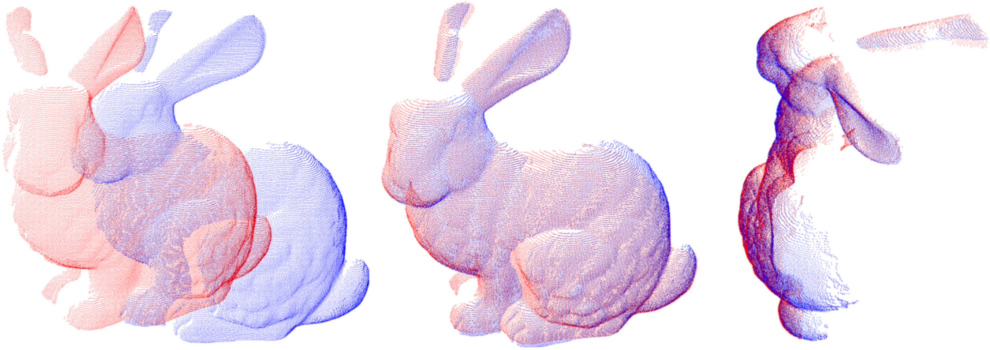
Probabilistic Robotics Course

ICP optimization on a Manifold

Giorgio Grisetti

grisetti@diag.uniroma1.it

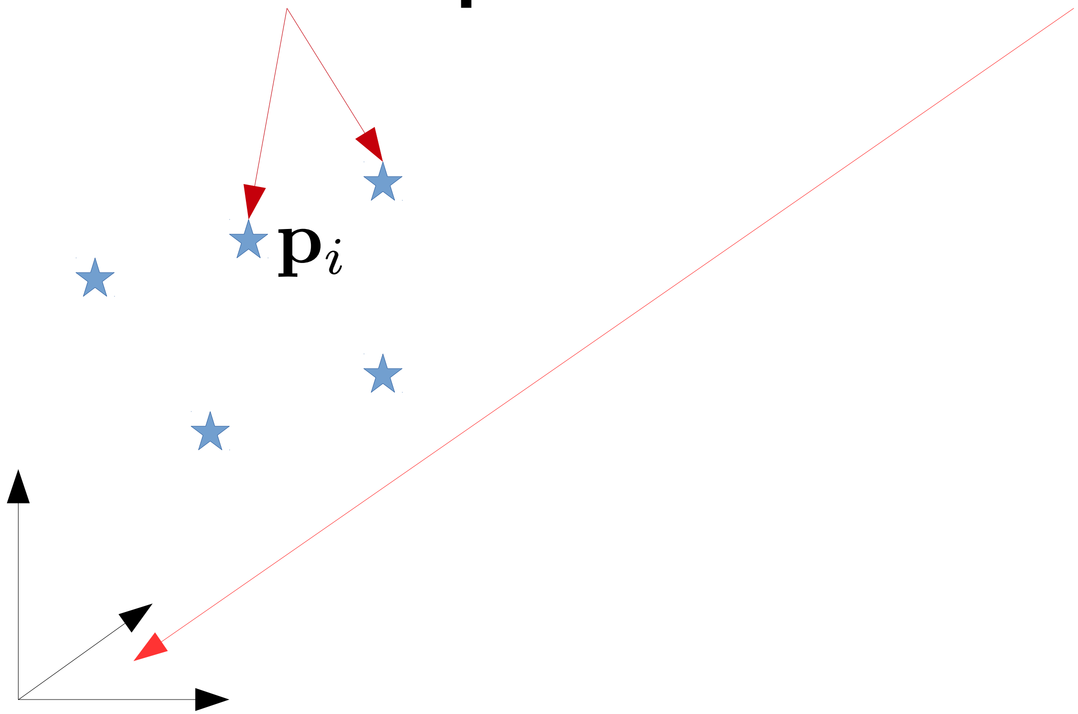
Department of Computer, Control, and Management Engineering
Sapienza University of Rome



Courtesy of Li et. al

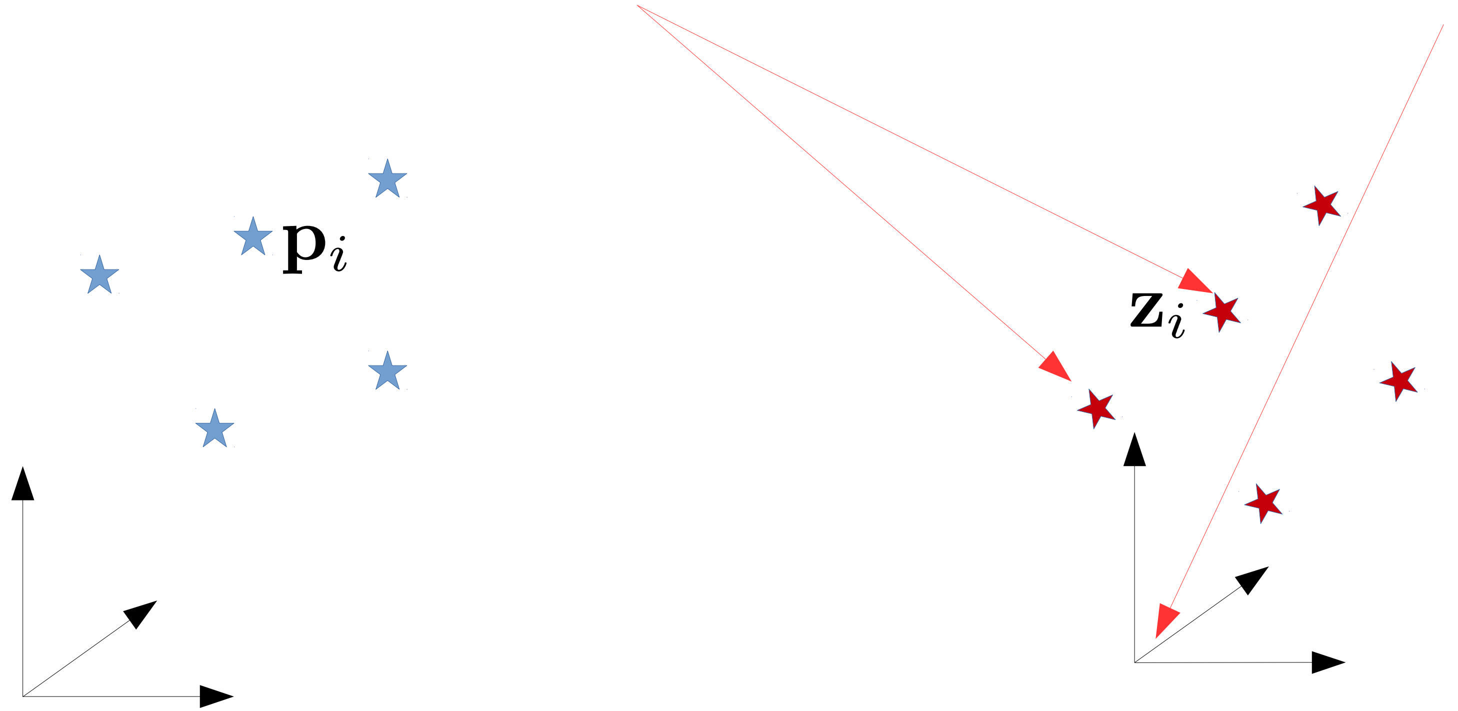
Example ICP Optimization 3D

Given a set of points in the world frame



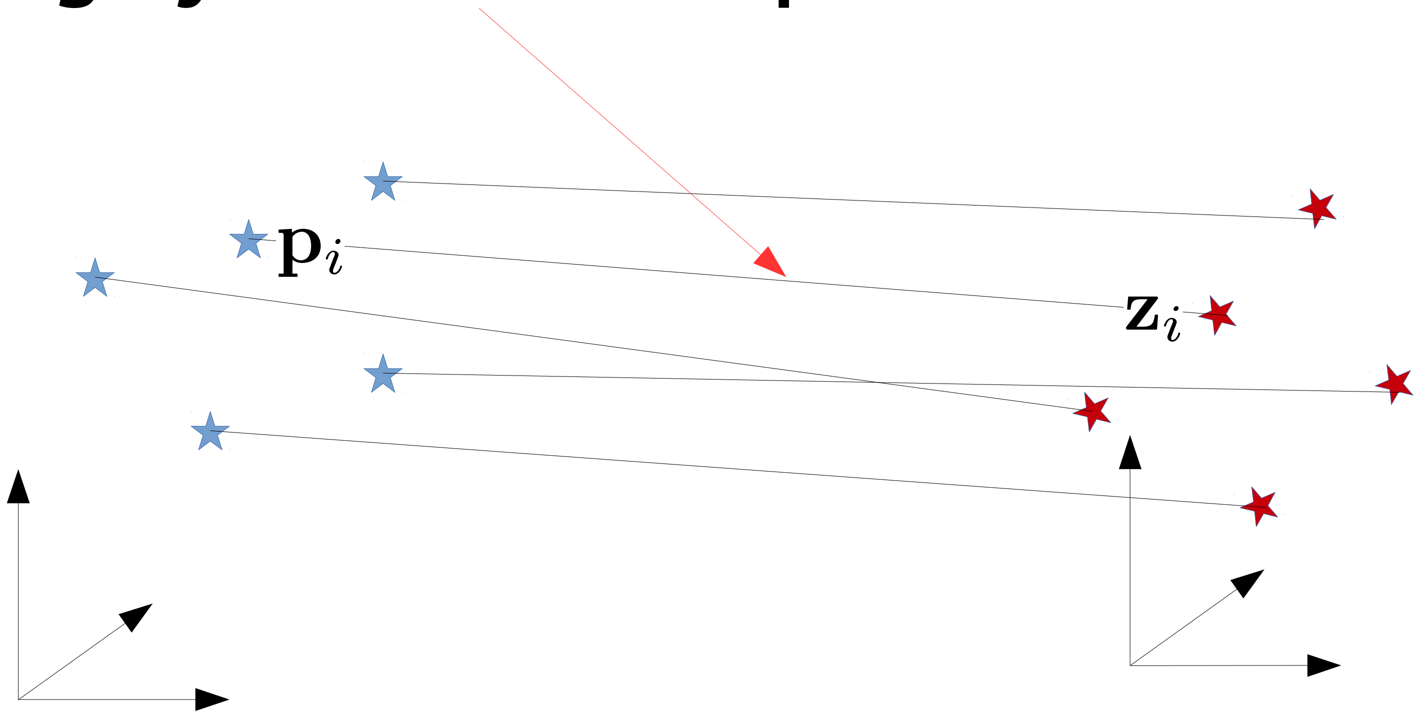
Example ICP Optimization 3D

A set of 3D measurements in the robot frame



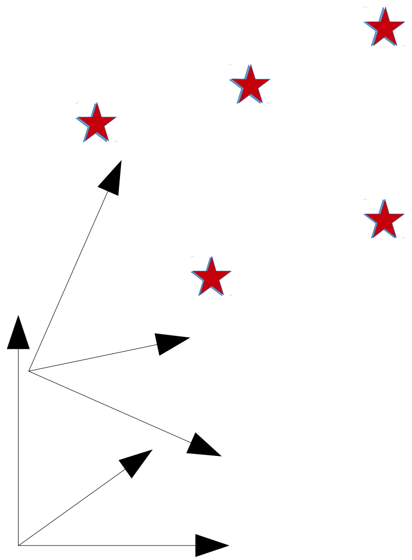
Example ICP Optimization 3D

Roughly known correspondences



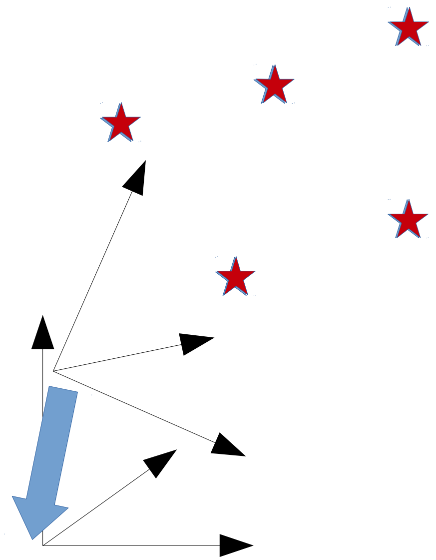
Example ICP Optimization 3D

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



Example ICP Optimization 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

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

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

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

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

Measurements

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

MICP: Error

The measurements are Euclidean, no need for boxminus

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

MICP: Jacobian

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

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

MICP: Code

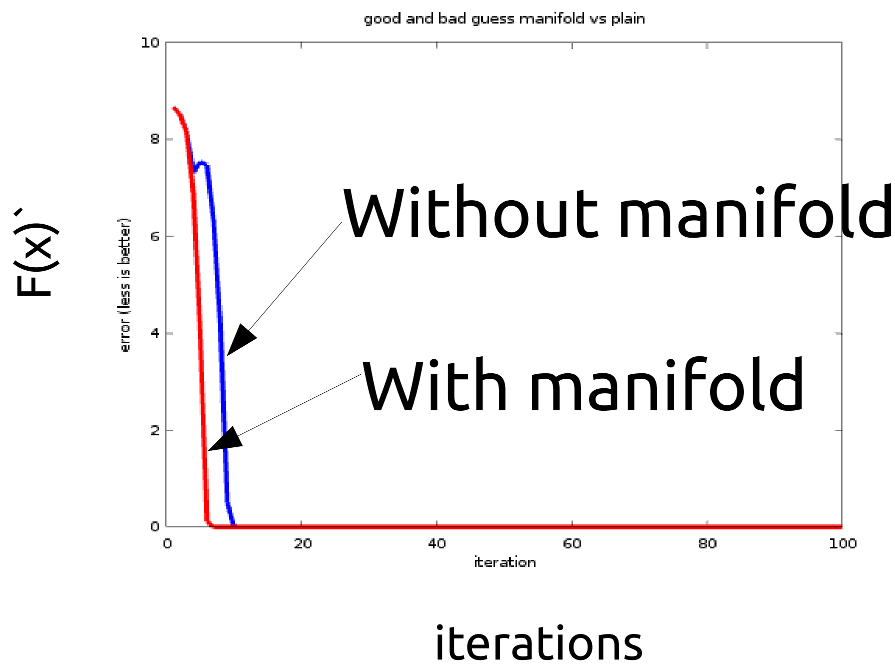
```
function T=v2t(v)
    T=eye(4);
    T(1:3,1:3)=Rx(v(4))*Ry(v(5))*Rz(v(6));
    T(1:3,4)=v(1:3);
endfunction;


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

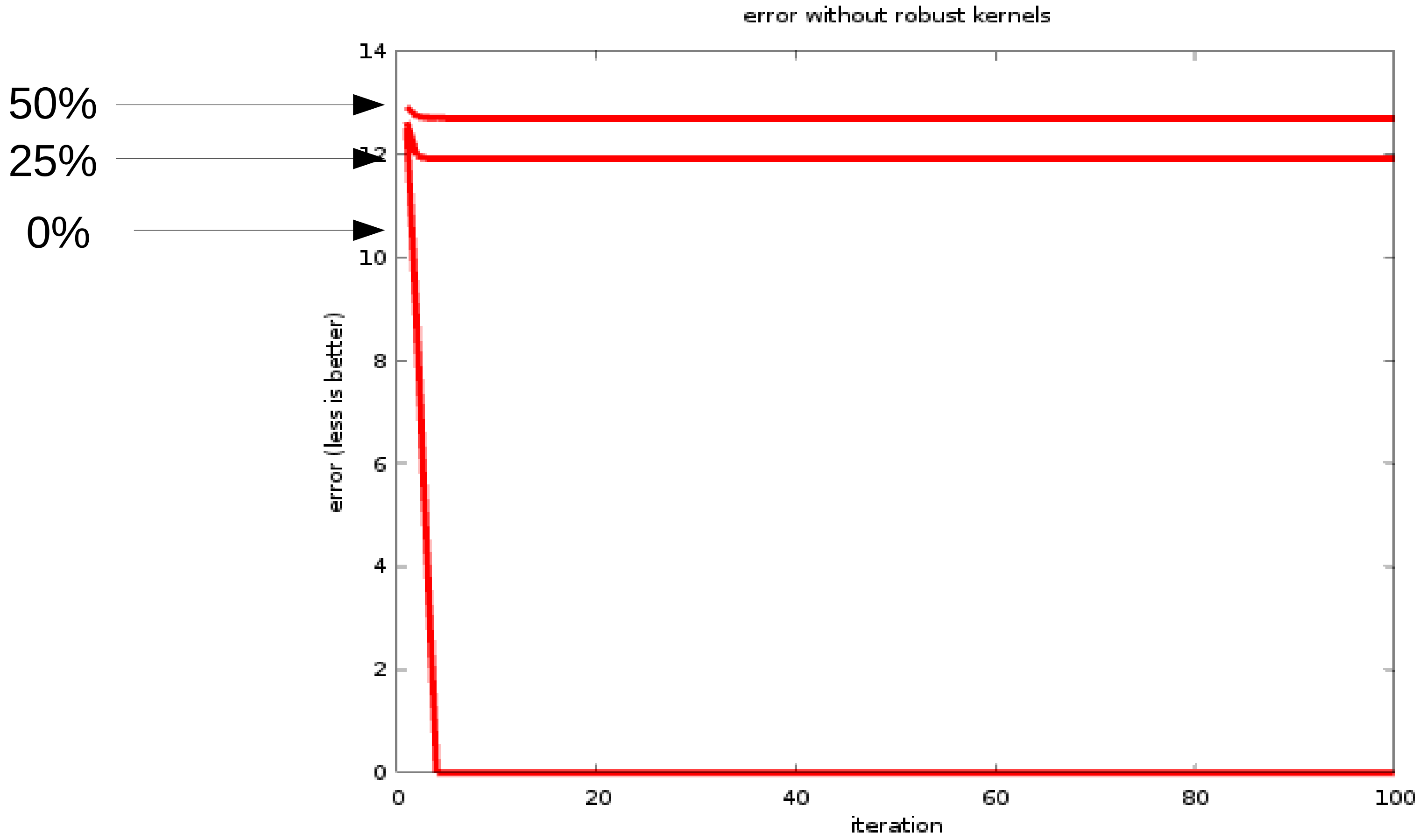
- Spawn a set of random points in 3D
- Define a location of the robot
- Compute syntetic measurements from that location
- Set the origin as initial guess
- Run ICP and plot the evolution of the error



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