Probabilistic Robotics Course
Localization with Kalman Filters
[Example Application]

G. Grisetti, B. Della Corte

{grisetti,dellacorte}@dis.uniroma1.it

Dept of Computer Control and Management Engineering
Sapienza University of Rome

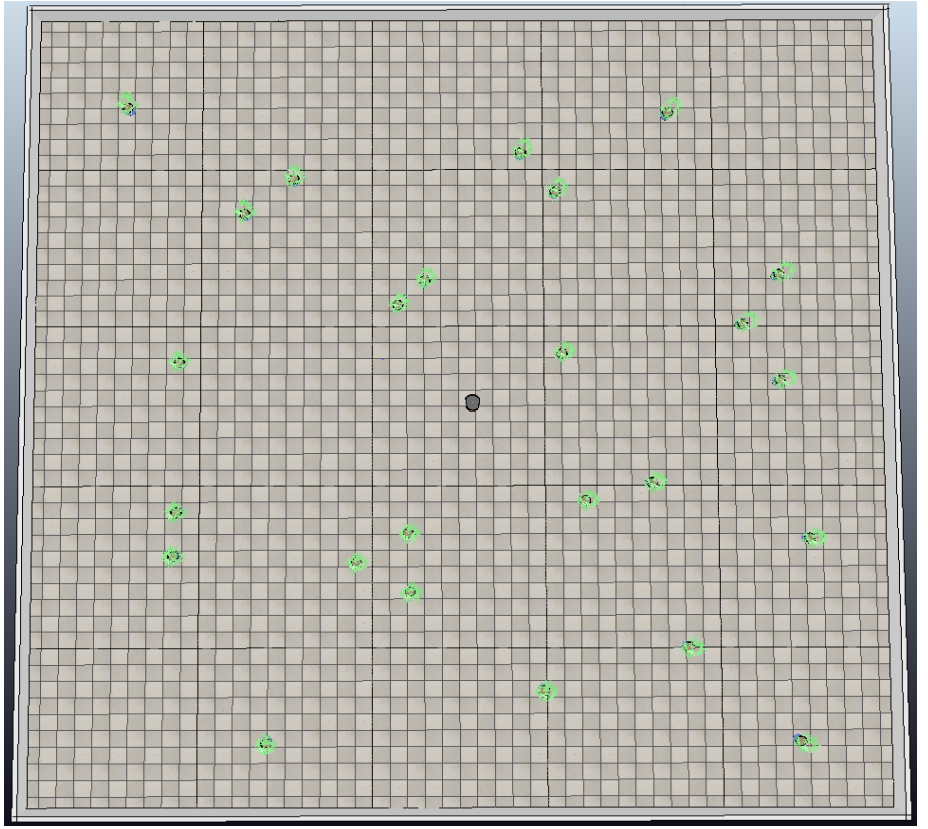
Outline

- Scenario
- Controls
- Observations
- Jacobians
- Implementation

Scenario

Orazio moves on a 2D plane

- It 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 KF based algorithm to track the position of Orazio as it moves

The inputs of our algorithms will be

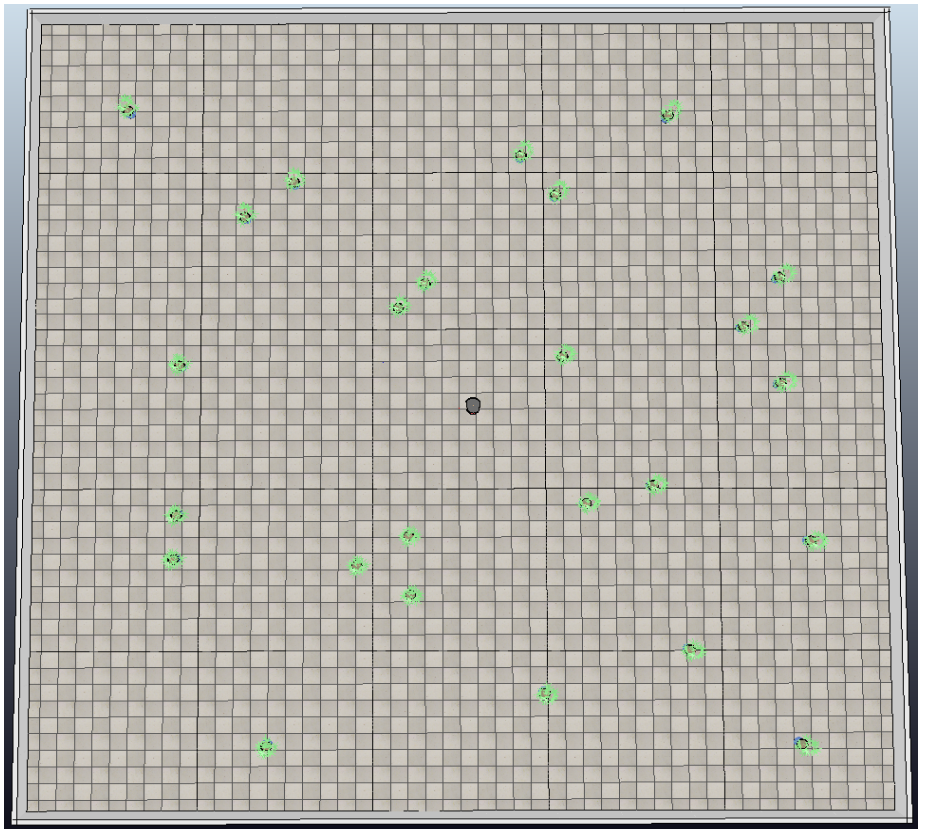
- velocity measurements
- landmark measurements

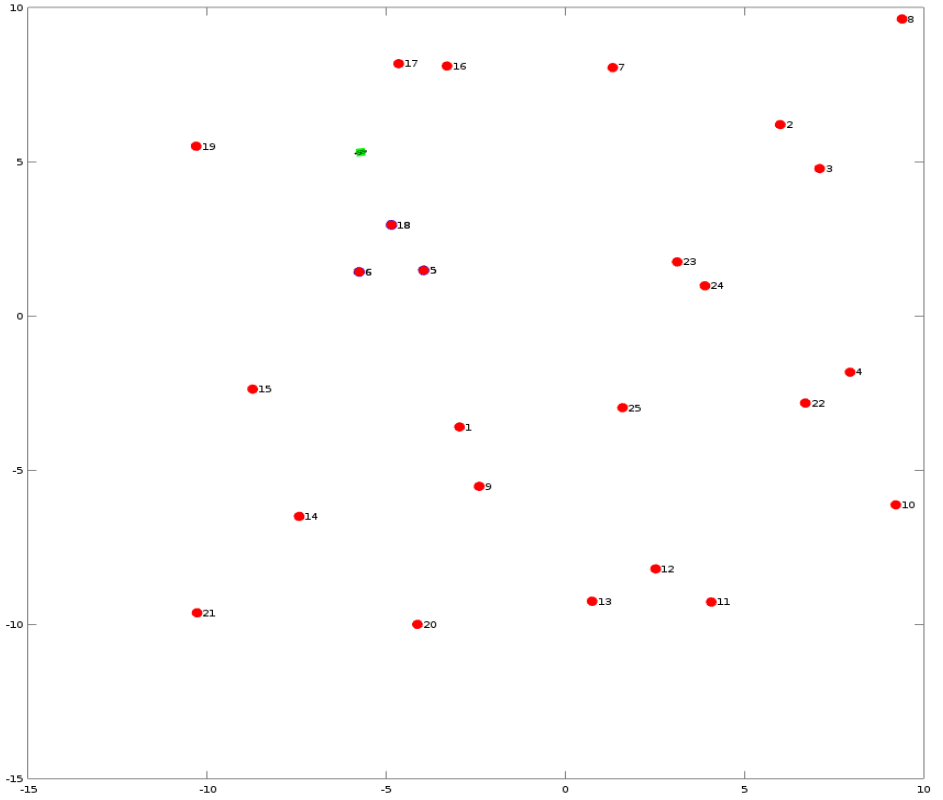
The prior knowledge about the map is represented by the location of each landmark in the world

Prior

The map is represented as a set of landmark coordinates

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





Domains

Define

- state space

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

Instead of considering rotational and translational velocities, we consider the integrated motion in the interval as input

This leads to a lighter notation

- space of controls (inputs)

$$\mathbf{u}_t = \begin{pmatrix} \Delta_t v_t \\ \Delta_t \omega_t \end{pmatrix} = \begin{pmatrix} u_t^1 \\ u_t^2 \end{pmatrix} \in \Re^2$$

- space of observations (measurements)

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

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

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

- space of controls (inputs)

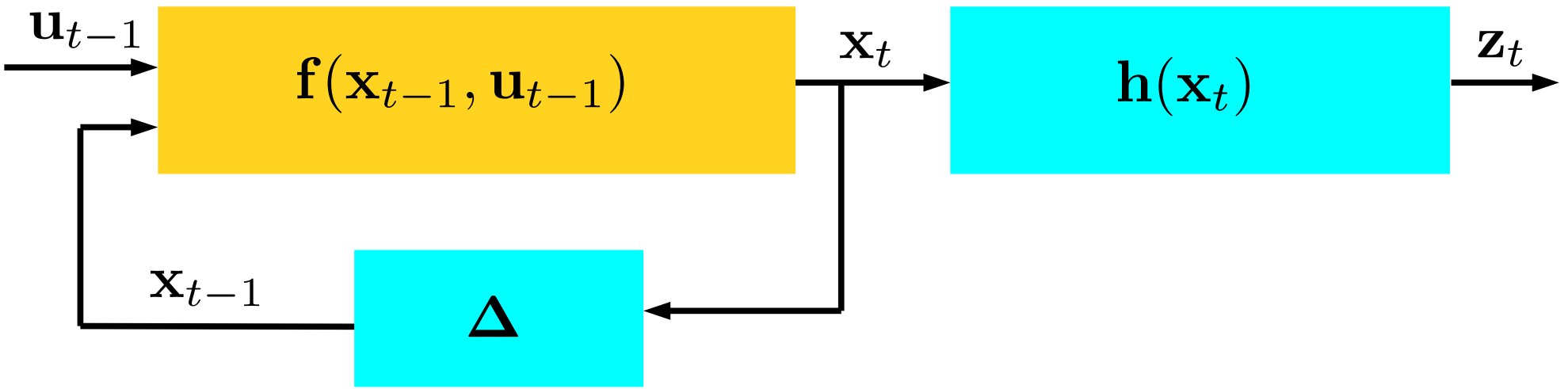
$$\mathbf{z}_t = \begin{pmatrix} x_t^{[i]} \\ y_t^{[i]} \end{pmatrix} \in \Re^2$$

- space of observations (measurements)

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

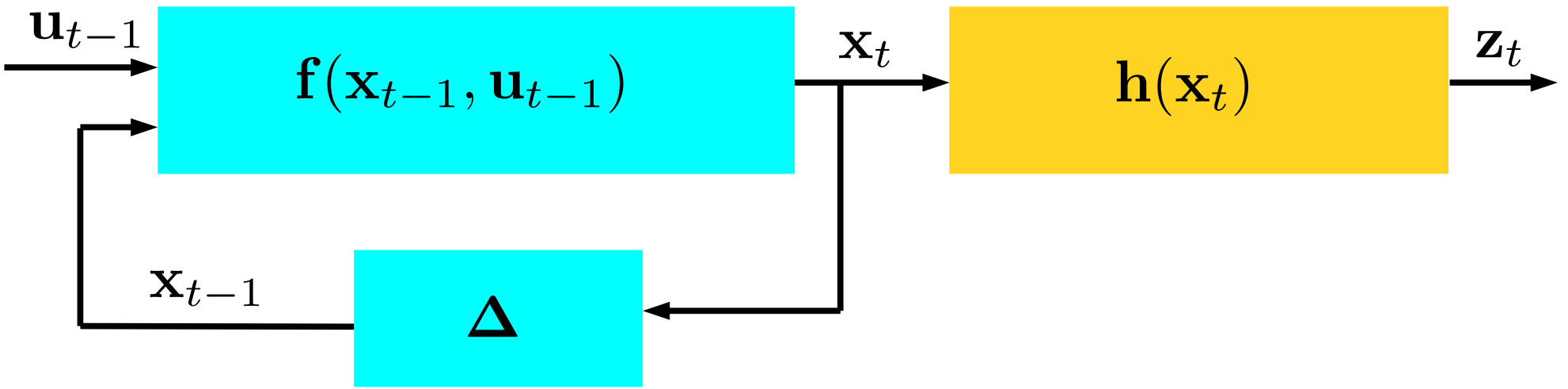
measurement and control, in this problem are already Euclidean

Transition Function



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

Measurement Function



We have [i] measurement functions, one per landmark

$$\mathbf{z}_t^{[i]} = \mathbf{h}^{[i]}(\mathbf{x}_t)$$

relative position of the I[th] landmark w.r.t the robot at time t

$$= \mathbf{R}_t^T(\mathbf{l}^{[i]} - \mathbf{t}_t)$$

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

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

rotation matrix of theta

Measurement Function

At each point in time, our robot will sense only a subset of *K* landmarks in the map

The measurement is thus consisting of a stack of measurements

$$\mathbf{z}_t = \begin{pmatrix} \mathbf{z}^{[i_1]} \\ \mathbf{z}^{[i_2]} \\ \dots \\ \mathbf{z}^{[i_K]} \end{pmatrix} = \mathbf{h}(\mathbf{x}_t) = \begin{pmatrix} \mathbf{h}^{[i_1]}(\mathbf{x}_t) \\ \mathbf{h}^{[i_2]}(\mathbf{x}_t) \\ \dots \\ \mathbf{h}^{[i_K]}(\mathbf{x}_t) \end{pmatrix}$$

index of the landmark generating the measurement

Control Noise

We assume the velocity measurements are affected 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

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

Jacobians!

At each time step our system will need to compute the derivatives of transition and measurement functions

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

$$\mathbf{A}_t = \frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{x}} = \begin{pmatrix} 1 & 0 & -u^1_{t-1} \sin(\theta_{t-1}) \\ 0 & 1 & u^1_{t-1} \cos(\theta_{t-1}) \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{B}_t = \frac{\partial \mathbf{f}(\cdot)}{\partial \mathbf{u}} = \begin{pmatrix} \cos(\theta_{t-1}) & 0 \\ \sin(\theta_{t-1}) & 0 \\ 0 & 1 \end{pmatrix}$$

Jacobians (cont)

We will have *n* measurement functions, one for each landmark

$$\mathbf{h}^{[i]}(\mathbf{x}_t) = \mathbf{R}_t^T(\mathbf{l}^{[i]} - \mathbf{t}_t)$$

this is a column vector!!!

$$\mathbf{C}_t^{[i]} = \frac{\partial \mathbf{h}^{[i]}(\cdot)}{\partial \mathbf{x}} = \begin{pmatrix} -\mathbf{R}_t^T & \frac{\partial \mathbf{R}_t^T}{\partial \theta_t}(\mathbf{l}^{[i]} - \mathbf{t}_t) \end{pmatrix}$$

derivative of rotation matrix w.r.t. theta

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

Jacobians (cont)

The total Jacobian of the measurement will be the stack of the individual measurement functions

$$\mathbf{C}_t = \frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial \mathbf{h}^{[i_1]}}{\partial \mathbf{x}} \\ \frac{\partial \mathbf{h}^{[i_2]}}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial \mathbf{h}^{[i_K]}}{\partial \mathbf{x}} \end{pmatrix} = \begin{pmatrix} \mathbf{C}_t^{[i_1]} \\ \mathbf{C}_t^{[i_2]} \\ \vdots \\ \mathbf{C}_t^{[i_K]} \end{pmatrix}$$

Hands on!

g2o Wrapper

Load your Vrep acquired dataset

```
[land, pose, transition, obs] = loadG2o('my_dataset.g2o');
```

It returns 4 Struct-Arrays(Landmark, Poses, Transitions, Observations), *i.e.* :

```
land =

  1x25 struct array containing the fields:

    id
    x_pose
    y_pose
```

```
pose =

  1x137 struct array containing the fields:

    id
    x
    y
    theta
```

```
transition =

  1x136 struct array containing the
fields:

    id_from
    id_to
    v
```

```
obs =

  1x136 struct array containing the fields:

    pose_id
    observation
```

EKF Localization

```
% load your own dataset dataset
[landmarks, poses, transitions, observations] = loadG2o('dataset.
    g2o');
mu = rand(3,1)*20-10; % init mean
mu(3) = normalizeAngle(mu(3));

sigma = eye(3)*0.001; % init covariance

%simulation cycle
for i=1:length(transitions)
     % predict with transitions
     [mu, sigma] = ekf_prediction(mu, sigma, transitions(i));
     % correct with observations
     [mu, sigma] = ekf_correction(mu, sigma, landmarks,
    observations(i));

     plot_state(landmarks, mu, sigma, observations(i));
endfor
```

EKF Localization

```
% load your own dataset dataset
[landmarks, poses, transitions, observations] = loadG2o('dataset.
    g2o');
mu = rand(3,1)*20-10; % init mean
mu(3) = normalizeAngle(mu(3));

sigma = eye(3)*0.001; % init covariance

%simulation cycle
for i=1:length(transitions)
     % predict with transitions TODO
     [mu, sigma] = ekf_prediction(mu, sigma, transitions(i));
     % correct with observations TODO
     [mu, sigma] = ekf_correction(mu, sigma, landmarks,
    observations(i));

     plot_state(landmarks, mu, sigma, observations(i));
endfor
```