

# **Probabilistic Robotics Course**

## **UKF Localization [Example Application]**

**Giorgio Grisetti**

`grisetti@diag.uniroma1.it`

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

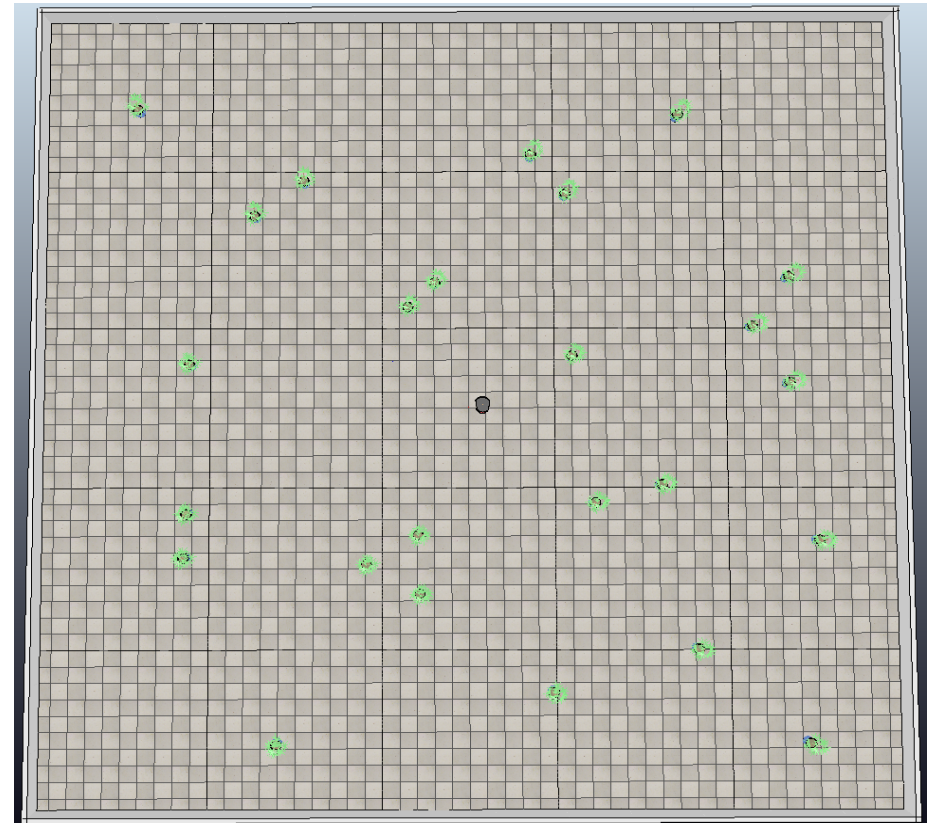
# Outline

- 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 landmark coordinates

$$\mathbf{l}^{[i]} = \begin{pmatrix} x^{[i]} \\ y^{[i]} \end{pmatrix} \in \mathbb{R}^2$$

# Domains

Define

- state space

$$\mathbf{X}_t = [\mathbf{R}_t | \mathbf{t}_t] \in SE(2)$$

- space of controls (inputs)

$$\mathbf{u}_t = \begin{pmatrix} u_t^1 \\ u_t^2 \end{pmatrix} \in \mathbb{R}^2$$

- space of observations (measurements)

$$\mathbf{z}_t^{[i]} = \begin{pmatrix} x_t^{[i]} \\ y_t^{[i]} \end{pmatrix} \in \mathbb{R}^2$$

# Domains

Find an Euclidean parameterization of non-Euclidean spaces

$$\mathbf{X}_t = [\mathbf{R}_t | \mathbf{t}_t] \in SE(2) \xrightarrow{\text{blue arrow}} \mathbf{x}_t = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} \in \mathbb{R}^3$$

- state space

$$\mathbf{u}_t = \begin{pmatrix} u_t^1 \\ u_t^2 \end{pmatrix} \in \mathbb{R}^2$$

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

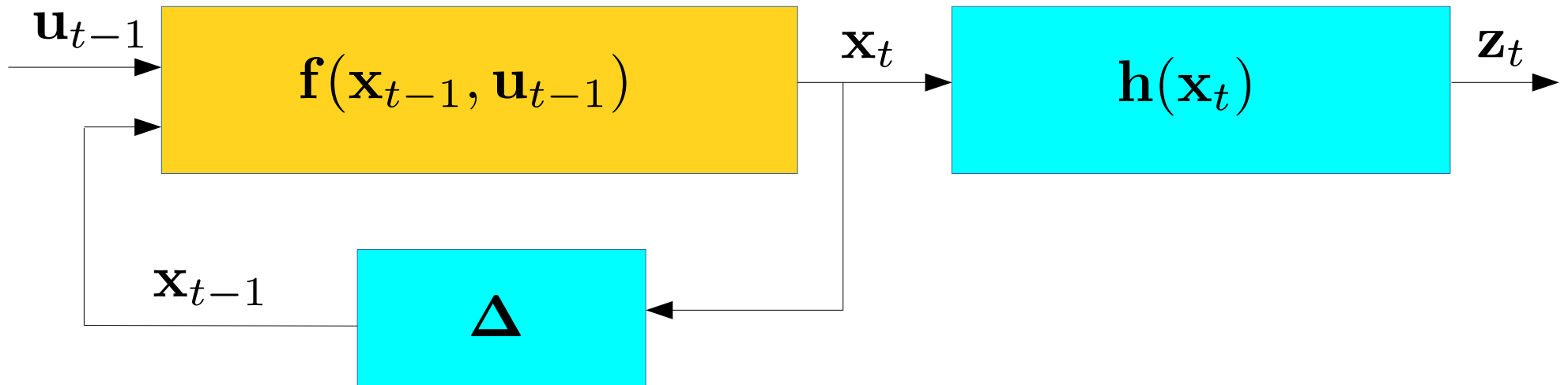
- space of controls (inputs)

$$\mathbf{z}_t = \begin{pmatrix} x_t^{[i]} \\ y_t^{[i]} \end{pmatrix} \in \mathbb{R}^2$$

measurement and control, in this problem are already Euclidean

- space of observations (measurements)

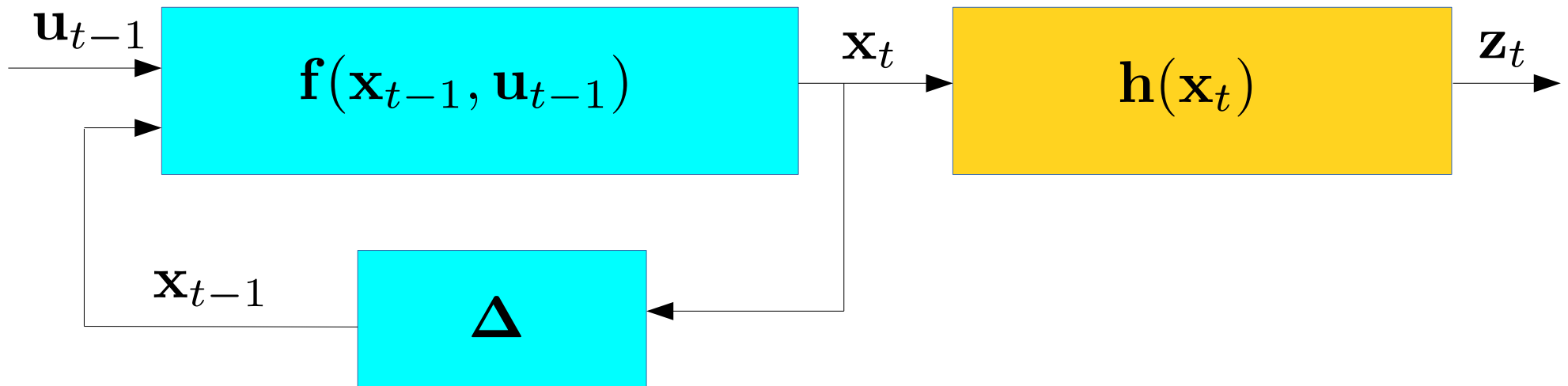
# Transition Function



$$\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \begin{pmatrix} x_{t-1} + u_{t-1}^1 \cdot \cos(\theta_{t-1}) \\ y_{t-1} + u_{t-1}^1 \cdot \sin(\theta_{t-1}) \\ \theta_{t-1} + u_{t-1}^2 \end{pmatrix}$$



# Measurement Function



We have  $[i]$  measurement functions, one per landmark

$z_t^{[i]}$

$=$

$h^{[i]}(x_t)$

$=$

$R_t^T(l^{[i]} - t_t)$

$=$

$$\begin{pmatrix} \cos \theta_t (x^{[i]} - x_t) + \sin \theta_t (y^{[i]} - y_t) \\ -\sin \theta_t (x^{[i]} - x_t) + \cos \theta_t (y^{[i]} - y_t) \end{pmatrix}$$

relative position of the  $i^{\text{th}}$  landmark w.r.t the robot at time  $t$

$$R_t = \begin{pmatrix} \cos \theta_t & -\sin \theta_t \\ \sin \theta_t & \cos \theta_t \end{pmatrix}$$

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

$$\mathbf{n}_{u,t} \sim \mathcal{N} \left( \mathbf{n}_{u,t}; \mathbf{0}, \begin{pmatrix} (u_t^1)^2 + \sigma_v^2 & 0 \\ 0 & (u_t^2)^2 + \sigma_\omega^2 \end{pmatrix} \right)$$

# 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)

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

apply transition

compute sigma points

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

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

**Predict**

compute sigma points and mean of measurement

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

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

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

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

compute cross correlation of joint distribution

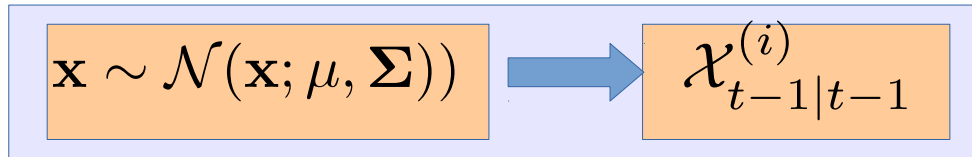
conditioning

$$\mu_{t|t} = \mu_{t|t-1} + \Sigma_{x,z} (\Sigma_{z|x} + \Sigma_{z,z})^{-1} (\mathbf{z}_t - \mu_z)$$

$$\Sigma_{t|t} = \Sigma_{t|t-1} - \Sigma_{x,z} (\Sigma_{z|x} + \Sigma_{z,z})^{-1} \Sigma_{z,x}$$

**Update**

# Sigma Points



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

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

$$\mathbf{x}^{(0)} = \mu$$

$$\mathbf{x}^{(i)} = \mu + [\mathbf{L}]_i \text{ for } i \in [1..n]$$

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

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

# Sigma Points: Code

```
1 function [sigmaP, weightsM, weightsC] = compute_sigma_points(mu,  
2     sigma)  
3     state_dim = size(mu,1);  
4     num_of_sigma_points = 2*state_dim+1;  
5  
6     %From theory  
7     k = 0;  
8     alpha = 1e-3;  
9     lambda = alpha^2 * ( state_dim + k );  
10    bet = 2;  
11  
12    %init the sigma points matrix and its weight vectors  
13    sigmaP = zeros(size(mu,1),num_of_sigma_points);  
14    weightsM = zeros(num_of_sigma_points,1);  
15    weightsC = zeros(num_of_sigma_points,1);  
16  
17    %the first weights can be computed as  
18    weightsM(1) = %TODO;  
19    weightsC(1) = %TODO;  
20  
21    %the first sigma point is the mean  
22    sigmaP(:,1) = %TODO;
```

First we compute  
the values for

$w_m^{(0)}, w_c^{(0)}, \mathbf{x}^{(0)}$

# Sigma Points: Code

weight\_value

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

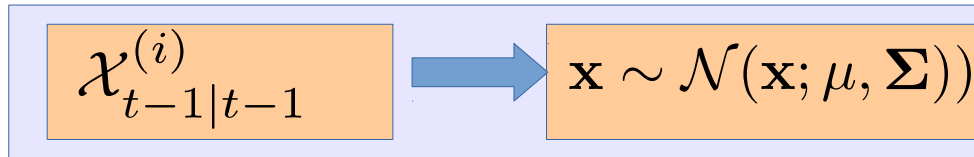
```
1  %compute the other weights
2  weight_value = %TODO;
3  weightsM(2:end) = repmat(weight_value , num_of_sigma_points -1,1);
4  weightsC(2:end) = weightsM(2:end);
5
6  %chol extract the Cholesky Decomposition
7  %with the arg "lower" it returns the lower triangular L
8  [L] = chol(sigma*(state_dim+lambda) , "lower");
9
10 point_idx = 2;
11 for i=1:state_dim
12     %half of the remaining 2*state_dim sigma points
13     sigmaP(:,point_idx) = %TODO;
14     point_idx++;
15     sigmaP(:,point_idx) = %TODO;
16     point_idx++;
17 end
18 endfunction
```

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

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

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

# Reconstruct Mean



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

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



# Reconstruct Mean: Code

```
1 function [mu, sigma] = reconstruct_mean_cov(sigmaP, wM, wC)
2
3 state_dim = size(sigmaP,1);
4 num_of_sigma_points = size(sigmaP,2);
5
6 %initialize mean
7 mu = zeros(state_dim,1);
8 %populate mean
9 for i=1:num_of_sigma_points
10     mu += %TODO;
11 endfor
12
13 %initialize covariance
14 sigma = zeros(state_dim, state_dim);
15 %populate covariance
16 for i=1:num_of_sigma_points
17     delta = %TODO;
18     sigma += %TODO;
19 endfor
20
21 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

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

apply transition

compute sigma points

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

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

**Predict**

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

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

$$\mathbf{x}^{(0)} = \mu$$

$$\mathbf{x}^{(i)} = \mu + [\mathbf{L}]_i \text{ for } i \in [1..n]$$

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

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

# Prediction: Transition

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

apply transition

compute sigma points

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

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

**Predict**

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

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


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

weights propagated

# Prediction: Code

```
1 function [sigmaP, weightsM, weightsC] = prediction(mu, sigma,
    transition)
2 u = transition.v;
3 u_x = u(1); u_theta = u(3);
4
5 noise = 0.1; %constant part
6 v_noise = u_x^2; %lin vel dependent part
7 w_noise = u_theta^2; %ang vel dependent part
8 sigma_u = [ noise+v_noise, 0;
9            0, noise+v_noise ];
10
11 sigma_xu = zeros(5,5);
12 sigma_xu(1:3,1:3) = sigma;
13 sigma_xu(4:5,4:5) = sigma_u;
14
15 mu_xu = zeros(5,1);
16 mu_xu(1:3) = mu;
17 mu_xu(4:5) = [u_x; u_theta];
18
19 % extract the sigma points
20 [sigmaP_xu, weightsM, weightsC] = compute_sigma_points(mu_xu,
    sigma_xu);
21
22 for i=1:size(sigmaP_xu,1)
23     curr_sigma_mu = sigmaP_xu(1:3,i);
24     curr_input_u = [sigmaP_xu(4,i); 0; sigmaP_xu(5,i)];
25     sigmaP(:,end+1) = motion_model(curr_sigma_mu, curr_input_u);
26 end
27 endfunction
```

Apply transition  
to the sigma  
points



# Update: Predict Measurement

compute sigma points and mean of measurement

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

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

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

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

compute cross correlation of joint distribution

**1/2 Update**

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

$$\mu_{t|t-1} = \sum_i w_m^{(i)} \mathbf{x}_{t|t-1}^{(i)}$$

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

# Update: Chain Rule

compute sigma points and mean of measurement

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

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

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

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

compute cross correlation of joint distribution

**1/2 Update**

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

$$\Sigma_{x,z} = \begin{pmatrix} \Sigma_x & \Sigma_{x,z} \\ \Sigma_{z,x} & \Sigma_{z|x} + \Sigma_{z,z} \end{pmatrix}$$

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

covariance reconstruction

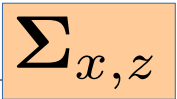
# Update 1: Code

```
1 function [mu, sigma] = correction(sigmaP, weightsM, weightsC,  
   landmarks, observations)  
2 %usual definition of state_dim and so on  
3 [ ... ]  
4 %define an iterator to work on sigmaZ_t  
5 id_x = 1;  
6 for i=1:num_landmarks_seen  
7     %retrieve info about the observed landmark  
8     measurement = observations.observation(i);  
9     z_t(end+1,:) = measurement.x_pose; % where we see the landmark  
10    z_t(end+1,:) = measurement.y_pose;  
11    current_land = searchById(landmarks, measurement.id);  
12    lx = current_land.x_pose;  
13    ly = current_land.y_pose;  
14  
15    %where I should see that landmark from each sigma point value  
16    for j=1:size(sigmaP,2)  
17        current_sigma_point = sigmaP(:,j);  
18        measure_prediction = measurement_function(current_sigma_point,  
19            [lx; ly]);  
20        sigmaZ_t(id_x:id_x+1,j) = measure_prediction;  
21    endfor  
22    id_x +=2;  
23 endfor
```



# Update 2: Code

```
1 %once we have computed our sigmaZ_t, we can reconstruct mu_z and
  relative covariace
2 [mu_z, Sigma_z] = reconstruct_mean_cov(sigmaZ_t, weightsM,
  weightsC);
3 % here we need to correct mu and sigma, so first of all we
  reconstruct mu and sigma from sigma points
4 [mu, sigma] = reconstruct_mean_cov(sigmaP, weightsM, weightsC);
5
6 %observation noise
7 noise = 0.01;
8 Sigma_noise = eye(2*num_landmarks_seen)*noise;
9
10 Sigma_xz = zeros(state_dim, 2*num_landmarks_seen);
11 for i=1:size(sigmaP,2)
12     delta_x= %TODO;
13     delta_z= %TODO;
14     Sigma_xz += %TODO;
15 end
```



# Update: Conditioning

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

**1/2 Update**

```
1 %Kalman gain
2 K = Sigma_xz * inv(Sigma_z + Sigma_noise);
3
4 %update mu
5 error = %TODO;
6 correction = %TODO;
7 mu = mu + correction;
8
9 %update sigma
10 sigma = %TODO;
11 endfunction
```

Compute the error and apply correction

All the ingredients are ready to update the covariance sigma