

Robot Programming

Localizing on a Distance Map

G. Grisetti

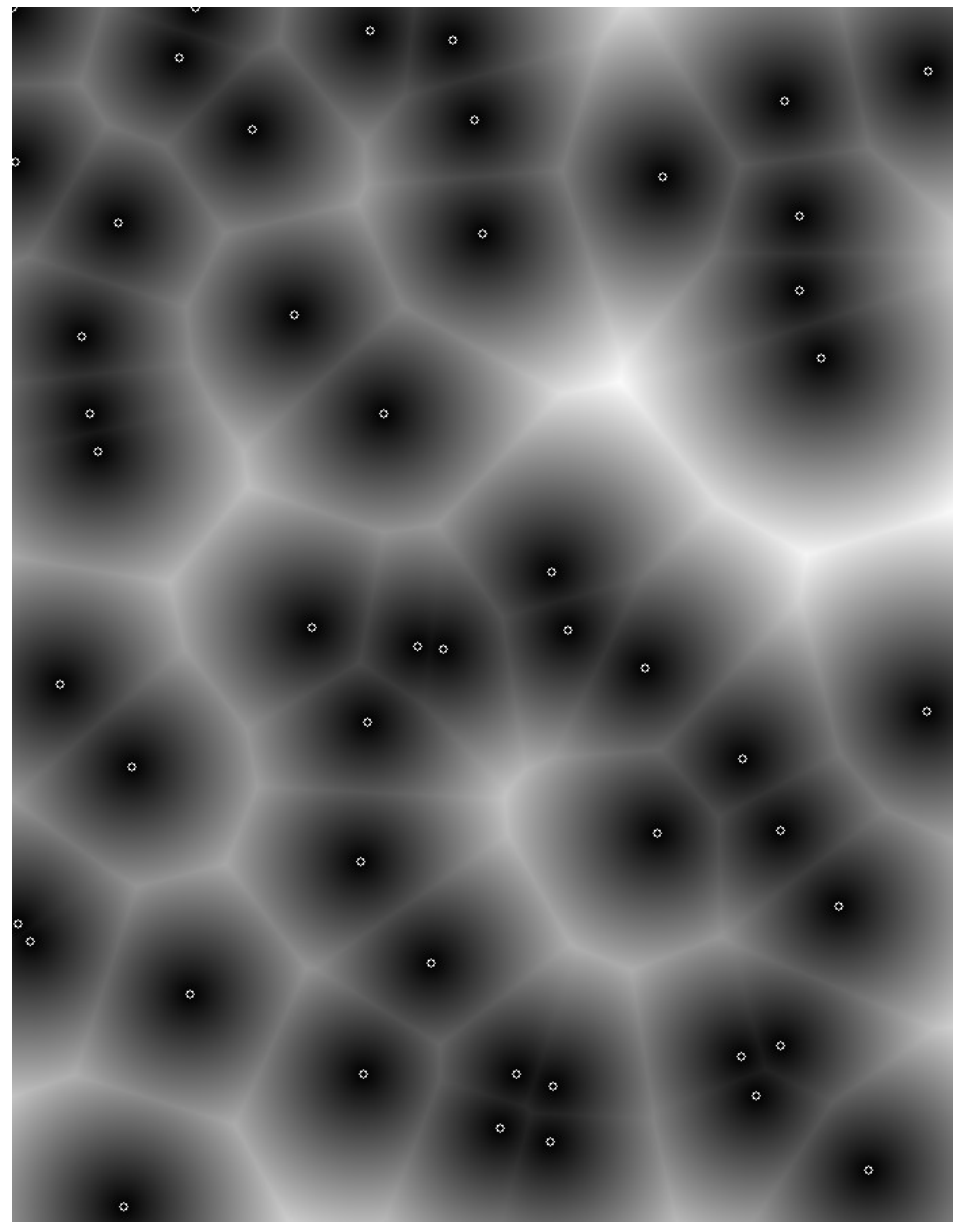
`{grisetti}@diag.uniroma1.it`

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



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

Endpoint in robot frame

Robot pose

Dmap for Localisation

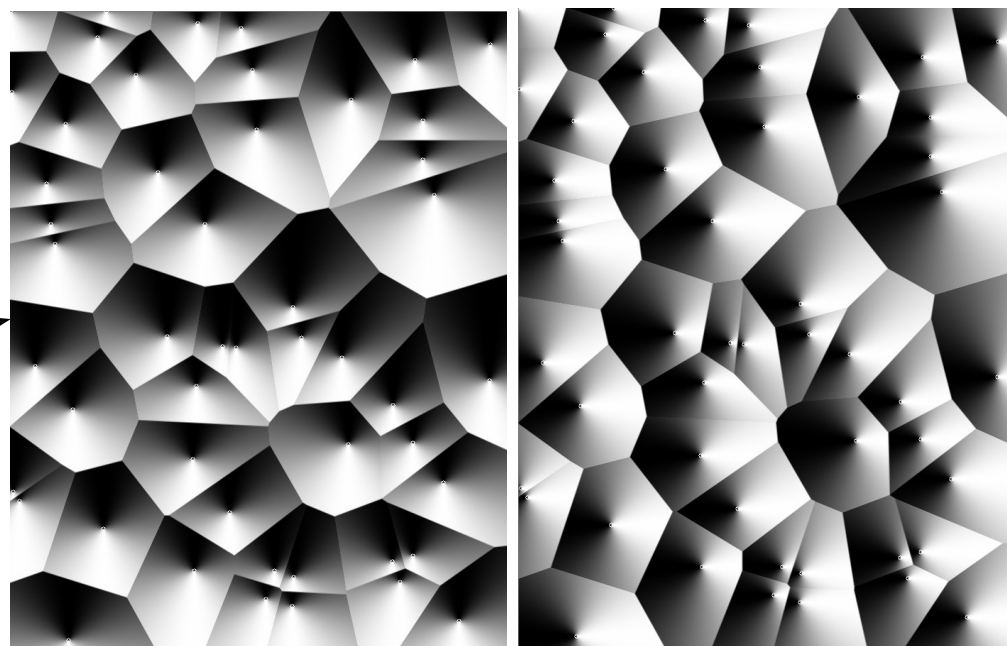
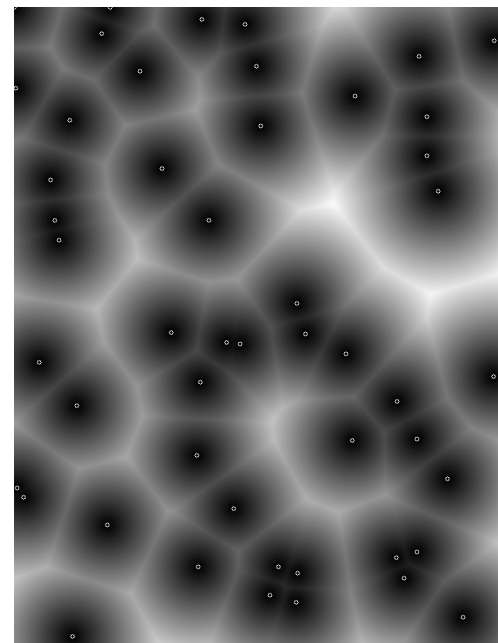
Let our reference be the set of points shown to the right

- 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

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



Gauss-Newton

Iteratively solve this problem

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

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

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

Requires $\mathbf{e}(\cdot)$ to be differentiable w.r.t. and euclidean perturbation around \mathbf{X}

Gauss-Newton(on manifold)

Clear \mathbf{H} and \mathbf{b}

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

For each measurement, update \mathbf{h} and \mathbf{b}

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

$$\mathbf{H} \leftarrow \mathbf{H} + \mathbf{E}_i^T \mathbf{E}_i$$

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

Update the estimate with the perturbation

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

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

Gauss-Newton(on manifold)

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

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

For each measurement, update h and b

$$\mathbf{p}_i \leftarrow \mathbf{X}^* \mathbf{z}_i$$

$$\mathbf{e}_i \leftarrow \mathbf{d}(\mathbf{p}_i)$$

$$\mathbf{E}_i \leftarrow \text{grad}^T(\mathbf{p}_i) \begin{pmatrix} 1 & 0 & -\mathbf{p}_i.y \\ 0 & 1 & \mathbf{p}_i.x \end{pmatrix}$$

$$\mathbf{H} \leftarrow \mathbf{H} + \mathbf{E}_i^T \Omega_i \mathbf{E}_i$$

$$\mathbf{b} \leftarrow \mathbf{b} + \mathbf{E}_i^T \Omega_i \mathbf{e}_i$$

Update the estimate with the perturbation

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

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

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

Hands On

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 odometry reading \mathbf{U} in **SE2** (relative position measured from the encoders), it updates its pose

$$\mathbf{U} = \text{odom_before.inverse()} * \text{odom_now}$$

$$\mathbf{X} = \mathbf{X} * \mathbf{U}$$

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

$$x_i = z_i * \cos(\text{angle}_i), y_i = z_i * \sin(\text{angle}_i)$$

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

- $\text{Map} \rightarrow \text{base_link} = T_{\text{localizer}}$
- $\text{Odom} \rightarrow \text{base_link} = T_{\text{odom}}$

hence it generates a transform $\text{Map} \rightarrow \text{Odom}$ s.t.

$$T_{\text{localizer}} = T_{\text{exported}} * T_{\text{odom}}$$

$$T_{\text{exported}} = T_{\text{localizer}} * T_{\text{odom}}.\text{inverse}()$$