Probabilistic Robotics Course

UKF Localization [Example Application]

Giorgio Grisetti
grisetti@diag.uniroma1.it

Department of Computer, Control, and Management Engineering
Sapienza University of Rome

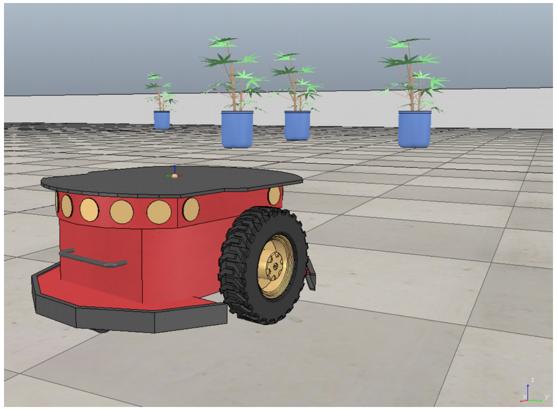
Outline

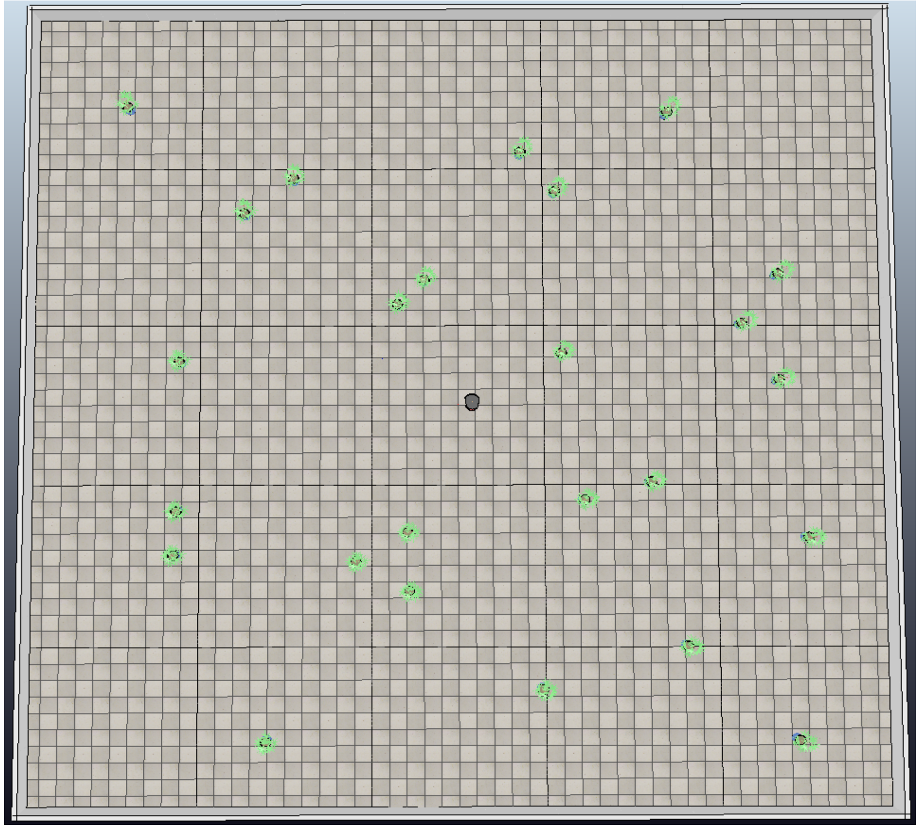
- Scenario
- Controls
- Observations
- Non-Linear Systems and Gaussian Noise
- Unscented Kalman Filter

Scenario

Orazio moves on a 2D plane

- Is controlled by translational and rotational velocities
- Senses a set of uniquely **distinguishable** landmarks through a “2D landmark sensors”
- The **location** of the landmarks in the world **is known**





Approaching the problem

We want to develop a UKF based algorithm to track the position of Orazio as it moves

The inputs of our algorithms will be

- velocity measurements
- landmark measurements

The prior knowledge about the map is represented by the location of each landmark in the world

Prior

The map is represented as a set of n 2D landmark coordinates:

$$\mathbf{l}^{[n]} = \begin{pmatrix} x^{[n]} \\ y^{[n]} \end{pmatrix} \in \Re^2$$

We use the letter $\mathbf{l}^{[n]}$ instead of $\mathbf{x}^{[n]}$ since the landmarks are not part of the state (we know their positions in this scenario)

Domains

Find an Euclidean parameterization of non-Euclidean spaces

$$\mathbf{X}_t = [\mathbf{R}_t | \mathbf{t}_t] \in SE(2) \Rightarrow \mathbf{x}_t = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} \in \Re^3$$

- state space

$$\mathbf{u}_t = \begin{pmatrix} u_t^x \\ u_t^\theta \end{pmatrix} \in \Re^2$$

- space of controls (inputs)

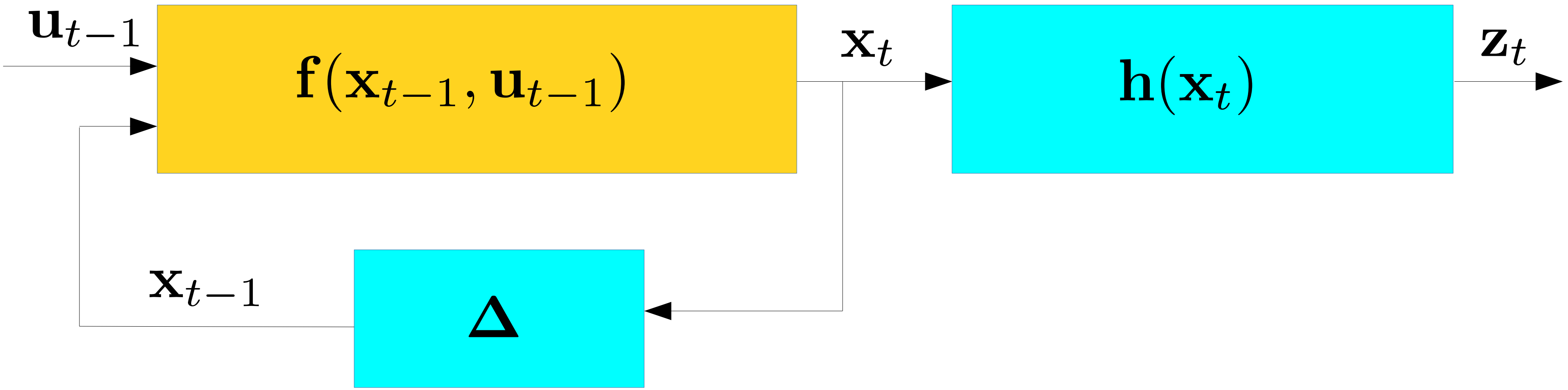
$$\mathbf{z}_t^{[n]} = \begin{pmatrix} x_t^{[n]} \\ y_t^{[n]} \end{pmatrix} \in \Re^2$$

- space of observations (measurements)

poses are not Euclidean, we map them to 3D vectors

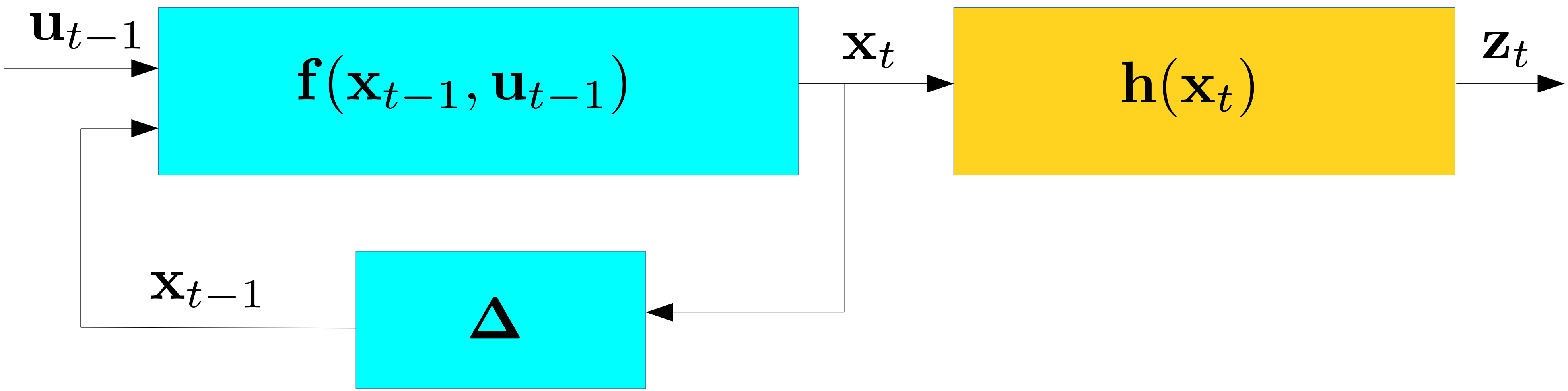
measurement and control, in this problem are already Euclidean

Transition Function



$$\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \begin{pmatrix} x_{t-1} + u^x_{t-1} \cdot \cos(\theta_{t-1}) \\ y_{t-1} + u^x_{t-1} \cdot \sin(\theta_{t-1}) \\ \theta_{t-1} + u^\theta_{t-1} \end{pmatrix}$$

Measurement Function



We have [n] measurement functions, one per landmark

$$\begin{aligned} \mathbf{z}_t^{[n]} &= \mathbf{h}^{[n]}(\mathbf{x}_t) \\ &= \mathbf{R}_t^T(\mathbf{l}^{[n]} - \mathbf{t}_t) \\ &= \begin{pmatrix} \cos\theta_t(x^{[n]} - x_t) + \sin\theta_t(y^{[n]} - y_t) \\ -\sin\theta_t(x^{[n]} - x_t) + \cos\theta_t(y^{[n]} - y_t) \end{pmatrix} \end{aligned}$$

relative position of the n^th landmark w.r.t. the robot at time t

$$\mathbf{R}_t = \begin{pmatrix} \cos\theta_t & -\sin\theta_t \\ \sin\theta_t & \cos\theta_t \end{pmatrix}$$

rotation matrix of theta

Control Noise

We assume the velocity measurements are effected by a Gaussian noise resulting from the sum of two aspects

- a constant noise
- a velocity dependent term whose standard deviation grows with the speed
- translational and rotational noise are assumed independent

$$\mathbf{n}_{u,t} \sim \mathcal{N}\left(\mathbf{n}_{u,t}; \mathbf{0}, \begin{pmatrix} (u_t^x)^2 + \sigma_v^2 & 0 \\ 0 & (u_t^\theta)^2 + \sigma_\omega^2 \end{pmatrix}\right)$$

Measurement Noise

We assume it is zero mean, constant

$$\mathbf{n}_z \sim \mathcal{N}\left(\mathbf{n}_z; \mathbf{0}, \begin{pmatrix} \sigma_z^2 & 0 \\ 0 & \sigma_z^2 \end{pmatrix}\right)$$

Wrapup (UKF)

$$\begin{pmatrix} \mathbf{x}_{t-1|t-1} \\ \mathbf{u}_{t-1} \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_{t-1|t-1} \\ \mu_{u,t-1} \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_{t-1|t-1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{u,t-1} \end{pmatrix} \right]$$

compute sigma points

$$\mathcal{X}^{(i)}_{t-1|t-1}$$

apply transition

$$\mathcal{X}^{(i)}_{t|t-1} = \mathbf{f}(\mathcal{X}^{(i)}_{t-1|t-1})$$

Predict

compute sigma points and mean of measurement

$$\mathbf{z}^{(i)}_t = \mathbf{h}(\mathbf{x}^{(i)}_{t|t-1})$$

$$\mu_z = \sum_i w^{(i)}_m \mathbf{z}^{(i)}_t$$

compute cross correlation of joint distribution

$$\boldsymbol{\Sigma}_{x,z} = \sum_i w^{(i)}_c (\mathbf{x}^{(i)}_{t|t-1} - \mu_{t|t-1})(\mathbf{z}^{(i)}_t - \mu_z)^T$$

$$\boldsymbol{\Sigma}_{z,z} = \sum w^{(i)}_c (\mathbf{z}^{(i)}_t - \mu_z)(\mathbf{z}^{(i)}_t - \mu_z)^T$$

conditioning

$$\begin{aligned} \mu_{t|t} &= \mu_{t|t-1} + \boldsymbol{\Sigma}_{x,z}\left(\boldsymbol{\Sigma}_{z|x} + \boldsymbol{\Sigma}_{z,z}\right)^{-1}(\mathbf{z}_t - \mu_z) \\ \boldsymbol{\Sigma}_{t|t} &= \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{\Sigma}_{x,z}\left(\boldsymbol{\Sigma}_{z|x} + \boldsymbol{\Sigma}_{z,z}\right)^{-1}\boldsymbol{\Sigma}_{z,x} \end{aligned}$$

Update

Sigma Points

$$\mathbf{x} \sim \mathcal{N}(\mathbf{x}; \mu, \mathbf{\Sigma})) \rightarrow \mathcal{X}^{(i)}_{t-1|t-1}$$

$$\lambda = \alpha^2(n + \kappa) - n$$

$$\mathbf{L} = \sqrt{(n + \lambda)\mathbf{A}}$$

$$\begin{aligned} w_m^{(0)} &= \tfrac{\lambda}{n+\lambda} \\ w_c^{(0)} &= w_m^{(0)} + (1 - \alpha^2 + \beta) \\ w_c^{(i)} &= w_m^{(i)} = \tfrac{1}{2(n+\lambda)} \end{aligned}$$

$$\begin{aligned} \mathbf{x}^{(0)} &= \mu \\ \mathbf{x}^{(i)} &= \mu + [\mathbf{L}]_i \text{ for } i \in [1..n] \\ \mathbf{x}^{(i)} &= \mu - [\mathbf{L}]_{n-i} \text{ for } i \in [n+1..2n] \end{aligned}$$

Sigma Points: Code

```
function [sigmaP, weightsM, weightsC] = compute_sigma_points(mu,
    sigma)

 state_dim = size(mu,1);
 num_of_sigma_points = 2*state_dim+1;

 %From theory
 k = 0;
 alpha = 1e-3;
 lambda = alpha^2 *   state_dim
 bet = 2;

 %init the sigma points matrix and its weight vectors
 sigmaP = zeros(size(mu,1),num_of_sigma_points);
 weightsM = zeros(num_of_sigma_points,1);
 weightsC = zeros(num_of_sigma_points,1);

 %the first weights can be computed as
 weightsM(1) = %TODO;
 weightsC(1) = %TODO;

 %the first sigma point is the mean
 sigmaP(:,1) = %TODO;
```

First we compute the values for $w_m^{(0)}, w_c^{(0)}, \mathbf{x}^{(0)}$

Sigma Points: Code

weight_value

$$w_m^{(i)} = w_c^{(i)}$$

```
%compute the other weights
weight_value = %TODO;
weightsM(2:end) = repmat(weight_value, num_of_sigma_points-1,1);
weightsC(2:end) = weightsM(2:end);

%chol extract the Cholesky Decomposition
%with the arg "lower" it returns the lower triangular L
[L] = chol(sigma*(state_dim+lambda) , "lower");

point_idx = 2;
for i=1:state_dim
  %half of the remaining 2*state_dim sigma points
  sigmaP(:,point_idx) = %TODO;
  point_idx++;
  sigmaP(:,point_idx) = %TODO;
  point_idx++;
end
endfunction
```

Values of $\mathbf{x}^{(i)}$

for $i \in [1..n]$

for $i \in [n+1..2n]$

Reconstruct Mean

$$\mathcal{X}^{(i)}_{t-1|t-1} \rightarrow \mathbf{x} \sim \mathcal{N}(\mathbf{x}; \mu, \mathbf{\Sigma}))$$

$$\mu = \sum w_m^{(i)} \mathbf{x}^{(i)}$$

$$\mathbf{\Sigma} = \sum w_c^{(i)} (\mathbf{x}^{(i)} - \mu)(\mathbf{x}^{(i)} - \mu)^T$$

Reconstruct Mean: Code

```
function [mu, sigma] = reconstruct_mean_cov(sigmaP, wM, wC)

 state_dim = size(sigmaP,1);
 num_of_sigma_points = size(sigmaP,2);

 %initialize mean
 mu = zeros(state_dim,1);
 %populate mean
 for i=1:num_of_sigma_points
  mu += %TODO;
 endfor

 %initialize covariance
 sigma = zeros(state_dim,state_dim);
 %populate covariance
 for i=1:num_of_sigma_points
  delta = %TODO;
  sigma += %TODO;
 endfor

endfunction
```

Hint

- Check that the forward and backward transformation work
  - Generate a random covariance and mean
  - Extract the sigma points
  - Reconstruct the mean and covariance
  - Verify you get the same input

Prediction: Sigma Points

$$\begin{pmatrix} \mathbf{x}_{t-1|t-1} \\ \mathbf{u}_{t-1} \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_{t-1|t-1} \\ \mu_{u,t-1} \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_{t-1|t-1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{u,t-1} \end{pmatrix}\right]$$

compute sigma points

$$\mathcal{X}^{(i)}_{t-1|t-1}$$

apply transition

$$\mathcal{X}^{(i)}_{t|t-1} = \mathbf{f}(\mathcal{X}^{(i)}_{t-1|t-1})$$

Predict

$$\lambda = \alpha^2 n$$

$$\mathbf{L} = \sqrt{(n+\lambda)\mathbf{A}}$$

$$\begin{aligned} \mathbf{x}^{(0)} &= \mu \\ \mathbf{x}^{(i)} &= \mu + [\mathbf{L}]_i \ \text{ for } i \in [1..n] \\ \mathbf{x}^{(i)} &= \mu - [\mathbf{L}]_{n-i} \ \text{ for } i \in [n+1..2n] \end{aligned}$$

$$\begin{aligned} w_m^{(0)} &= \frac{\lambda}{n+\lambda} \\ w_c^{(0)} &= w_m^{(0)} + (1-\alpha^2+\beta) \\ w_c^{(i)} &= w_m^{(i)} = \frac{1}{2(n+\lambda)} \end{aligned}$$

Prediction: Transition

$$\begin{pmatrix} \mathbf{x}_{t-1|t-1} \\ \mathbf{u}_{t-1} \end{pmatrix} \sim \mathcal{N}\left[\begin{pmatrix} \mu_{t-1|t-1} \\ \mu_{u,t-1} \end{pmatrix}, \begin{pmatrix} \mathbf{\Sigma}_{t-1|t-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Sigma}_{u,t-1} \end{pmatrix} \right]$$

compute sigma points

$$\mathcal{X}^{(i)}_{t-1|t-1}$$

apply transition

$$\mathcal{X}^{(i)}_{t|t-1} = \mathbf{f}(\mathcal{X}^{(i)}_{t-1|t-1})$$

Predict

Same motion model is applied to the values of the sigma points

$$\mathbf{x}^{(i)}_{t|t-1} = \mathbf{f}(\mathbf{x}^{(i)}_{t-1|t-1}, \mathbf{u}^{(i)}_{t-1|t-1})$$

$$w^{(i)}_{t|t-1} = w^{(i)}_{t-1|t-1}$$

weights propagated

Prediction: Code

```
function [sigmaP, weightsM, weightsC] = prediction(mu, sigma, transition)
 u = transition.v;
 u_x = u(1);  u_theta = u(3);

 noise = 0.1;                       %constant part
 v_noise = u_x^2;                   %lin vel dependent part
 w_noise = u_theta^2;               %ang vel dependent part
 sigma_u = [ noise+v_noise, 0;
             0, noise+v_noise];

 sigma_xu = zeros(5,5);
 sigma_xu(1:3,1:3) = sigma;
 sigma_xu(4:5,4:5) = sigma_u;

 mu_xu = zeros(5,1);
 mu_xu(1:3) = mu;
 mu_xu(4:5) = [u_x; u_theta];

 % extract the sigma points
 [sigmaP_xu, weightsM, weightsC] = compute_sigma_points(mu_xu, sigma_xu);

 for i=1:size(sigmaP_xu,1)
  curr_sigma_mu = sigmaP_xu(1:3,i);
  curr_input_u = [sigmaP_xu(4,i); 0; sigmaP_xu(5,i)];
  sigmaP(:,end+1) = motion_model(curr_sigma_mu, curr_input_u);
 end
endfunction
```

Apply transition to the sigma points

Update: Predict Measurement

compute sigma points and mean of measurement

$$\mathbf{z}_t^{(i)} = \mathbf{h}(\mathbf{x}_{t|t-1}^{(i)})$$

$$\mu_z = \sum_i w_m^{(i)} \mathbf{z}_t^{(i)}$$

$$\mathbf{\Sigma}_{x,z} = \sum_i w_c^{(i)} (\mathbf{x}_{t|t-1}^{(i)} - \mu_{t|t-1})(\mathbf{z}_t^{(i)} - \mu_z)^T$$

$$\mathbf{\Sigma}_{z,z} = \sum_i w_c^{(i)} (\mathbf{z}_t^{(i)} - \mu_z)(\mathbf{z}_t^{(i)} - \mu_z)^T$$

compute cross correlation of joint distribution

½ Update

$$\mu_z = \sum_i w_m^{(i)} \mathbf{z}_t^{(i)} \qquad \mu_{t|t-1} = \sum w_m^{(i)} \mathbf{x}_{t|t-1}^{(i)}$$

From the sigma points the mean can be reconstructed in this way

Update: Chain Rule

compute sigma points and mean of measurement

$$\mathbf{z}_t^{(i)} = \mathbf{h}(\mathbf{x}_{t|t-1}^{(i)})$$

$$\mu_z = \sum_i w_m^{(i)} \mathbf{z}_t^{(i)}$$

$$\mathbf{\Sigma}_{x,z} = \sum_i w_c^{(i)} (\mathbf{x}_{t|t-1}^{(i)} - \mu_{t|t-1})(\mathbf{z}_t^{(i)} - \mu_z)^T$$

$$\mathbf{\Sigma}_{z,z} = \sum_i w_c^{(i)} (\mathbf{z}_t^{(i)} - \mu_z)(\mathbf{z}_t^{(i)} - \mu_z)^T$$

compute cross correlation of joint distribution

½ Update

$$\mu_{x,z} = \begin{pmatrix} \mu_x \\ \mu_z \end{pmatrix} = \begin{pmatrix} \mu_x \\ \mu_z \end{pmatrix}$$

$$\mathbf{\Sigma}_{x,z} = \begin{pmatrix} \mathbf{\Sigma}_x & \boxed{\mathbf{\Sigma}_{x,z}} \\ \mathbf{\Sigma}_{z,x} & \mathbf{\Sigma}_{z|x} + \boxed{\mathbf{\Sigma}_{z,z}} \end{pmatrix}$$

$$\mathbf{\Sigma}_{t|t-1} = \sum w_c^{(i)} (\mathbf{x}_{t|t-1}^{(i)} - \mu_{t|t-1})(\mathbf{x}_{t|t-1}^{(i)} - \mu_{t|t-1})^T$$

covariance reconstruction

Update 1: Code

```
function [mu, sigma] = correction(sigmaP, weightsM, weightsC,
    landmarks, observations)
 %usual definition of state_dim and so on
 [...]
 %define an iterator to work on sigmaZ_t
 id_x = 1;
 for i=1:num_landmarks_seen
  %retrieve info about the observed landmark
  measurement = observations.observation(i);
  z_t(end+1,:) = measurement.x_pose; % where we see the landmark
  z_t(end+1,:) = measurement.y_pose;
  current_land = searchById(landmarks, measurement.id);
  lx = current_land.x_pose;
  ly = current_land.y_pose;

  %where I should see that landmark from each sigma point value
  for j=1:size(sigmaP,2)
    current_sigma_point = sigmaP(:,j);
    measure_prediction = measurement_function(current_sigma_point,
    [lx; ly]);
    sigmaZ_t(id_x:id_x+1,j) = measure_prediction;
  endfor
  id_x +=2;
 endfor
```

Update 2: Code

```
%once we have computed our sigmaZ_t, we can reconstruct mu_z and relative covariace
[mu_z, Sigma_z] = reconstruct_mean_cov(sigmaZ_t, weightsM, weightsC);
% here we need to correct mu and sigma, so first of all we reconstruct mu and sigma from sigma points
[mu, sigma] = reconstruct_mean_cov(sigmaP, weightsM, weightsC);

%observation noise
noise = 0.01;
Sigma_noise = eye(2*num_landmarks_seen)*noise;

Sigma_xz = zeros(state_dim, 2*num_landmarks_seen);
for i=1:size(sigmaP,2)
 delta_x= %TODO;
 delta_z= %TODO;
 Sigma_xz += %TODO;
end
```

$\Sigma_{x,z}$

Update: Conditioning

$$
\begin{aligned}
\mu_{t|t} &= \mu_{t|t-1} + \mathbf{\Sigma}_{x,z}\left(\mathbf{\Sigma}_{z|x} + \mathbf{\Sigma}_{z,z}\right)^{-1}(\mathbf{z}_t - \mu_z) \\
\mathbf{\Sigma}_{t|t} &= \mathbf{\Sigma}_{t|t-1} - \mathbf{\Sigma}_{x,z}\left(\mathbf{\Sigma}_{z|x} + \mathbf{\Sigma}_{z,z}\right)^{-1}\mathbf{\Sigma}_{z,x}
\end{aligned}
$$

½ Update

```
%Kalman gain
K = Sigma_xz * inv(Sigma_z + Sigma_noise);

%update mu
 error = %TODO;
 correction = %TODO;
mu = mu + correction;

%update sigma
 sigma = %TODO;
endfunction
```

Compute the error and apply correction

All the ingredients are ready to update the covariance sigma

Hands On!