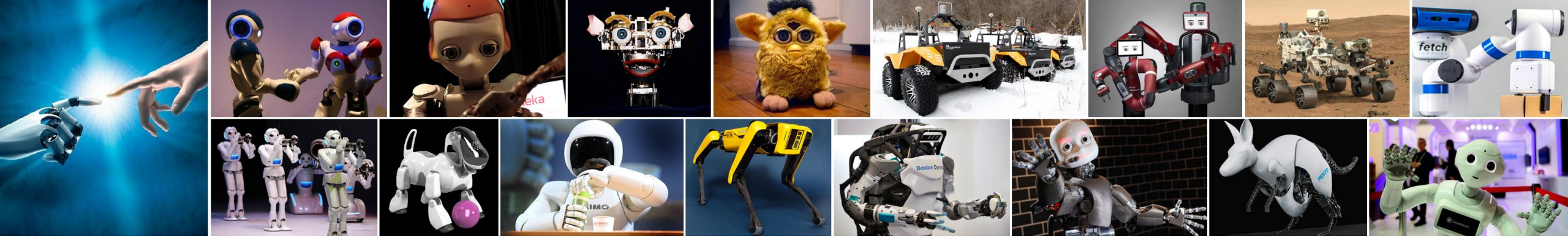
Problems with EKFs

- Taylor series expansion = poor approximation of non-linear functions, success of linearization depends on
  - Limited uncertainty and
  - Limited amount of local non-linearity
- (Manually) calculating partial derivatives is a pain
- Cannot handle multi-modal (multi-hypothesis) distributions
- What's next?
  - Unscented Kalman Filter (how to better generalize to non-linear models)
  - Non-Gaussian noise Kalman Filter (how to generalize the Kalman Filter when noise distribution is Non-Gaussian)
  - Stability and Divergence (how to design a stable KF that does not diverge)



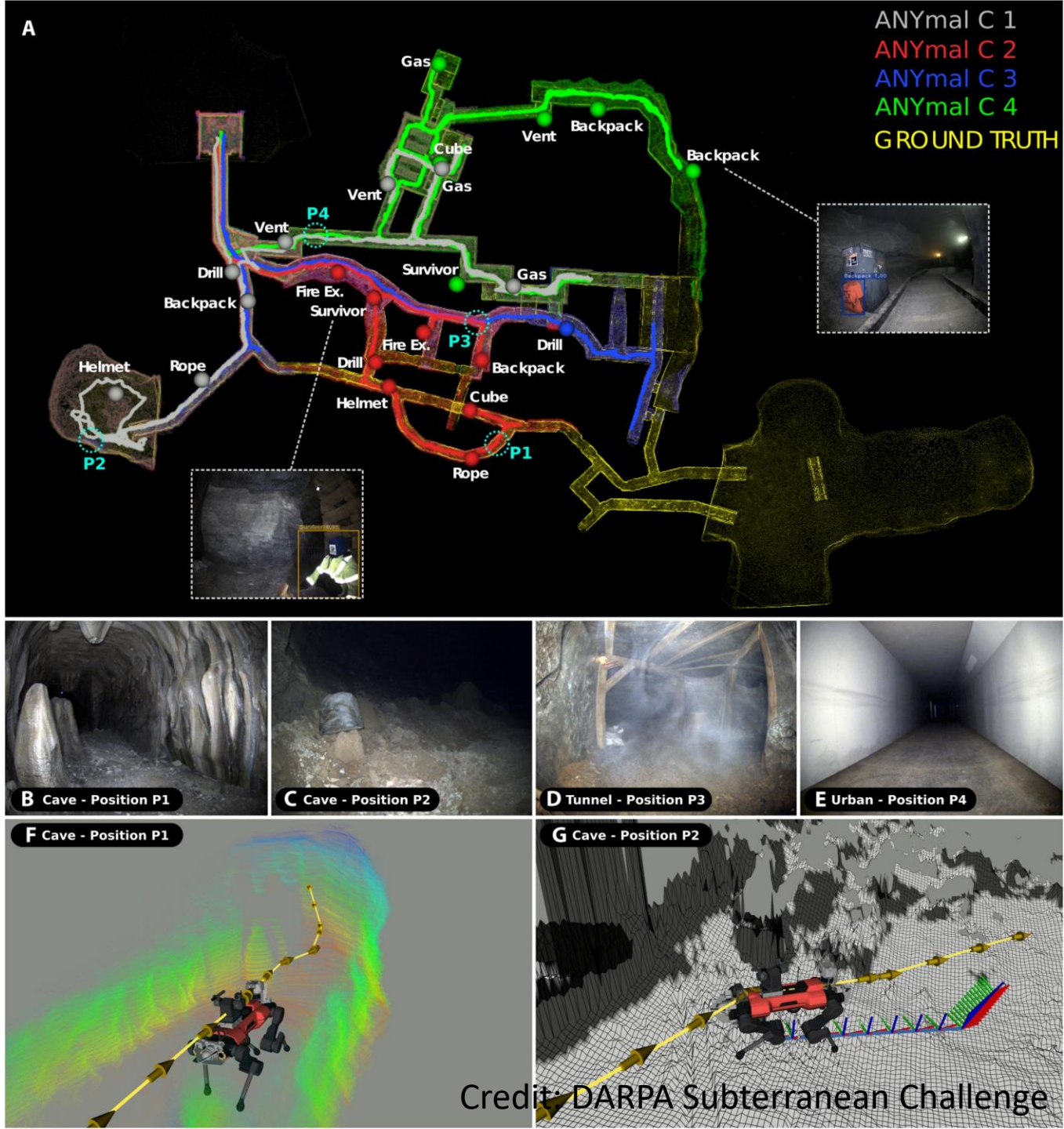
COMPSCI-603: Robotics

Simultaneous Localization and Mapping (SLAM)

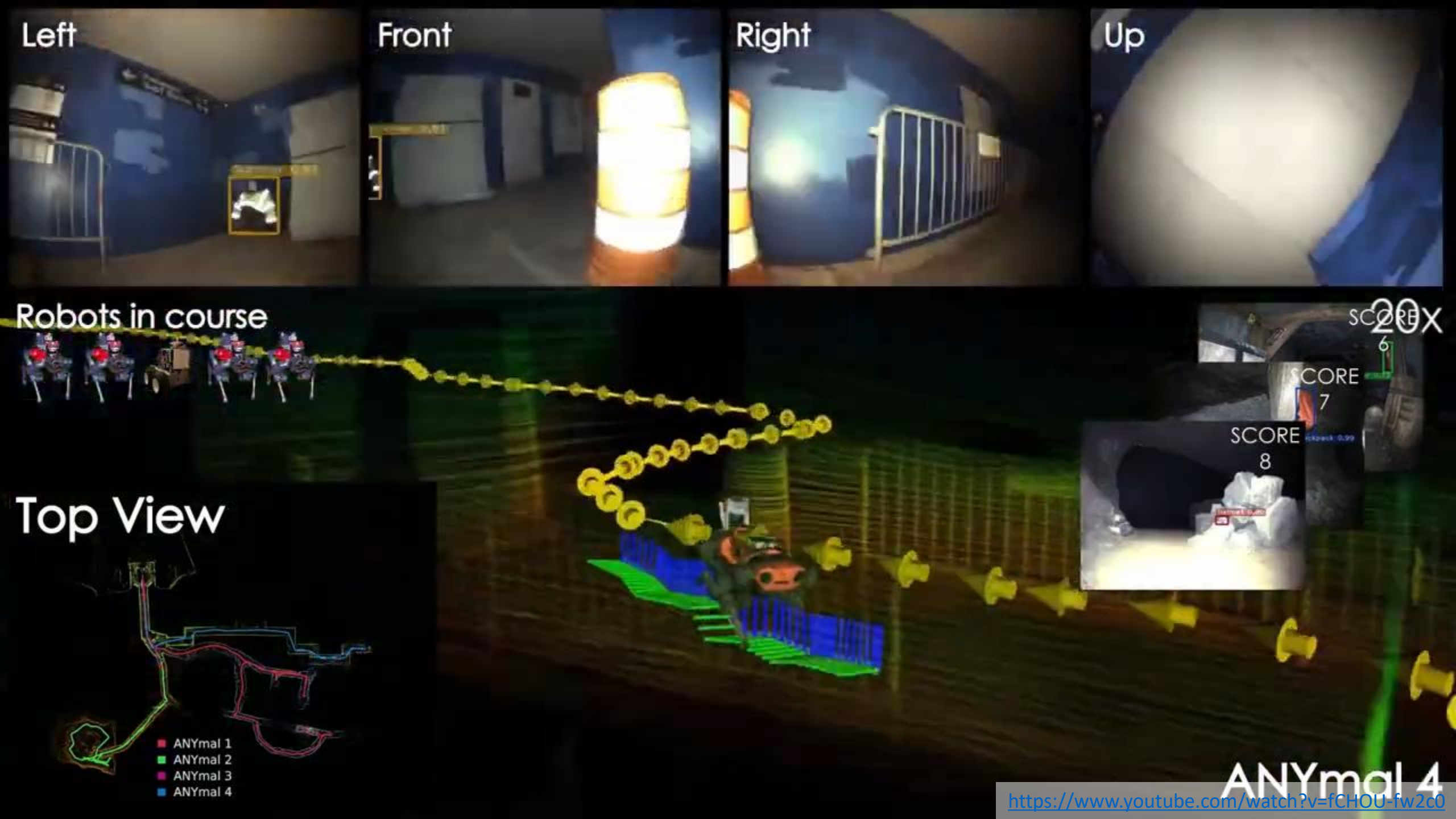
Partial slide courtesy by C. Stachniss

SLAM Overview

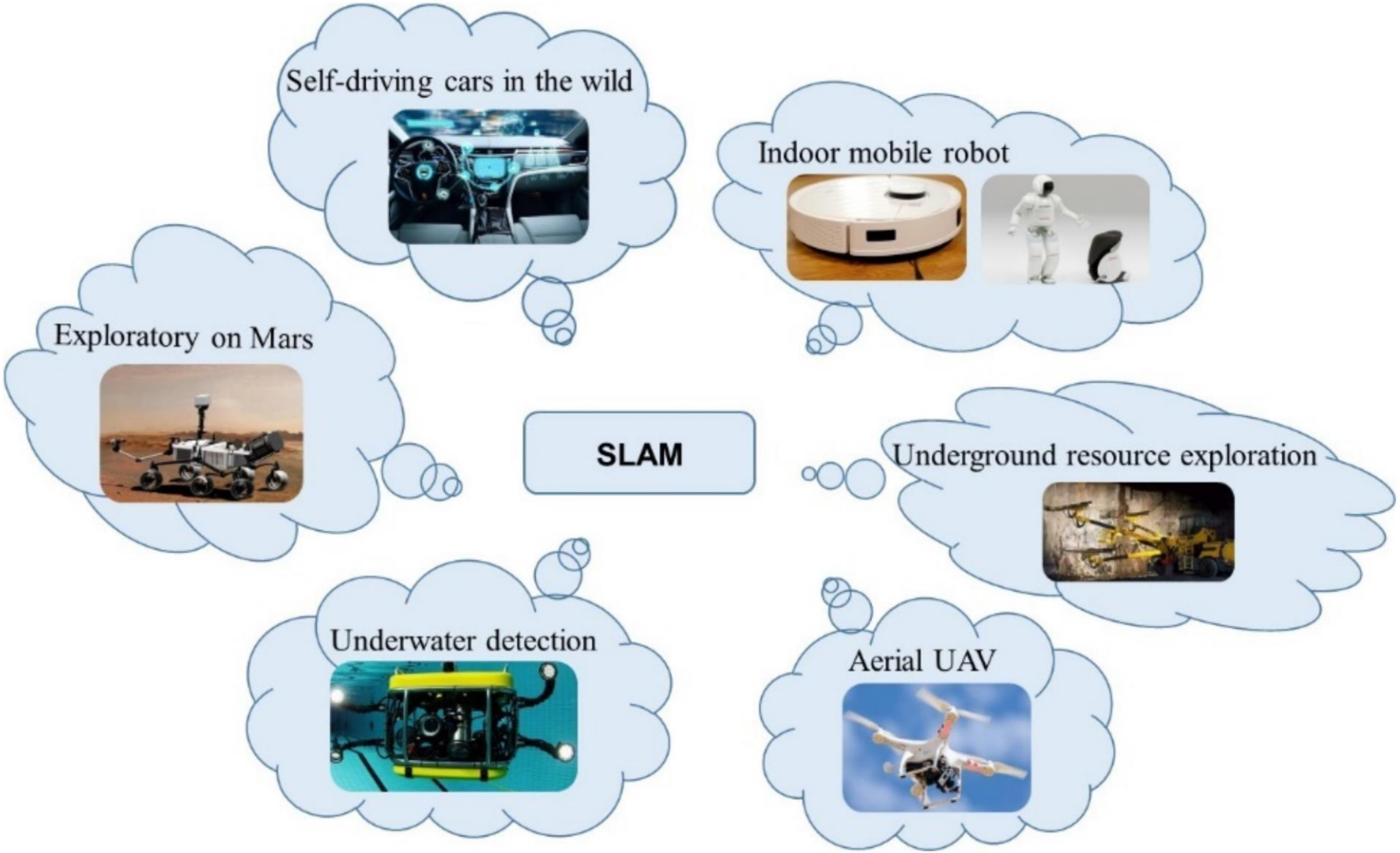
- Simultaneous Localization and Mapping (SLAM) builds a map of the environment from a mobile robot (or a mobile sensing platform).
- At the same time, SLAM localizes the mobile robot in the map build so far.
- SLAM is a chicken-and-egg problem.



Credit: DARPA Subterranean Challenge


Left
Front
Right
Up
Robots in course
20x
SCORE
6
SCORE
7
SCORE
8
Top View
ANYmal 1
ANYmal 2
ANYmal 3
ANYmal 4
ANYmal 4
https://www.youtube.com/watch?v=fCHOU-fw2c0

SLAM Applications



Guanwei Jia, Xiaoying Li, Dongming Zhang, Weiqing Xu, Haojie Lv, Yan Shi, and Maolin Cai. "Visual-SLAM Classical framework and key Techniques: a review." Sensors 22, no. 12 (2022): 4582.

Extended Kalman Filter (EKF)-based SLAM

Definition of the SLAM Problem

- Given
  - Controls

$$u_{1:T} = \{u_1, u_2, u_3, \ldots, u_T\}$$

  - Observations

$$z_{1:T} = \{z_1, z_2, z_3, \ldots, z_T\}$$

- Wanted
  - Map of the environment

$$m$$

  - Path (or current pose) of the robot

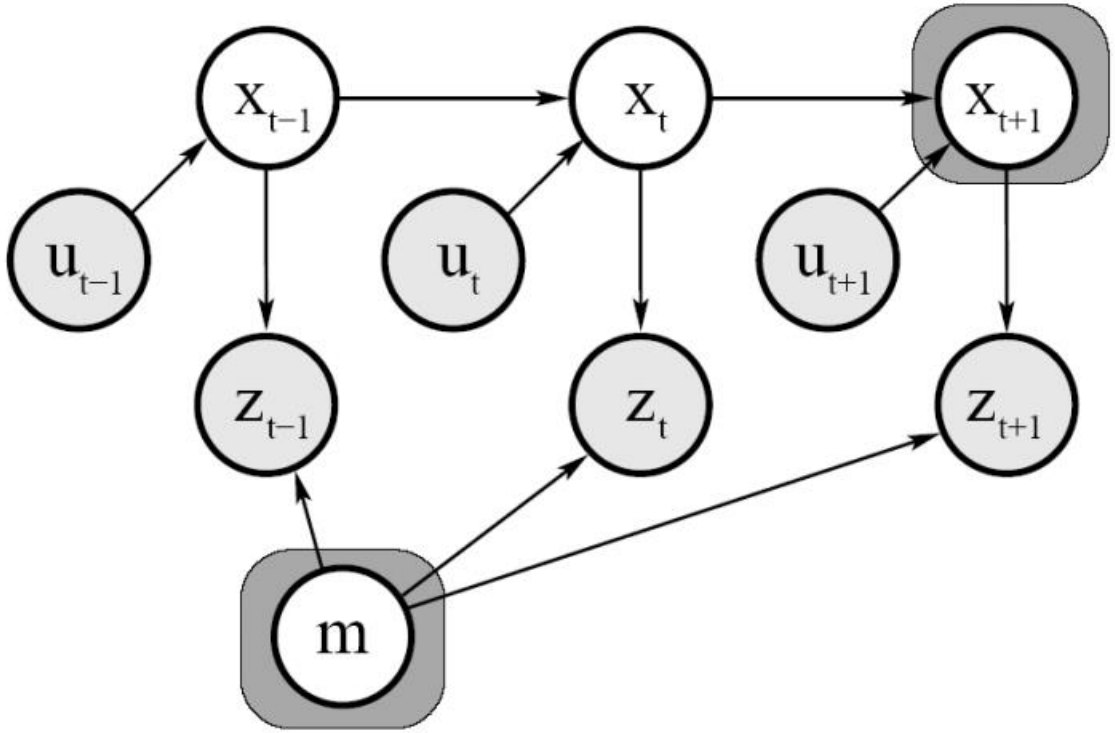
$$x_{0:T} = \{x_0, x_1, x_2, \ldots, x_T\} \quad \text{or} \quad x_t$$

EKF SLAM

- SLAM can be formulated under the Bayes filtering framework to estimate:

$$p(x_t, m \mid z_{1:t}, u_{1:t})$$

- Extended Kalman Filter (EKF) can be used to solve the SLAM problem.
  - Kalman Filter is a recursive Bayes Filter for the linear Gaussian case.
  - EKF for dealing with non-linearities.



Recall EKF

1: **Extended_Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$

3: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$

Estimation using nonlinear motion model g

4: $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$

5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6: $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$

7: $return \; \mu_t, \Sigma_t$

Correction using nonlinear sensor model h

EKF SLAM

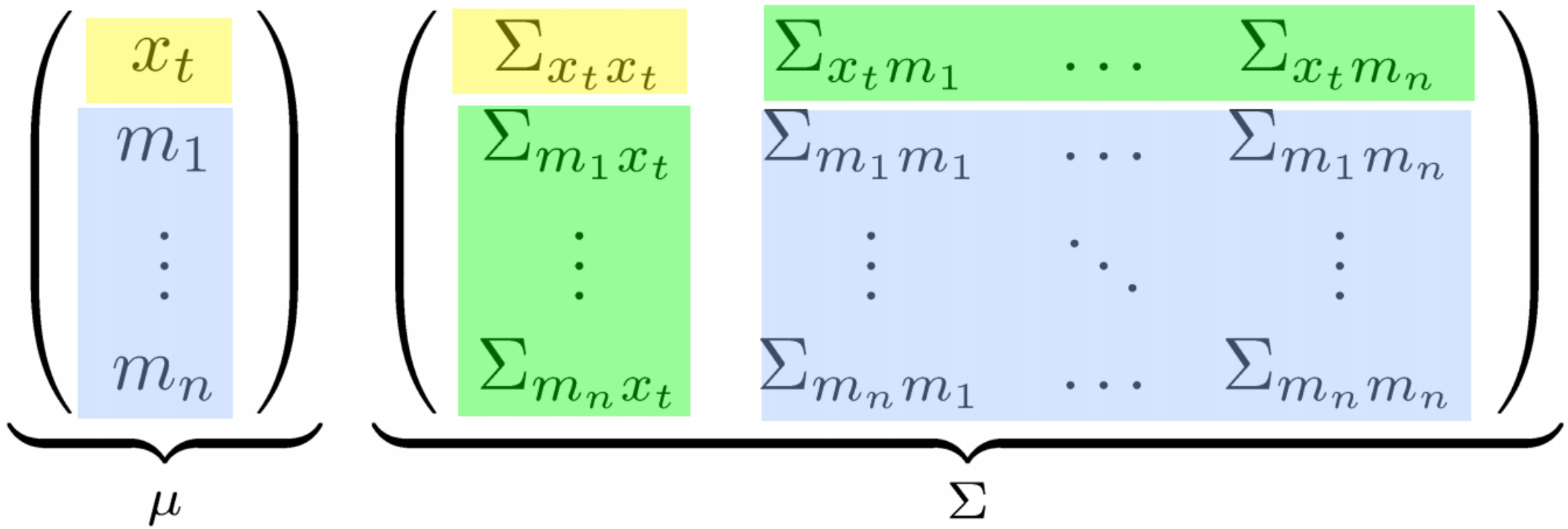
- EKF SLAM applies EKF to SLAM.
- EKF SLAM estimates robot's pose and locations of landmarks (e.g., points in the world) in the environment.
- State space (for the 2D plane) is:

$$x_t = (\underbrace{x, y, \theta}_{\text{robot's pose}}, \underbrace{m_{1,x}, m_{1,y}}_{\text{landmark 1}}, \ldots, \underbrace{m_{n,x}, m_{n,y}}_{\text{landmark n}})^T$$

  - Robot pose includes x, y, and θ.
  - Landmark locations includes x and y coordinates.
  - Assumption (for now): known landmark correspondences.

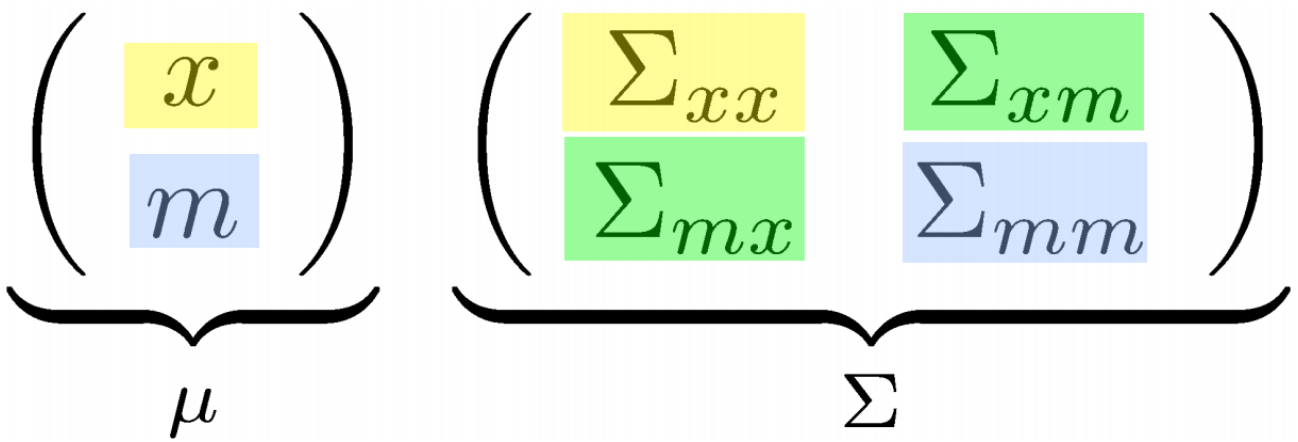
EKF SLAM

- Map with n landmarks: $(3 + 2n)$-dimensional Gaussian.
- Belief is represented by:



EKF SLAM Steps

1. State prediction
2. Landmark prediction
3. Measurement
4. Data association
5. Update

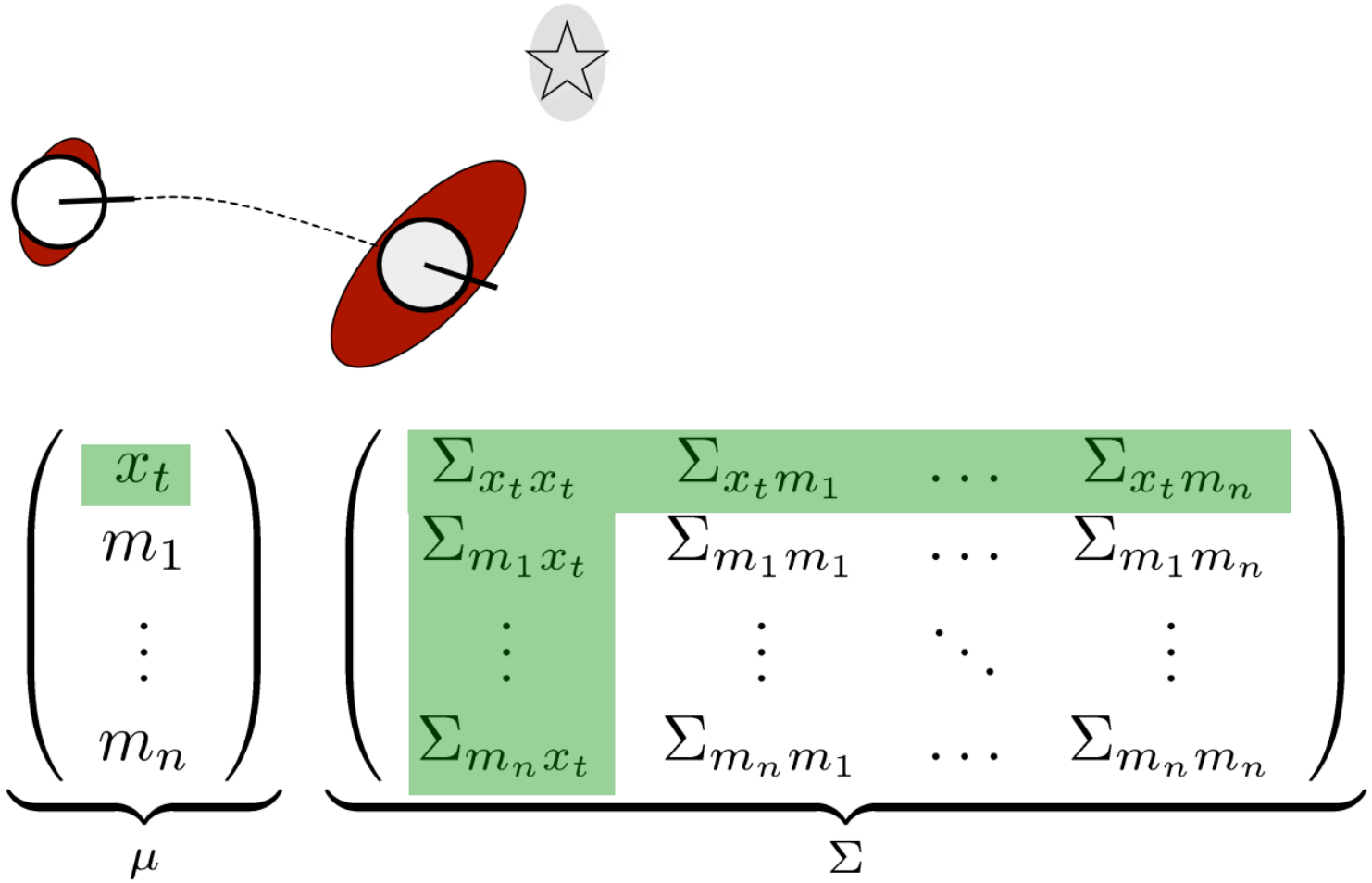


More compact math notation

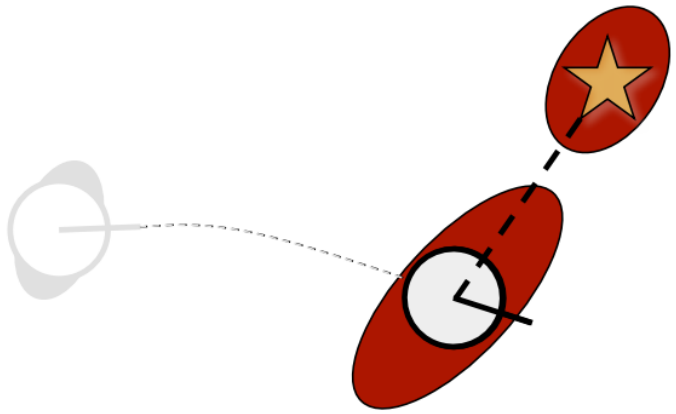
EKF SLAM Overview: Initialize State

$$\underbrace{\begin{pmatrix} x_t \\ m_1 \\ \vdots \\ m_n \end{pmatrix}}_{\mu} \quad \underbrace{\begin{pmatrix} \Sigma_{x_t x_t} & \Sigma_{x_t m_1} & \cdots & \Sigma_{x_t m_n} \\ \Sigma_{m_1 x_t} & \Sigma_{m_1 m_1} & \cdots & \Sigma_{m_1 m_n} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{m_n x_t} & \Sigma_{m_n m_1} & \cdots & \Sigma_{m_n m_n} \end{pmatrix}}_{\Sigma}$$

EKF SLAM Overview: Predict State

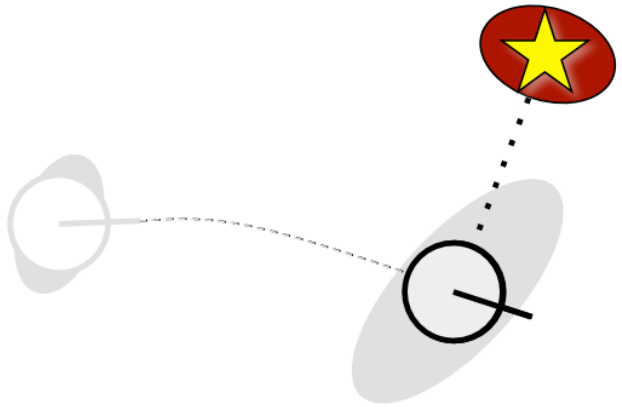


EKF SLAM Overview: Predict Landmark Locations



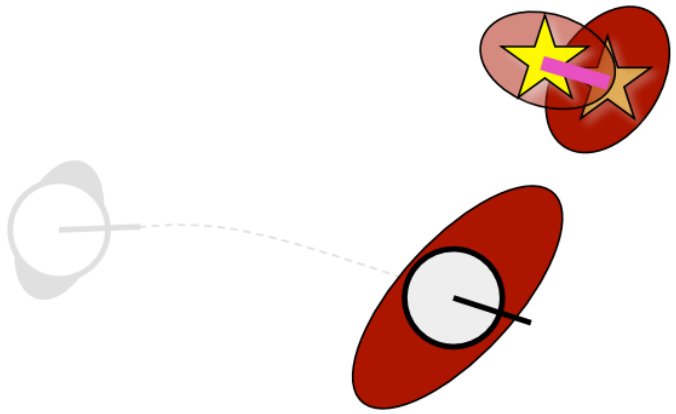
$$\underbrace{\begin{pmatrix} x_t \\ m_1 \\ \vdots \\ m_n \end{pmatrix}}_{\mu} \quad \underbrace{\begin{pmatrix} \Sigma_{x_t x_t} & \Sigma_{x_t m_1} & \cdots & \Sigma_{x_t m_n} \\ \Sigma_{m_1 x_t} & \Sigma_{m_1 m_1} & \cdots & \Sigma_{m_1 m_n} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{m_n x_t} & \Sigma_{m_n m_1} & \cdots & \Sigma_{m_n m_n} \end{pmatrix}}_{\Sigma}$$

EKF SLAM Overview: Obtain Measurement



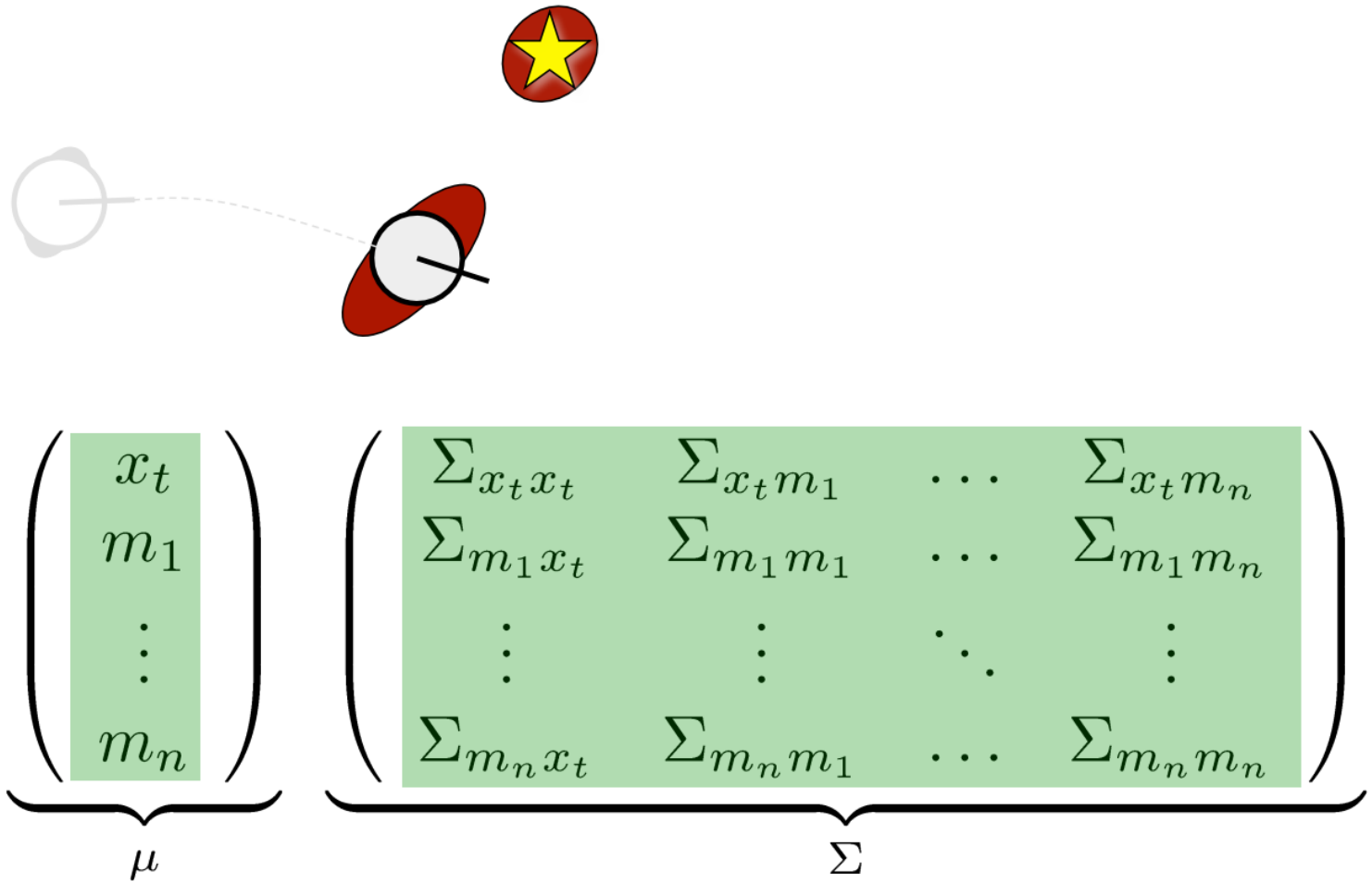
$$\underbrace{\begin{pmatrix} x_t \\ m_1 \\ \vdots \\ m_n \end{pmatrix}}_{\mu} \quad \underbrace{\begin{pmatrix} \Sigma_{x_t x_t} & \Sigma_{x_t m_1} & \cdots & \Sigma_{x_t m_n} \\ \Sigma_{m_1 x_t} & \Sigma_{m_1 m_1} & \cdots & \Sigma_{m_1 m_n} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{m_n x_t} & \Sigma_{m_n m_1} & \cdots & \Sigma_{m_n m_n} \end{pmatrix}}_{\Sigma}$$

EKF SLAM Overview: Perform Data Association



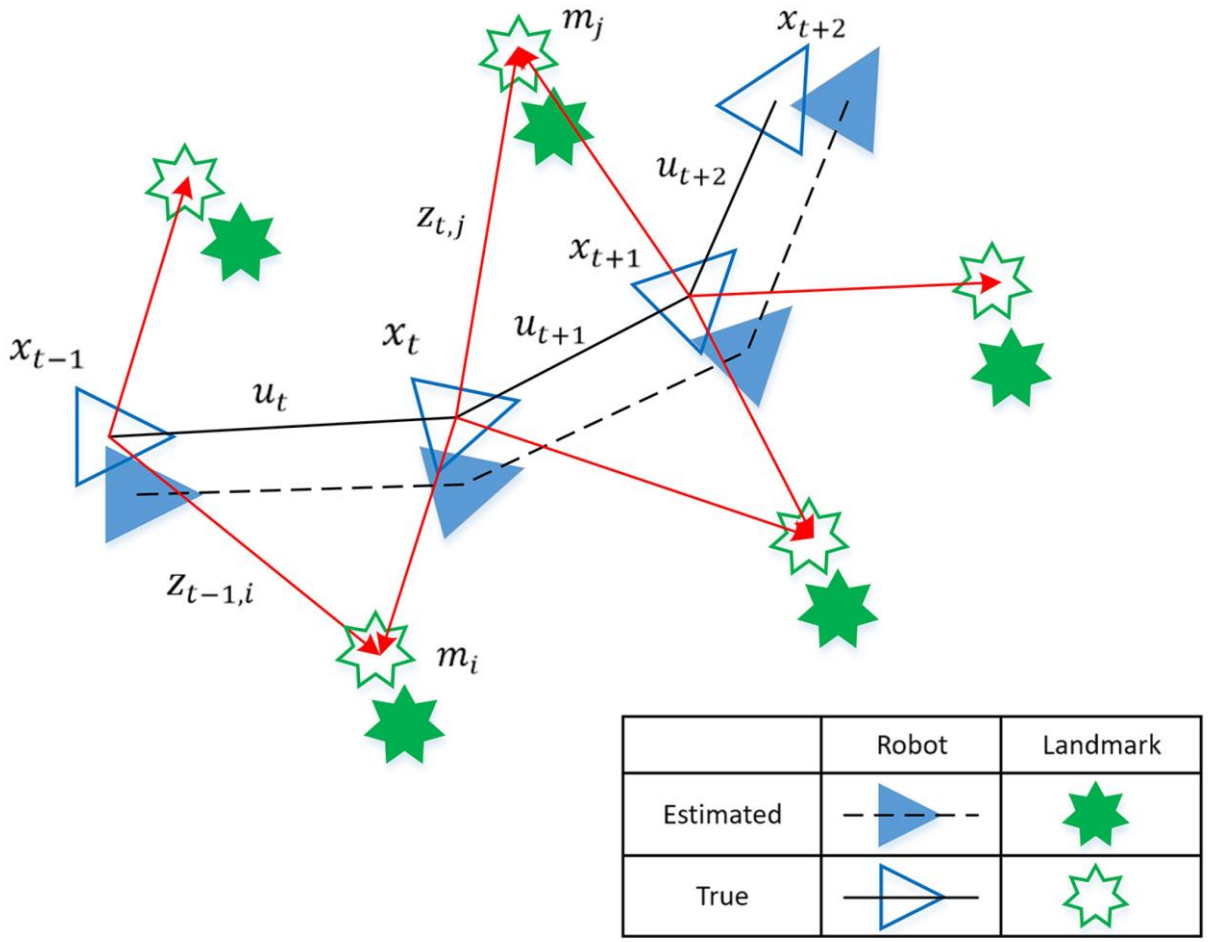
$$\underbrace{\begin{pmatrix} x_t \\ m_1 \\ \vdots \\ m_n \end{pmatrix}}_{\mu} \quad \underbrace{\begin{pmatrix} \Sigma_{x_t x_t} & \Sigma_{x_t m_1} & \cdots & \Sigma_{x_t m_n} \\ \Sigma_{m_1 x_t} & \Sigma_{m_1 m_1} & \cdots & \Sigma_{m_1 m_n} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{m_n x_t} & \Sigma_{m_n m_1} & \cdots & \Sigma_{m_n m_n} \end{pmatrix}}_{\Sigma}$$

EKF SLAM Overview: Update State

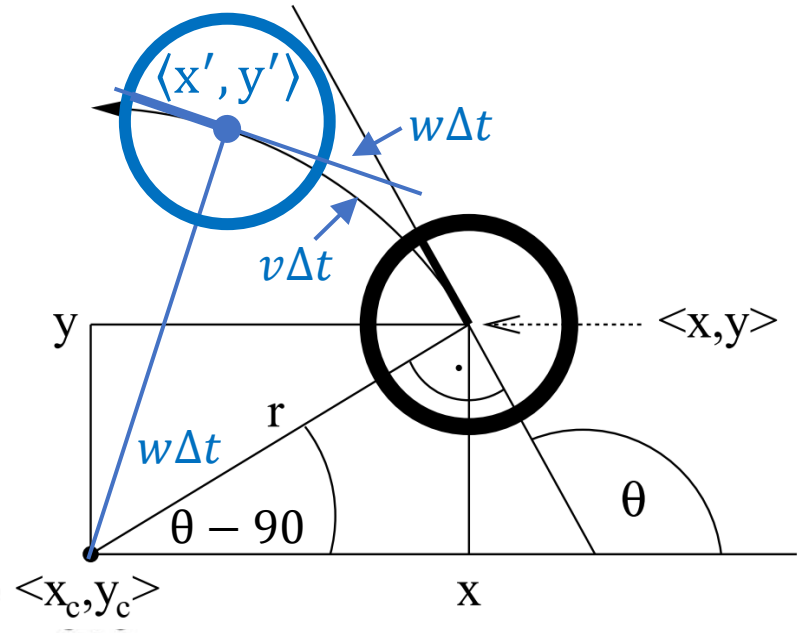


EKF SLAM

- Setup
  - Moves in a 2D space
  - Observation of point landmarks
  - Known number of landmarks
  - Known data association
  - Range-bearing sensor
  - Velocity-based motion model



Velocity-Based Motion Model



control $u = \begin{bmatrix} v \\ \omega \end{bmatrix}$

- Arc length:

$$l = v \cdot \Delta t = r \cdot w\Delta t$$

- Rotation radius:

$$\mathrm{r} = \left|\frac{v}{w}\right|$$

- Circle center:

$$\begin{aligned} \mathrm{x_c} &= \mathrm{x} - \mathrm{r} \cdot \cos(\theta - 90) \\ &= \mathrm{x} - \mathrm{r} \cdot \sin\theta \end{aligned}$$

$$\begin{aligned} \mathrm{y_c} &= \mathrm{y} - \mathrm{r} \cdot \sin(\theta - 90) \\ &= \mathrm{x} + \mathrm{r} \cdot \cos\theta \end{aligned}$$

Velocity-Based Motion Model

- New pose after Δt:

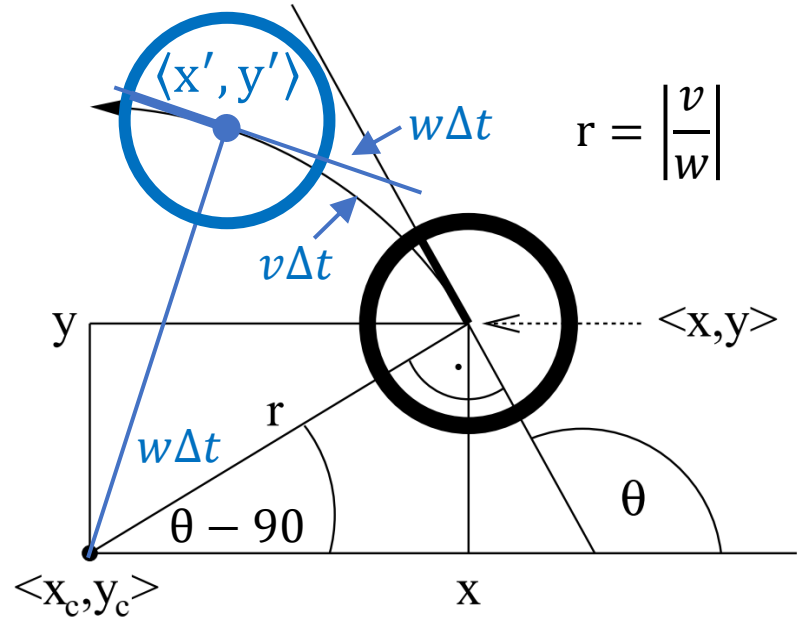
$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x_c + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ y_c - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \theta + \omega\Delta t \end{pmatrix}$$

$$= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega}\sin\theta + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ \frac{v}{\omega}\cos\theta - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \omega\Delta t \end{pmatrix}$$

x_c

$$x' = x_c + r \cdot \cos(\theta - 90 + w\Delta t)$$
$$= x_c + r \cdot \sin(\theta + w\Delta t)$$

$$y' = y_c + r \cdot \sin(\theta - 90 + w\Delta t)$$
$$= y_c - r \cdot \cos(\theta + w\Delta t)$$



EKF for SLAM: Predict State Using Motion

- Goal: Update state space based on the motion.
- Velocity-based motion model in the 2D plane:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \underbrace{\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega_t \, \Delta t \end{pmatrix}}_{g_{x,y,\theta}(u_t,(x,y,\theta)^T)}$$

- How to map this motion model in 3D space to the $(3 + 2N)$-dimensional state space in the EKF-SLAM?

EKF for SLAM: Initialize State

- Platform starts in its own reference frame (all landmarks unknown).
- State has $(3 + 2N)$ dimensions:

$$\mu_0 = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \end{pmatrix}^T$$

$$\Sigma_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \infty & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \infty \end{pmatrix}$$

EKF for SLAM

1: **Extended_Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$

3: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$

4: $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$

5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6: $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$

7: $return \; \mu_t, \Sigma_t$

EKF for SLAM: Update State

- From the motion in the plane:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta+\omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta+\omega_t\Delta t) \\ \omega_t\,\Delta t \end{pmatrix}$$

- To the $(3+2N)$-dimensional state space:

$$\begin{pmatrix} x' \\ y' \\ \theta' \\ \vdots \end{pmatrix} = \underbrace{\begin{pmatrix} x \\ y \\ \theta \\ \vdots \end{pmatrix} + \underbrace{\begin{pmatrix} 1 & 0 & 0 & \underbrace{0\ldots0}_{2N cols} \\ 0 & 1 & 0 & 0\ldots0 \\ 0 & 0 & 1 & 0\ldots0 \\ & & & \end{pmatrix}^T}_{F_x^T} \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta+\omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta+\omega_t\Delta t) \\ \omega_t\,\Delta t \end{pmatrix}}_{g(u_t,x_t)}$$

F_x : Projection function that projects variables from 3D space to the $3+2N$ space

EKF for SLAM

1: **Extended_Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: ~~$\bar{\mu}_t = g(u_t, \mu_{t-1})$~~ Done

3: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$

4: $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$

5: $\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$

6: $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$

7: $return \; \mu_t, \Sigma_t$

EKF for SLAM: Update Covariance

- The motion model only affects the motion of the robot, but NOT the landmarks.
- Representing the motion model's Jacobian in the $(3 + 2N)$ space:

Jacobian of the motion (3x3)

$$G_t = \begin{pmatrix} G_t^x & 0 \\ 0 & I \end{pmatrix}$$

Identity (2N x 2N)

Jacobian of Motion Model

$$G_t^x = \frac{\partial}{\partial (x, y, \theta)^T} \left[\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t} \sin\theta + \frac{v_t}{\omega_t} \sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos\theta - \frac{v_t}{\omega_t} \cos(\theta + \omega_t \Delta t) \\ \omega_t \, \Delta t \end{pmatrix} \right]$$

Jacobian of Motion Model

$$
\begin{aligned}
G_t^x &= \frac{\partial}{\partial (x, y, \theta)^T} \left[\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t} \sin\theta + \frac{v_t}{\omega_t} \sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos\theta - \frac{v_t}{\omega_t} \cos(\theta + \omega_t \Delta t) \\ \omega_t \, \Delta t \end{pmatrix} \right] \\
&= I + \frac{\partial}{\partial (x, y, \theta)^T} \begin{pmatrix} -\frac{v_t}{\omega_t} \sin\theta + \frac{v_t}{\omega_t} \sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos\theta - \frac{v_t}{\omega_t} \cos(\theta + \omega_t \Delta t) \\ \omega_t \, \Delta t \end{pmatrix}
\end{aligned}
$$

Jacobian of Motion Model

$$
\begin{aligned}
G_t^x &= \frac{\partial}{\partial (x, y, \theta)^T} \left[\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t} \sin\theta + \frac{v_t}{\omega_t} \sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos\theta - \frac{v_t}{\omega_t} \cos(\theta + \omega_t \Delta t) \\ \omega_t \, \Delta t \end{pmatrix} \right] \\
&= I + \frac{\partial}{\partial (x, y, \theta)^T} \begin{pmatrix} -\frac{v_t}{\omega_t} \sin\theta + \frac{v_t}{\omega_t} \sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos\theta - \frac{v_t}{\omega_t} \cos(\theta + \omega_t \Delta t) \\ \omega_t \, \Delta t \end{pmatrix} \\
&= I + \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t} \cos\theta + \frac{v_t}{\omega_t} \cos(\theta + \omega_t \Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t} \sin\theta + \frac{v_t}{\omega_t} \sin(\theta + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix}
\end{aligned}
$$

$\partial x \quad \partial y \qquad\qquad \partial \theta$

Jacobian of Motion Model

$$
\begin{aligned}
G_t^x &= \frac{\partial}{\partial (x,y,\theta)^T} \left[\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta+\omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta+\omega_t\Delta t) \\ \omega_t\,\Delta t \end{pmatrix} \right] \\
&= I + \frac{\partial}{\partial (x,y,\theta)^T} \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta+\omega_t\Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta+\omega_t\Delta t) \\ \omega_t\,\Delta t \end{pmatrix} \\
&= I + \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t}\cos\theta + \frac{v_t}{\omega_t}\cos(\theta+\omega_t\Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta+\omega_t\Delta t) \\ 0 & 0 & 0 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 & -\frac{v_t}{\omega_t}\cos\theta + \frac{v_t}{\omega_t}\cos(\theta+\omega_t\Delta t) \\ 0 & 1 & -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta+\omega_t\Delta t) \\ 0 & 0 & 1 \end{pmatrix}
\end{aligned}
$$

EKF for SLAM

1: **Extended_Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: ~~$\bar{\mu}_t = g(u_t, \mu_{t-1})$~~ Done

3: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$

$$\begin{aligned}
\bar{\Sigma}_t &= G_t \Sigma_{t-1} G_t^T + R_t \\
&= \begin{pmatrix} G_t^x & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \Sigma_{xx} & \Sigma_{xm} \\ \Sigma_{mx} & \Sigma_{mm} \end{pmatrix} \begin{pmatrix} (G_t^x)^T & 0 \\ 0 & I \end{pmatrix} + R_t \\
&= \begin{pmatrix} G_t^x \Sigma_{xx} (G_t^x)^T & G_t^x \Sigma_{xm} \\ (G_t^x \Sigma_{xm})^T & \Sigma_{mm} \end{pmatrix} + R_t
\end{aligned}$$

Summary of Estimation in EKF for SLAM

EKF_SLAM_Prediction$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t, R_t)$**:**

2: $F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$

3 x $(3 + 2N)$

Projection function that projects variables from 3D space to the $3 + 2N$ space

3: $\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$

$(3 + 2N)$ x 1

4: $G_t = I + F_x^T \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$

$(3 + 2N)$ x $(3 + 2N)$

5: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + \underbrace{F_x^T \, R_t^x \, F_x}_{R_t}$

$(3 + 2N)$ x $(3 + 2N)$

EKF for SLAM

1: **Extended_Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: ~~$\bar{\mu}_t = g(u_t, \mu_{t-1})$~~ Done

3: ~~$\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$~~ Done

4: $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$

5: $\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$

6: $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$

7: $return \; \mu_t, \Sigma_t$

EKF SLAM: Correction in EKF for SLAM

- Assume know data association on landmarks:
  - $c_t^i = j$: i-th measurement at time t observes the landmark with index j.

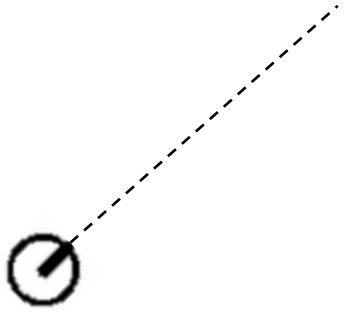
- If a landmark is observed for the first time, initialize the landmark.

- Compute the expected observation based on sensor model
- Calculate the Jacobian H of a sensor model h.
- Proceed with computing the Kalman gain.

Range-Bearing Sensor Model

- We use range-bearing observations, for observation beam i:

$$z_t^i = (r_t^i, \phi_t^i)^T$$



- If a landmark has not been observed, we can initialize it with:

$$\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} r_t^i \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \end{pmatrix}$$

location of landmark j | estimated location of the robot | relative measurement from beam i

Range-Bearing Sensor Model

- Convert between 2D location and range-bearing observation for landmark *j*:

$$
\begin{aligned}
\delta &= \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix} \\
q &= \delta^T \delta \\
\hat{z}_t^i &= \begin{pmatrix} \sqrt{q} \\ \operatorname{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \end{pmatrix} \\
&= h(\bar{\mu}_t)
\end{aligned}
$$

- Compute the Jacobian:

$$
{}^{\text{low}}H_t^i = \frac{\partial h(\bar{\mu}_t)}{\partial \bar{\mu}_t}
$$

low-dim space $(x, y, \theta, m_{j,x}, m_{j,y})$

Range-Bearing Sensor Model

$$
\begin{aligned}
\delta &= \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix} \\
q &= \delta^T \delta \\
\hat{z}_t^i &= \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \end{pmatrix} \\
&= h(\bar{\mu}_t)
\end{aligned}
$$

- Jacobian:

$$
\begin{aligned}
{}^{\text{low}}H_t^i &= \frac{\partial h(\bar{\mu}_t)}{\partial \bar{\mu}_t} \\
&= \begin{pmatrix} \frac{\partial \sqrt{q}}{\partial x} & \frac{\partial \sqrt{q}}{\partial y} & \cdots \\ \frac{\partial \text{atan2}(\ldots)}{\partial x} & \frac{\partial \text{atan2}(\ldots)}{\partial y} & \cdots \end{pmatrix} \\
&= \frac{1}{q} \begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & +\sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{pmatrix}
\end{aligned}
$$

low-dim space

$$(x, y, \theta, m_{j,x}, m_{j,y})$$

- E.g.,: $\frac{\partial \sqrt{q}}{\partial x} = \frac{1}{2} \frac{1}{\sqrt{q}} 2\,\delta_x\,(-1) = \frac{1}{q}\,(-\sqrt{q}\delta_x)$

Range-Bearing Sensor Model

- Project the Jacobian to the $(3 + 2N)$-dimensional space

$$
{}^{\text{low}}H_t^i = \frac{1}{q}\begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & +\sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{pmatrix}
$$

2 x 5

$$
H_t^i = {}^{\text{low}}H_t^i \, F_{x,j} \qquad F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0\cdots0 & 0 & 0 & 0\cdots0 \\ 0 & 1 & 0 & 0\cdots0 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 1 & 0\cdots0 & 0 & 0 & 0\cdots0 \\ 0 & 0 & 0 & 0\cdots0 & 1 & 0 & 0\cdots0 \\ 0 & 0 & 0 & \underbrace{0\cdots0}_{2j-2} & 0 & 1 & \underbrace{0\cdots0}_{2N-2j} \end{pmatrix}
$$

5 x $(3 + 2N)$

landmark j

EKF for SLAM

1: **Extended_Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: ~~$\bar{\mu}_t = g(u_t, \mu_{t-1})$~~ Done

3: ~~$\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$~~ Done

4: $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$

5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6: $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$

7: $return \; \mu_t, \Sigma_t$

EKF SLAM: Correction (1/2)

EKF_SLAM_Correction

6: $Q_t = \begin{pmatrix} {\sigma_r}^2 & 0 \\ 0 & {\sigma_\phi}^2 \end{pmatrix}$

7: *for all observed features* $z_t^i = (r_t^i, \phi_t^i)^T$ *do*

8: $j = c_t^i$

9: *if landmark* j *never seen before*

10: $\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} r_t^i \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \end{pmatrix}$

11: *endif*

12: $\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix}$

13: $q = \delta^T \delta$

14: $\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \end{pmatrix}$

EKF SLAM: Correction (2/2)

15: $F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & \underbrace{0 \cdots 0}_{2j-2} & 0 & 1 & \underbrace{0 \cdots 0}_{2N-2j} \end{pmatrix}$

16: $H_t^i = \frac{1}{q} \begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & +\sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & +\delta_x \end{pmatrix} F_{x,j}$

17: $K_t^i = \bar{\Sigma}_t \, H_t^{iT} (H_t^i \, \bar{\Sigma}_t \, H_t^{iT} + Q_t)^{-1}$

18: $\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i)$

19: $\bar{\Sigma}_t = (I - K_t^i \, H_t^i) \, \bar{\Sigma}_t$

20: *endfor*

21: $\mu_t = \bar{\mu}_t$

22: $\Sigma_t = \bar{\Sigma}_t$

23: *return* μ_t, Σ_t

EKF for SLAM

1: **Extended_Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: ~~$\bar{\mu}_t = g(u_t, \mu_{t-1})$~~ Done

3: ~~$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$~~ Done

4: ~~$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$~~ Done

5: ~~$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$~~ Done

6: ~~$\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$~~ Done

7: *return* μ_t, Σ_t

EKF SLAM Algorithm

EKF_SLAM_Prediction$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, c_t, R_t)$**:**

2: $F_x = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 \end{pmatrix}$

3: $\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix} -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} - \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega_t \Delta t \end{pmatrix}$

4: $G_t = I + F_x^T \begin{pmatrix} 0 & 0 & -\frac{v_t}{\omega_t} \cos \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & -\frac{v_t}{\omega_t} \sin \mu_{t-1,\theta} + \frac{v_t}{\omega_t} \sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix} F_x$

5: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + \underbrace{F_x^T \, R_t^x \, F_x}_{R_t}$

EKF_SLAM_Correction

6: $Q_t = \begin{pmatrix} {\sigma_r}^2 & 0 \\ 0 & {\sigma_\phi}^2 \end{pmatrix}$

7: *for all observed features* $z_t^i = (r_t^i, \phi_t^i)^T$ *do*

8: $j = c_t^i$

9: *if landmark* j *never seen before*

10: $\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} r_t^i \cos(\phi_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\phi_t^i + \bar{\mu}_{t,\theta}) \end{pmatrix}$

11: *endif*

12: $\delta = \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{pmatrix}$

13: $q = \delta^T \delta$

14: $\hat{z}_t^i = \begin{pmatrix} \sqrt{q} \\ \text{atan2}(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \end{pmatrix}$

15: $F_{x,j} = \begin{pmatrix} 1 & 0 & 0 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 1 & 0 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 0 & \underbrace{0 \cdots 0}_{2j-2} & 0 & 1 & \underbrace{0 \cdots 0}_{2N-2j} \end{pmatrix}$

16: $H_t^i = \frac{1}{q} \begin{pmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & +\sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & +\delta_x \end{pmatrix} F_{x,j}$

17: $K_t^i = \bar{\Sigma}_t \, H_t^{iT} (H_t^i \, \bar{\Sigma}_t \, H_t^{iT} + Q_t)^{-1}$

18: $\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i)$

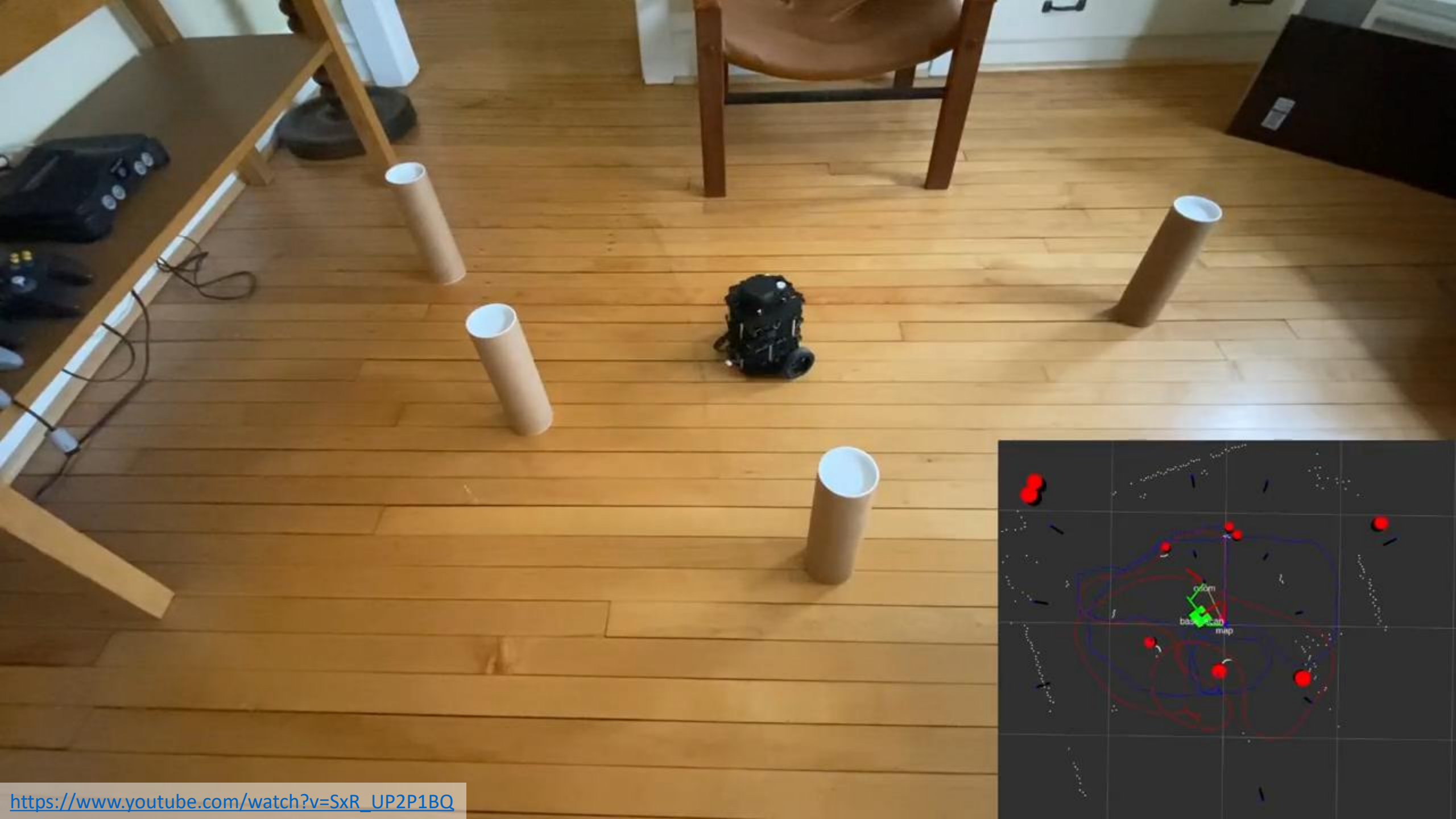
19: $\bar{\Sigma}_t = (I - K_t^i \, H_t^i) \, \bar{\Sigma}_t$

20: *endfor*

21: $\mu_t = \bar{\mu}_t$

22: $\Sigma_t = \bar{\Sigma}_t$

23: *return* μ_t, Σ_t



https://www.youtube.com/watch?v=SxR_UP2P1BQ

Graph-based SLAM using Pose Graphs

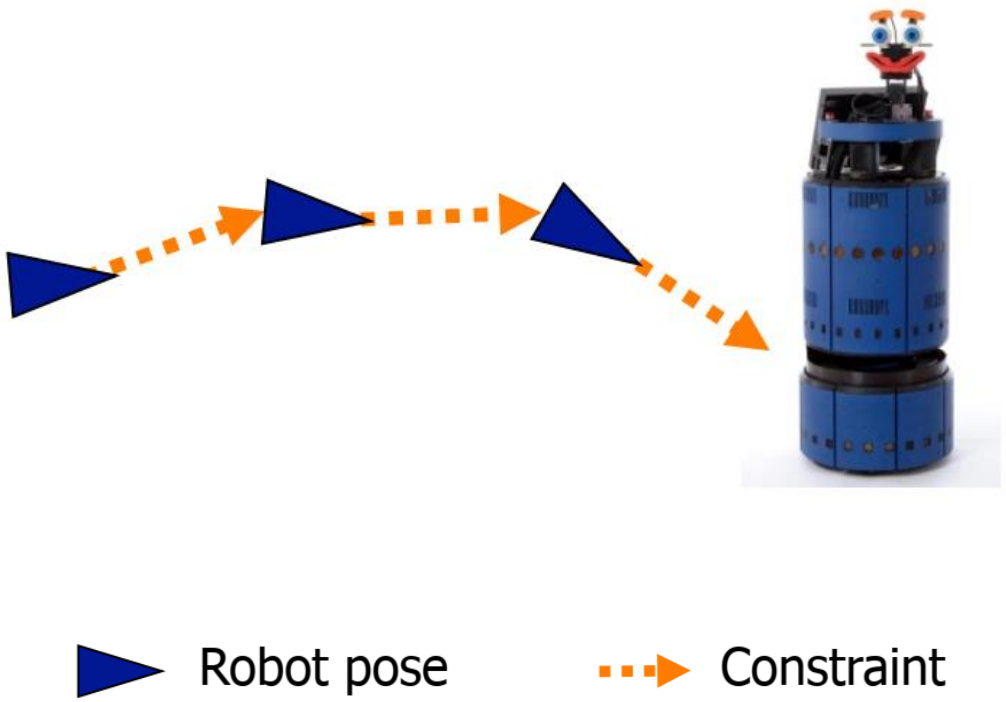
Pose-Graph SLAM

- Graphs can be used to represent a set of robot poses where pairs of poses are connected by edges that encode spatial constraints between the robot poses.
  - Graph represents the SLAM problem.
  - Each node in the graph represents a pose of the robot during mapping
  - Each edge between two nodes encodes a spatial constraint between them.

- Pose-Graph SLAM: Build the graph of robot poses and find a pose configuration that minimize the error introduced by the constraints.

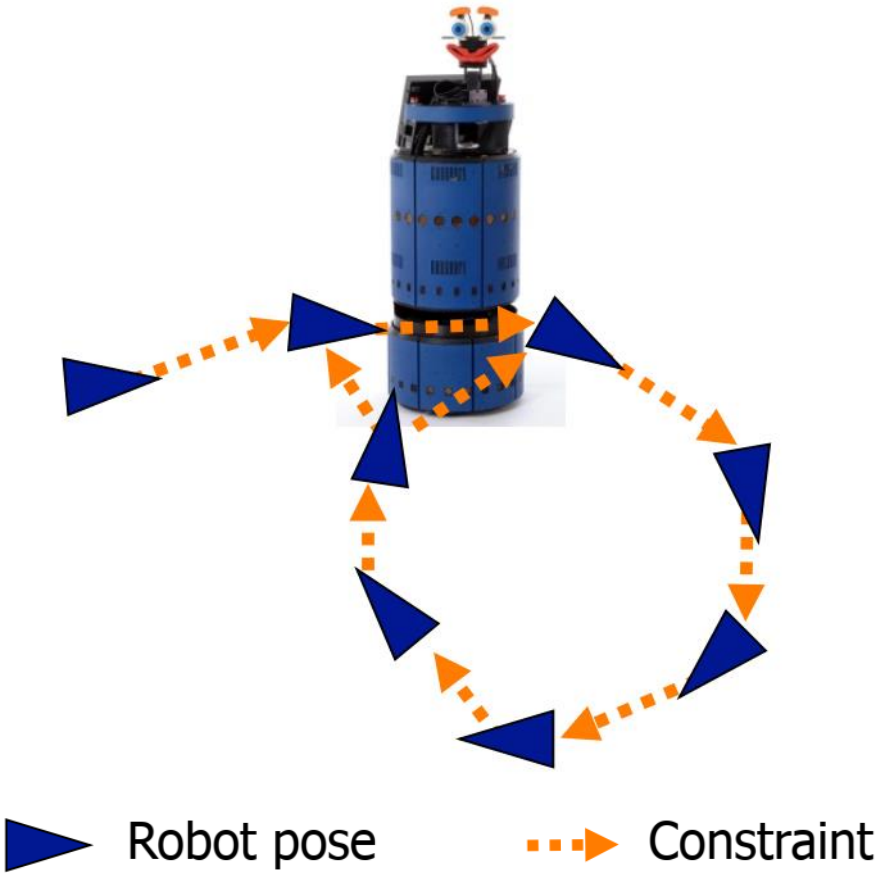
Pose-Graph SLAM

- Constraints connect the poses of the robot while it is moving.
- Constraints are inherently uncertain.



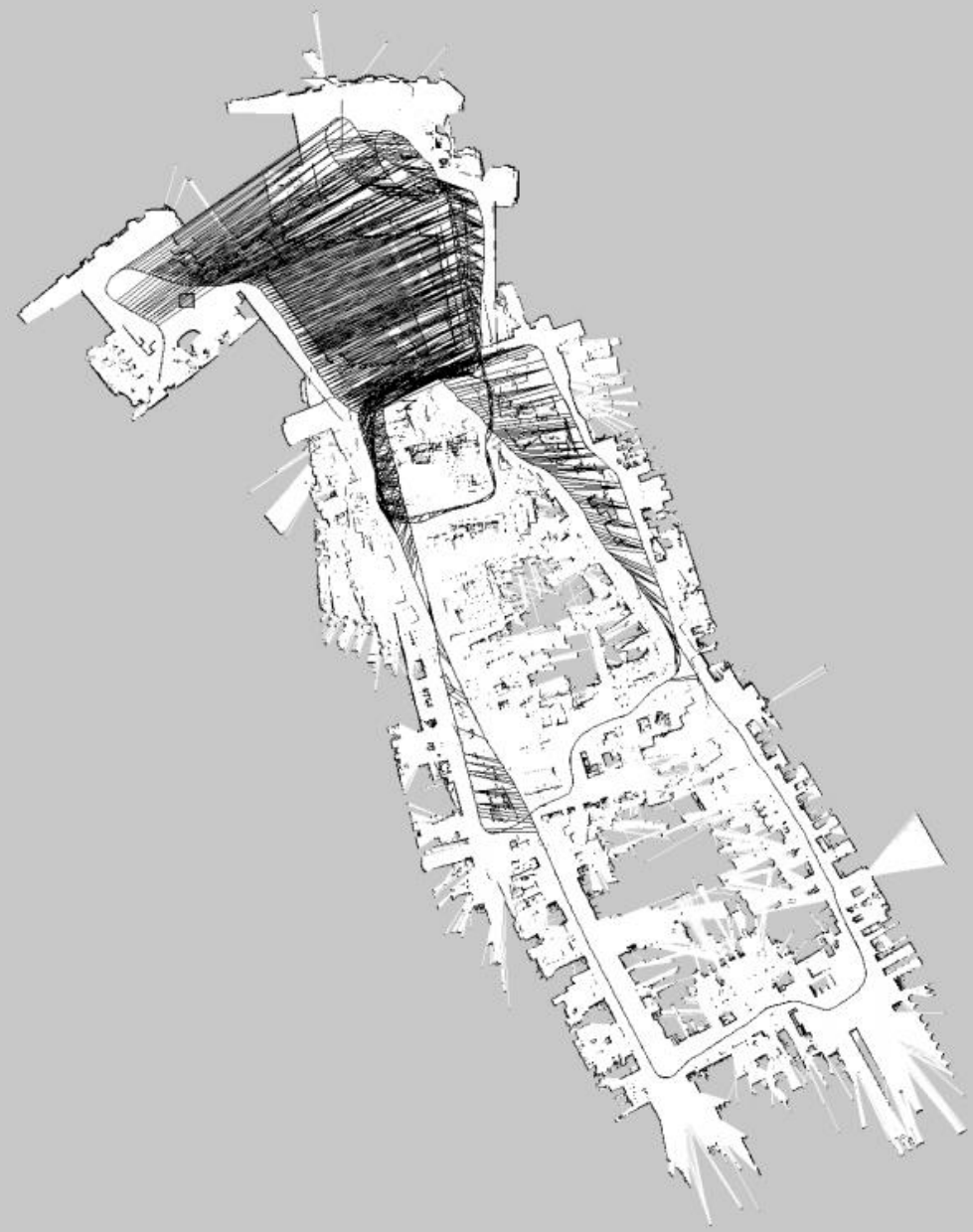
Pose-Graph SLAM

- Observing previously seen areas generates constraints between non-successive poses.



Map constructed using odometry only

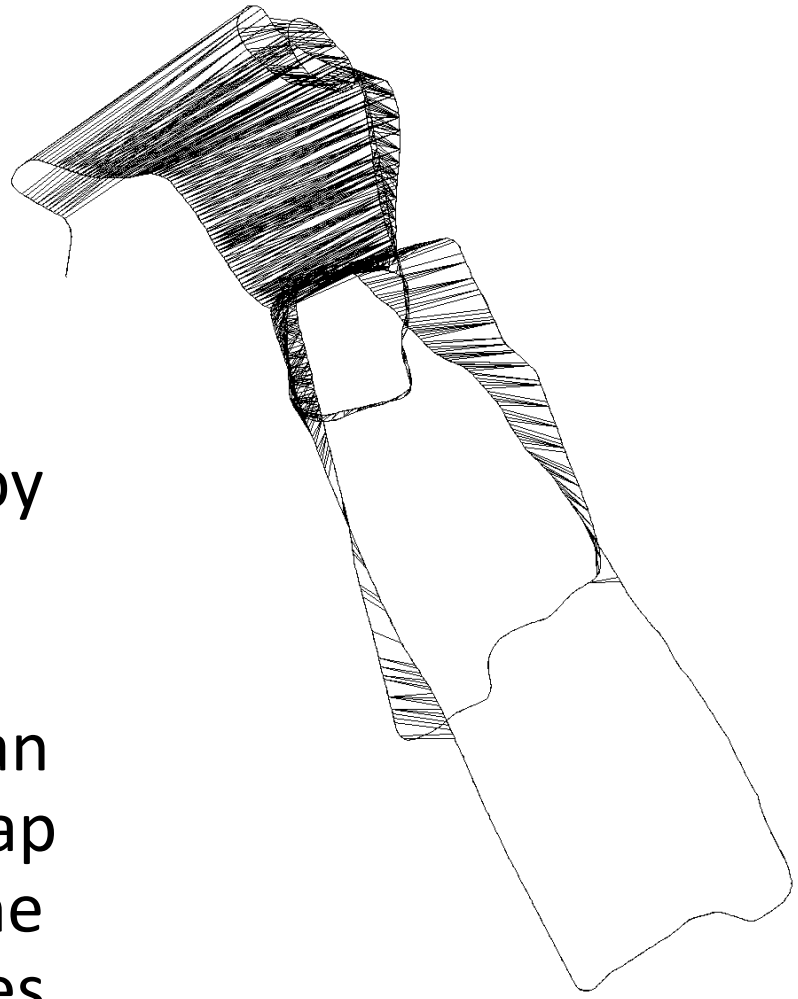
Adding a pose graph to SLAM

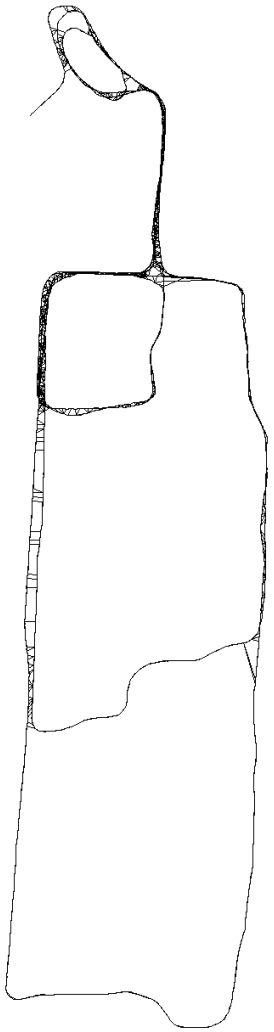


KUKA Halle 22, courtesy of P. Pfaff

Pose-Graph SLAM Example

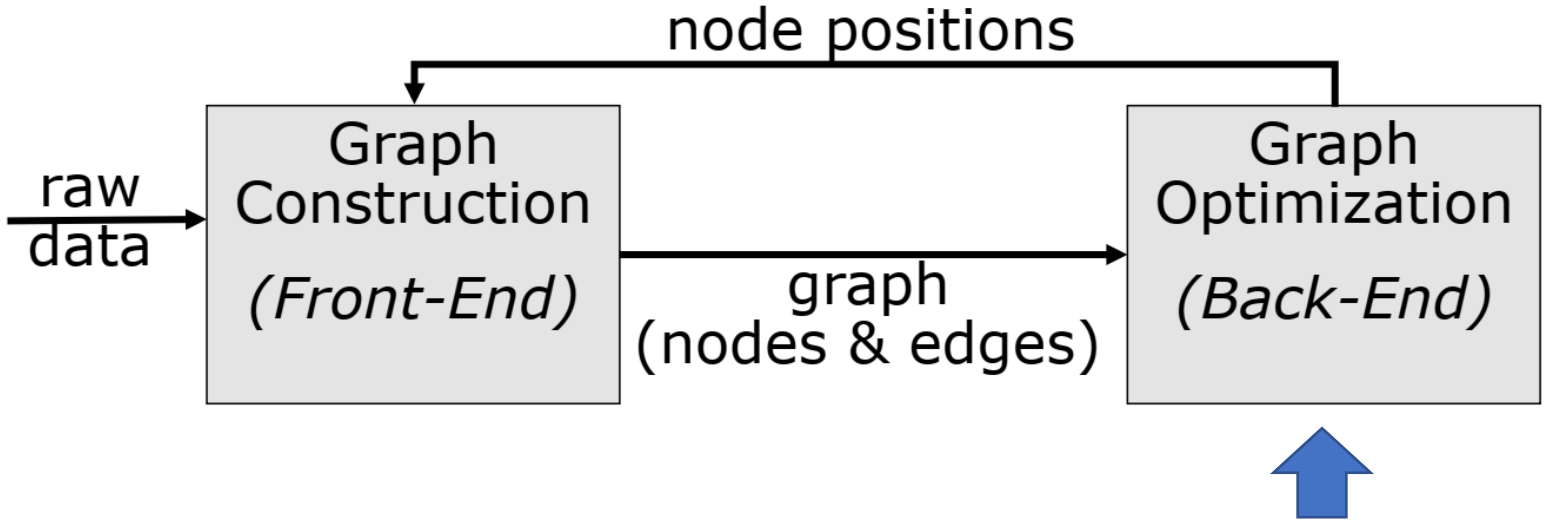
- Once we have the graph, we determine the most likely map by correcting the nodes.
- Then, we can render a map based on the known poses.





Overall SLAM System

- An overall SLAM system includes front-end and back-end that interact with each other.
- A consistent map helps to determine new constraints by reducing the search space.
- We first focus on the back-end for graph optimization.



Least Squares Optimization in General

- It is an optimization approach for computing a solution for an overdetermined system:
  - “More equations than unknowns”
- It minimizes the sum of the squared errors in the equations.
- It is a classic approach to a large set of problems:
  - Graph optimization for SLAM
  - Various machine learning methods

Least Squares Optimization in General

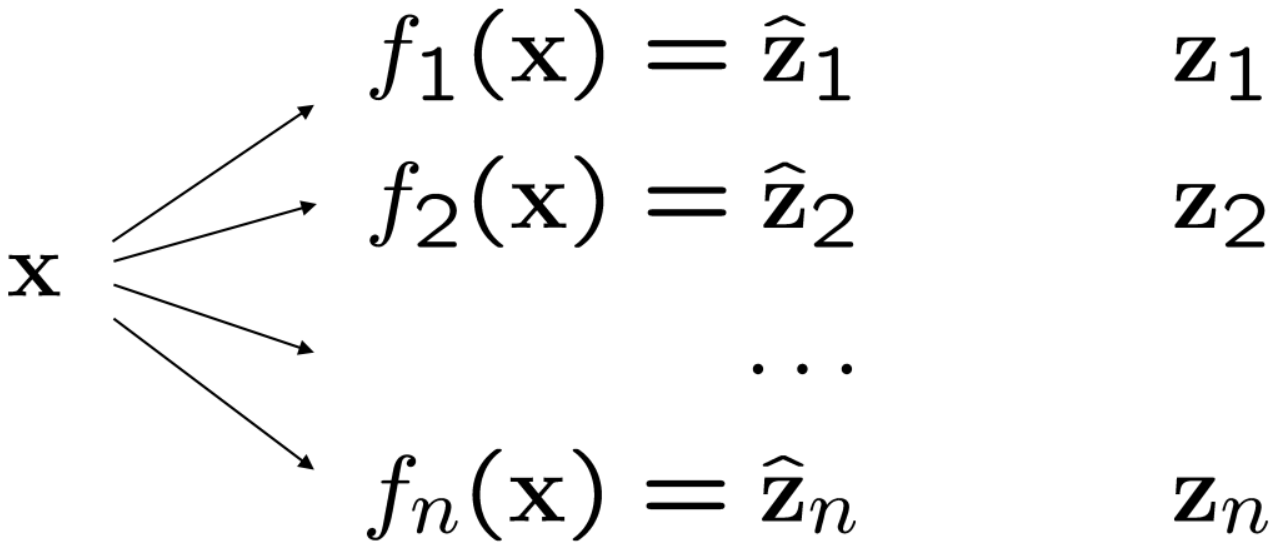
- Given a problem described by a set of n observation functions:

$$\{f_i(\mathbf{x})\}_{i=1:n}$$

- Let
  - $\mathbf{x}$ be the state vector
  - $\mathbf{z}_i$ be a measurement of the state $\mathbf{x}$
  - $\hat{\mathbf{z}}_i = f_i(\mathbf{x})$ be a function which projects $\mathbf{x}$ to a predicted measurement
- Given n noisy measurement $\mathbf{z}_{1:n}$ about the state $\mathbf{x}$
- Goal: Estimate the state $\mathbf{x}$ which best explains the measurements $\mathbf{z}_{1:n}$

Recall: how is this done in probabilistic formulations?

Least Squares Optimization in General



state (unknown)	predicted measurements	real measurements

Error Function for Least Squares Optimization

- Error $\mathbf{e}_i$ is often defined as the difference between the prediction and the actual measurement:

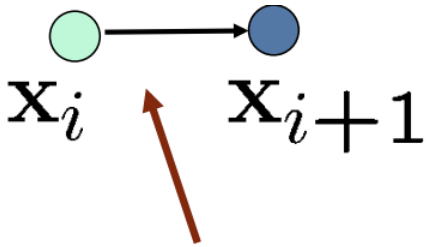
$$\mathbf{e}_i(\mathbf{x}) \quad = \quad \mathbf{z}_i - f_i(\mathbf{x})$$

- We assume that the error has zero mean and is normally distributed.
  - Gaussian error with covariance matrix $\mathbf{\Omega}_i$ (also called information matrix)
- The squared error of a measurement depends only on the noisy states and is a scalar:

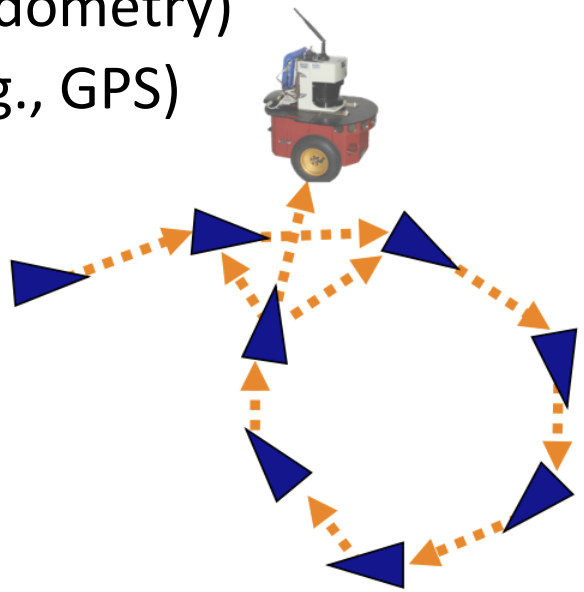
$$e_i(\mathbf{x}) \quad = \quad \mathbf{e}_i(\mathbf{x})^T \mathbf{\Omega}_i \mathbf{e}_i(\mathbf{x})$$

Graph Representation

- The graph is assumed to include n nodes $\mathbf{x} = \mathbf{x}_{1:n}$
- Each node $\mathbf{x}_i$ is a robot pose at time t_i
- A constraint encoded by the edge exists between the nodes if the robot moves from $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$
  - Edge is computed using a motion model (e.g., odometry)
  - Measurements $\mathbf{z}_{ij}$ are obtained by sensors (e.g., GPS)

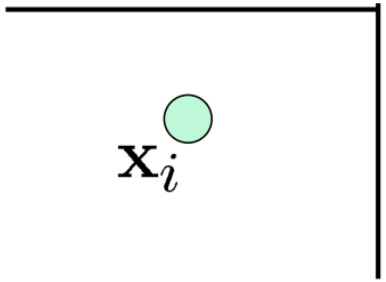


The edge is estimated using a motion model

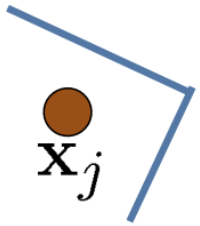


The Graph

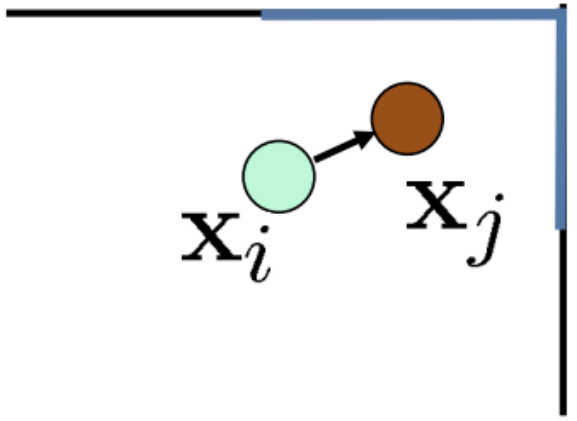
- An edge also exists between the nodes if the robot observes the same scene from $\mathbf{x}_i$ and from $\mathbf{x}_j$, computed iteratively by a motion model.
- Construct a "virtual measurement" about the position of $\mathbf{x}_j$ that is seen from $\mathbf{x}_i$ by using the environment as a reference.
  - Edge represents the position of $\mathbf{x}_j$ seen from $\mathbf{x}_i$ based on the observation.



Measurement from $\mathbf{x}_i$

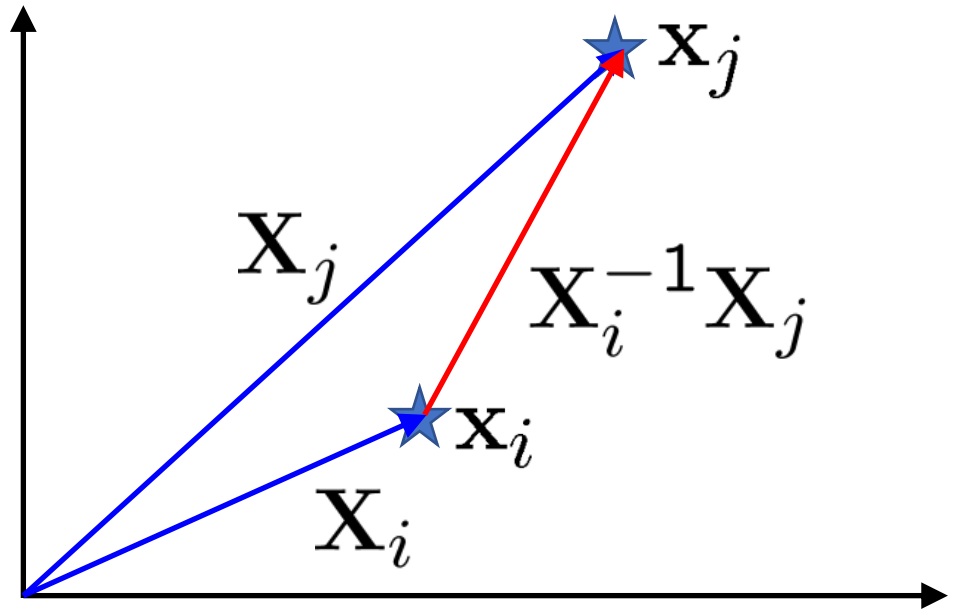


Measurement from $\mathbf{x}_j$



Transformations

- How does $\mathbf{x}_i$ sees $\mathbf{x}_j$?
- Express this through transformations
- Let $\mathbf{X}_i$ be transformation of the origin into $\mathbf{x}_i$
- Let $\mathbf{X}_i^{-1}$ be the inverse transformation
- We can express relative transformation $\mathbf{X}_i^{-1}\mathbf{X}_j$
- Transformations can be expressed using homogenous coordinates



Transformations

- Homogenous coordinates (also called projective coordinates)
  - Homogenous coordinates are widely used in projective geometry to provide an alternative representation of gematric objects and translations.
  - N-dimensional space expressed in N+1 dimensional space.
  - Projection to homogeneous space: $(x, y, z)^T \rightarrow (x, y, z, 1)^T = (a, b, c, d)^T$
  - Back-projection to 3D space: $(a, b, c, d)^T \rightarrow \left(\frac{a}{d}, \frac{b}{d}, \frac{c}{d}\right)^T = (x, y, z)^T$

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{R} = \begin{pmatrix} \mathbf{R}^{3D} & 0 \\ 0 & 1 \end{pmatrix} \qquad \mathbf{X} = \begin{pmatrix} \mathbf{R}^{3D} & \mathbf{t} \\ 0 & 1 \end{pmatrix}$$

translation rotation rigid-body

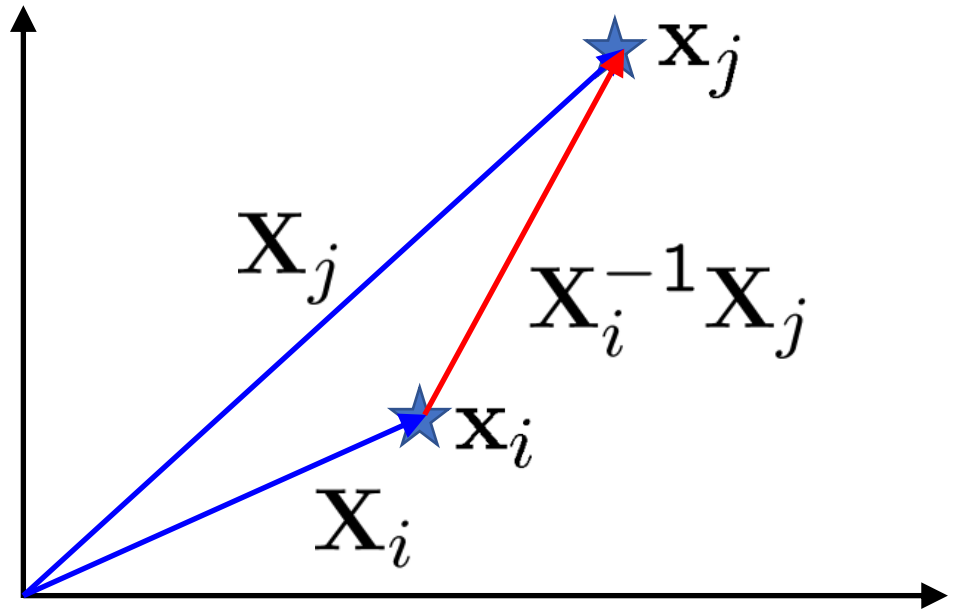
Transformations

- Transformations can be expressed using homogenous coordinates.

- Estimation-Based edge:

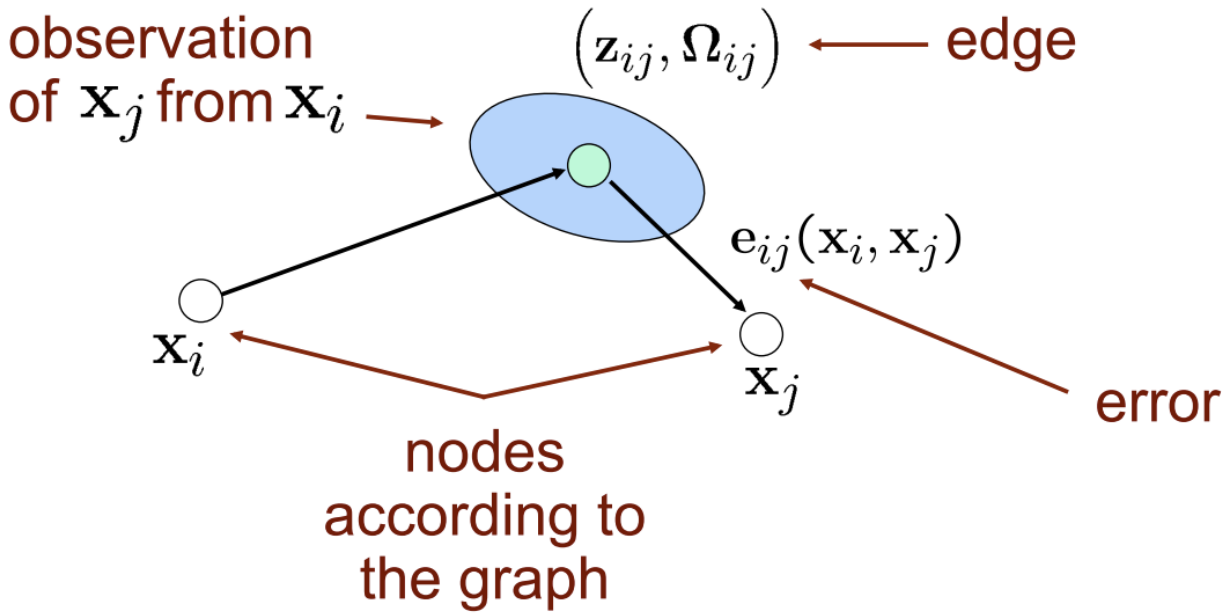
$$(\mathbf{X}_i^{-1}\mathbf{X}_j)$$

describes “how node i sees node j”



Pose Graph Formulation

- Measurements $\mathbf{z}_{ij}$ are affected by noise (e.g., GPS).
- Covariance matrix for each edge $\mathbf{\Omega}_{ij}$ to encode its uncertainty.
- The "bigger" $\mathbf{\Omega}_{ij}$, the more the edge "matters" in the optimization.



Error Function

- Error function for a single constraint

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \text{t2v}(\mathbf{Z}_{ij}^{-1}(\mathbf{X}_i^{-1}\mathbf{X}_j))$$

measurement

x_j seen from x_i

- Error as a function of the whole state vector

$$\mathbf{e}_{ij}(\mathbf{x}) = \text{t2v}(\mathbf{Z}_{ij}^{-1}(\mathbf{X}_i^{-1}\mathbf{X}_j))$$

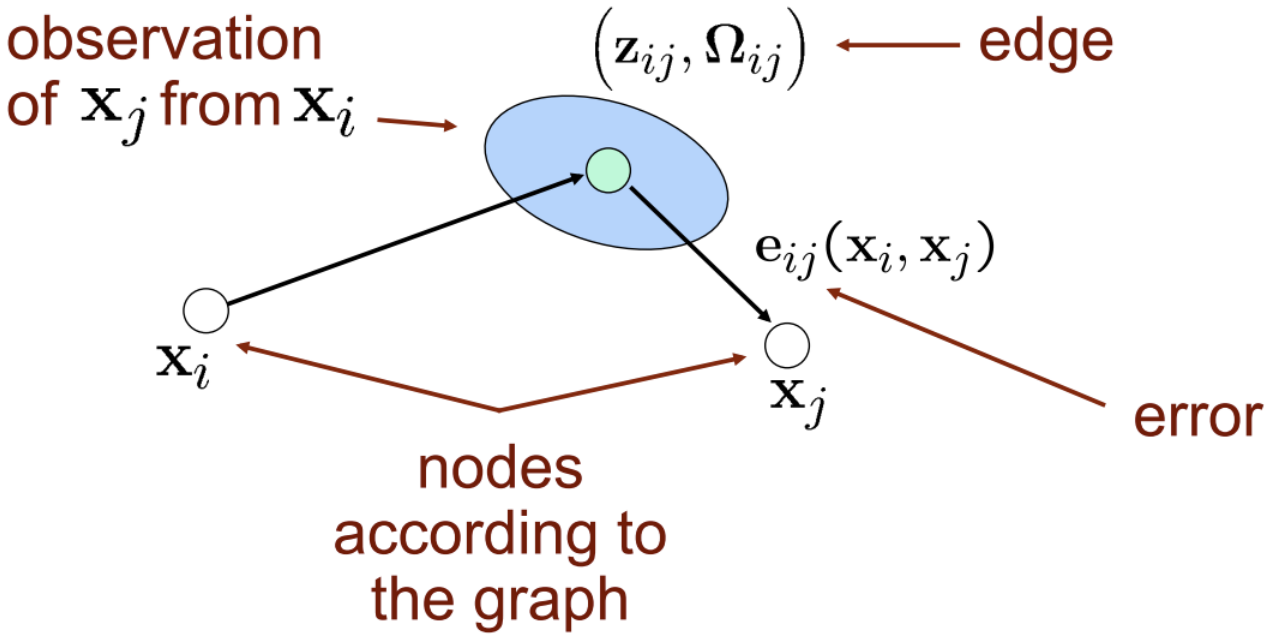
- Error takes a value of 0 if

$$\mathbf{Z}_{ij} = (\mathbf{X}_i^{-1}\mathbf{X}_j)$$

Pose Graph Optimization

- Goal:

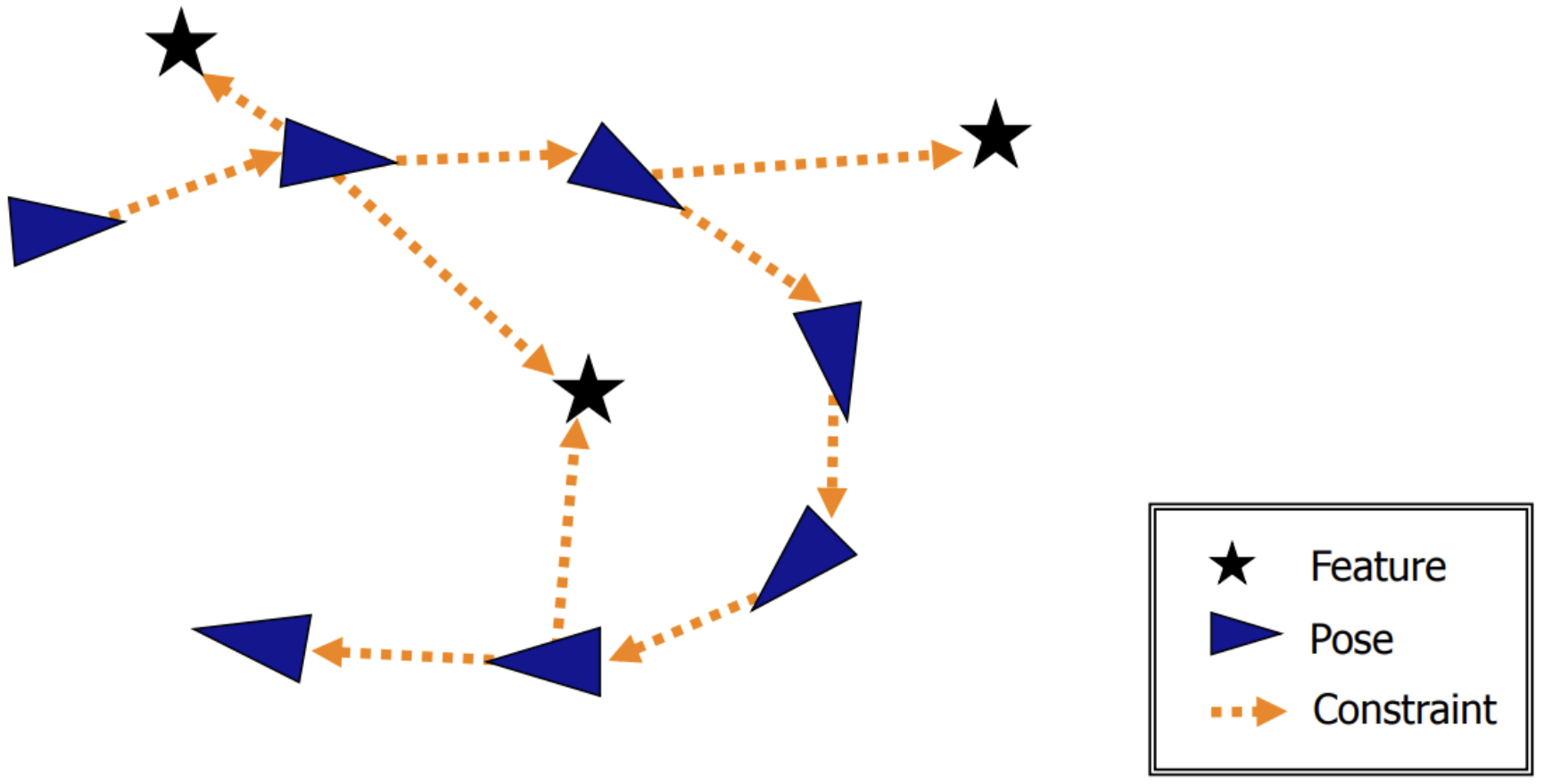
$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{ij} \mathbf{e}_{ij}^T \boldsymbol{\Omega}_{ij} \mathbf{e}_{ij}$$



Optimization Using the Gauss-Newton Algorithm

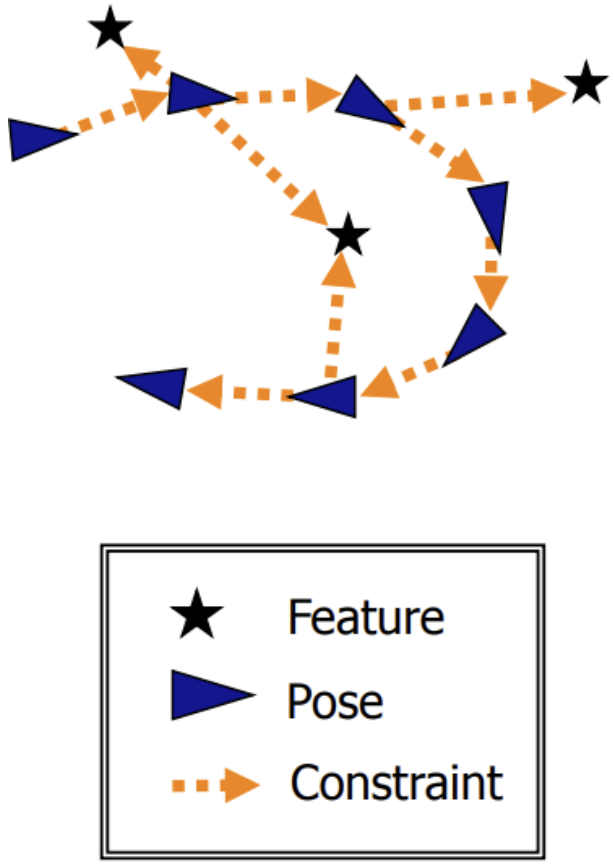
- Define the error function
- If nonlinear, linearize the error function
- Compute its derivative
- Set the derivative to zero
- Solve the linear system
- Iterate this procedure until convergence

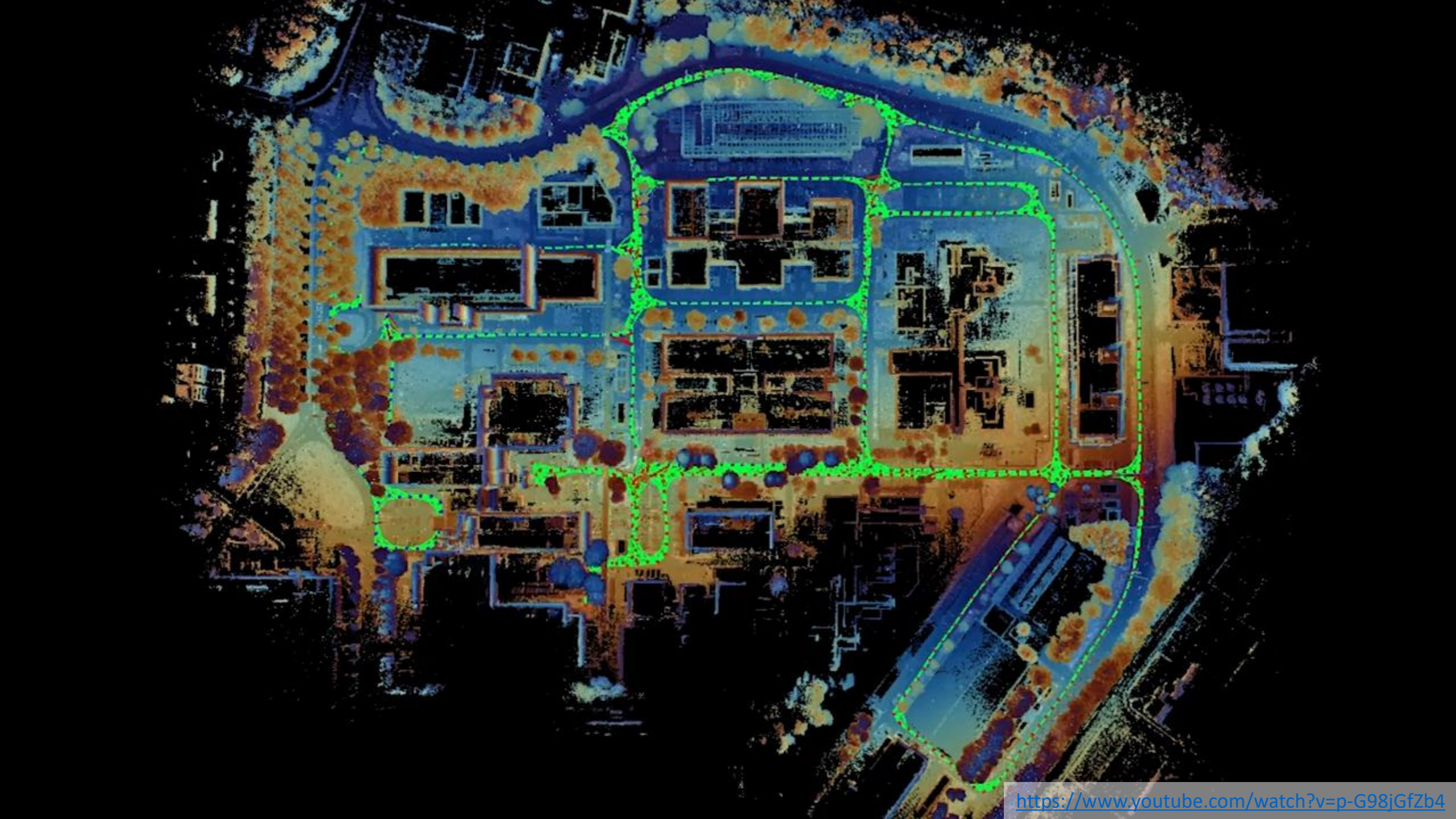
Pose Graph with Landmarks



Pose Graph with Landmarks

- Nodes can represent:
  - Robot poses
  - Landmark locations
- Edges can represent:
  - Pose displacement from a motion model
  - Landmark location estimation
- The minimization optimizes the landmark locations and robot poses
  - E.g., in the (3+2N) space for 2D SLAM



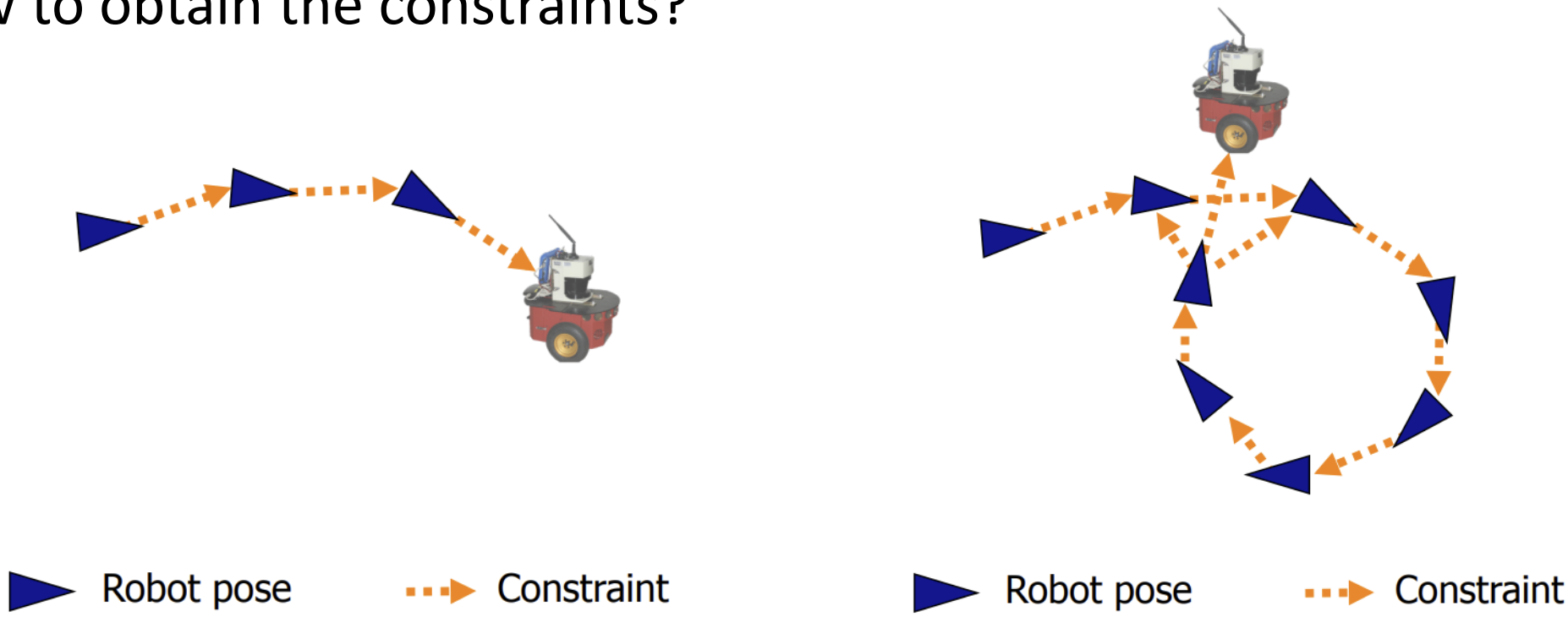


https://www.youtube.com/watch?v=p-G98jGfZb4

SLAM Front-Ends

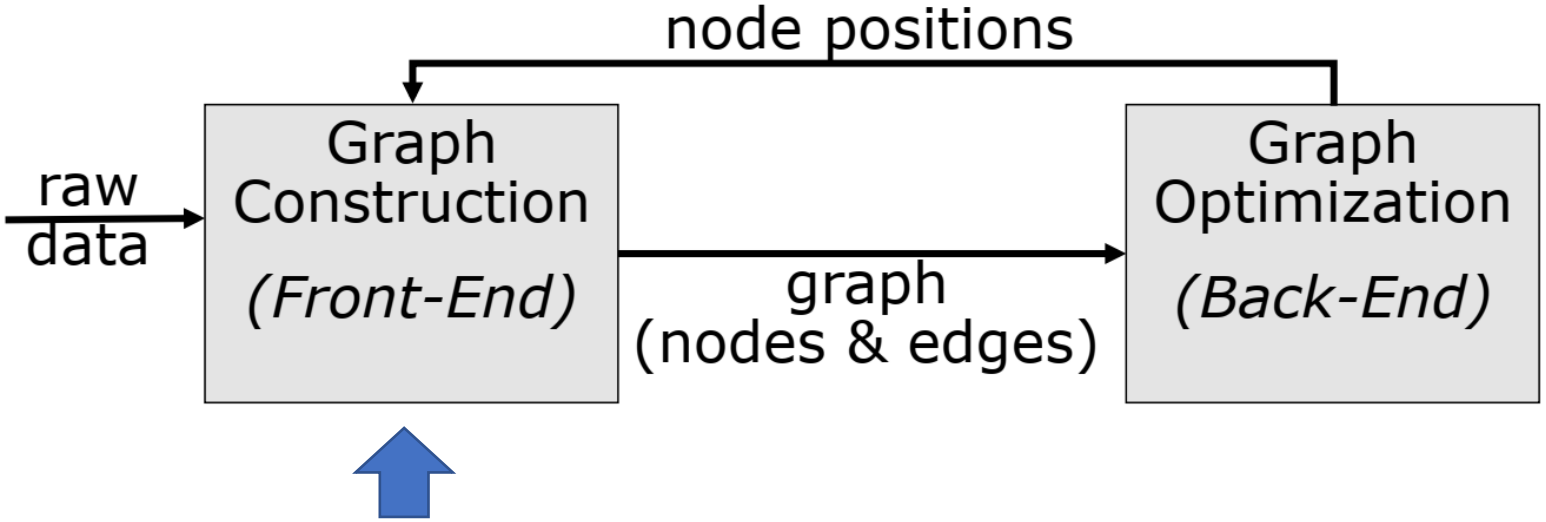
Graph-Based SLAM

- Constraints are represented by the edges that connect the nodes computed using a motion model.
- How to obtain the constraints?



SLAM Front-End

- Front-end creates constraints that can be obtained from a motion model (for short-term edges) and matching observations (for long-term edges).
- Popular methods by matching observations: Dense scan-matching, landmark-based matching



Problem Formulation

- Given: two corresponding point sets (e.g., obtained from LiDAR):

$$Q = \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_N\} \qquad P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_M\}$$

  with correspondences $\mathcal{C} = \{(i, j)\}$.

- Wanted: translation t and rotation R that minimizes the sum of the squared error:

$$E(R, \boldsymbol{t}) = \sum_{(i,j)\in\mathcal{C}} \|\boldsymbol{q}_i - R\boldsymbol{p}_j - \boldsymbol{t}\|^2$$

  where $\boldsymbol{p}_j$ and $\boldsymbol{q}_i$ are corresponding points.

Center of Mass

- The centers of mass of the corresponding points in both sets:

$$\boldsymbol{\mu}_Q = \frac{1}{|C|} \sum_{(i,j) \in \mathcal{C}} \boldsymbol{q}_i \qquad \boldsymbol{\mu}_P = \frac{1}{|C|} \sum_{(i,j) \in \mathcal{C}} \boldsymbol{p}_j$$

- Mean-reduced points by subtracting the corresponding center of mass for every point:

$$\begin{aligned} Q' &= \{\boldsymbol{q}_i - \boldsymbol{\mu}_Q\} = \{\boldsymbol{q}'_i\} \\ P' &= \{\boldsymbol{p}_j - \boldsymbol{\mu}_P\} = \{\boldsymbol{p}'_j\} \end{aligned}$$

Orthogonal Procrustes Problem

- After translating to overlay the centers of mass for the two point-clouds, minimizing:

$$E'(R) = \|[\boldsymbol{q}'_1 \dots \boldsymbol{q}'_n] - R[\boldsymbol{p}'_1 \dots \boldsymbol{p}'_n]\|_F^2$$

  is equivalent to minimizing:

$$E(R, \boldsymbol{t}) = \sum_{(i,j)\in C} \|\boldsymbol{q}_i - R\boldsymbol{p}_j - \boldsymbol{t}\|^2$$

Point Alignment and Transformation

$$\boldsymbol{p}_j \leftarrow R(\boldsymbol{p}_j - \boldsymbol{\mu}_P) + \mu_Q$$

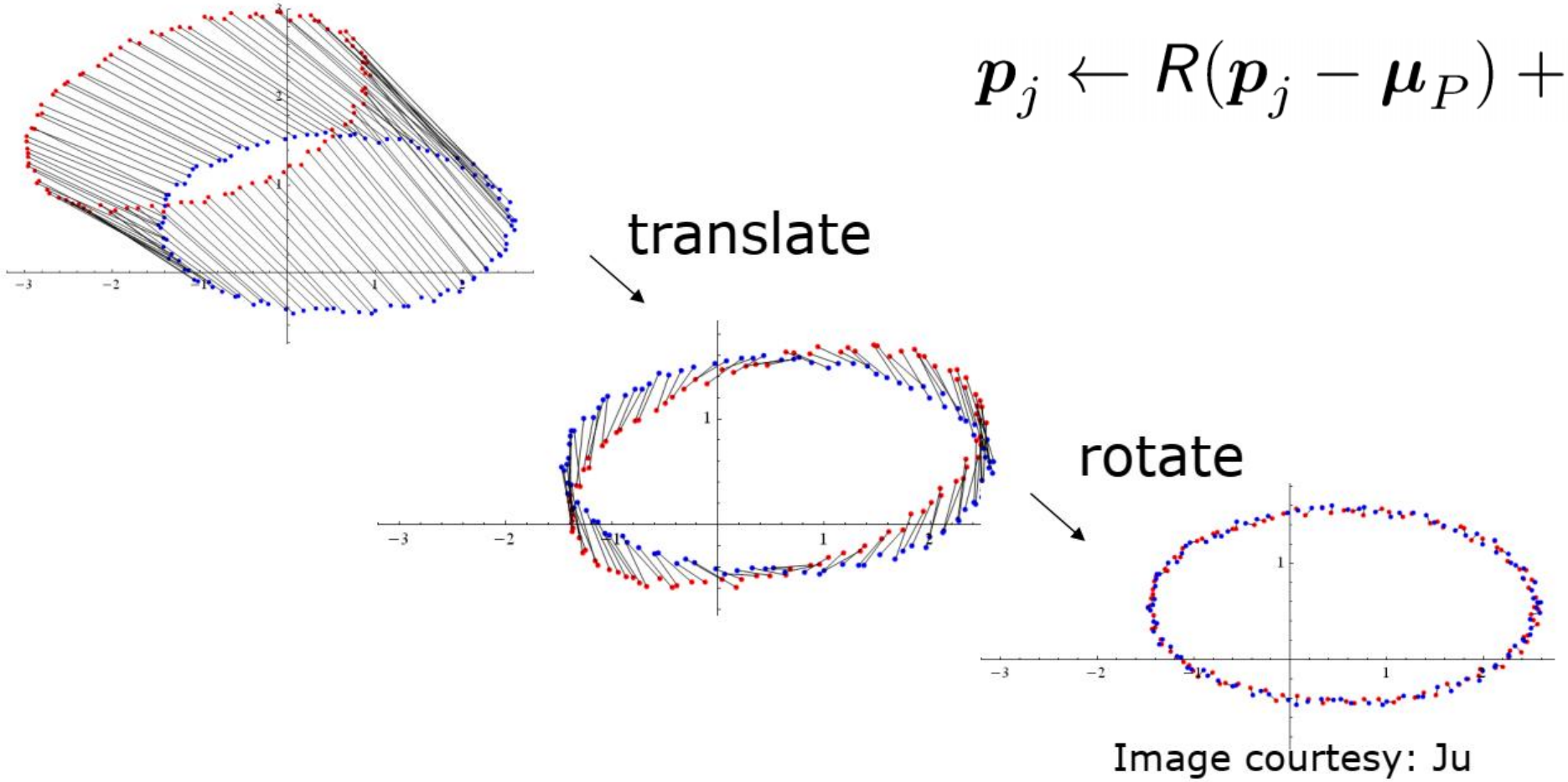
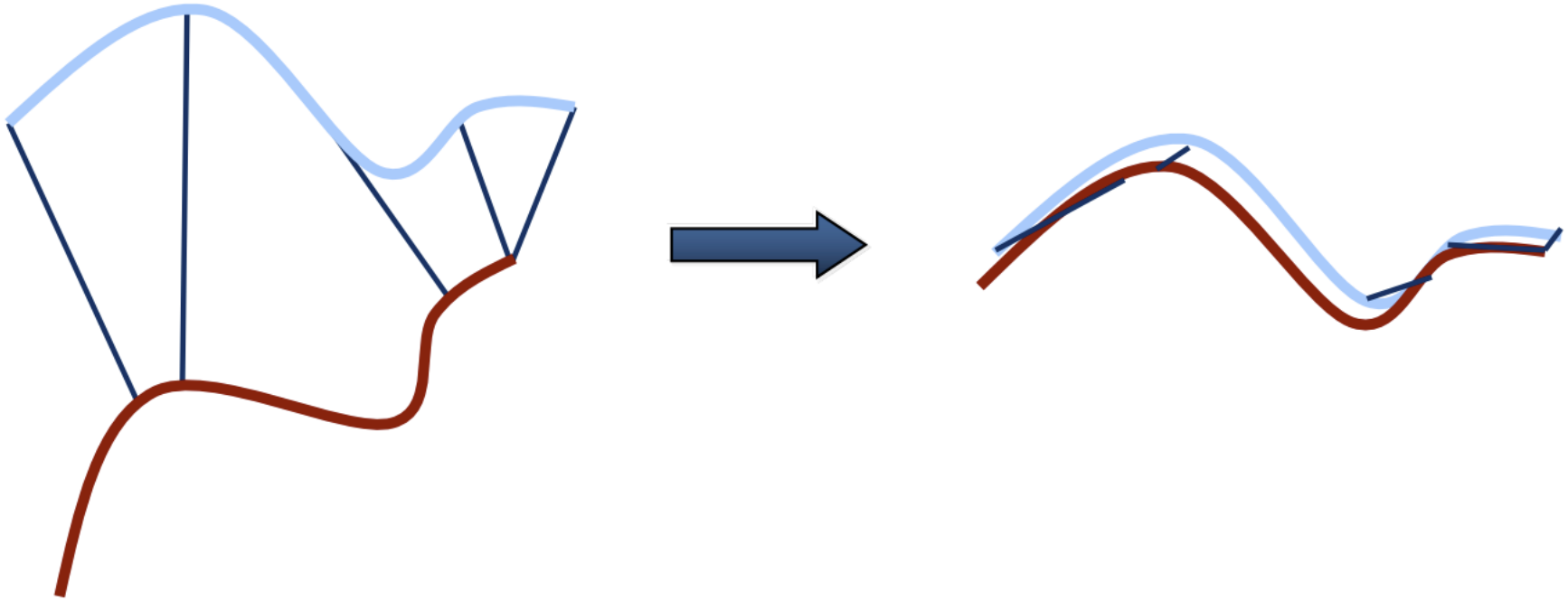


Image courtesy: Ju

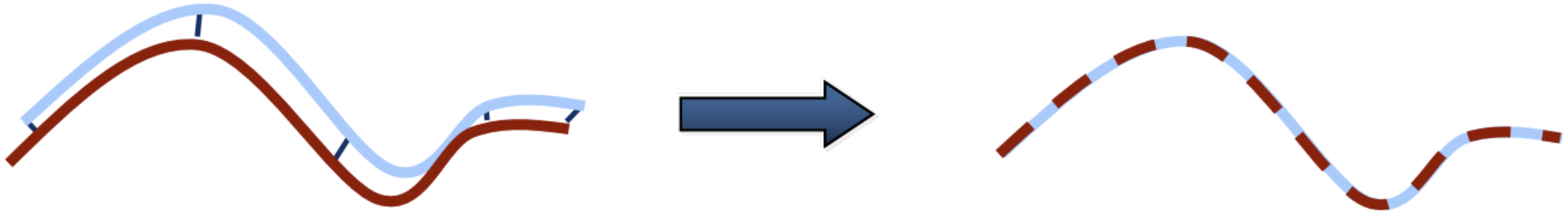
Point Alignment with Unknown Correspondence

- If correct correspondences are not known, it is generally impossible to determine the optimal relative rotation/translation in one step --- but we can iterate!

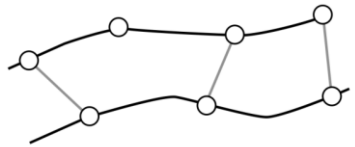


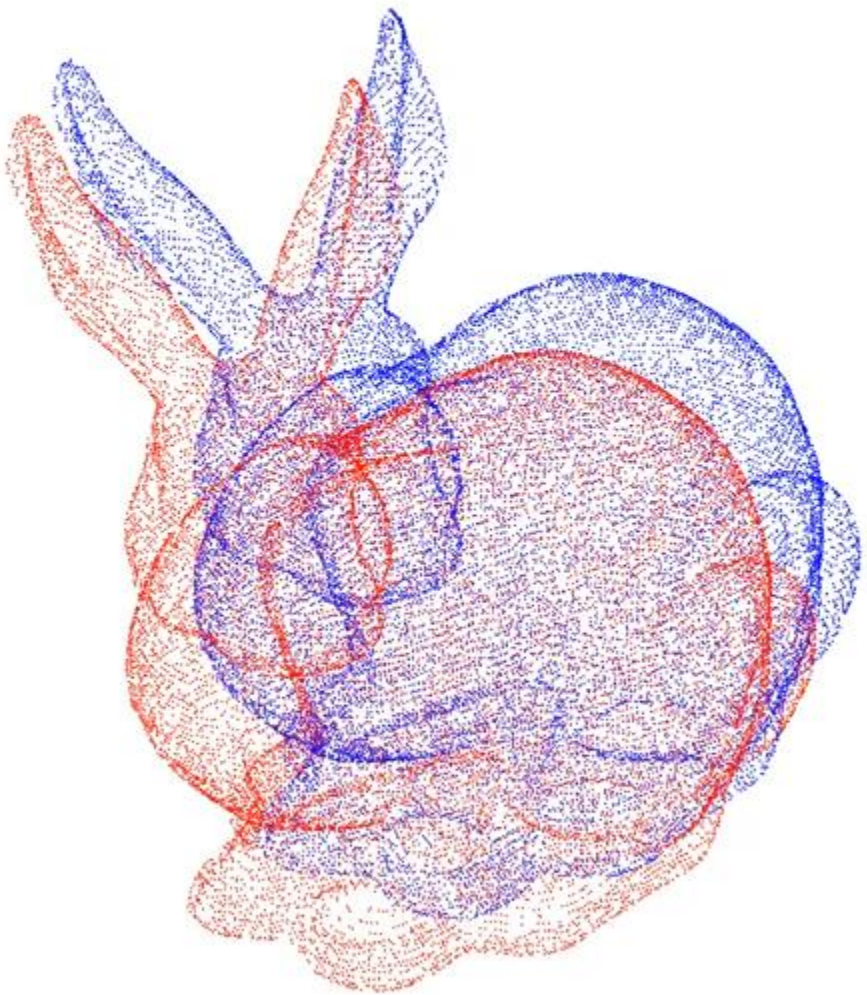
Iterative Closest Point (ICP) Algorithm

- Key idea: iterate to find alignment.
- Two major steps:
  - Data association (also called correspondence identification, matching)
    - E.g., closest point, point-to-plane association, surface normal, local descriptor, etc.
  - Transformation
- ICP converges if sharing positions are “close enough” with sufficient overlap.

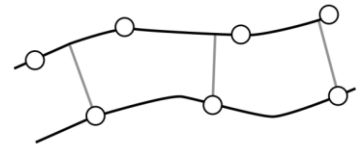


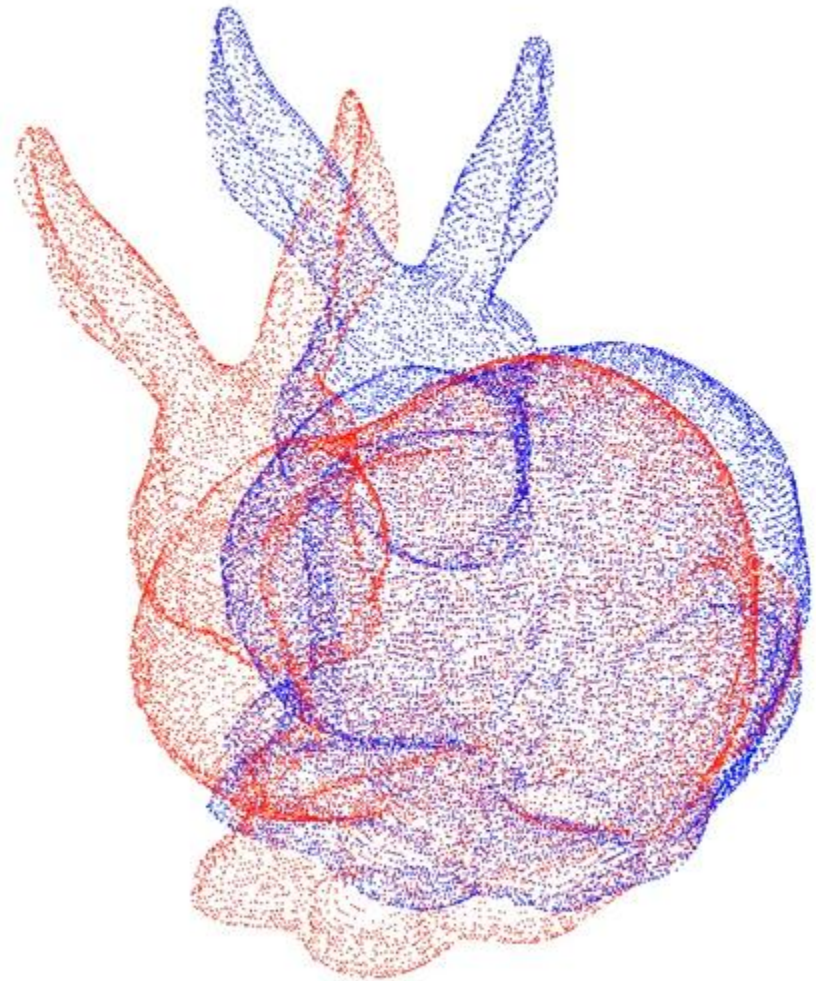
Point-to-point / Iteration 2





Point-to-plane / Iteration 2

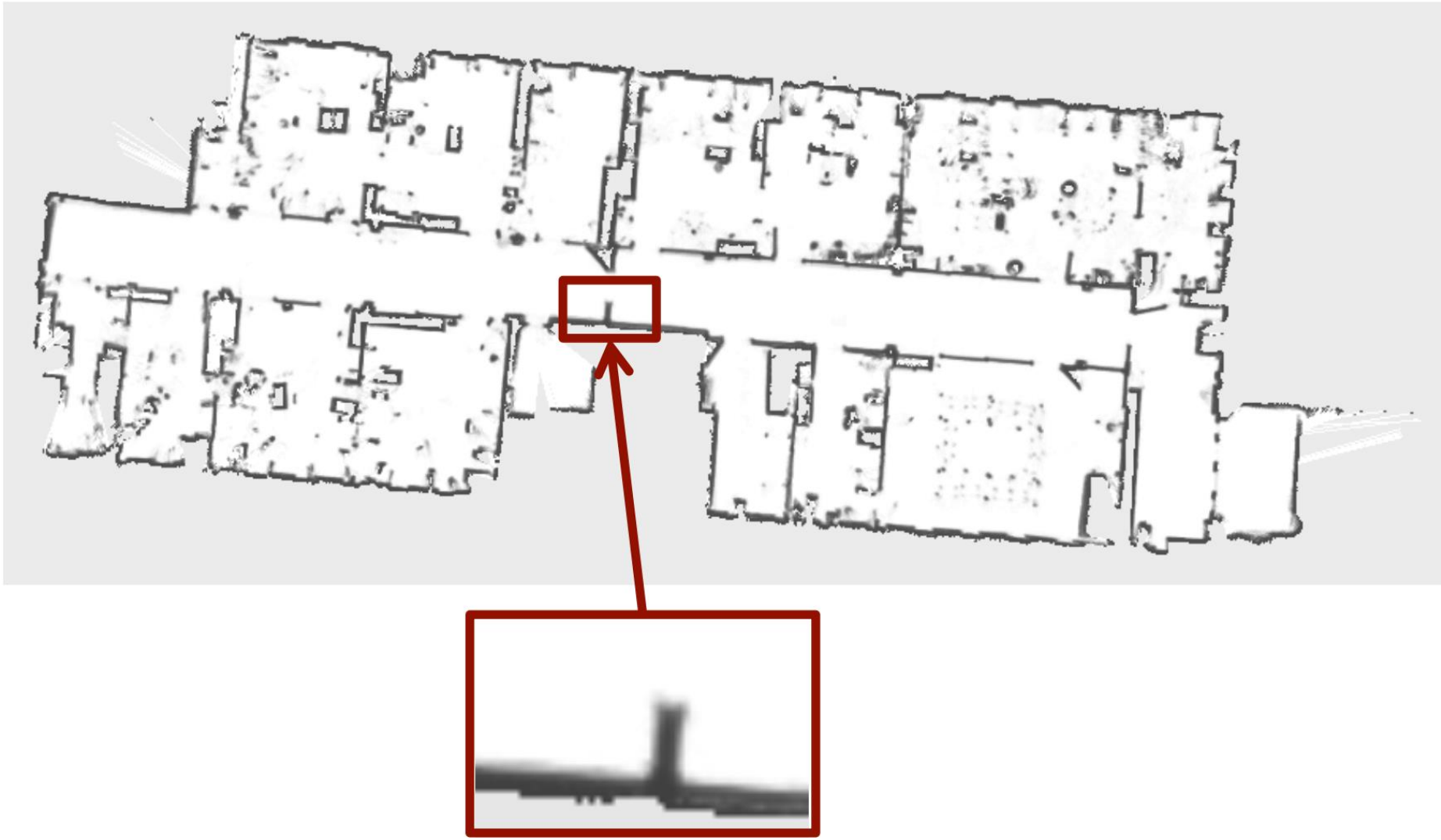




https://www.youtube.com/watch?v=LcghboLgTiA

Path Planning Using the Map

Maps from Pose-Graph SLAM

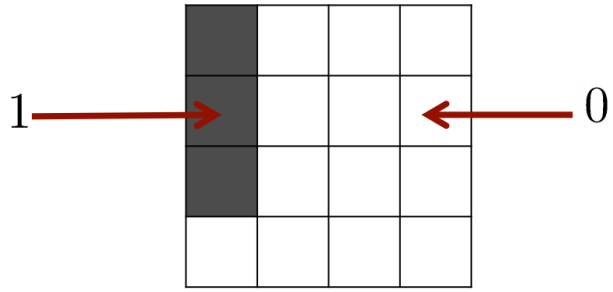


Grid Occupancy Maps

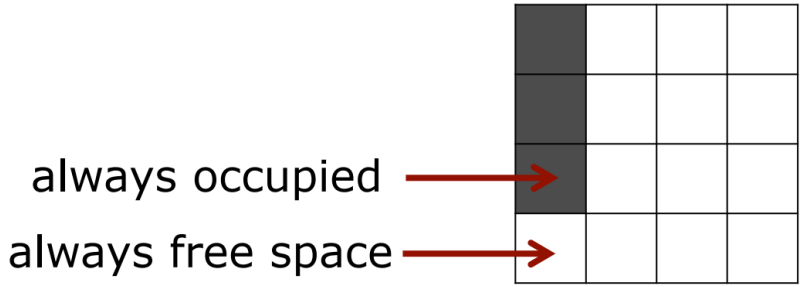
- Discretize the world into cells/grids (also called cell decomposition)
- Grid structure is rigid
- Each cell is assumed to be occupied or free space
- Non-parametric model
- Large maps require substantial memory resources
- Do not rely on a feature detector (e.g., no landmarks)

Grid Occupancy Map Assumptions

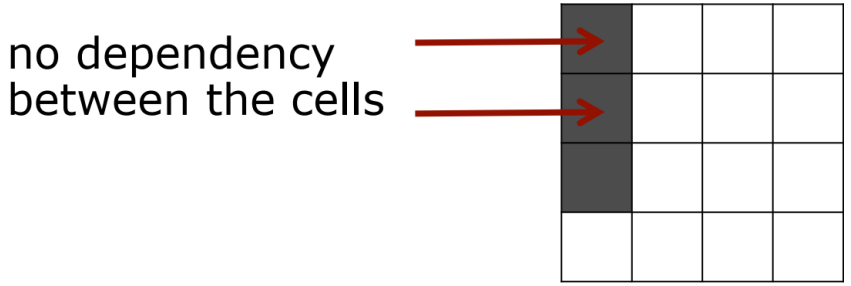
1. Each cell is a binary random variable that models the occupancy



2. The world is static (most mapping systems make this assumption)



3. The cells (the random variables) are independent of each other



Path Planning on Grid Occupancy Maps

- The goal of path planning is to find a collision-free route from a starting point to a target point (or from one pose to another pose).

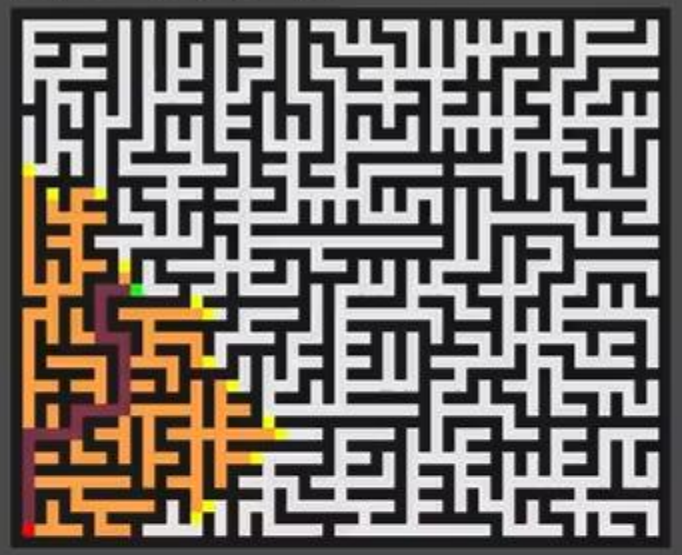
- Path planning methods:
  - Search-based methods
  - Sampling-based methods
  - Learning-based methods, e.g., reinforcement learning methods to find a way through a maze

Search-Based Methods

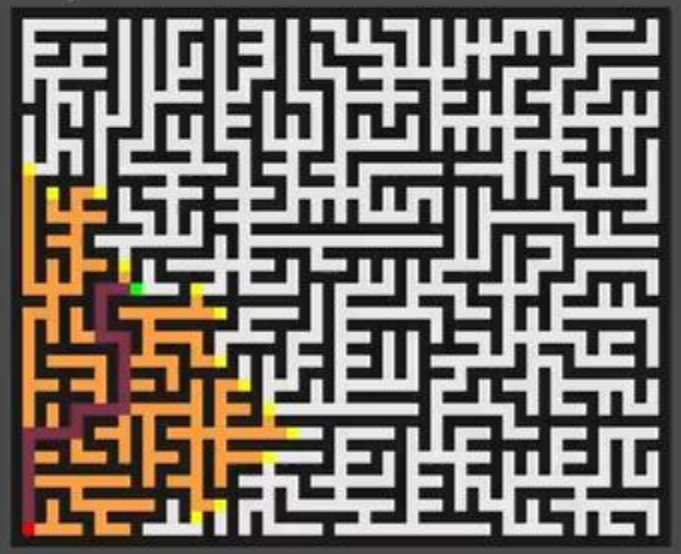
- Depth-first search (DFS)
  - explores as far as possible along each branch before backtracking.
- Breath-first search (BFS)
  - explores all cells at the present depth prior to moving on to the cells at the next depth level.
- Dijkstra's Algorithm: BFS + Priority
  - Changes due to priority that accounts for edge costs
  - Relaxation: Edit previous planned path only if new option is better
- A*: Dijkstra's Algorithm + Heuristics
  - Priority defined as cost to go + heuristic to goal

DFS - 55 x 45

BFS - 55 x 45



Dijkstra - 55 x 45



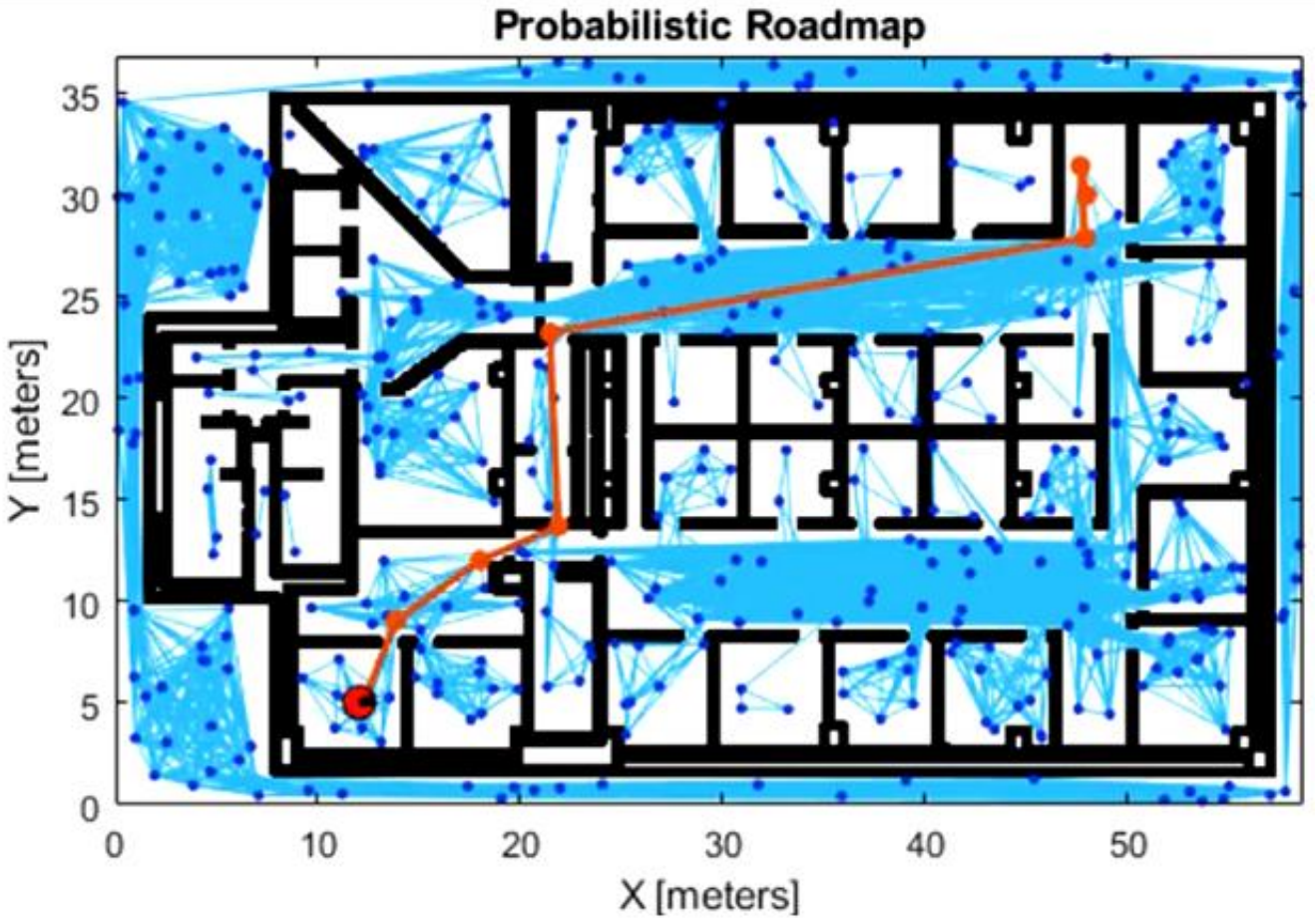
AStar - 55 x 45

	Distance	Expanded	Place
DFS	33	976	4
BFS	33	210	2
Dijkstra	33	218	3
AStar	33	129	1

https://www.youtube.com/watch?v=GC-nBgi9r0U

Sampling-Based Methods

- Probabilistic Roadmap (PRM)
  - Randomly sampling nodes from the map to create a roadmap
  - Query a path using a graph search algorithm (e.g., A*)

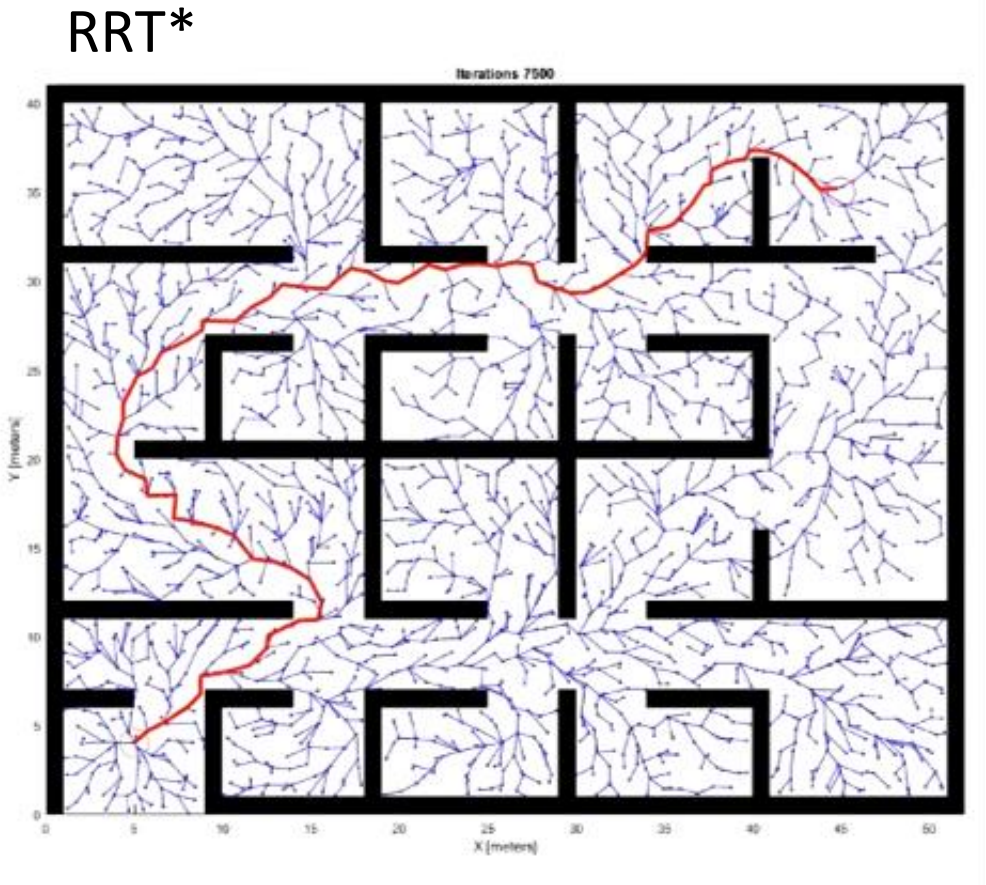


Video credit: MathWorks, https://www.youtube.com/watch?v=-fePRPyeKnc

Sampling-Based Methods

- Rapidly-exploring Random Tree (RRT)
  - Search tree is built incrementally from random nodes.
  - Tree is expended with nearest neighbor search.
- RRT*
  - RRT* expands the tree in a similar way like RRT.
  - RRT* is an optimized version of RRT to find a shortest path.

RRT*



Video credit: MathWorks
https://www.youtube.com/watch?v=-fePRPyeKnc