From Least-Squares to ICP

Giorgio Grisetti
grisetti@dis.uniroma1.it

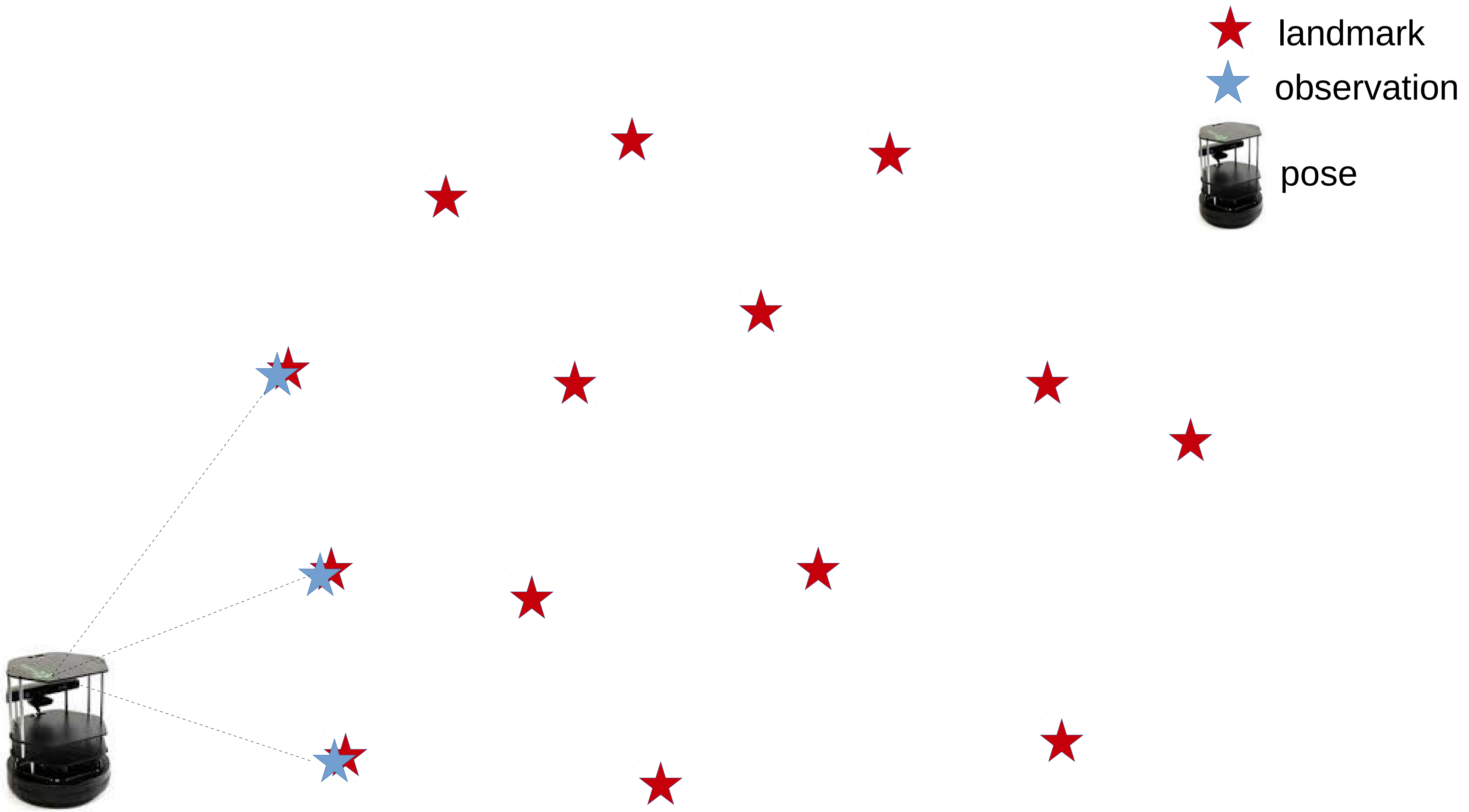
Dept of Computer Control and Management Engineering
Sapienza University of Rome

Download Code Here

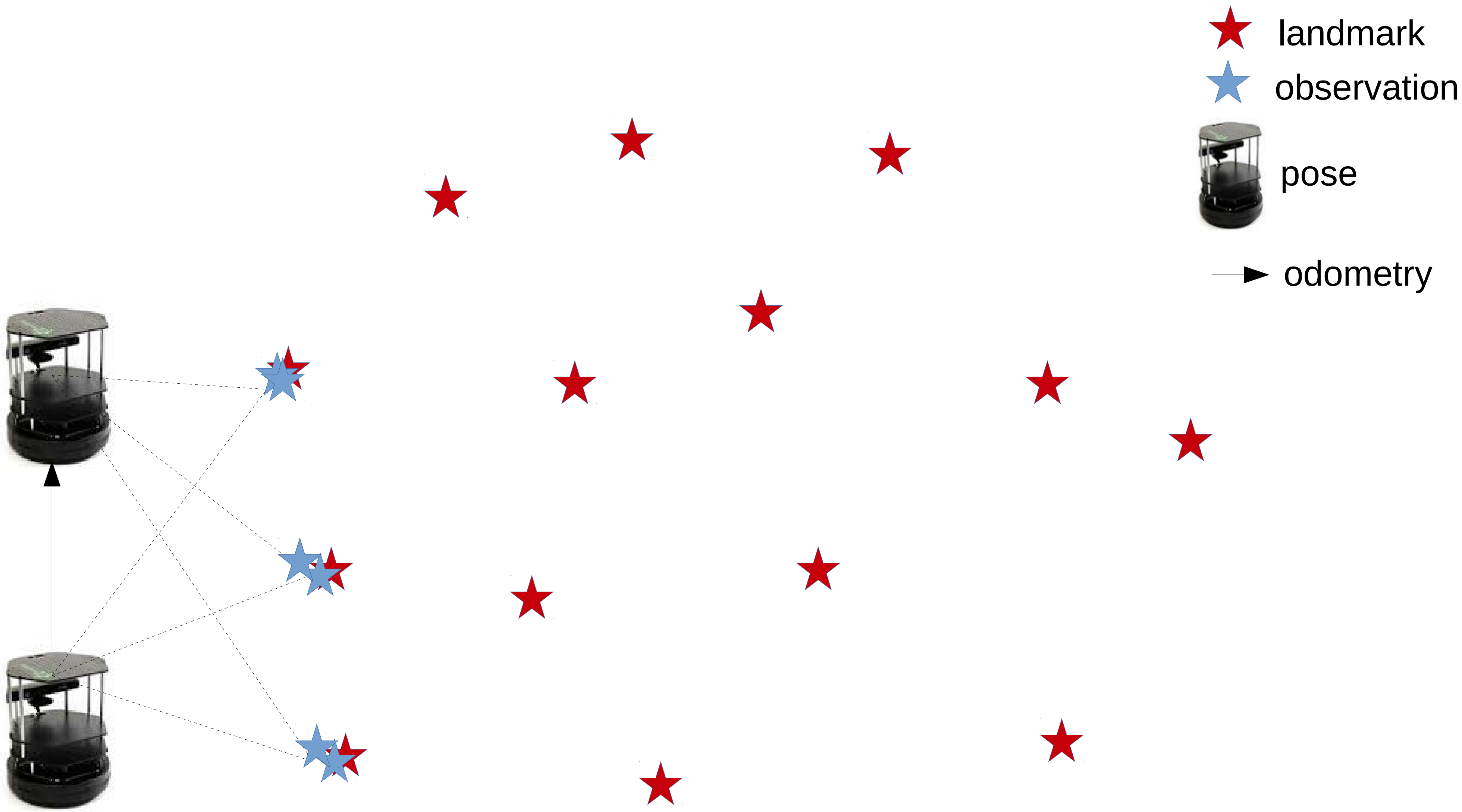
http://www.dis.uniroma1.it/~labrococo/tutorial_icra_2016/ICP_3D.tgz

Special thanks to **Ulrich Wollath** for reporting errors in the early version of slides and code

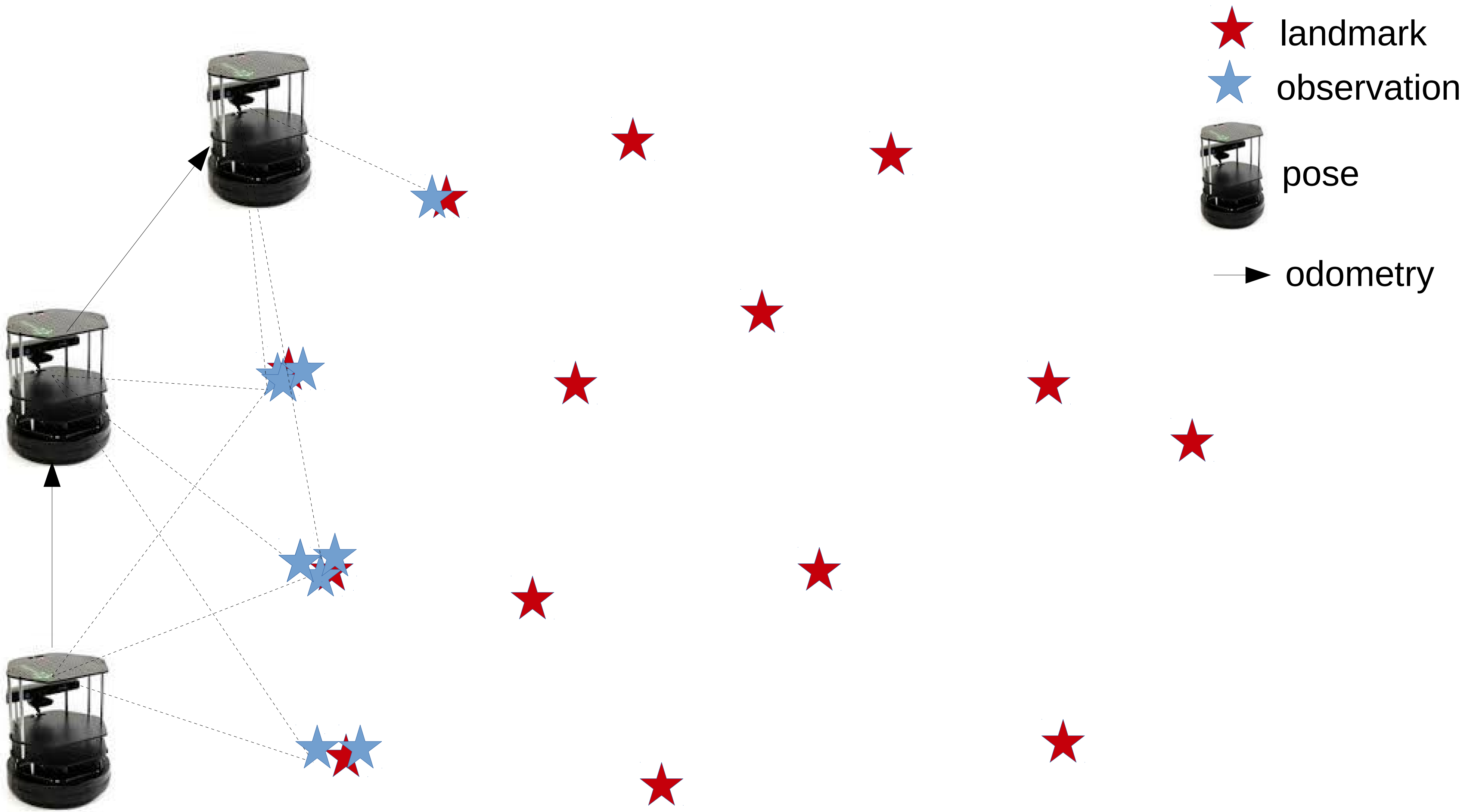
SLAM as Estimation



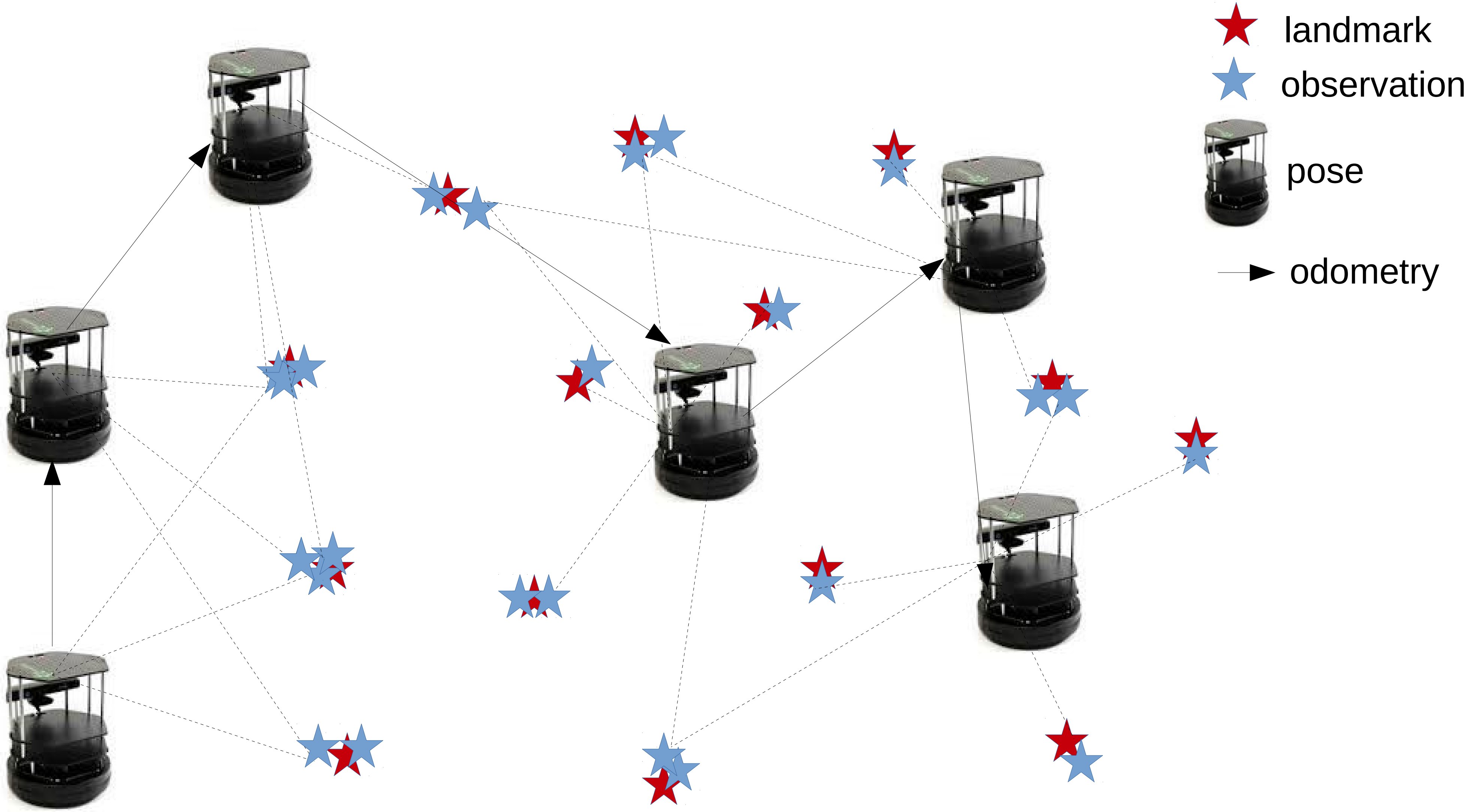
SLAM as Estimation



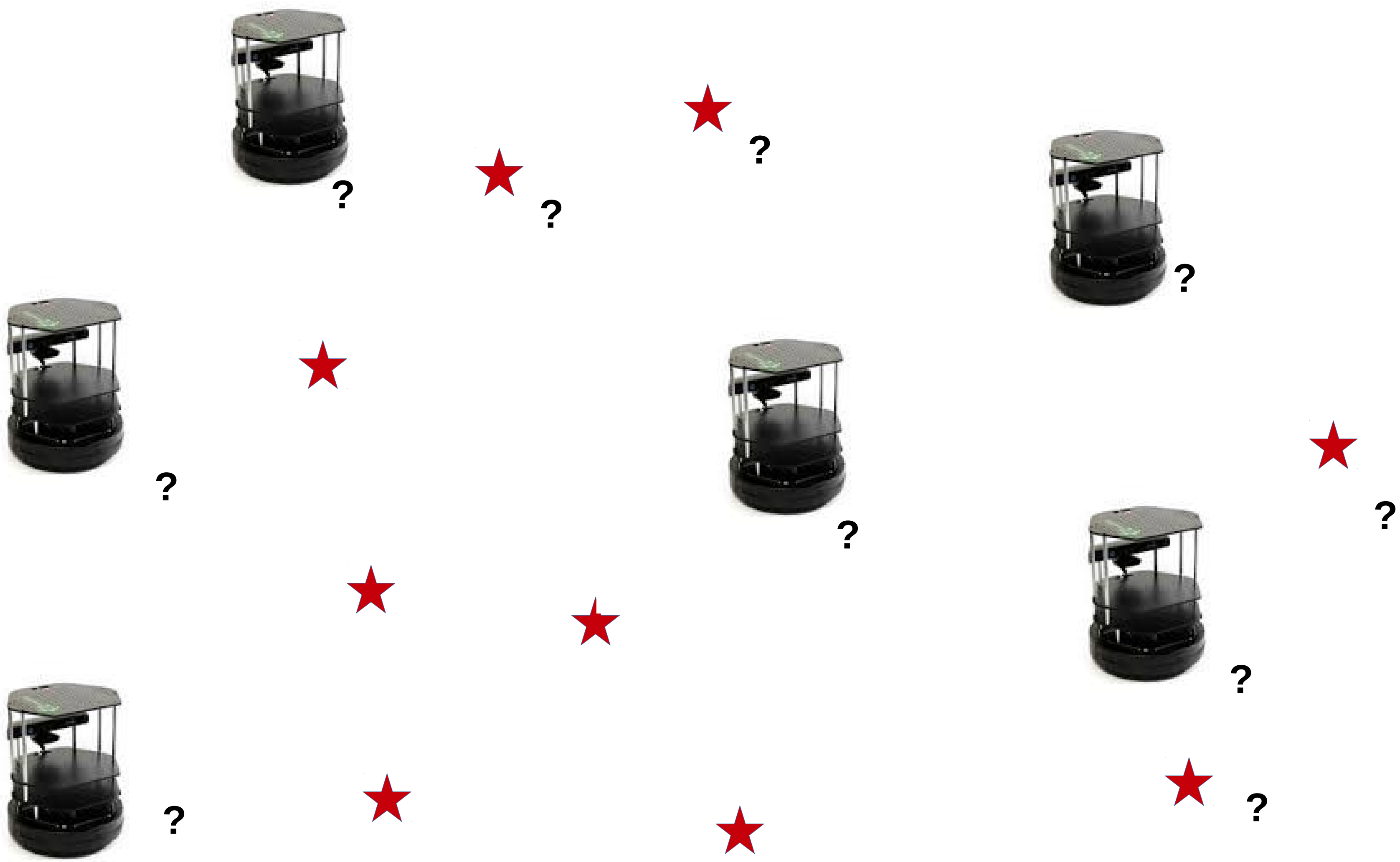
SLAM as Estimation



SLAM as Estimation

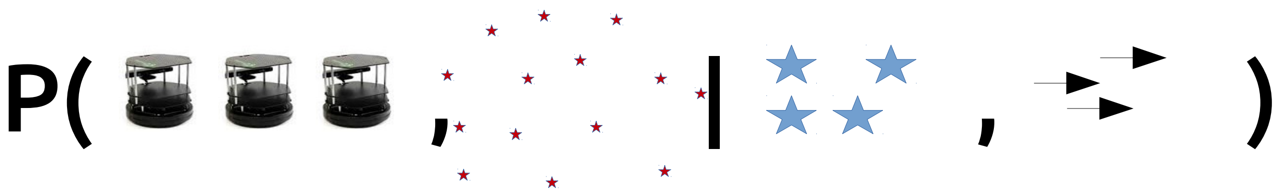


SLAM as Estimation



Maximum Likelihood Estimation

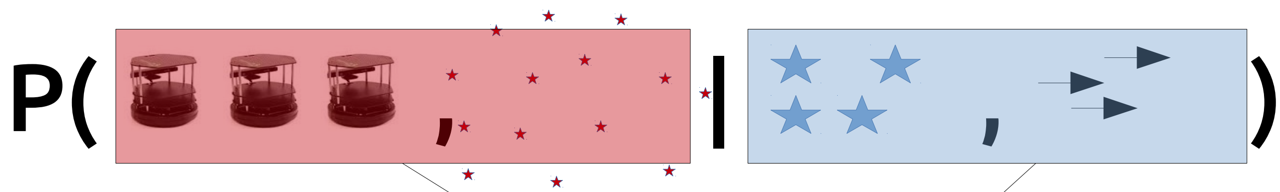
Estimate



$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x}|\mathbf{z})$$

Maximum Likelihood Estimation

Estimate



$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x} | \mathbf{z})$$

Measurements

State

x*: state most consistent with observations

Maximum Likelihood Estimation

Using

$$\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}} p(\mathbf{x}|\mathbf{z})$$

- Bayes' Rule

$$p(\mathbf{x}|\mathbf{z}) = \frac{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{z})}$$

$$\propto p(\mathbf{z}|\mathbf{x})$$

- Independence,

$$= \prod_i p(\mathbf{z}_i|\mathbf{x})$$

We can further simplify the task

Maximum Likelihood Estimation

Using

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{x}|\mathbf{z})$$

- Bayes' Rule

$$p(\mathbf{x}|\mathbf{z}) = \frac{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{z})}$$

$$\propto p(\mathbf{z}|\mathbf{x})$$

- Independence,

$$= \prod_i p(\mathbf{z}_i|\mathbf{x})$$

We can further simplify the task

Gaussian Assumption

Measurements affected by Gaussian noise

$$
\begin{aligned}
p(\mathbf{z}_i|\mathbf{x}) &= \mathcal{N}(\mathbf{z}_i; \mathbf{h}_i(\mathbf{x}), \mathbf{\Sigma}_i) \\
&\propto \exp\left(-\underbrace{(\mathbf{h}_i(\mathbf{x}) - \mathbf{z}_i)}_{\mathbf{e}_i(\mathbf{x})}{}^T \underbrace{\mathbf{\Sigma}^{-1}}_{\mathbf{\Omega}_i}(\mathbf{h}_i(\mathbf{x}) - \mathbf{z}_i)\right)
\end{aligned}
$$

Gaussian Assumption

Measurements affected by Gaussian noise

$$p(\mathbf{z}_i|\mathbf{x}) = \mathcal{N}(\mathbf{z}_i; \mathbf{h}_i(\mathbf{x}), \mathbf{\Sigma}_i)$$

$$\propto \exp\left(-\underbrace{(\mathbf{h}_i(\mathbf{x}) - \mathbf{z}_i)}_{\mathbf{e}_i(\mathbf{x})}^T \underbrace{\mathbf{\Sigma}^{-1}}_{\mathbf{\Omega}_i}(\mathbf{h}_i(\mathbf{x}) - \mathbf{z}_i)\right)$$

measurement ($\mathbf{z}_i$)

prediction ($\mathbf{h}_i$)

state ($\mathbf{x}$)

error function ($\mathbf{e}_i(\mathbf{x})$)

Gaussian Assumption

Through Gaussian assumption

- Maximization becomes minimization
- Product turns into sum

$$
\begin{aligned}
\mathbf{x}^* &= \underset{x}{\operatorname{argmax}} \prod_i p(\mathbf{z}_i | \mathbf{x}) \\
&= \underset{x}{\operatorname{argmax}} \prod_i \exp(-\mathbf{e}_i(\mathbf{x})^T \mathbf{\Omega}_i \mathbf{e}_i(\mathbf{x})) \\
&= \underset{x}{\operatorname{argmin}} \sum_i \mathbf{e}_i(\mathbf{x})^T \mathbf{\Omega}_i \mathbf{e}_i(\mathbf{x})
\end{aligned}
$$

Gauss Method Overview

Iterative minimization of

$$F(\mathbf{x}) = \sum_i \mathbf{e}_i(\mathbf{x})^T \boldsymbol{\Omega}_i \mathbf{e}_i(\mathbf{x})$$

Each iteration refines the current estimate by applying a perturbation

$$\mathbf{x} \leftarrow \mathbf{x} + \boldsymbol{\Delta}\mathbf{x}$$

Perturbation obtained by minimizing a quadratic approximation of the problem in $\boldsymbol{\Delta}\mathbf{x}$

$$F(\mathbf{x} + \boldsymbol{\Delta}\mathbf{x}) \simeq \boldsymbol{\Delta}\mathbf{x}^T \mathbf{H} \boldsymbol{\Delta}\mathbf{x} + 2\mathbf{b}^T \boldsymbol{\Delta}\mathbf{x} + c$$

Linearization

The quadratic approximation is obtained by linearizing the error functions around $\mathbf{x}$

$$\mathbf{e}_i(\mathbf{x}^* + \mathbf{\Delta x}) \quad \simeq \quad \underbrace{\mathbf{e}_i(\mathbf{x}^*)}_{\mathbf{e}} + \underbrace{\left.\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial(\mathbf{x})}\right|_{\mathbf{x}=\mathbf{x}^*}}_{\mathbf{J}_i} \mathbf{\Delta x}$$

...expanding the products

$$\mathbf{e}_i(\mathbf{x}^* + \mathbf{\Delta x})^T \mathbf{\Omega}_i \mathbf{e}_i(\mathbf{x}^* + \mathbf{\Delta x}) \simeq$$

$$\mathbf{\Delta x}^T \underbrace{\mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{J}_i}_{\mathbf{H}_i} \mathbf{\Delta x} + 2 \underbrace{\mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{e}_i}_{\mathbf{b}_i^T} \mathbf{\Delta}_x + \underbrace{\mathbf{e}_i^T \mathbf{\Omega}_i \mathbf{e}_i}_{c_i}$$

...and grouping the terms

$$\mathbf{H} = \sum_i \mathbf{H}_i \qquad \mathbf{b} = \sum_i \mathbf{b}_i \qquad c = \sum_i c_i$$

Quadratic form

Find the $\Delta \mathrm{x}$ that minimizes the quadratic approximation of the objective function

$$
\begin{aligned}
\mathbf{\Delta x}^* &= \underset{\mathbf{\Delta x}}{\operatorname{argmin}} F(\mathbf{x}^* + \mathbf{\Delta x}) \\
&\simeq \underset{\mathbf{\Delta x}}{\operatorname{argmin}} \mathbf{\Delta x}^T \mathbf{H} \mathbf{\Delta x} + 2\mathbf{b}^T \mathbf{\Delta x} + c
\end{aligned}
$$

Find $\Delta \mathrm{x}$ that nulls the derivative of quadratic form

$$
\begin{aligned}
\mathbf{0} &= \frac{\partial \left[\mathbf{\Delta x}^T \mathbf{H} \mathbf{\Delta x} + 2\mathbf{b}^T \mathbf{\Delta x} + c \right]}{\partial \mathbf{\Delta x}} \\
-\mathbf{b} &= \mathbf{H} \mathbf{\Delta x}
\end{aligned}
$$

Algorithm (one Iteration)

Clear **H** and **b**

$$\mathbf{H} \leftarrow 0 \qquad \mathbf{b} \leftarrow 0$$

For each measurement, update h and b

$$\begin{aligned}
\mathbf{e}_i &\leftarrow \mathbf{h}_i(\mathbf{x}^*) - \mathbf{z}_i \\
\mathbf{J}_i &\leftarrow \left.\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}^*} \\
\mathbf{H} &\leftarrow \mathbf{H} + \mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{J}_i \\
\mathbf{b} &\leftarrow \mathbf{b} + \mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{e}_i
\end{aligned}$$

Update the estimate with the perturbation

$$\begin{aligned}
\mathbf{\Delta x} &\leftarrow \text{solve}(\mathbf{H}\mathbf{\Delta x} = -\mathbf{b}) \\
\mathbf{x}^* &\leftarrow \mathbf{x}^* + \mathbf{\Delta x}
\end{aligned}$$

Methodology

Identify the state space **X**

- Qualify the domain
- Find a locally Euclidean parameterization

Identify the measurement space(s) **Z**

- Qualify the domain
- Find a locally Euclidean parameterization

Identify the prediction functions **h(x)**

Gauss-Newton in SLAM

Typical problems where GN is used

- Calibration
- Registration
  - Cloud to Cloud (ICP)
  - Image to Cloud (Posit)
- Global Optimization
  - Pose-SLAM
  - Bundle Adjustment

Warning

- **Data association is assumed to be known known**
- **Gauss-Newton alone is not sufficient to solve a full problem**
- **One needs a strategy to compute data association**

Gauss-Newton in SLAM

Typical problems where GN is used

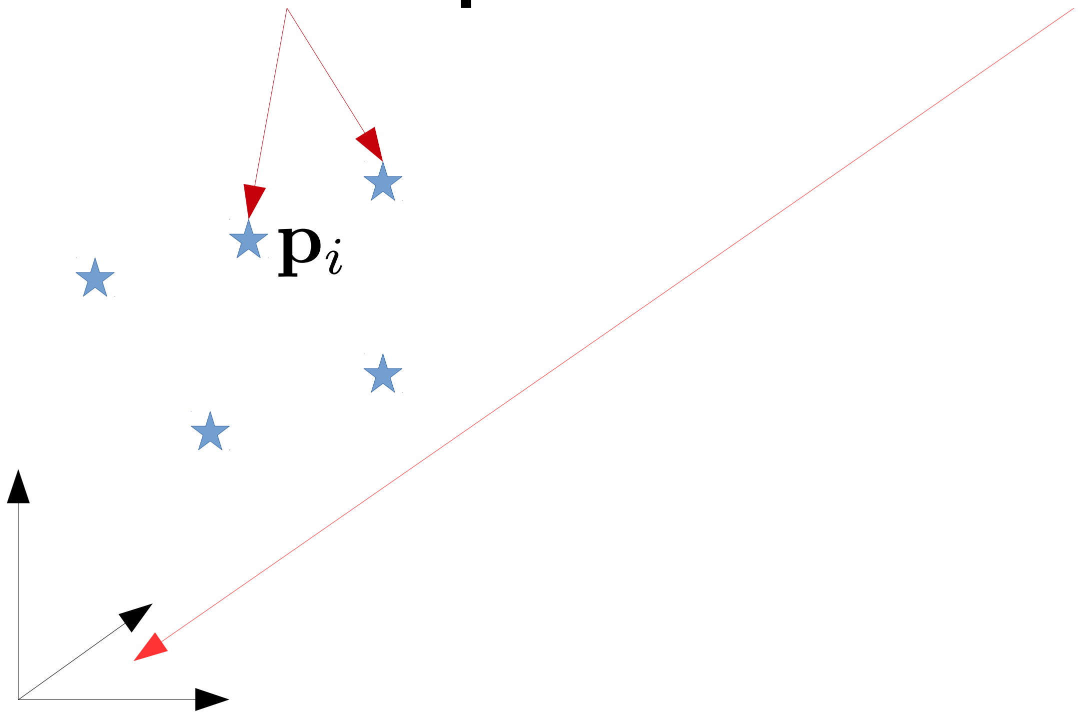
- Calibration
- Registration
  - Cloud to Cloud (ICP)
  - Image to Cloud (Posit)
- Global Optimization
  - Pose-SLAM
  - Bundle Adjustment

Warning

- **Data association is assumed to be known known**
- **Gauss-Newton alone is not sufficient to solve a full problem**
- **One needs a strategy to compute data association**

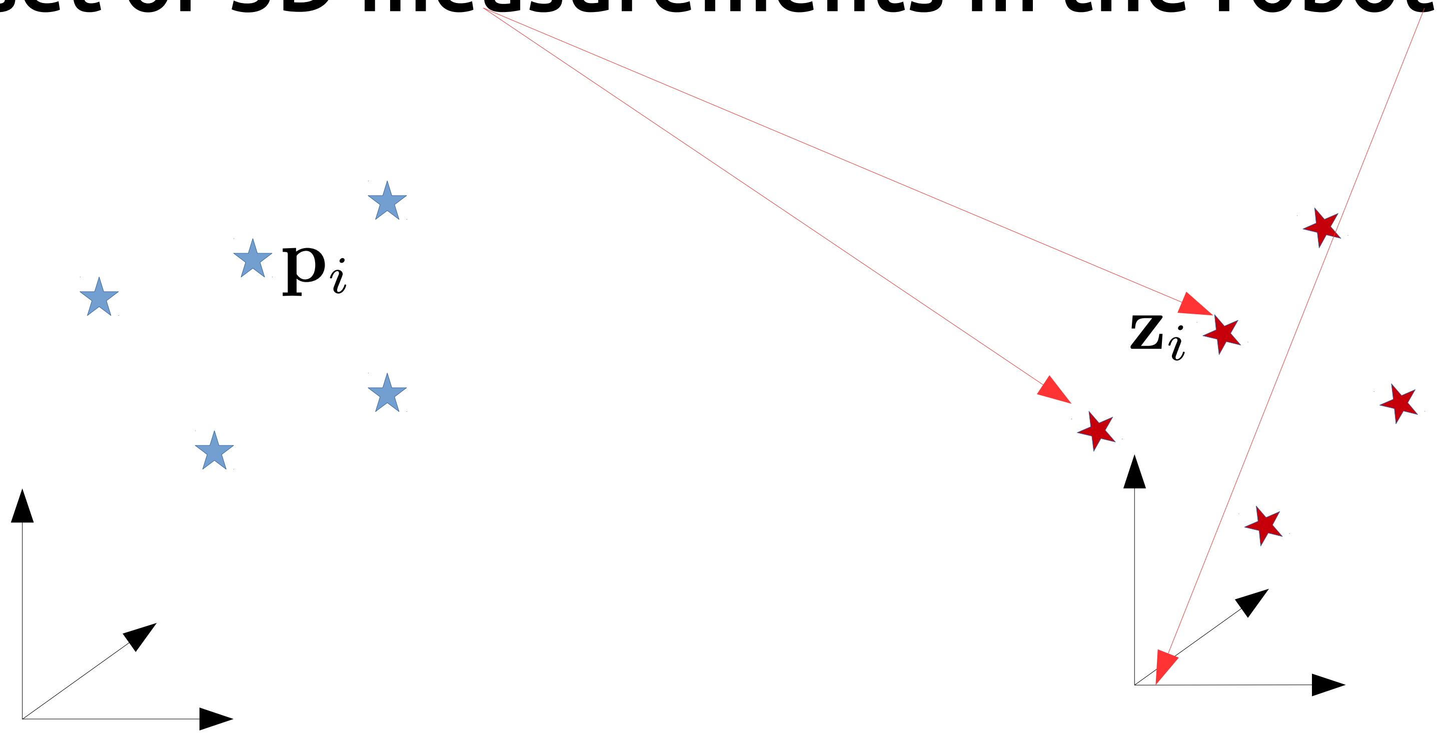
Example ICP Optimization in 3D

Given a set of points in the world frame



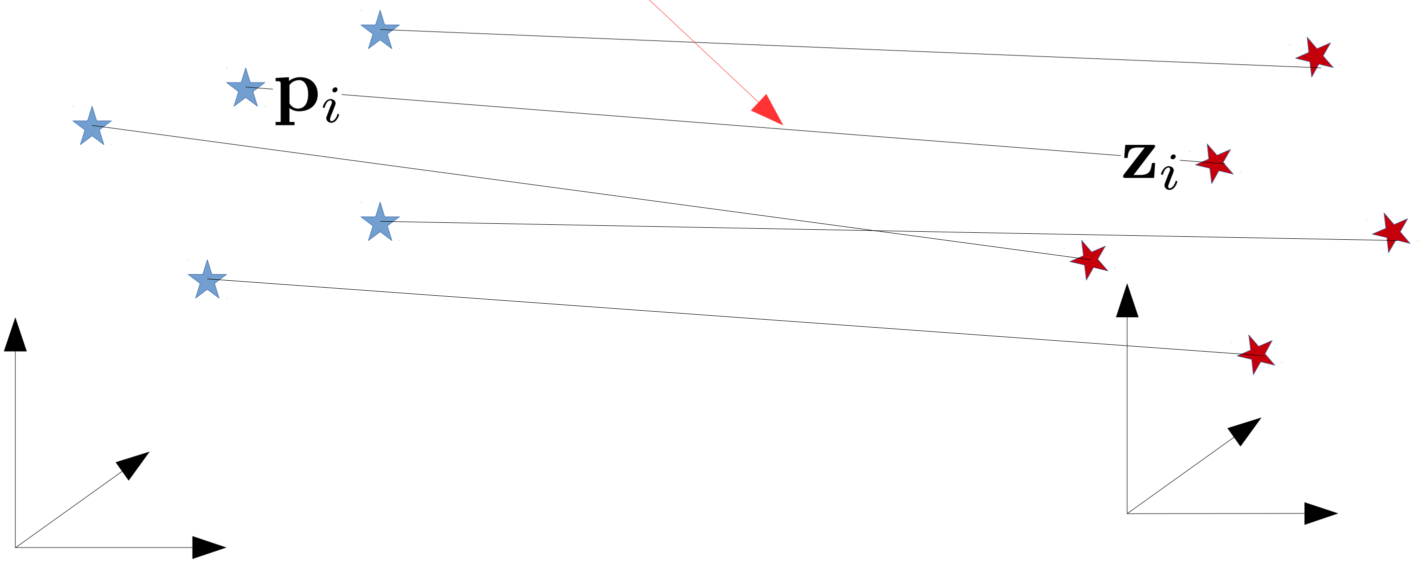
Example ICP Optimization in 3D

A set of 3D measurements in the robot frame



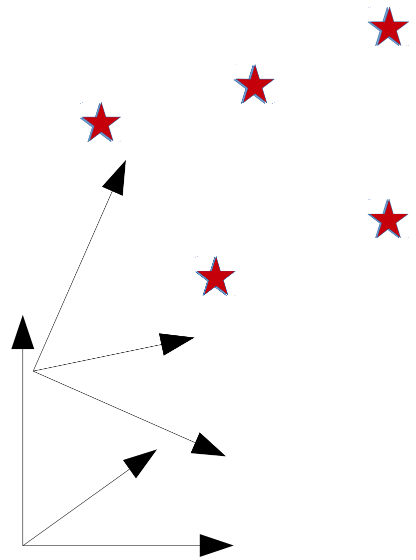
Example ICP Optimization in 3D

Roughly known correspondences



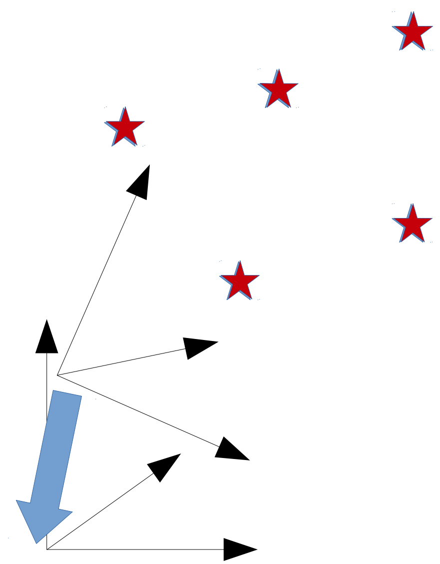
Example ICP Optimization in 3D

We want to find a transform that minimizes distance between corresponding points



Example ICP Optimization in 3D

Such a transform will be the pose of world w.r.t. robot



Note: we can also estimate robot w.r.t world, but it leads to longer calculations

ICP: State and Measurements

State

$$\mathbf{x} \in SE(3)$$

$$\mathbf{x} = (\underbrace{x\ y\ z}_{\mathbf{t}}\ \underbrace{\alpha_x\ \alpha_y\ \alpha_z}_{\alpha})^T$$

Measurements

$$\mathbf{z} \in \Re^3$$

$$\mathbf{h}_i(\mathbf{x}) = \mathbf{R}(\alpha)\mathbf{p}_i + \mathbf{t}$$

On Rotation Matrices

A rotation is obtained by composing the rotations along *x-y-z*

$$\mathbf{R}(\alpha) \quad = \quad \mathbf{R}_x(\alpha_x)\mathbf{R}_y(\alpha_y)\mathbf{R}_z(\alpha_z)$$

Small lookup of rotations (and derivatives)

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{pmatrix} \quad \mathbf{R}_y = \begin{pmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{pmatrix} \quad \mathbf{R}_z = \begin{pmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{R}'_x = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -s & -c \\ 0 & c & -s \end{pmatrix} \quad \mathbf{R}'_y = \begin{pmatrix} -s & 0 & c \\ 0 & 0 & 0 \\ -c & 0 & -s \end{pmatrix} \quad \mathbf{R}'_z = \begin{pmatrix} -s & -c & 0 \\ c & -s & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

ICP: Jacobian

$$\frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \mathbf{t}} \; \frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \alpha_x} \; \frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \alpha_y} \; \frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \alpha_z} \right)$$

$$\frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \mathbf{t}} = \mathbf{I}$$

$$\frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \alpha_x} = \mathbf{R}'_x \mathbf{R}_y \mathbf{R}_z \mathbf{p}_i$$

$$\frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \alpha_y} = \mathbf{R}_x \mathbf{R}'_y \mathbf{R}_z \mathbf{p}_i$$

$$\frac{\partial \mathbf{h}_i(\mathbf{x})}{\partial \alpha_z} = \mathbf{R}_x \mathbf{R}_y \mathbf{R}'_z \mathbf{p}_i$$

ICP: Octave Code

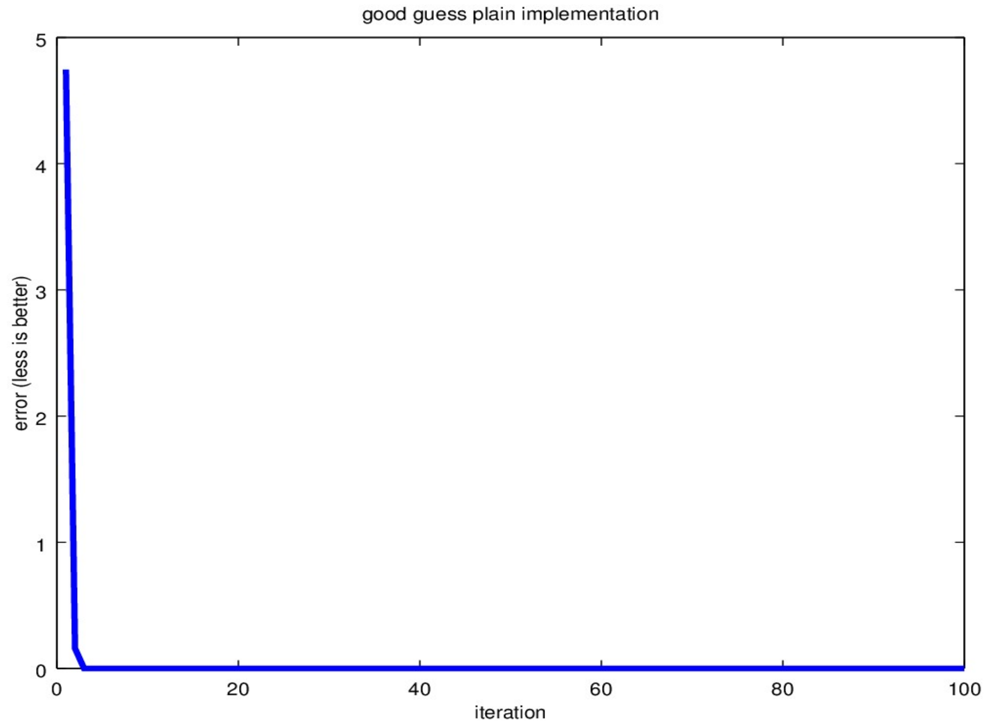
```
function [e,J]=errorAndJacobian(x,p,z)
  rx=Rx(x(4)); #rotation matrices at x
  ry=Ry(x(5));
  rz=Rz(x(6));
  rx_p=Rx_prime(x(4)); #derivatives at x
  ry_p=Ry_prime(x(5));
  rz_p=Rz_prime(x(6));
  t=x(1:3);

  z_hat=rx*ry*rz*p+t; #prediction
  e=z_hat-z;          #error
  J=zeros(3,6);       #jacobian
  J(1:3,1:3)=eye(3);  #translational part of jacobian


  J(1:3,4)=rx_p*ry*rz*p; #de/dax
  J(1:3,5)=rx*ry_p*rz*p; #de/day
  J(1:3,6)=rx*ry*rz_p*p; #de/daz
endfunction
```

ICP: Octave Code

```
function [x, chi_stats]=doICP(x_guess, P, Z, num_iterations)
  x=x_guess;
  chi_stats=zeros(1,num_iterations); #ignore this for now
  for (iteration=1:num_iterations)
    H=zeros(6,6);
    b=zeros(6,1);
    chi=0;
    for (i=1:size(P,2))
      [e,J] = errorAndJacobian(x, P(:,i), Z(:,i));
      H+=J'*J;
      b+=J'*e;
      chi+=e'*e;
    endfor
    chi_stats(iteration)=chi;
    dx=-H\b;
    x+=dx;
  endfor
endfunction
```

Testing, good initial guess

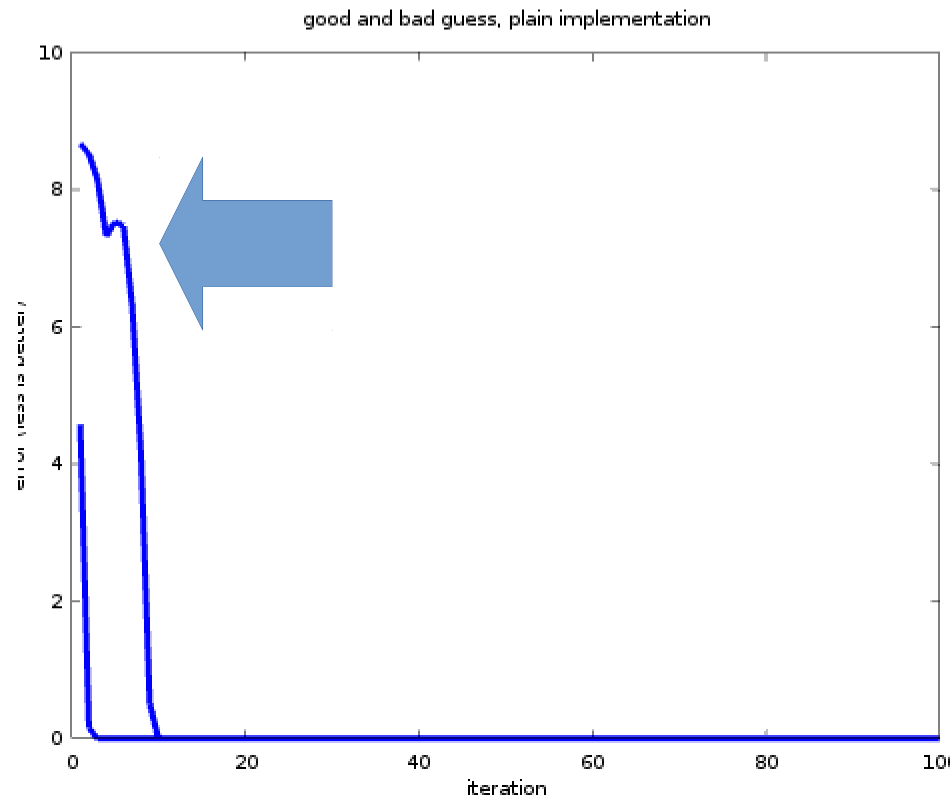
- Spawn a set of random points in 3D
- Define a location of the robot
- Compute synthetic measurements from that location
- Set the a point close to the true location as initial guess
- Run ICP and plot the evolution of the error



- When started from a good guess, the system converges nicely

Testing, bad initial guess

- Spawn a set of random points in 3D
- Define a location of the robot
- Compute synthetic measurements from that location
- Set the origin as initial guess
- Run ICP and plot the evolution of the error

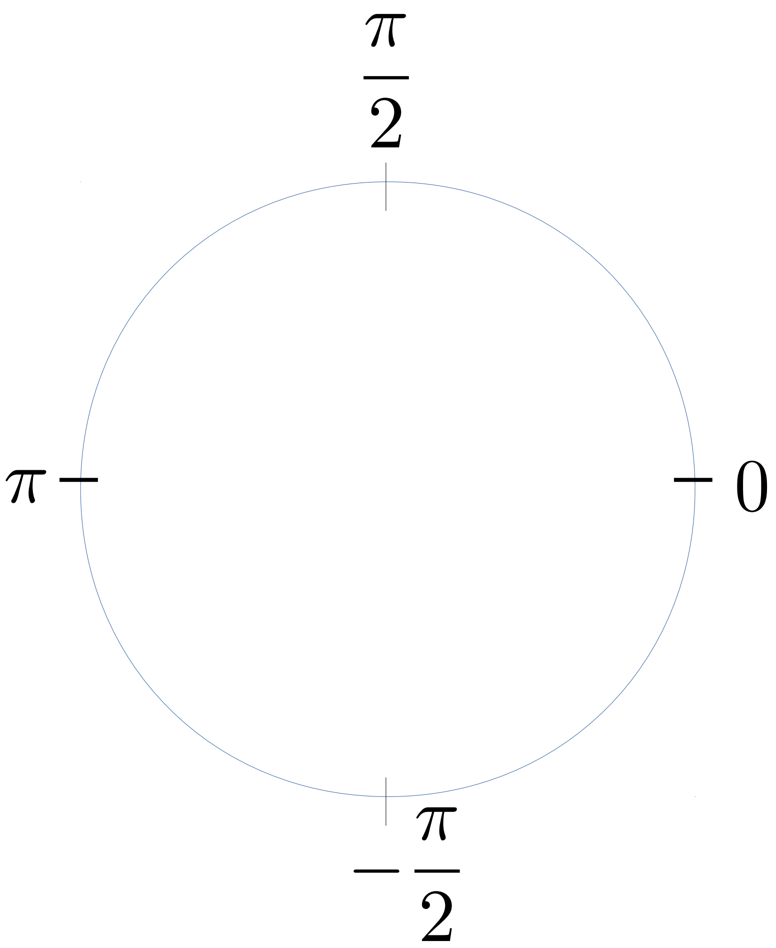


- If the guess is poor, the system might take long to converge
- The error might increase

Non-Euclidean Spaces

In SLAM we often encounter spaces that have a non-euclidean topology

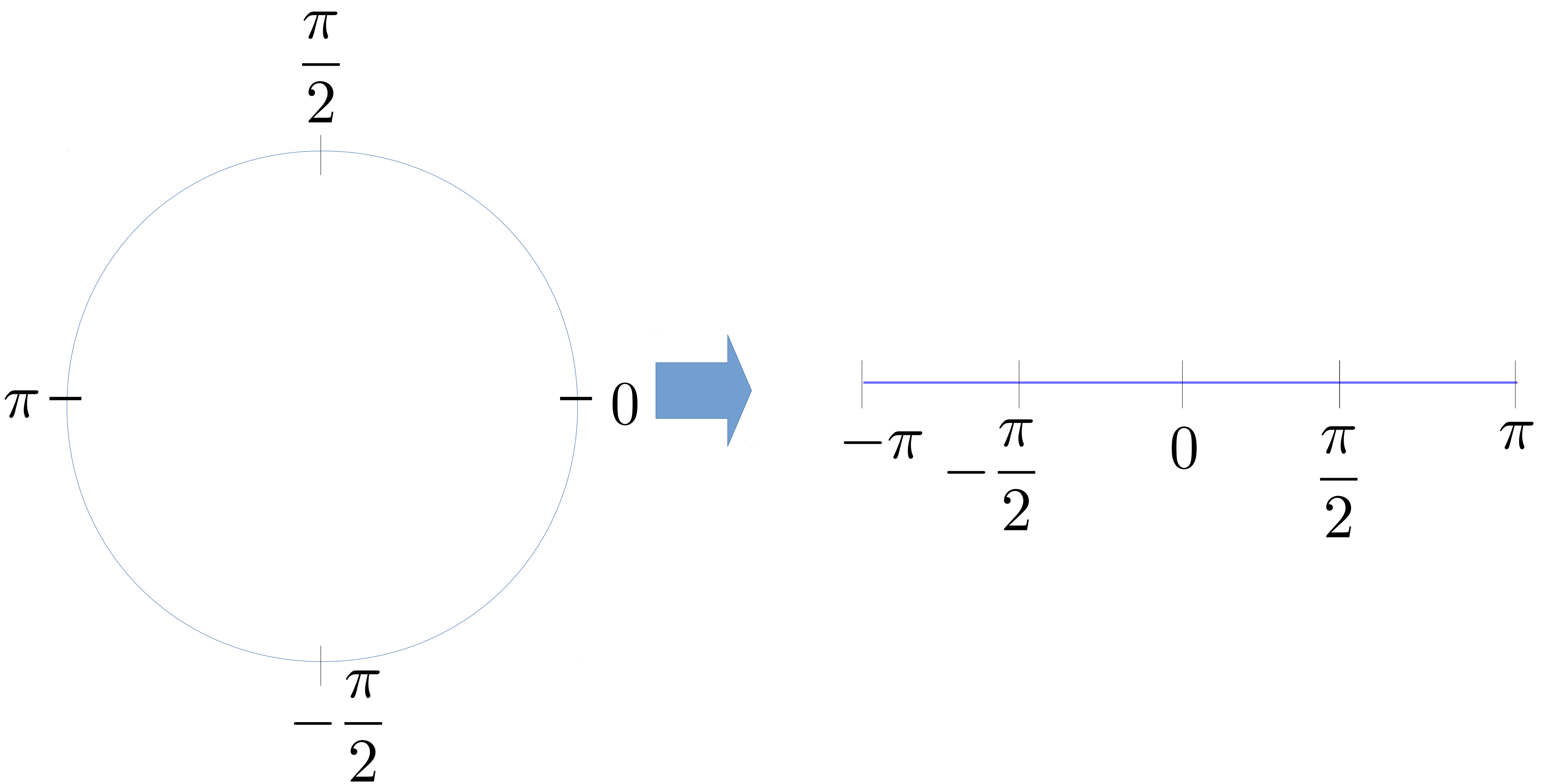
- E.g.: 2D angles



Non-Euclidean Spaces

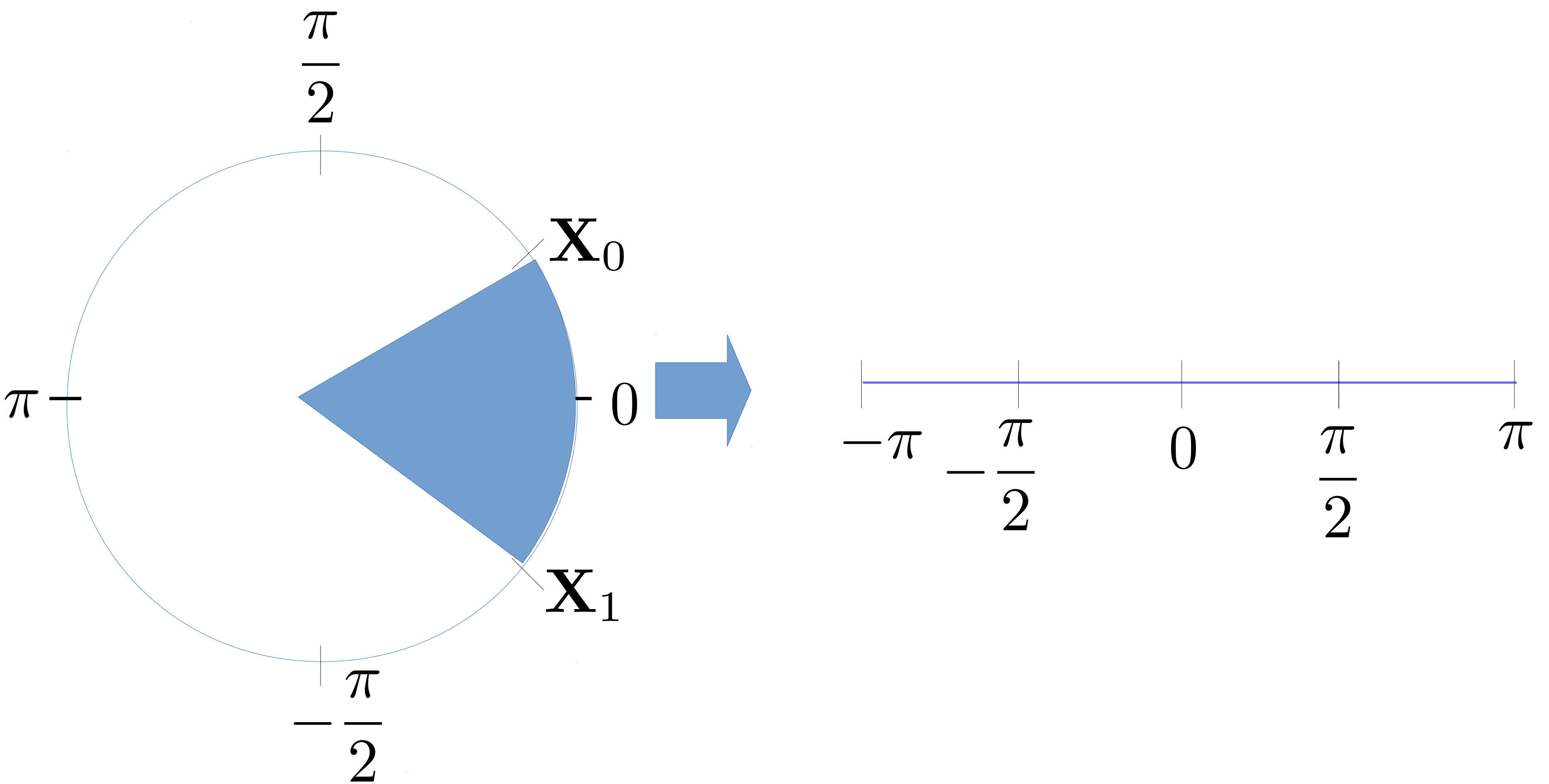
In such cases we commonly operate on a locally Euclidean parameterization

- E.g. we map the angles in the interval [-pi:pi]



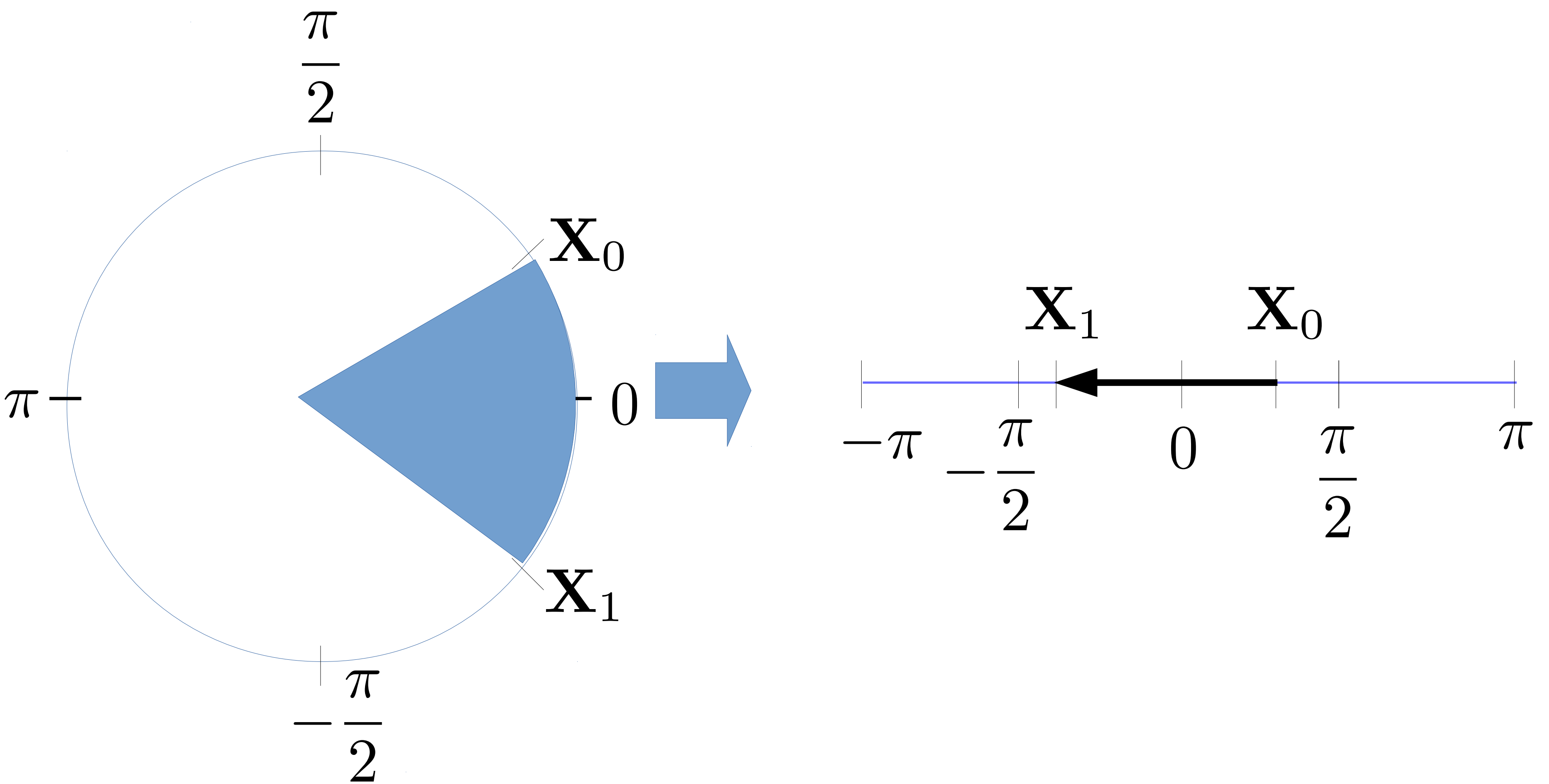
Non-Euclidean Spaces

We can then measure distances in the Euclidean mapping through a regular subtraction



Non-Euclidean Spaces

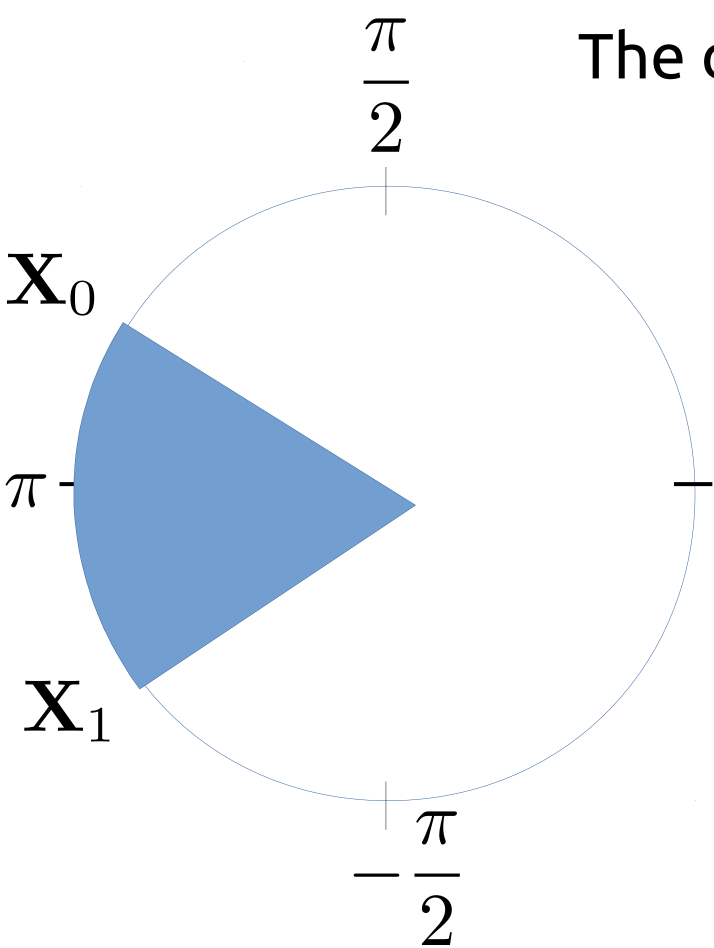
We can then measure distances in the Euclidean mapping through a regular subtraction

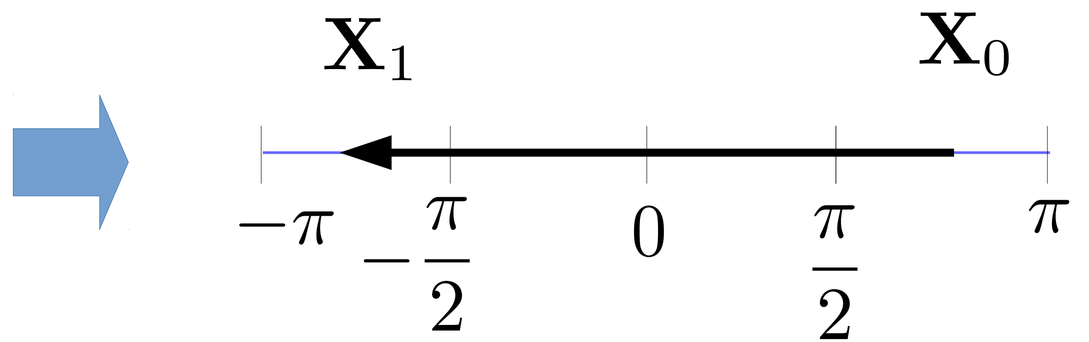


Non-Euclidean Spaces

We can then measure distances in the Euclidean mapping ~~through a regular subtraction~~

The direction of the distance is the opposite!!!

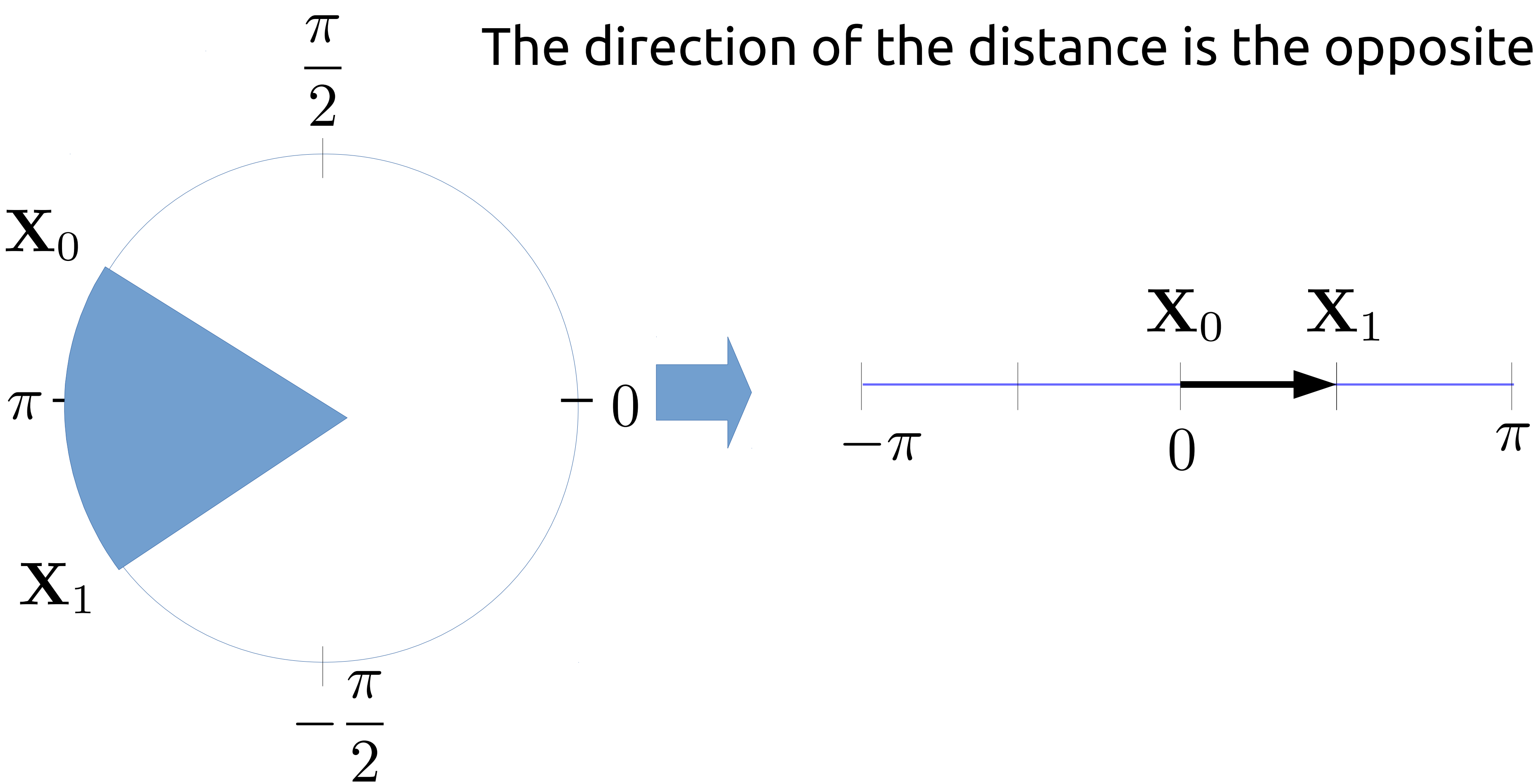




Non-Euclidean Spaces

Idea: when computing the distances, build the Euclidean mapping in the neighborhood of one of the points: the **chart around X0.**

The direction of the distance is the opposite!!!



Computing Differences

$\mathbf{x}_0$: start point, on manifold

$\mathbf{x}_1$: end point, on manifold

$\boldsymbol{\Delta}\mathbf{x}$: difference, on chart

- Compute a chart around $\mathbf{x}_0$
- Compute the location of $\mathbf{x}_1$ on the chart
- Measure the difference between points in the chart
- Chart is Euclidean: $\mathbf{x}_0 = \mathbf{x}_1 \Rightarrow \boldsymbol{\Delta}\mathbf{x} = 0$
- Use an operator $\mathbf{x}_1 \boxminus \mathbf{x}_0 = \boldsymbol{\Delta}\mathbf{x}$

Hertzberg, Wagner, Frese and Schroeder 2011

Applying Differences

X_0: start point, on manifold

$\boldsymbol{\Delta}\mathrm{x}$: difference on chart

X_1: end point, on manifold reachable from X_0
by moving of $\boldsymbol{\Delta}\mathrm{x}$ on the chart

- Compute a chart around X_0
- Move of $\boldsymbol{\Delta}\mathrm{x}$ in the chart and go back to the manifold
- Encapsulate the operation with an operator

$$\mathbf{X}_0 \boxplus \boldsymbol{\Delta}\mathbf{x} = \mathbf{X}_1$$

Algorithm (One Iteration)

Clear **H** and **b**

$$\mathbf{H} \leftarrow 0 \qquad \mathbf{b} \leftarrow 0$$

For each measurement

$$\begin{aligned}
\mathbf{e}_i &\leftarrow \mathbf{h}_i(\mathbf{X}^*) \boxminus \mathbf{Z}_i \\
\mathbf{J}_i &\leftarrow \left.\frac{\partial \mathbf{e}(\mathbf{X}^* \boxplus \mathbf{\Delta x})}{\partial \mathbf{\Delta x}}\right|_{\mathbf{\Delta x}=\mathbf{0}} \\
\mathbf{H} &\leftarrow \mathbf{H} + \mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{J}_i \\
\mathbf{b} &\leftarrow \mathbf{b} + \mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{e}_i
\end{aligned}$$

Compute and apply the perturbation

$$\begin{aligned}
\mathbf{\Delta x} &\leftarrow \mathrm{solve}(\mathbf{H}\mathbf{\Delta x} = -\mathbf{b}) \\
\mathbf{X}^* &\leftarrow \mathbf{X}^* \boxplus \mathbf{\Delta x}
\end{aligned}$$

Gauss in Non Euclidean Spaces

Beware of the + and – operators

- Error function

$$\mathbf{e}_i(\mathbf{x}) \quad = \quad \mathbf{h}_i(\mathbf{x}) \boxminus \mathbf{z}_i$$

- Taylor expansion

$$\mathbf{e}_i(\mathbf{X} \boxplus \mathbf{\Delta x}) \quad = \quad \underbrace{\mathbf{e}_i(\mathbf{X})}_{\mathbf{e}_i} + \underbrace{\left.\frac{\partial \mathbf{e}_i(\mathbf{X} \boxplus \mathbf{\Delta x})}{\partial \mathbf{\Delta x}}\right|_{\mathbf{\Delta x}=\mathbf{0}}}_{\mathbf{J}_i} \mathbf{\Delta x}$$

- Increments

$$\mathbf{X} \leftarrow \mathbf{X} \boxplus \mathbf{\Delta x}$$

Algorithm (One Iteration)

Clear **H** and **b**

$$\mathbf{H} \leftarrow 0 \qquad \mathbf{b} \leftarrow 0$$

For each measurement

$$\mathbf{e}_i \leftarrow \mathbf{h}_i(\mathbf{X}^*) \boxminus \mathbf{Z}_i$$

$$\mathbf{J}_i \leftarrow \left.\frac{\partial \mathbf{e}(\mathbf{X}^* \boxplus \mathbf{\Delta x})}{\partial \mathbf{\Delta x}}\right|_{\mathbf{\Delta x}=\mathbf{0}}$$

$$\mathbf{H} \leftarrow \mathbf{H} + \mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{J}_i$$

$$\mathbf{b} \leftarrow \mathbf{b} + \mathbf{J}_i^T \mathbf{\Omega}_i \mathbf{e}_i$$

Compute and apply the perturbation

$$\mathbf{\Delta x} \leftarrow \mathrm{solve}(\mathbf{H}\mathbf{\Delta x} = -\mathbf{b})$$

$$\mathbf{X}^* \leftarrow \mathbf{X}^* \boxplus \mathbf{\Delta x}$$

Methodology

State space **X**

- Qualify the Domain
- Define an Euclidean parameterization for the perturbation
- Define boxplus operator

Measurement space(s) **Z**

- Qualify the Domain
- Define an Euclidean parameterization for the perturbation
- Define boxminus operator

Identify the prediction functions **h(X)**

MICP: State and Measurements

State

$$\begin{aligned}
\mathbf{X} &= [\mathbf{R}|\mathbf{t}] \in SE(3) \\
\mathbf{\Delta x} &= (\underbrace{\Delta x\ \Delta y\ \Delta z}_{\mathbf{\Delta t}}\ \underbrace{\Delta \alpha_x\ \Delta \alpha_y\ \Delta \alpha_z}_{\mathbf{\Delta}\alpha})^T \\
\mathbf{X} \boxplus \mathbf{\Delta x} &= \mathrm{v2t}(\mathbf{\Delta x})\mathbf{X} \\
&= [\mathbf{R}(\mathbf{\Delta}\alpha)\mathbf{R}|\mathbf{R}(\mathbf{\Delta}\alpha)\mathbf{t} + \mathbf{\Delta t}]
\end{aligned}$$

Measurements

$$\begin{aligned}
\mathbf{z} &\in \Re^3 \\
\mathbf{h}_i(\mathbf{X} \boxplus \mathbf{\Delta x}) &= \mathbf{R}(\mathbf{\Delta}\alpha)\underbrace{[\mathbf{R}\mathbf{p}_i + \mathbf{t}]}_{\mathbf{p}'_i} + \mathbf{\Delta t}
\end{aligned}$$

MICP: Error

The measurements are Euclidean, no need for boxminus

$$
\begin{aligned}
\mathbf{e}_i(\mathbf{X} \boxplus \mathbf{\Delta x}) &= \mathbf{h}_i(\mathbf{X} \boxplus \mathbf{\Delta x}) - \mathbf{z}_i \\
&= \mathbf{R}_x(\mathbf{\Delta}\alpha)\mathbf{p}'_i + \mathbf{\Delta t} - \mathbf{z}_i
\end{aligned}
$$

MICP: Jacobian

Linearizing around the **0** of the chart simplifies the calculations

$$
\begin{aligned}
\mathbf{J}_i &= \left.\frac{\partial \mathbf{e}_i(\mathbf{X} \boxplus \boldsymbol{\Delta}\mathbf{x})}{\partial \boldsymbol{\Delta}\mathbf{x}}\right|_{\boldsymbol{\Delta}\mathbf{x}=0} \\
&= \left.\begin{pmatrix} \frac{\partial \mathbf{e}_i(\cdot)}{\partial \boldsymbol{\Delta}\mathbf{t}} & \frac{\partial \mathbf{e}_i(\cdot)}{\partial \boldsymbol{\Delta}\alpha} \end{pmatrix}\right|_{\boldsymbol{\Delta}\mathbf{x}=0} \\
&= \left.\begin{pmatrix} \frac{\partial \boldsymbol{\Delta}\mathbf{t}}{\partial \boldsymbol{\Delta}\mathbf{t}} & \frac{\partial \mathbf{R}_(\boldsymbol{\Delta}\alpha)\mathbf{p}'_i}{\partial \boldsymbol{\Delta}\alpha} \end{pmatrix}\right|_{\boldsymbol{\Delta}\mathbf{x}=0} \\
&= \begin{pmatrix} \mathbf{I} & [-\mathbf{p}'_i]_\times \end{pmatrix}
\end{aligned}
$$

MICP: Code

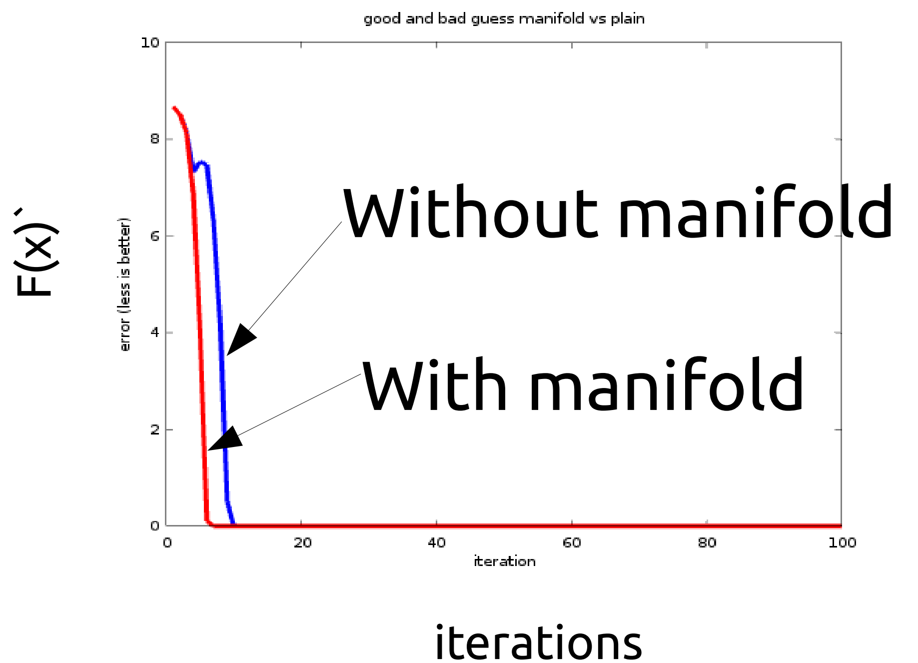
```
function T=v2t(v)
    T=eye(4);
    T(1:3,1:3)=Rx(v(4))*Ry(v(5))*Rz(v(6));
    T(1:3,4)=v(1:3);
endfunction;


function [e,J]=errorAndJacobianManifold(X,p,z)
   z_hat=X(1:3,1:3)*p+X(1:3,4); #prediction
   e=z_hat-z;
   J=zeros(3,6);
   J(1:3,1:3)=eye(3);
   J(1:3,4:6)=skew(z_hat);
endfunction
```

MICP: Code

```
function [X, chi_stats]=doICPManifold(X_guess, P, Z, n_it)
  X=X_guess;
  chi_stats=zeros(1,n_it);
  for (iteration=1:n_it)
    H=zeros(6,6);
    b=zeros(6,1);
    chi=0;
    for (i=1:size(P,2))
      [e,J] = errorAndJacobianManifold(X, P(:,i), Z(:,i));
      H+=J'*J;
      b+=J'*e;
      chi+=e'*e;
    endfor
    chi_stats(iteration)=chi;
    dx=-H\b;
    X=v2t(dx)*X;
  endfor
endfunction
```

Testing

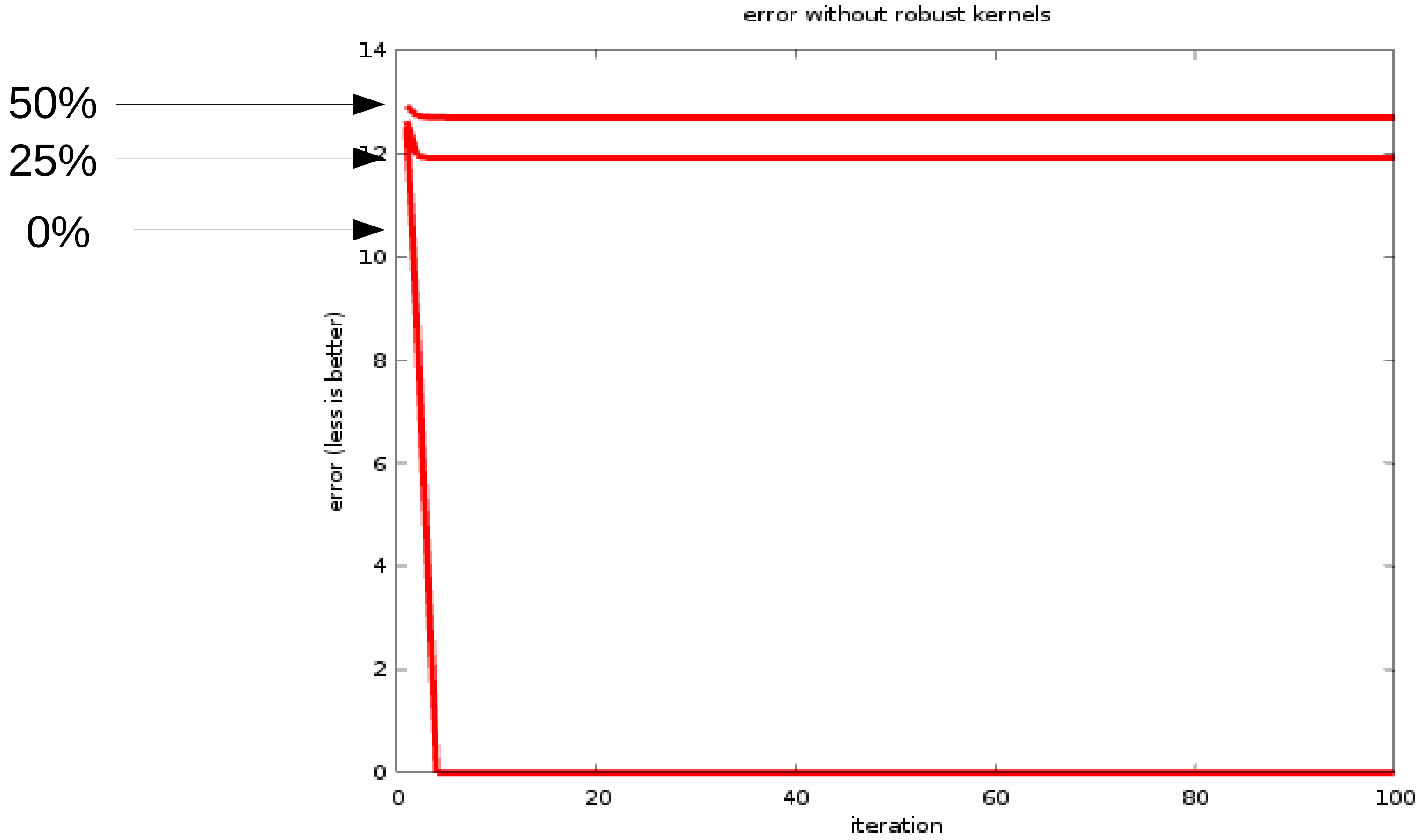
- Spawn a set of random points in 3D
- Define a location of the robot
- Compute syntetic measurements from that location
- Set the origin as initial guess
- Run ICP and plot the evolution of the error



- I need about 5 iterations to get a decent error

Outliers

Let's inject an increasing number of outliers



Robust Kernels

Outliers in the data due to data association result in performance loss

There will be outliers

Hint: Lessen the contribution of measurements having higher error (e.g. using Robust Kernels)

Trivial Kernel Implementation

```
If (error>threshold){
  scale_error_so_that_its_norm_is_the_threshold();
}
```

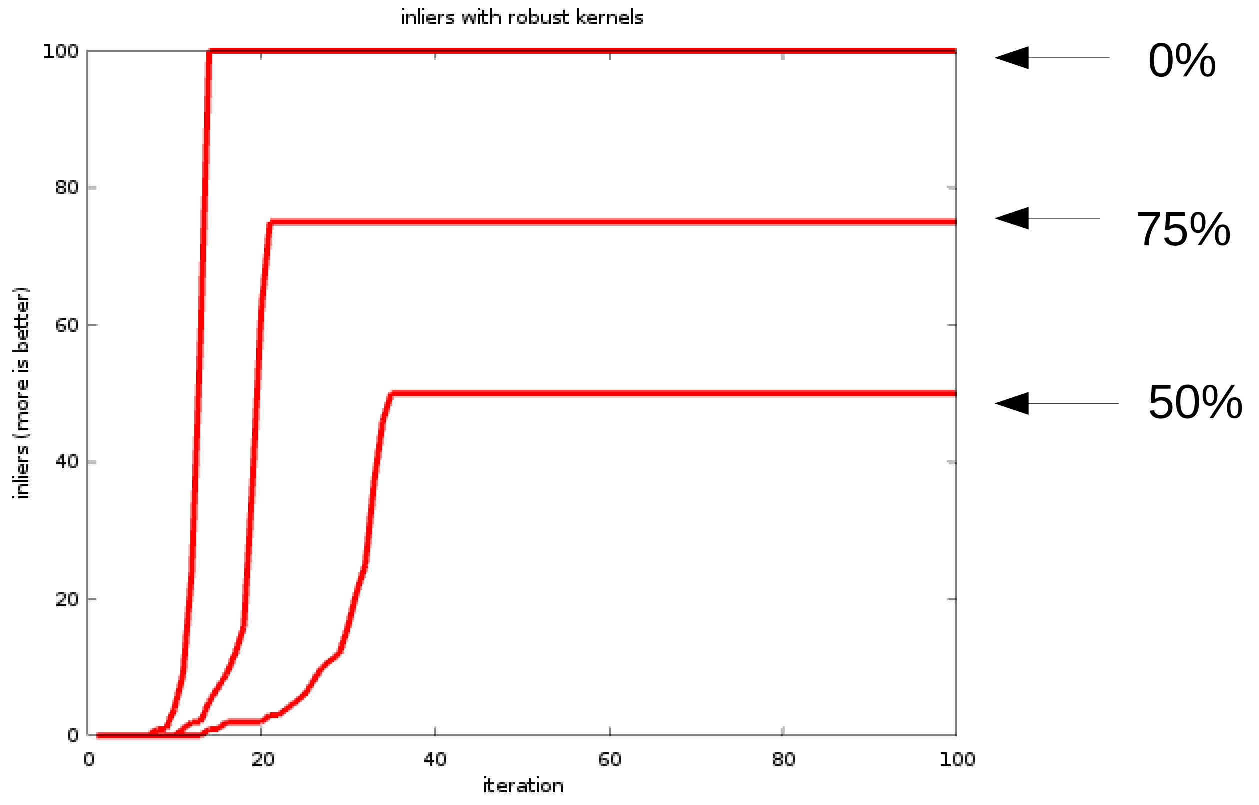
MICP with Outliers: Code

```
function [X, chi_stats]=doICPManifold(X_guess, P, Z, n_it)
  X=X_guess;
  chi_stats=zeros(1,n_it);
  for (iteration=1:n_it)
    H=zeros(6,6);
    b=zeros(6,1);
    for (i=1:size(P,2))
      [e,J] = errorAndJacobianManifold(X, P(:,i), Z(:,i));
      chi=e'*e;
      if(chi>threshold)
        e*=sqrt(threshold/chi);
      endif;
      H+=J'*J;
      b+=J'*e;
      chi_stats(iteration)+=chi;
    endfor
    dx=-H\b;
    X=v2t(dx)*X;
  endfor
endfunction
```

Behavior with Outliers

Instead of measuring the F(x) we measure the number of inliers as the algorithm evolves

The closer is the estimated # of inliers to the true fraction the better is our system



Take Home Message

Gauss-Newton is a powerful tool used as building block of modern SLAM systems

It is used both within

- front-end (like in this lecture)
- back end (like in the next)

- In this talk we provided basics for
  - Formalizing the problem
  - Hacking a solver
  - Dealing with non-Euclidean spaces
  - Cope with some outliers

Take Home Message

Works under its assumptions, that are

- Mild measurement functions
- Decent initial guess
- The system is observable

Some software Implementing GN

Calibration

ICP

Sparse LS