

Probabilistic Robotics Course

Multi-Point Registration

Giorgio Grisetti

`grisetti@dis.uniroma1.it`

Dept of Computer Control and Management Engineering
Sapienza University of Rome

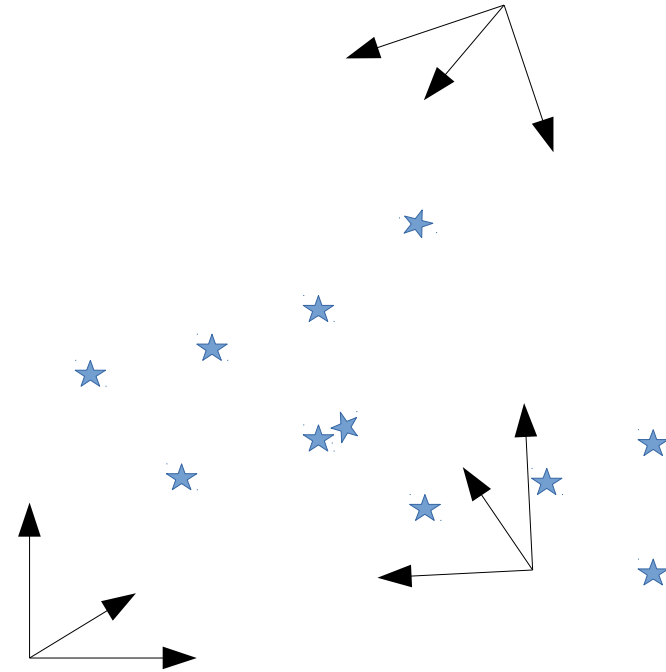
Problem Definition

The environment is populated with identifiable landmarks whose position is unknown

The landmarks are observed by a 3D sensor from multiple positions

We want to determine

- The landmark locations
- The robot positions



State

The state is a collection of transforms and landmark positions

$$\begin{array}{ll}
 \mathbf{X} & : \quad \mathbf{X} = \{\mathbf{X}_r^{[1]}, \dots, \mathbf{X}_r^{[N]}, \mathbf{X}_l^1, \dots, \mathbf{X}_l^{[M]}\} \\
 \text{robot } \mathbf{X}_r^{[n]} \in SE(3) & : \quad \mathbf{X}^{[n]} = (\mathbf{R}^{[n]} | \mathbf{t}^{[n]}) \\
 \text{landmark } \mathbf{X}_l^{[m]} \in \mathbb{R}^3 & : \quad \mathbf{X}_l^{[m]} = (x^{[m]} \ y^{[m]} \ z^{[m]})^T
 \end{array}$$

The increments are represented by a large vector containing the minimal perturbation for each state variable

$$\begin{array}{ll}
 \Delta \mathbf{x} \in \mathbb{R}^{6N+3M} & : \quad \Delta \mathbf{x} = \left(\Delta \mathbf{x}_r^{[1]T}, \dots, \Delta \mathbf{x}_r^{[N]T}, \Delta \mathbf{x}_l^{[1]T}, \dots, \Delta \mathbf{x}_l^{[M]T} \right)^T \\
 \text{robot } \Delta \mathbf{x}_r^{[n]T} \in \mathbb{R}^6 & : \quad \Delta \mathbf{x}_r^{[n]T} = \underbrace{(\Delta x^{[n]} \ \Delta y^{[n]} \ \Delta z^{[n]})^T}_{\Delta \mathbf{t}^{[n]}} \underbrace{(\Delta \alpha_x^{[n]} \ \Delta \alpha_y^{[n]} \ \Delta \alpha_z^{[n]})^T}_{\Delta \alpha^{[n]}} \\
 \text{landmark } \Delta \mathbf{x}_l^{[m]T} \in \mathbb{R}^3 & : \quad \Delta \mathbf{x}_l^{[m]T} = \underbrace{(\Delta x^{[m]} \ \Delta y^{[m]} \ \Delta z^{[m]})^T}_{\Delta \mathbf{t}^{[m]}}
 \end{array}$$

State (implementation)

In an Octave Implementation, we will represent the states as

- an array of **4 x 4** transformation matrices, for the robot poses
- a matrix of **3 x num_landmarks** elements, for the landmarks. Each column is a landmark

State (implementation)

With such an articulated representation of the state it might come in handy to have functions that translate

- a landmark or
- pose index

in the corresponding position in the perturbation vector

For a pose:

$$v_idx = 1 + (pose_index - 1) * 6;$$

For a landmark:

$$v_idx = 1 + (num_poses) * 6 + (landmark_index - 1) * 3;$$

State (implementation)

```
function v_idx=poseMatrixIndex(pose_index, num_poses, num_landmarks)
    global pose_dim; # 6 in our case
    global landmark_dim; # 3 in our case
    if (pose_index>num_poses)
        v_idx=-1;
        return;
    endif;
    v_idx=1+(pose_index-1)*pose_dim;
endfunction;
```

```
function v_idx=landmarkMatrixIndex(landmark_index, num_poses, num_landmarks)
    global pose_dim;
    global landmark_dim;
    if (landmark_index>num_landmarks)
        v_idx=-1;
        return;
    endif;
    v_idx=1 + (num_poses)*pose_dim + (landmark_index-1) * landmark_dim;
endfunction;
```

Boxplus

The boxplus has to be adapted to apply the individual perturbations for each variable block

$$\begin{aligned}\mathbf{X}' &= \mathbf{X} \boxplus \Delta \mathbf{x} \\ \mathbf{X}_r^{[n]'} &= \Delta \mathbf{x}_r^{[n]} \boxplus \mathbf{X}_r^{[n]} \\ &= v2t(\Delta \mathbf{x}_r^{[n]}) \mathbf{X}_r^{[n]} \\ \mathbf{X}_l^{[m]'} &= \mathbf{X}_l^{[m]} + \Delta \mathbf{x}_l^{[n]}\end{aligned}$$

Boxplus (implementation)

```
function [XR, XL]=boxPlus(XR, XL, num_poses, num_landmarks, dx)
    global pose_dim;
    global landmark_dim;
    for(pose_index=1:num_poses)
        pose_matrix_index=poseMatrixIndex(pose_index,
                                            num_poses,
                                            num_landmarks);

        dxr=dx(pose_matrix_index:pose_matrix_index+pose_dim-1);
        XR(:, :, pose_index)=v2t(dxr)*XR(:, :, pose_index);
    endfor;
    for(landmark_index=1:num_landmarks)
        landmark_matrix_index=landmarkMatrixIndex(landmark_index,
                                                    num_poses,
                                                    num_landmarks);

        dxl=dx(landmark_matrix_index:landmark_matrix_index+landmark_dim-1, :);
        XL(:, landmark_index)+=dxl;
    endfor;
endfunction;
```


Measurements and Predictions

A measurement of the landmark m , performed by robot pose n is as follows

$$\mathbf{z}^{[n,m]} \in \mathbb{R}^3 \quad : \quad \mathbf{z}^{[m,n]} = \left(x^{[n,m]} \ y^{[n,m]} \ y^{[n,m]} \right)^T$$

The prediction and the error of this measurements are very similar to the ICP ones

$$\begin{aligned} \mathbf{h}^{[n,m]}(\mathbf{X}) &= \mathbf{X}_r^{[n]} \mathbf{X}_l^{[m]} \\ \mathbf{e}^{[n,m]}(\mathbf{X}) &= \mathbf{X}_r^{[n]} \mathbf{X}_l^{[m]} - \mathbf{z}^{[n,m]} \\ \mathbf{e}^{[n,m]}(\mathbf{X} \boxplus \Delta \mathbf{x}) &= v2t(\Delta \mathbf{x}_r^{[n]}) \mathbf{X}_r^{[n]} (\mathbf{X}_l^{[m]} + \Delta \mathbf{x}_l^{[m]}) - \mathbf{z}^{[n,m]} \end{aligned}$$

Jacobians

The prediction depends only on the robot pose m and the landmark n , so it will be mostly 0

Non 0 only for

- pose block n
- landmark block m

$$\begin{aligned}
 \frac{\partial \mathbf{e}^{[n,m]}(\mathbf{X} \boxplus \Delta \mathbf{x})}{\partial \Delta \mathbf{x}} &= \left(\cdots \mathbf{0}_{3 \times 6} \cdots \frac{\partial \mathbf{e}^{[n,m]}(\mathbf{X} \boxplus \Delta \mathbf{x})}{\partial \Delta \mathbf{x}_r^n} \cdots \mathbf{0}_{3 \times 6} \mathbf{0}_{3 \times 3} \cdots \frac{\partial \mathbf{e}^{[n,m]}(\mathbf{X} \boxplus \Delta \mathbf{x})}{\partial \Delta \mathbf{x}_l^m} \cdots \mathbf{0}_{3 \times 3} \cdots \right) \\
 \hat{\mathbf{z}}^{[m,n]} &= \\
 \frac{\partial \mathbf{e}^{[n,m]}(\mathbf{X} \boxplus \Delta \mathbf{x})}{\partial \Delta \mathbf{x}_r^n} \Big|_{\Delta \mathbf{x}_r^{[n]}=0} &= \underbrace{\left(\mathbf{I}_{3 \times 3} \parallel \left[-\hat{\mathbf{z}}^{[m,n]} \right]_{\times} \right)}_{\mathbf{J}_r^{n,m}} \\
 &\quad \underbrace{\mathbf{R}^{[n]}}_{\mathbf{J}_l^{n,m}} \\
 \frac{\partial \mathbf{e}^{[n,m]}(\mathbf{X} \boxplus \Delta \mathbf{x})}{\partial \Delta \mathbf{x}_l^m} \Big|_{\Delta \mathbf{x}_l^{[m]}=0} &= \\
 \mathbf{J}^{[n,m]} &= \left(\cdots \mathbf{0}_{3 \times 6} \cdots \mathbf{J}_r^{[n,m]} \cdots \mathbf{0}_{3 \times 6} \mathbf{0}_{3 \times 3} \cdots \mathbf{J}_l^{[n,m]} \cdots \mathbf{0}_{3 \times 3} \cdots \right)
 \end{aligned}$$

Jacobians (implementation)

```
function [e,Jr,Jl]=errorAndJacobian(Xr,Xl,z)
    R=Xr(1:3,1:3);
    t=Xr(1:3,4);
    z_hat=R*Xl+t; #prediction
    e=z_hat-z;
    Jr=zeros(3,6);
    Jr(1:3,1:3)=eye(3);
    Jr(1:3,4:6)=-skew(z_hat);
    Jl=R;
endfunction;
```

H Matrix and B vector

H and b for a measurement have only few non zero blocks

$$\begin{aligned}\mathbf{H}^{[n,m]} &= \mathbf{J}^{[n,m]T} \mathbf{\Omega}^{[n,m]} \mathbf{J}^{[n,m]} \\ &= \begin{pmatrix} \mathbf{J}_r^{[n,m]T} \mathbf{\Omega}^{[n,m]} \mathbf{J}_r^{[n,m]} & \mathbf{J}_r^{[n,m]T} \mathbf{\Omega}^{[n,m]} \mathbf{J}_l^{[n,m]} \\ \mathbf{J}_l^{[n,m]T} \mathbf{\Omega}^{[n,m]} \mathbf{J}_l^{[n,m]} & \mathbf{J}_l^{[n,m]T} \mathbf{\Omega}^{[n,m]} \mathbf{J}_l^{[n,m]} \end{pmatrix} \\ \mathbf{b}^{[n,m]} &= \mathbf{J}^{[n,m]T} \mathbf{\Omega}^{[n,m]} \mathbf{e}^{[n,m]} \\ &= \begin{pmatrix} \mathbf{J}_r^{[n,m]T} \mathbf{\Omega}^{[n,m]} \mathbf{e}^{[n,m]} \\ \mathbf{J}_l^{[n,m]T} \mathbf{\Omega}^{[n,m]} \mathbf{e}^{[n,m]} \end{pmatrix}\end{aligned}$$

H and B Implementation

```
H=zeros(system_size, system_size);
b=zeros(system_size,1);
chi_stats(iteration)=0;
for (measurement_num=1:size(Z,2))
    pose_index=associations(1,measurement_num);
    landmark_index=associations(2,measurement_num);
    z=Z(:,measurement_num);
    Xr=XR(:, :, pose_index);
    Xl=XL(:, landmark_index);
    [e,Jr,Jl] = errorAndJacobian(Xr, Xl, z);
    Hrr=Jr'*Jr;
    Hrl=Jr'*Jl;
    Hll=Jl'*Jl;
    br=Jr'*e;
    bl=Jl'*e;

    pose_matrix_index=poseMatrixIndex(pose_index, num_poses, num_landmarks);
    landmark_matrix_index=landmarkMatrixIndex(landmark_index, num_poses, num_landmarks);

    H(pose_matrix_index:pose_matrix_index+pose_dim-1,
        pose_matrix_index:pose_matrix_index+pose_dim-1)+=Hrr;
    H(pose_matrix_index:pose_matrix_index+pose_dim-1,
        landmark_matrix_index:landmark_matrix_index+landmark_dim-1)+=Hrl;
    H(landmark_matrix_index:landmark_matrix_index+landmark_dim-1,
        landmark_matrix_index:landmark_matrix_index+landmark_dim-1)+=Hll;
    H(landmark_matrix_index:landmark_matrix_index+landmark_dim-1,
        pose_matrix_index:pose_matrix_index+pose_dim-1)+=Hrl';

    b(pose_matrix_index:pose_matrix_index+pose_dim-1)+=br;
    b(landmark_matrix_index:landmark_matrix_index+landmark_dim-1)+=bl;
endfor
```

Degrees of Freedom

A state configuration X , has the same error of all states obtained by applying a rigid transform to all variables.

The system will be under-determined

- 6 degrees of freedom (dof of a rigid transform)

To solve the system we need to either

- eliminate the redundant variables
- adding a prior

Eliminating Variables

We can impose that one robot pose stays fixed

To this extent we can

- reduce the linear system by suppressing the rows and the columns of the variables that stay fixed.
- solve the reduced system
- propagate the computed perturbation to all variables that have not been eliminated

Eliminating Variables (impl)

```
dx=zeros(system_size,1);
```

```
% we solve the linear system, blocking the first pose  
% this corresponds to "remove" from H and b the locks  
% of the 1st pose, while solving the system
```

```
dx(pose_dim+1:end)=  
    -(H(pose_dim+1:end,pose_dim+1:end)\  
      b(pose_dim+1:end,1));  
[XR, XL]=boxPlus(XR,XL,num_poses, num_landmarks, dx);
```


Adding a Prior

Alternatively we can add a “flexible” constraint of the type $\Delta \mathbf{x}_r^{[1]} = 0$ to the least squares problem.

This is done by adding an error term

$$\begin{aligned} \mathbf{e}_{\text{prior}} &= \Delta \mathbf{x}_r^{[1]} \\ \mathbf{J}_{\text{prior}} &= \mathbf{I} \\ \mathbf{H}_{\text{prior}} &= \Omega_{\text{prior}} \\ \mathbf{b}_{\text{prior}} &= \mathbf{0} \end{aligned}$$

Can be implemented by adding to the final H a 6x6 (large) information matrix in correspondence of the first variable

The larger the Omega, the more stiff the constraint will be

Adding a Prior (impl)

```
dx=zeros(system_size,1);
```

```
H(1:pose_dim,1:pose_dim)+=eye(6)*large_value;
```

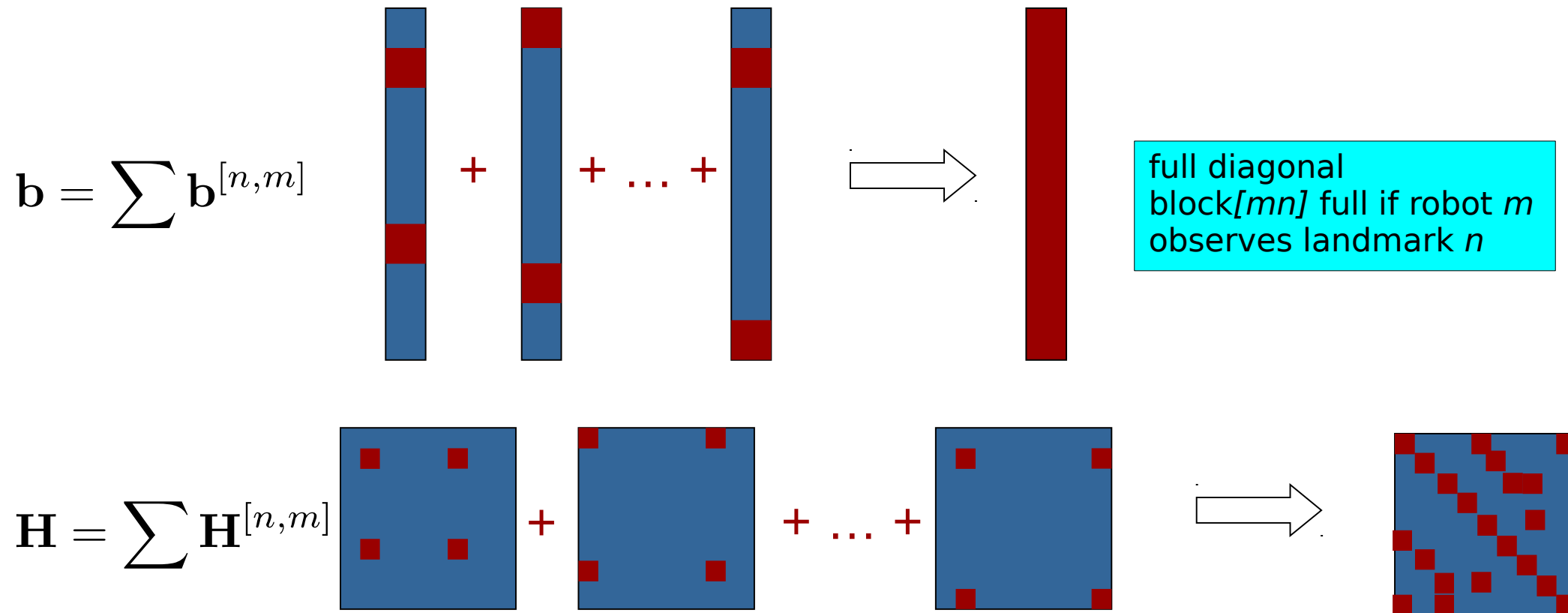
```
dx=-(H\b);
```

```
[XR, XL]=boxPlus(XR,XL,num_poses,  
                  num_landmarks, dx);
```

Structure

Structure of **H**: location of the non zero elements

The structure of **H** depends only on the structure of the measurements



Conclusions

We approached a complex multi-robot multi-landmark problem

- The measurement independence leads to a sparse structure of H
- The structure of H does not change during the iterations
- It can be efficiently solved by using sparse methods (we will see these later in this course)