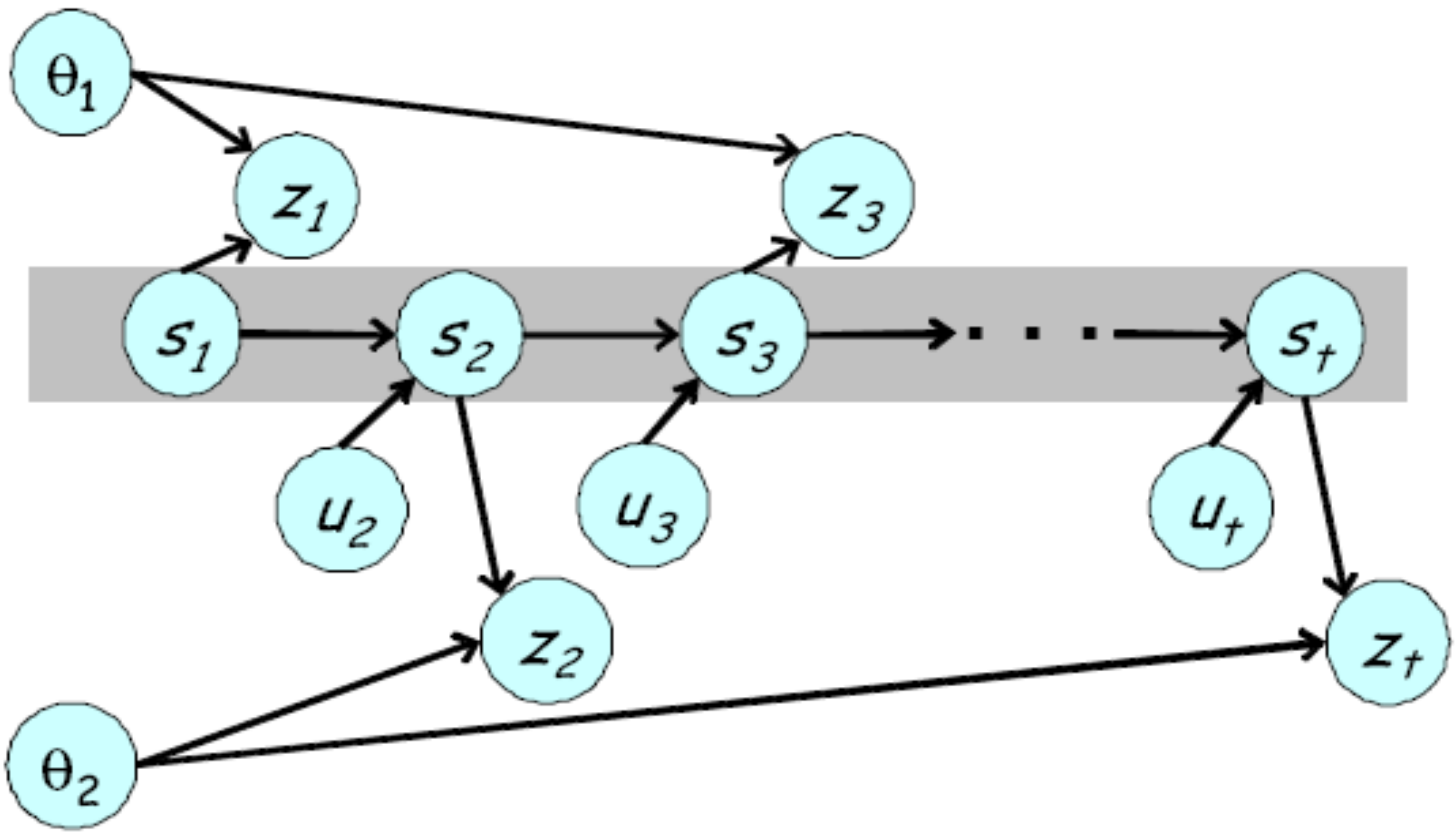


Figure from [Montemerlo et al – Fast SLAM]

RBPF Implementation for SLAM

- 2 steps:
  - Particle filter to estimate robot's pose
  - Set of low-dimensional, independent EKF's (one **per** feature **per** particle)
- E.g. FastSLAM which includes several computational speedups to achieve O(M logN) complexity (with M number of particles)

Questions

- For more information on PF:

  http://www.cim.mcgill.ca/~yiannis/ParticleTutorial.html