Robot Programming

Localizing on a Distance Map

G. Grisetti

{grisetti}@diag.uniroma1.it

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

DMap for Localization

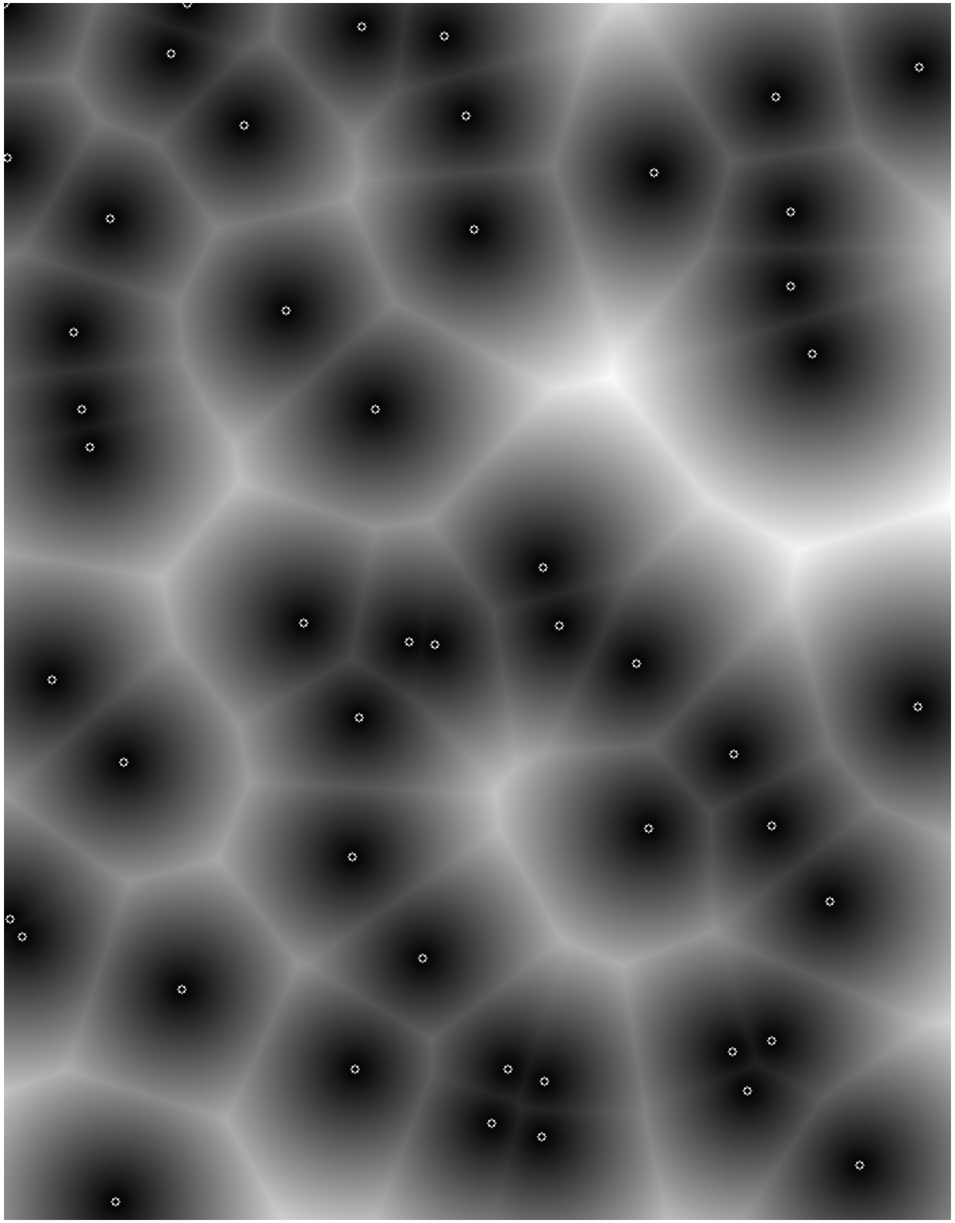
Let our reference be the set of points shown to the right

Formulate localization as a minimization problem

Endpoint in robot frame

$$\mathbf{X}^* = \underset{\mathbf{X}}{\operatorname{argmin}} \sum_j \|d(\mathbf{X}\mathbf{z}_j)\|^2$$

Robot pose



Gauss-Newton

Iteratively solve this problem

$$\mathbf{X}* = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i ||\mathbf{e}_i(\mathbf{X})||^2$$

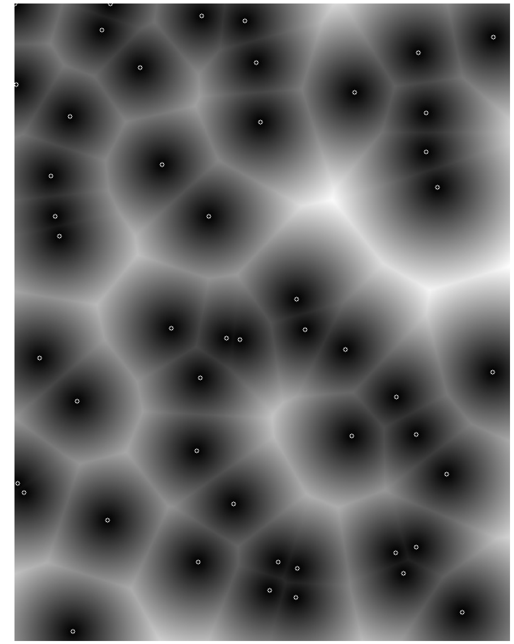
$$\mathbf{e} : Dom(\mathbf{X}) \rightarrow \Re^m$$

$$||\mathbf{v}||^2 := \mathbf{v}^T\mathbf{v}$$

Requires **e**(..) to be differentiable w.r.t. and euclidean perturbation around **X**

Dmap for Localiation

Let our reference be the set of points shown to the right

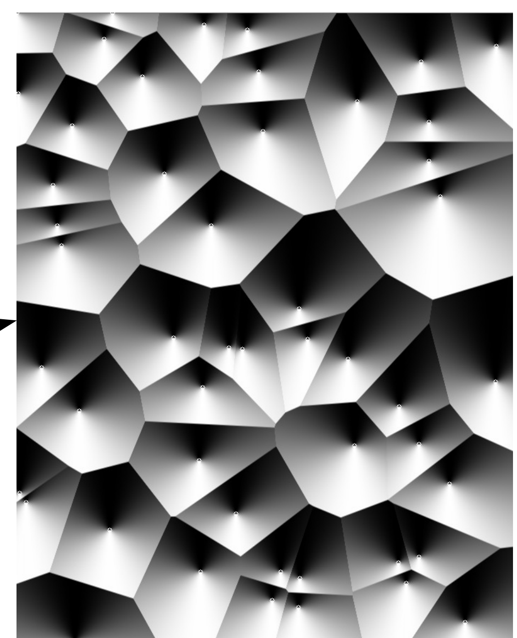


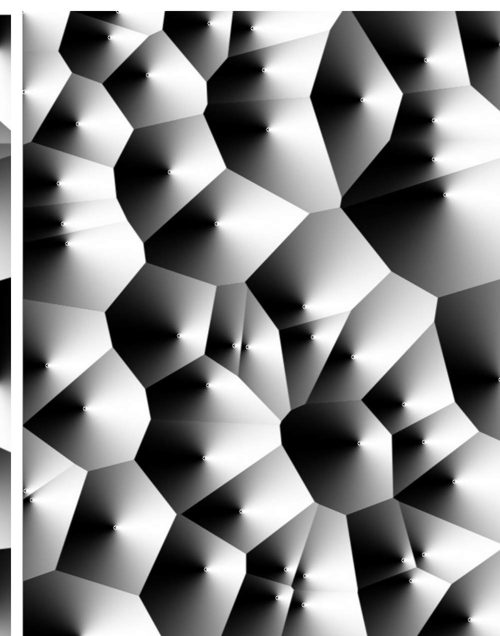
- Embed the data association in the cost function

$$\mathbf{X}^* = \operatorname*{argmin}_{\mathbf{X}} \sum_j \|d(\mathbf{X}\mathbf{z}_j)\|^2$$

- The distance function is differentiable so is the cost

$$\frac{\partial d(\mathbf{X}\mathbf{z}_j)}{\partial \mathbf{X}} = \boxed{\frac{\partial d(\mathbf{y})}{\partial \mathbf{y}}}\Bigg|_{\mathbf{y}=\mathbf{X}\mathbf{z}_j} \frac{\partial \mathbf{X}\mathbf{z}_j}{\partial \mathbf{X}}$$





Gauss-Newton(on manifold)

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

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

For each measurement, update h and b

$$\begin{aligned}
\mathbf{e}_i &\leftarrow \mathbf{d}_i(\mathbf{X}^*\mathbf{z}_i) \\
\mathbf{E}_i &\leftarrow \left.\frac{\partial \mathbf{d}_i(\mathbf{T}(\mathbf{\Delta x})\mathbf{X}^*\mathbf{z}_i)}{\partial \mathbf{\Delta x}}\right|_{\mathbf{\Delta x}=\mathbf{0}} \\
\mathbf{H} &\leftarrow \mathbf{H} + \mathbf{E}_i^T \mathbf{E}_i \\
\mathbf{b} &\leftarrow \mathbf{b} + \mathbf{E}_i^T \mathbf{e}_i
\end{aligned}$$

Update the estimate with the perturbation

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

$$\mathbf{X}^* \leftarrow \mathbf{T}(\mathbf{\Delta x})\mathbf{X}^*$$

Gauss-Newton(on manifold)

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

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

For each measurement, update h and b

$$\begin{aligned}
\mathbf{p}_i &\leftarrow \mathbf{X}^* \mathbf{z}_i \\
\mathbf{e}_i &\leftarrow \mathbf{d}(\mathbf{p}_i) \\
\mathbf{E}_i &\leftarrow \mathrm{grad}^T(\mathbf{p}_i) \begin{pmatrix} 1 & 0 & -\mathbf{p}_i.y \\ 0 & 1 & \mathbf{p}_i.x \end{pmatrix} \\
\mathbf{H} &\leftarrow \mathbf{H} + \mathbf{E}_i^T \mathbf{E}_i \\
\mathbf{b} &\leftarrow \mathbf{b} + \mathbf{E}_i^T \mathbf{e}_i
\end{aligned}$$

Update the estimate with the perturbation

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

$$\mathbf{X}^* \leftarrow \mathbf{T}(\mathbf{\Delta x})\mathbf{X}^*$$

Implementation

Ingredients

- Functions to display
- Functions to map from world to grid and viceversa
- Calculation of Gradients on dmap
- Facilities for Linear Algebra (Eigen helps)

Testing

- Spawn some points on a grid
- Compute the dmap and the gradients
- Express these points in world coordinates
- Express these points in sensor coordinates (to simulate a measurement)
- Implement GN

Full Fledged Implementation

In the repo, you find a working Dmap Localization

- The algorithm has a state **X** (the current estimate of the robot position)
- On startup the algorithm subscribes to
  - \map topic to get the occupancy grid on which to update distance and gradients
  - the \initial_pose topic to allow manually setting the initial guess
  - the \odom msg that expresses the position of the robot w.r.t. the odom frame (dead reckoning)
  - The \scan msg containing a laser scan

Full Fledged Implementation

- Whenever it receives an odomery reading **U**\in **SE2** (relative position measured from the encoders), it updates its pose

  U= odom_before.inverse()*odom_now

  X=**X***U**

- Whenever it receives a scan, it computes the endpoints in the robot frame

  x_i=z_i*cos(angle_i), y_i=z_i*sin(angle_i)

  and updates the estimate based using GN registration (seen before)

Full Fledged Implementation

The localizer publishes the pose of the robot w.r.t the map through a tf message, but not directly, as this would generate more than one root in the transform tree

- Map→base_link = T_localizer
- Odom→base_link = T_odom

hence it generates a transform Map→Odom s.t.

T_localizer=T_exported*T_odom

T_exported=T_localizer*T_odom.inverse()