Advanced Techniques for Mobile Robotics

Graph-based SLAM with Landmarks

Wolfram Burgard, Cyrill Stachniss,

Kai Arras, Maren Bennewitz

The Graph

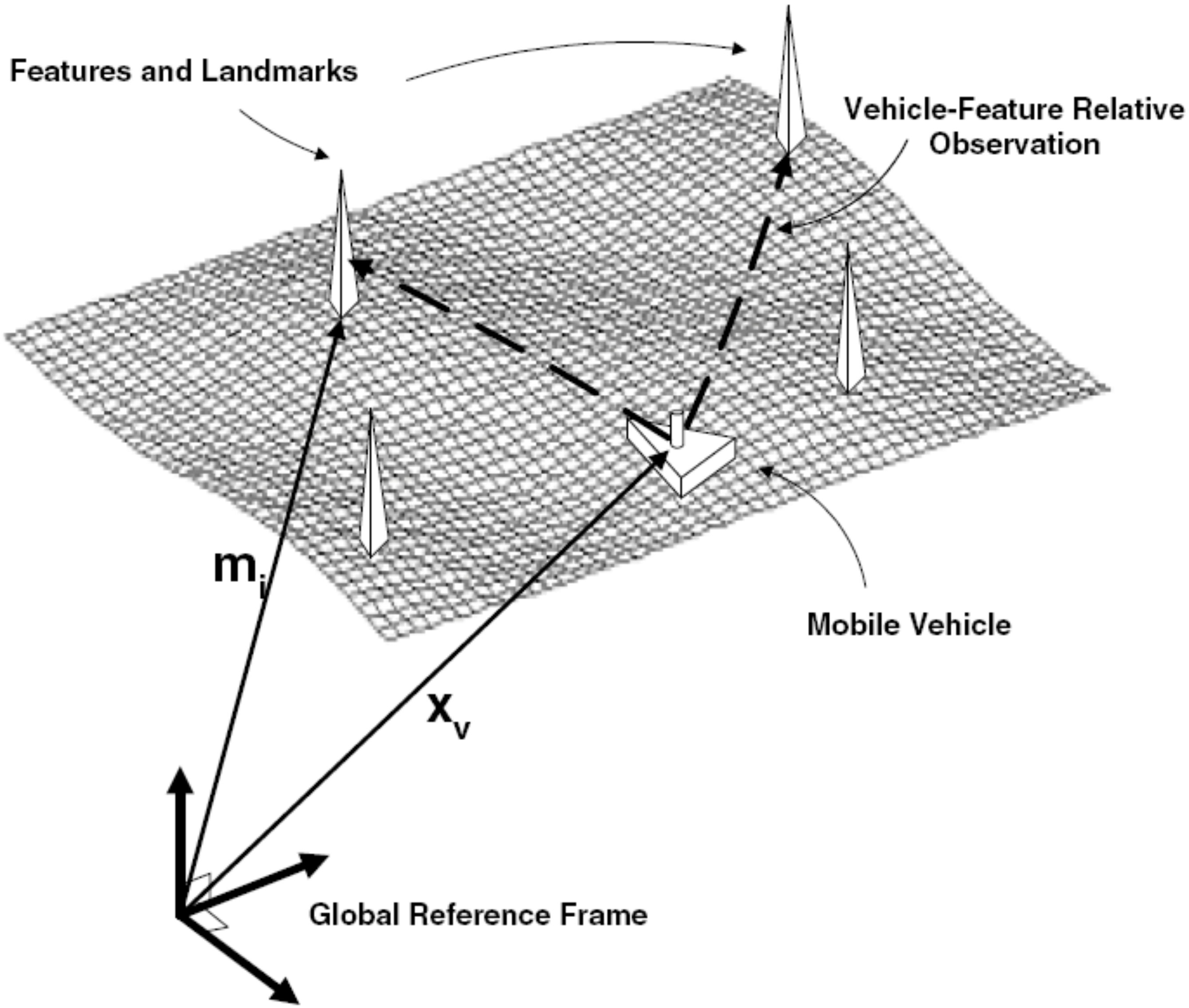
So far:

- Vertices for robot poses (x, y, θ)
- Edges for virtual observations between robot poses $\boldsymbol{z_{ij}} = < (x, y, \theta)^T_{ij}, \Omega_{ij} >$.

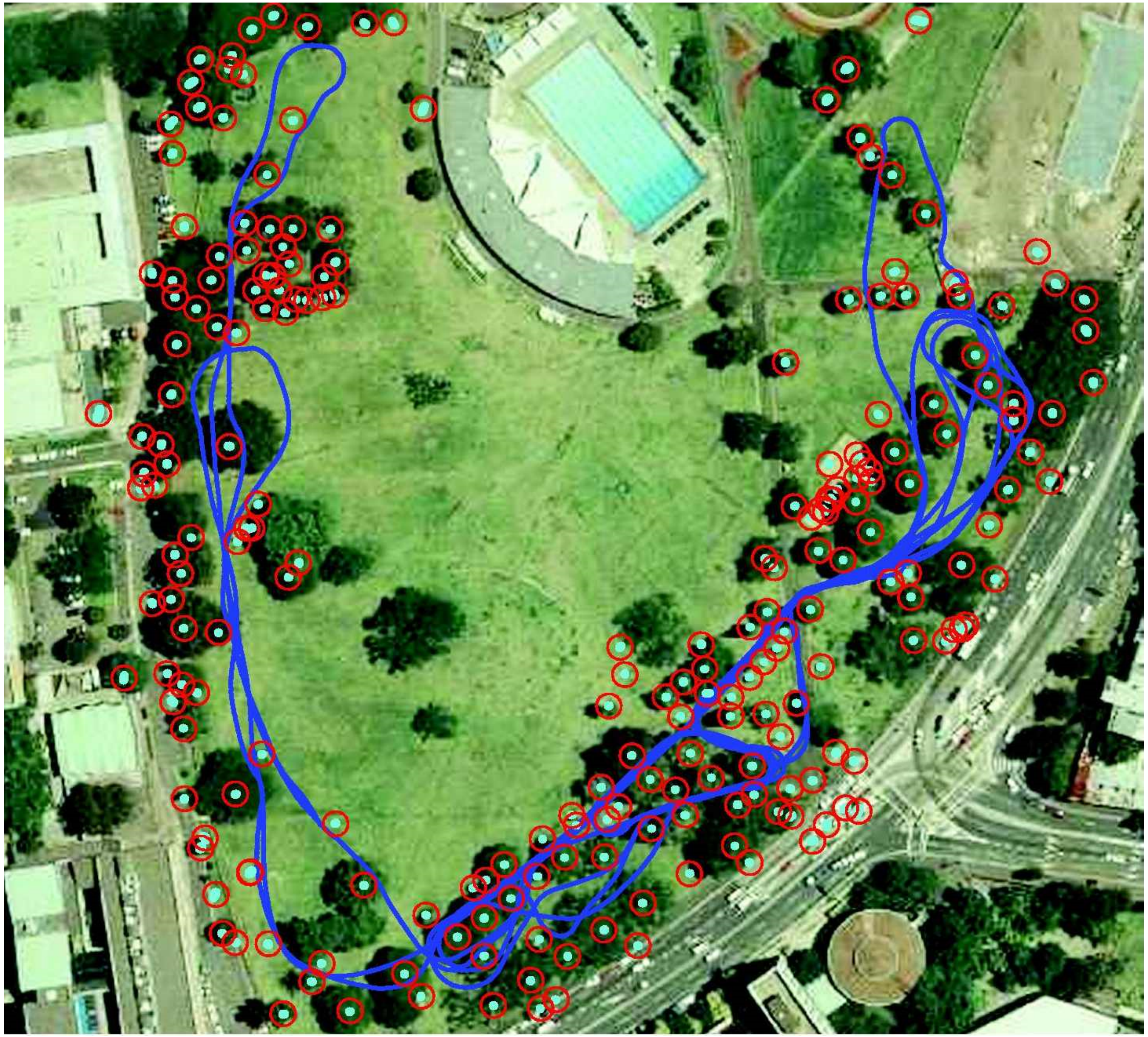
Topic today:

- How to deal with landmarks

Landmark-Based SLAM

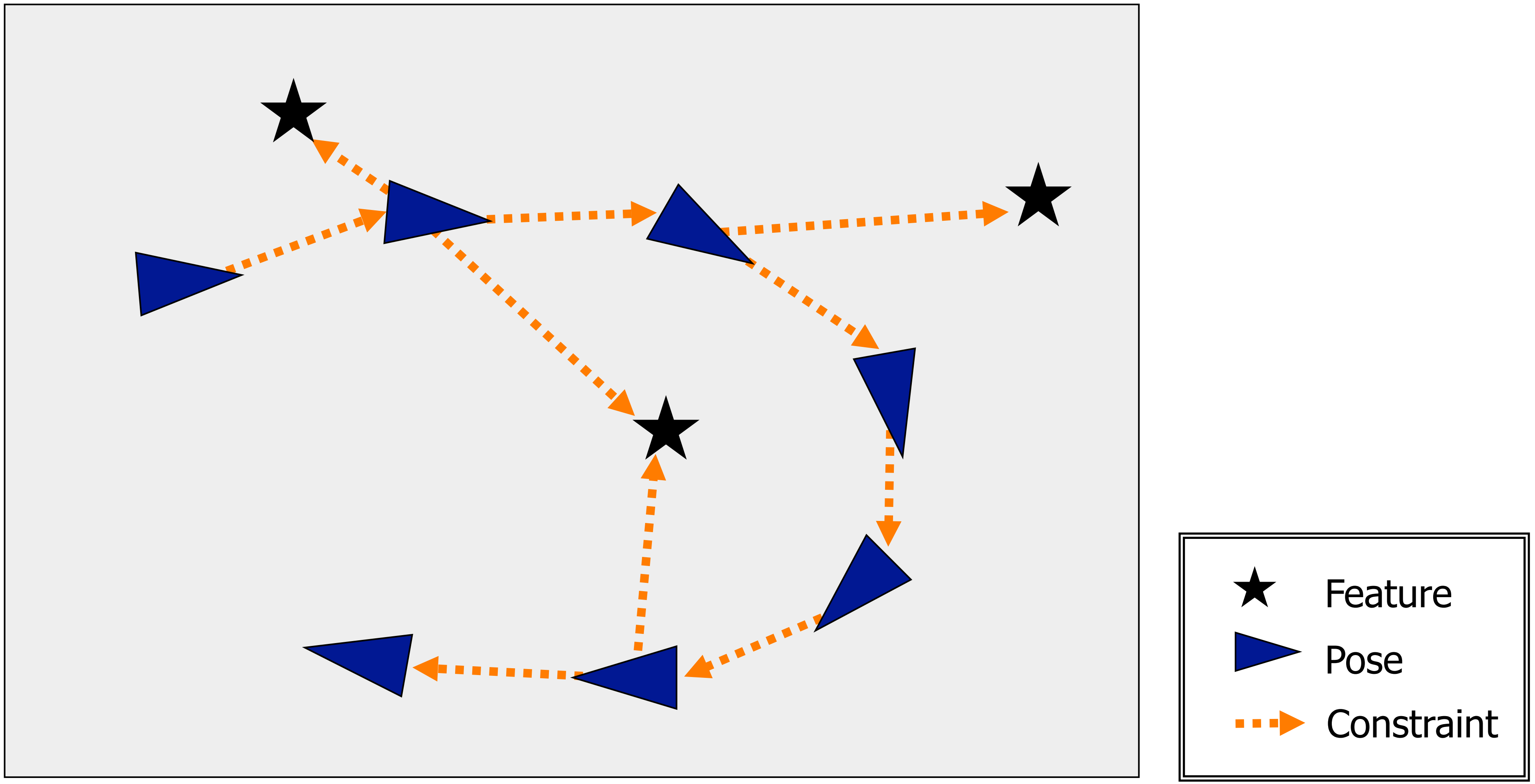


Real Landmark Map Example



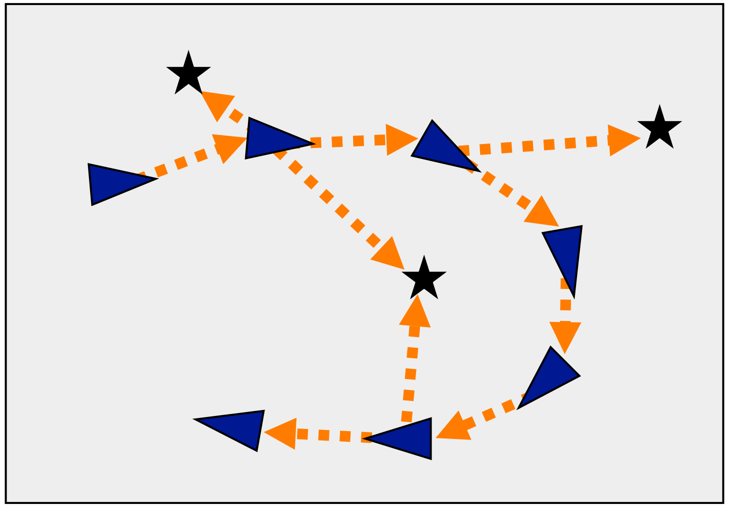
[courtesy by E. Nebot]

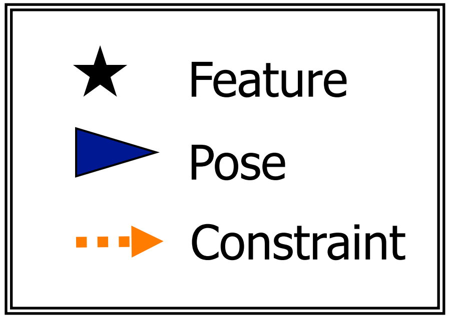
The Graph with Landmarks



The Graph with Landmarks

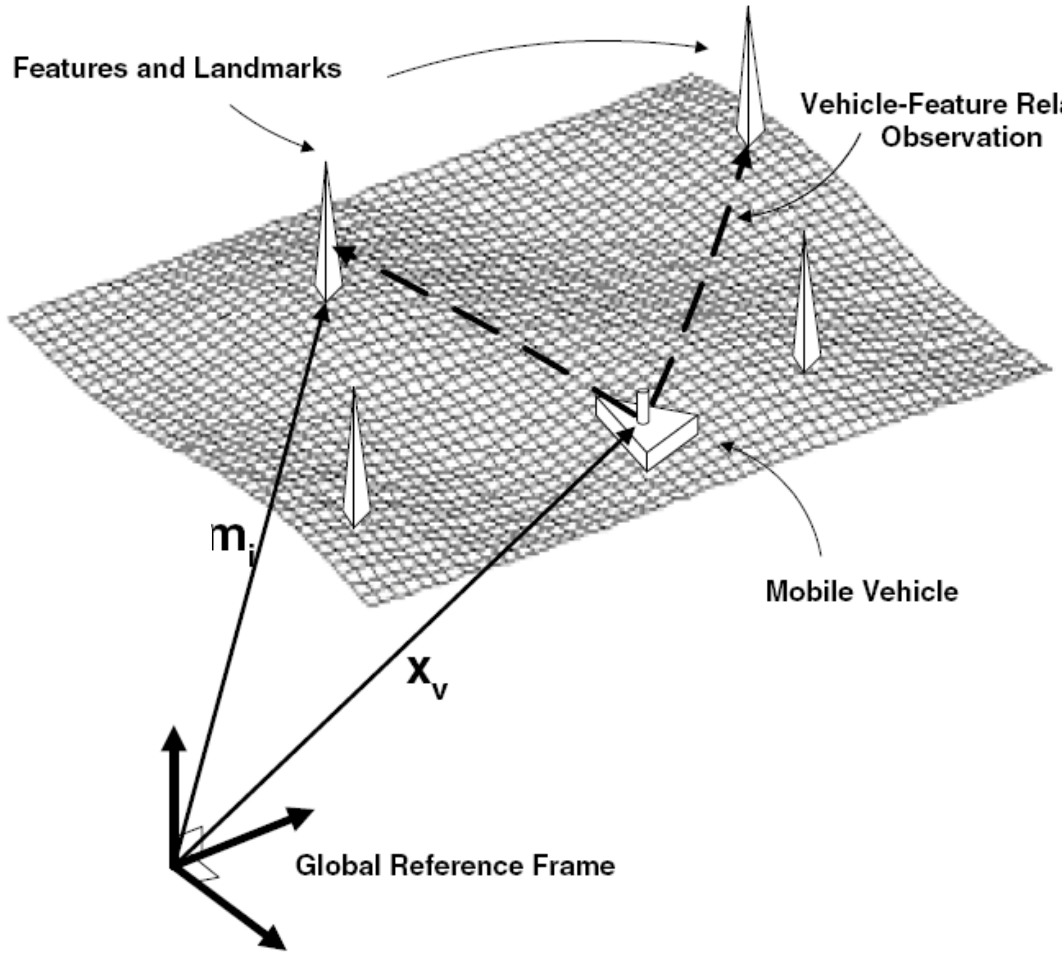
- **Nodes** can represent:
  - Robot poses
  - Landmark locations
- **Edges** can represent:
  - Landmark observations
  - Odometry measurements
- The minimization optimizes the landmark locations and robot poses





2D Landmarks

- A landmark is a 2D point in the world *(x,y)*
- Relative observation



Landmarks Observation

- Expected observation

$$\underset{\boxed{\text{Robot}}\quad\boxed{\text{Landmark}}}{\widehat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)} = \mathbf{R}_i^T(\mathbf{x}_j - \underbrace{\mathbf{t}_i}_{\boxed{\text{Robot translation}}})$$

Landmarks Observation

- Expected observation

$$\hat{\mathbf{z}}_{ij}(\underbrace{\mathbf{x}_i}_{\text{Robot}}, \underbrace{\mathbf{x}_j}_{\text{Landmark}}) \quad = \quad \mathbf{R}_i^T(\mathbf{x}_j - \underbrace{\mathbf{t}_i}_{\text{Robot translation}})$$

- Error function

$$\begin{aligned} \mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) &= \hat{\mathbf{z}}_{ij} - \mathbf{z}_{ij} \\ &= \mathbf{R}_i^T(\mathbf{x}_j - \mathbf{t}_i) - \mathbf{z}_{ij} \end{aligned}$$

Bearing Only Observations

- A landmark is still a 2D point
- The robot observe only the bearing (orientation towards the landmark)
- Observation function

$$\hat{\mathbf{z}}_{ij}(\mathbf{x}_i,\mathbf{x}_j) \quad = \quad \mathrm{atan}\frac{(\mathbf{x}_j-\mathbf{t}_i).y}{(\mathbf{x}_j-\mathbf{t}_i).x} - \theta_i$$

Robot

Landmark

Robot-landmark angle

Robot orientation

Bearing Only Observations

- Observation function

$$\hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \operatorname{atan}\frac{(\mathbf{x}_j - \mathbf{t}_i).y}{(\mathbf{x}_j - \mathbf{t}_i).x} - \theta_i$$

Robot

Landmark

Robot-landmark angle

Robot orientation

- Error function

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \operatorname{atan}\frac{(\mathbf{x}_j - \mathbf{t}_i).y}{(\mathbf{x}_j - \mathbf{t}_i).x} - \theta_i - \mathbf{z}_j$$

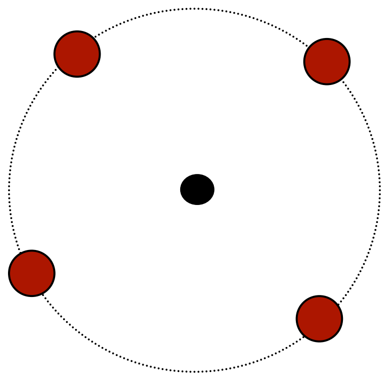
The Rank of the Matrix H

- What is the rank of the matrix H of a 2D landmark-pose constraint?
  - The blocks of the Jacobian are a 2x3 matrices
  - ***H*** cannot have more than rank 2
    ($\mathrm{rank}(A^T A) = \mathrm{rank}(A^T) = \mathrm{rank}(A)$)

- What is the rank of the matrix H for a bearing-only constraint?
  - The blocks of the Jacobian are a 1x3 matrices
  - ***H*** has rank=1

Where is the Robot?

- The robot observes one landmark (x-y)
- Where can the robot be?

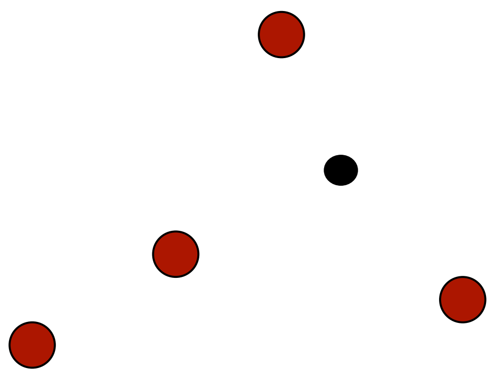


The robot can be somewhere on a circle around the landmark

It is a 1D solution space (constraint on the distance and the robot's orientation)

Where is the Robot?

- The robot observes one landmark (bearing-only)
- Where can the robot be?



The robot can be anywhere in the x-y plane

It is a 2D solution space (constraint on the robot's orientation)

Rank

- In landmark-based SLAM, the system can be under-determined
- The rank of ***H*** is **at most** equal to the sum of the ranks of the constraints
- Looking at the rank:
  - How many 2D landmark observations are needed to resolve for a robot pose?
  - How many bearing-only observations are needed to resolve for a robot pose?
- To determine a **unique solution**, the system should have **full rank**

Under-determined Systems

- No guarantee for a system with full rank
  - Landmarks may be observed only once
  - The robot might have no odometry
- We can still deal with these situations by adding a "damping" factor to $\boldsymbol{H}$.
  - Instead of solving $\boldsymbol{H}\,\boldsymbol{\Delta x} = -\boldsymbol{b}$, we solve

$$(\boldsymbol{H} + \lambda \boldsymbol{I})\,\boldsymbol{\Delta x} = -\boldsymbol{b}$$

What is the effect of that?

$(\boldsymbol{H} + \lambda \boldsymbol{I})\, \boldsymbol{\Delta x} = \boldsymbol{-b}$

- Damping factor for $\boldsymbol{H}$
- $(\boldsymbol{H} + \lambda \boldsymbol{I})\, \boldsymbol{\Delta x} = \boldsymbol{-b}$ instead of $\boldsymbol{H}\, \boldsymbol{\Delta x} = \boldsymbol{-b}$
- The damping factor $\lambda \boldsymbol{I}$ makes the system positive definite
- It adds an additional constraints that "drag" the increments towards 0.
- What happens when $\lambda \gg |\boldsymbol{H}|$?

Simplified Levenberg Marquardt

- Damping to regulate the convergence using backup/restore actions

```
x: the initial guess
while (! converged)
    λ = λ_init
   <H,b> = buildLinearSystem(x);
   E = error(x)
   x_old = x;
   Δx  = solveSparse( (H + λ I) Δx = -b);
   x += Δx;
   If (E < error(x)){
      x  = x_old;
      λ *= 2;
   } else { λ */ 2; }
```

Fixing a Subset of Variables

- Assume that the value of certain variables during the optimization is known a priori
- We may want to optimize all others and keep these fixed
- How?

Fixing a Subset of Variables

- Assume that the value of certain variables during the optimization is known a priori
- We may want to optimize all others and keep these fixed
- How?
- If a variable is not optimized, it should “disappears” from the linear system

Fixing a Subset of Variables

- Assume that the value of certain variables during the optimization is known a priori
- We may want to optimize all others and keep these fixed
- How?
- If a variable is not optimized, it should “disappears” from the linear system
- Construct the full system
- Suppress the rows and the columns corresponding to the variables to fix

Uncertainty

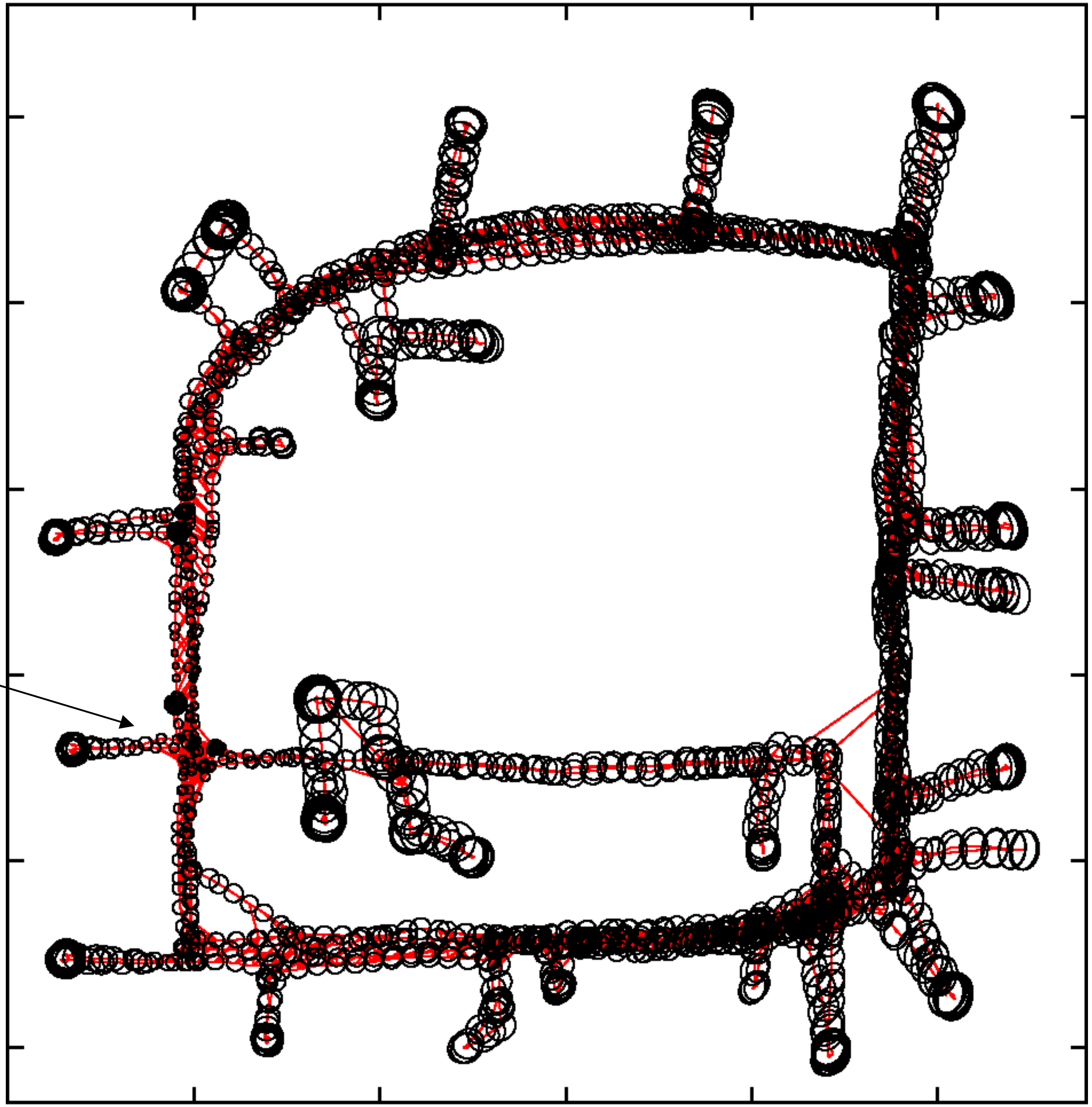
- ***H*** represents the inverse covariance of the likelihood around the linearization point
- Inverting ***H*** gives the covariance matrix (which is dense)
- The diagonal blocks of the covariance matrix represent the (absolute) uncertainties of the corresponding variables

Relative Uncertainty

To determine the relative uncertainty between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$:

- Construct the full matrix $\boldsymbol{H}$
- Suppress the rows and the columns of $\boldsymbol{x}_i$ (fix it)
- Compute the *j,j* block of the inverse
- This block will contain the covariance matrix of $\mathbf{x}_j$ w.r.t. $\boldsymbol{x}_i$, which has been fixed

Example



robot

You Should have Learned…

- How to incorporate landmarks in the map
- How to embed prior knowledge about the position of some parts of the map
- How to determine the relative uncertainties