Robot Programming

Distance Map

G. Grisetti

{grisetti}@diag.uniroma1.it

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

Neighbor Search

Given:

- a collection of vectors $\mathcal{P} = \{\mathbf{p}_n\}_{n=1:N}$ $\mathbf{p}_n \in \Re^k$
- a query vector $\mathbf{p}_q \in \Re^k$
- a distance metric $d(\mathbf{p}_n, \mathbf{p}_q) \in \Re^+$

Find either:

- the point in the collection that is the ***closest*** to the query, according to the metric

$$\mathbf{p}_i = \operatorname*{argmin}_{\mathbf{p}_n \in \mathcal{P}} d(\mathbf{p}_q, \mathbf{p}_n)$$

- the points in the collection whose distance from the query is smaller than a value ϵ

$$\mathcal{P}' = \{\mathbf{p}_i \in \mathcal{P}, d(\mathbf{p}_q, \mathbf{p}_i) < \epsilon\}$$

Distance Metrics

Examples:

- Squared Norm

$$\left\|\mathbf{p}_i - \mathbf{p}_j\right\|^2 = \left(\mathbf{p}_i - \mathbf{p}_j\right)^T \left(\mathbf{p}_i - \mathbf{p}_j\right)$$

- Omega Norm

this should look familiar

$$\left\|\mathbf{p}_i - \mathbf{p}_j\right\|^2_{\mathbf{\Omega}} = \left(\mathbf{p}_i - \mathbf{p}_j\right)^T \mathbf{\Omega} \left(\mathbf{p}_i - \mathbf{p}_j\right)$$

- Hamming distance (for binary descriptors)

Integer valued distance between two bit strings having the same dimension. Its value is the number of different bits.

example:

```
hamming(100100,101000)= 2
```

Trivial Approach

Brute Force:

compute the distance metric between the query point and *each* of the points in the collection and update the minimum.

Complexity: `O(N*cost_distance_metric)`

If we need to perform many queries, this results in unacceptable delays.

Idea: *use auxiliary search structures.*

Distance Map

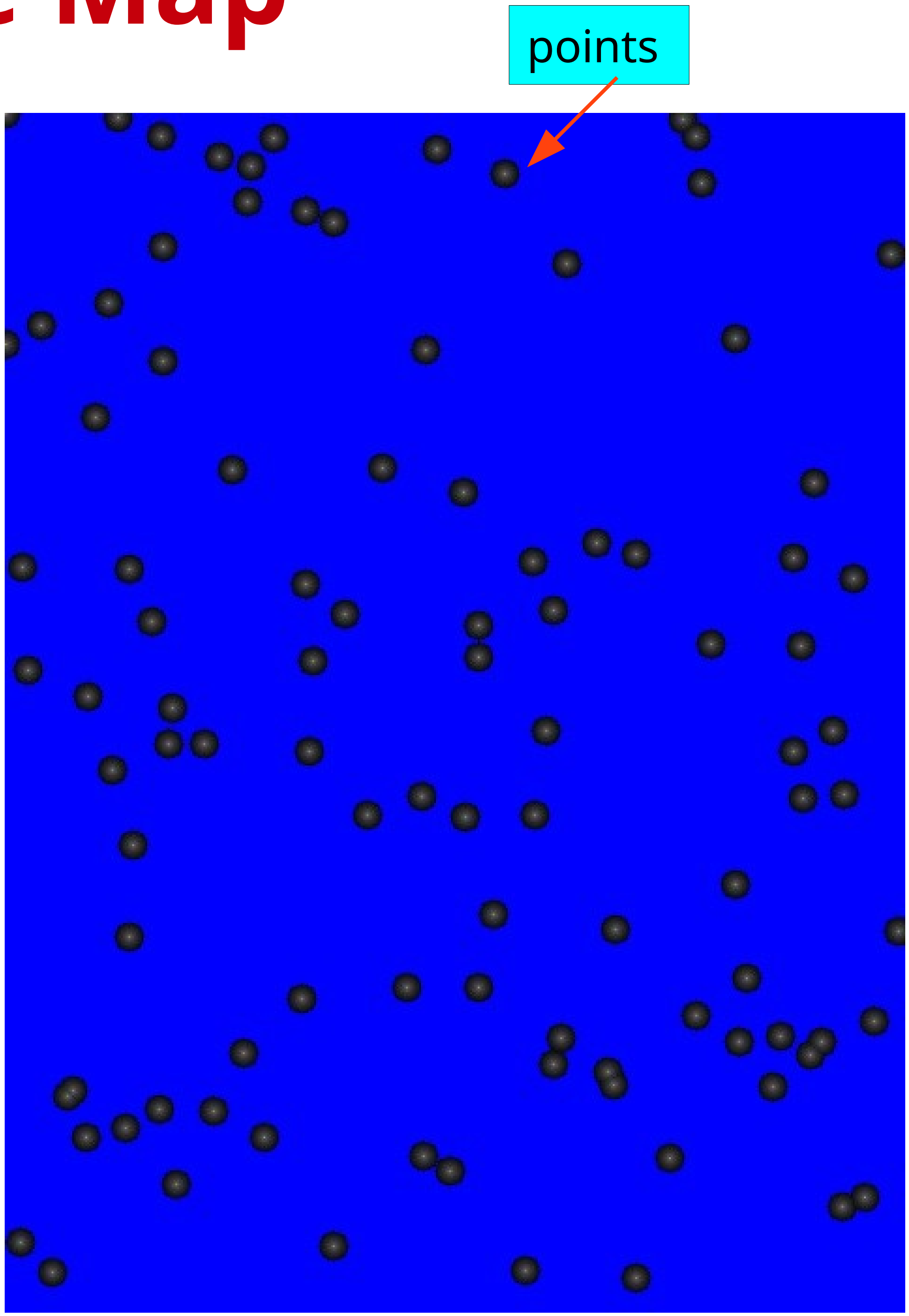
If

- the dimension of the vectors is small (< 3)
- they are spread in a relatively small region of the space

we can *pre-compute* a **grid lookup table**.

Each cell of the grid contains:

- the distance from the closest point.
- the identity of the closest point.



Distance Map

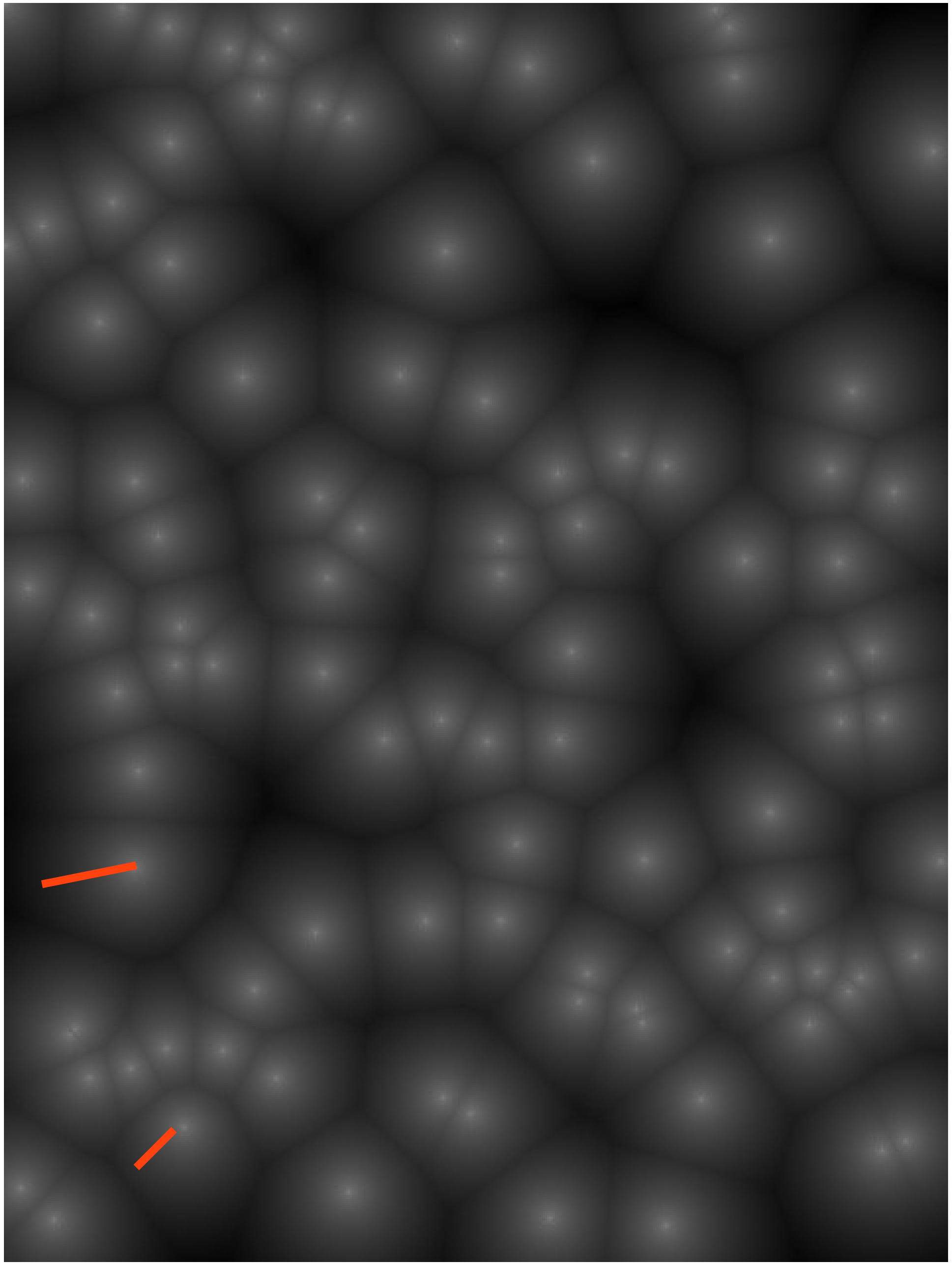
If

- the dimension of the vectors is small (< 3)
- they are spread in a relatively small region of the space

we can *pre-compute* a **grid lookup table**.

Each cell of the grid contains:

- the distance from the closest point (gray value)
- the identity of the closest point (orange line)

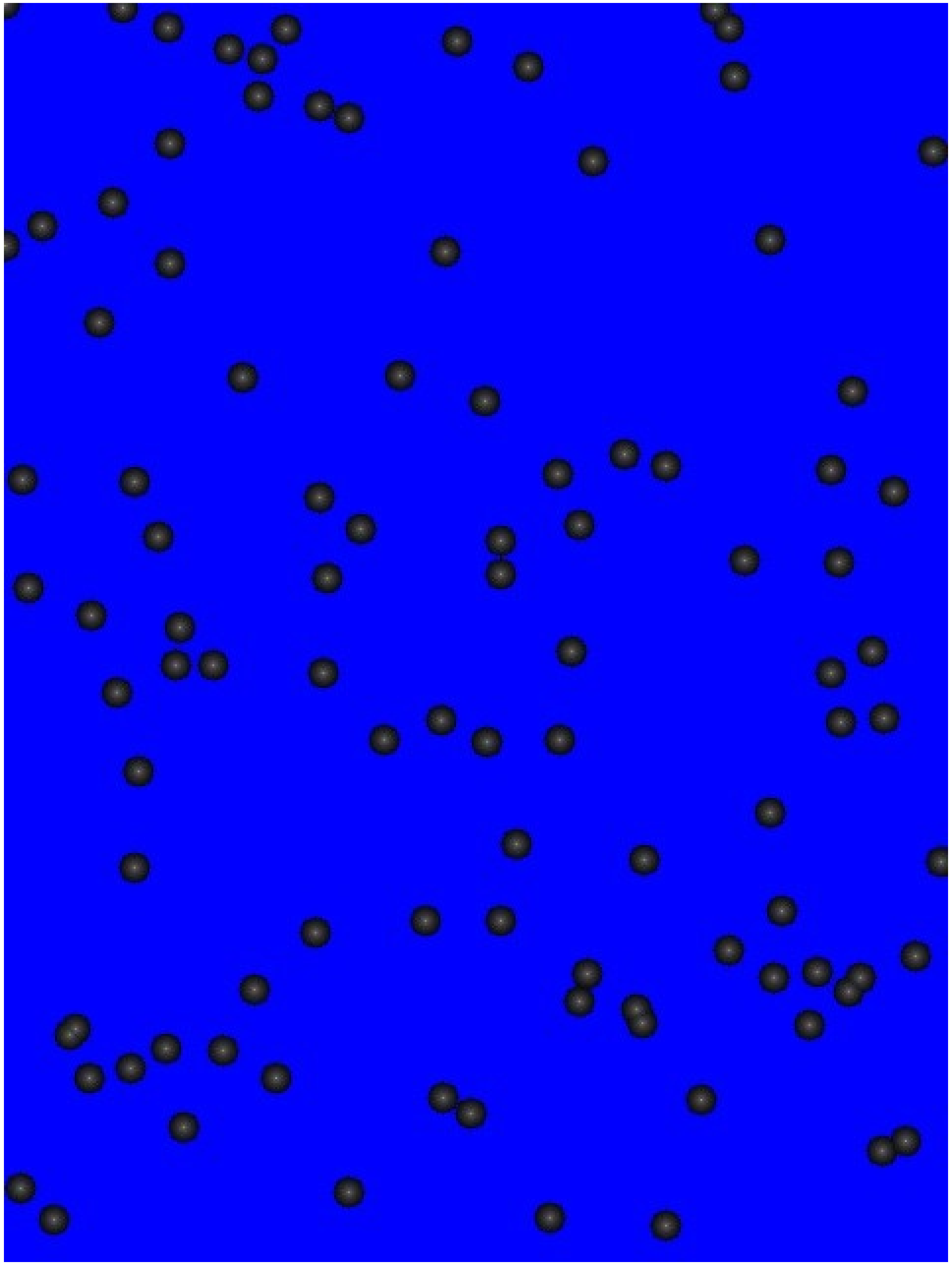


Computing the Distance Map

Modified Version of *Dijkstra algorithm.*

Each grid cell C stores:

- d: distance from nearest point
- `parent`: pointer to the nearest point



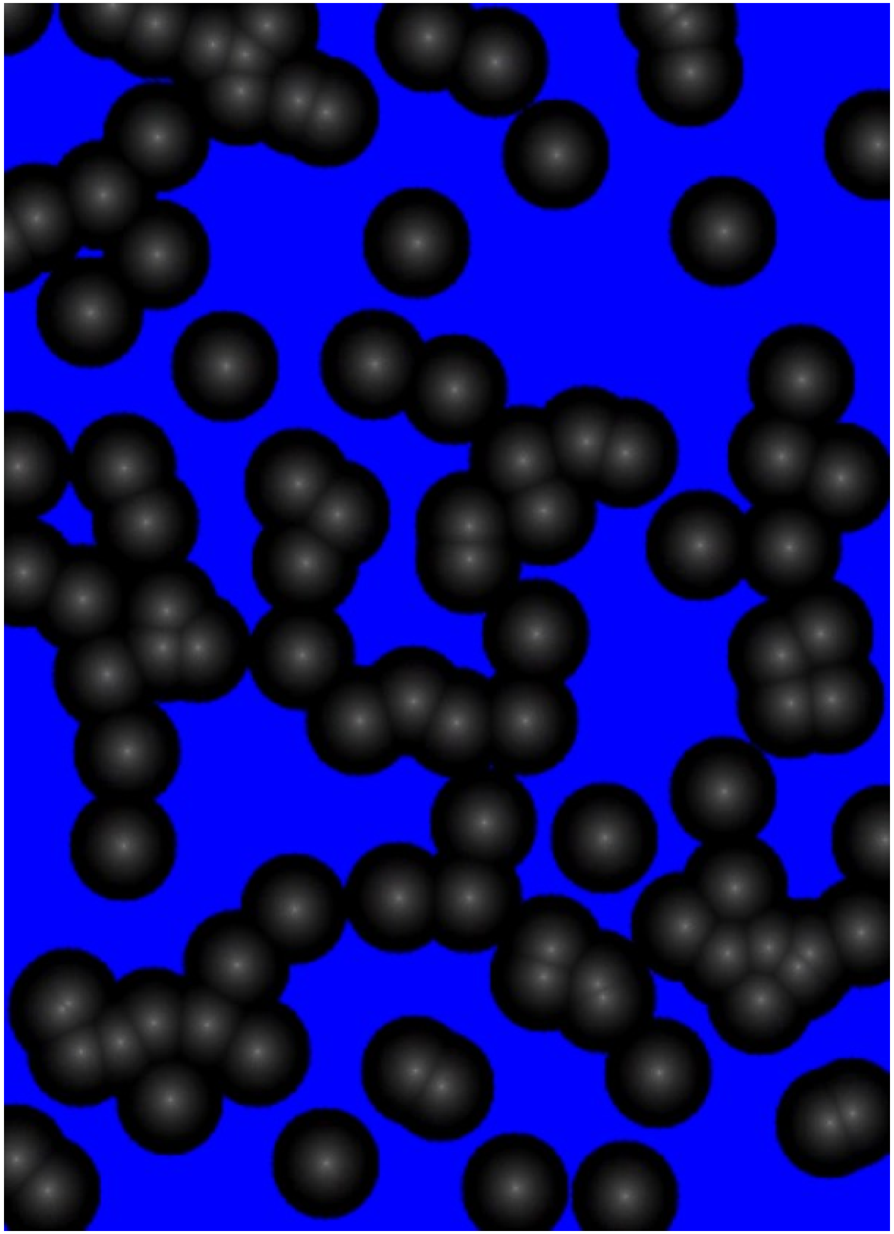
- We initialize the grid cells which corresponds to the points in $\mathcal{P}$ as:
  - `p.d = 0`
  - `p.parent = p`

Computing the Distance Map

- We expand each initialized grid cell p, *i.e.* we take the 8 neighbors.
- For each of these expanded cells c_i

```
d_{p,c} = distance(p.parent,c_i)
if (c_i.isEmpty()) {
    c_i.d = d_{p,c}
    c_i.parent = p.parent
} else if (c_i.d > d_{p,c}) {
        c_i.d = d_{p,c}
        c_i.parent = p.parent
}
```

- Continue iteratively for all the expanded cells.

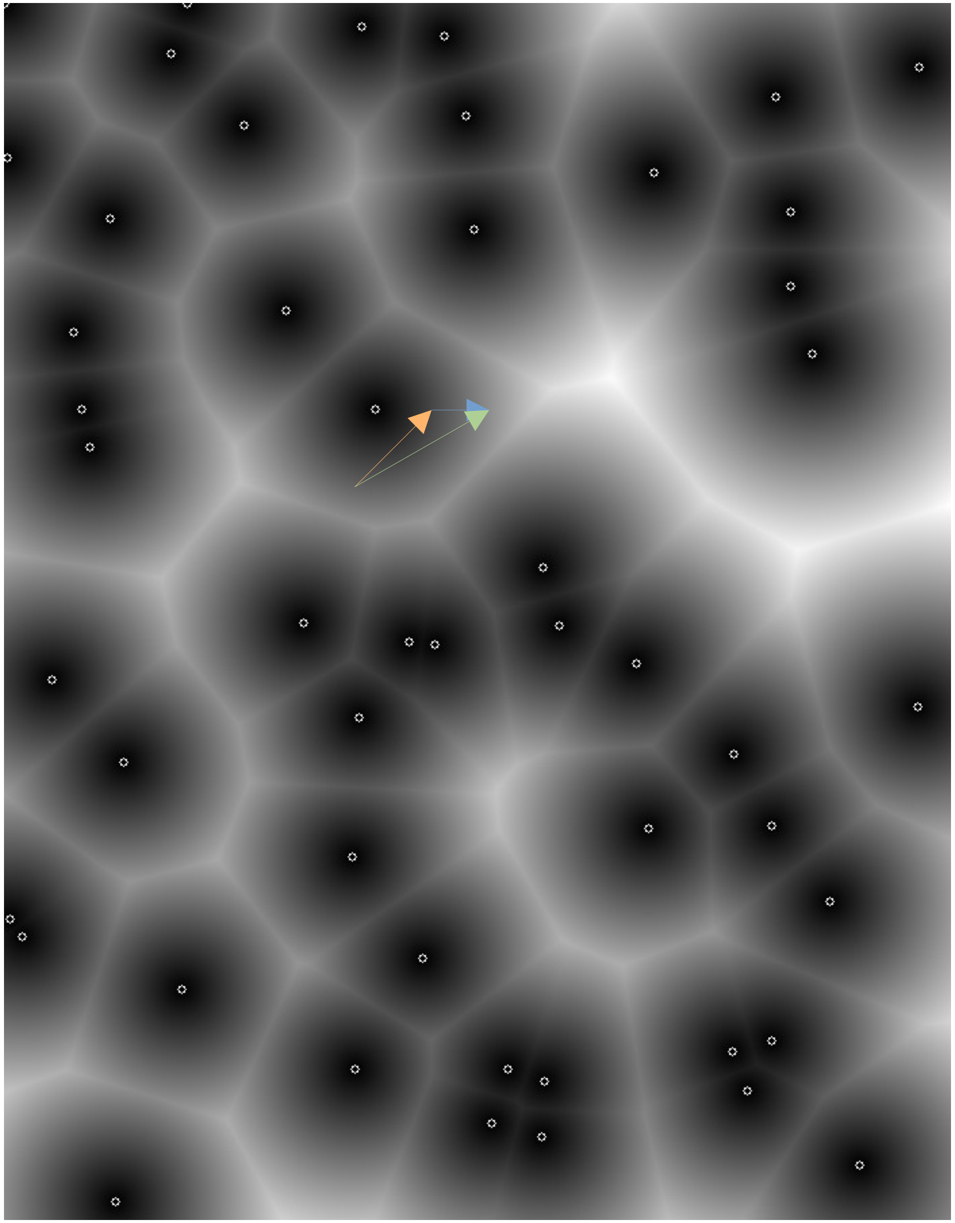


DMap for Obstacle Avoidance

Apply a force field where the force acting on the robot follows the direction of the gradient in the distance map

The intensity of the force is inversely proportional to distance

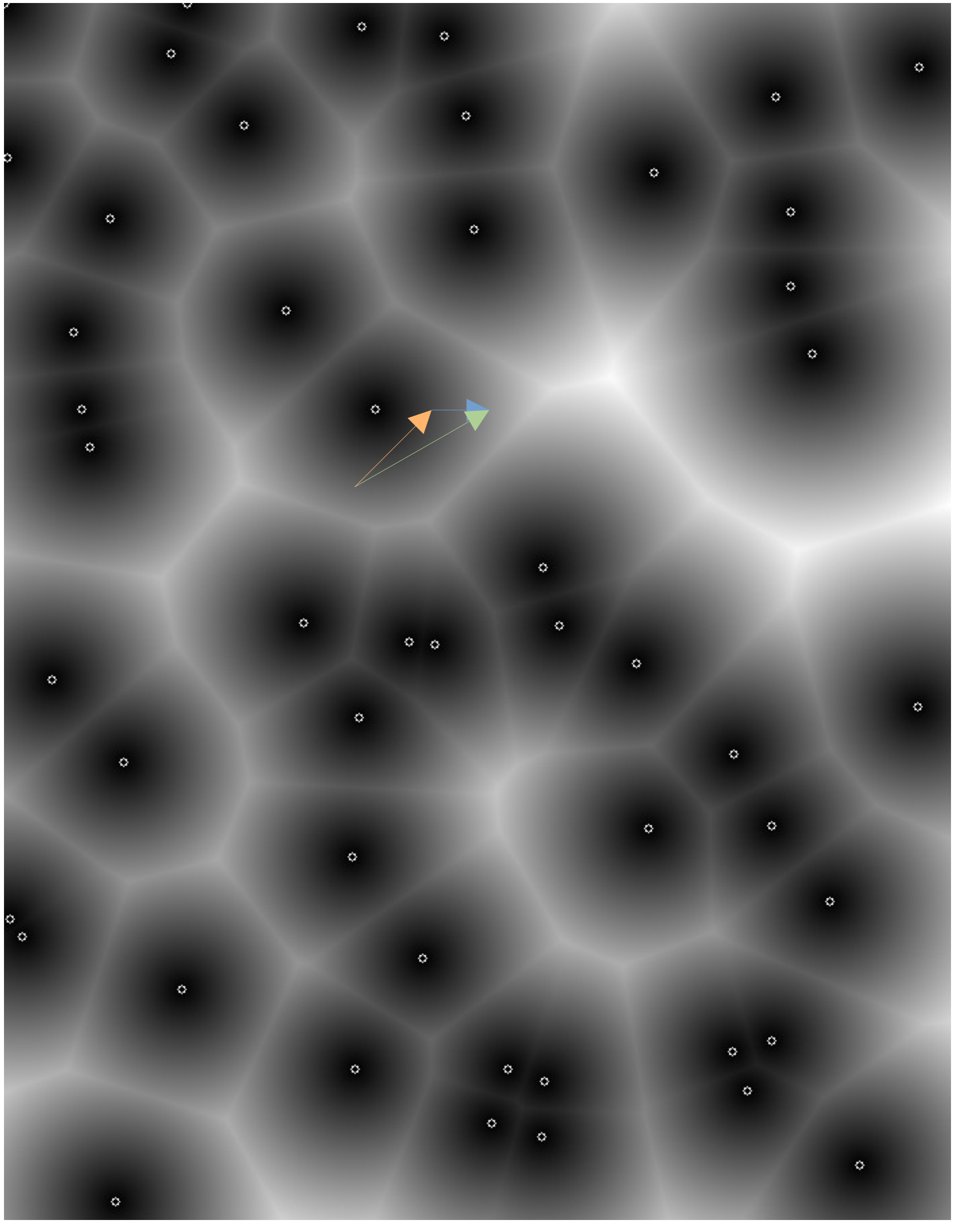
The **resulting** force applied to the robot is the sum of the "**control**" and the "**repulsion**"



DMap for Path Planning

To compute a path on the grid:

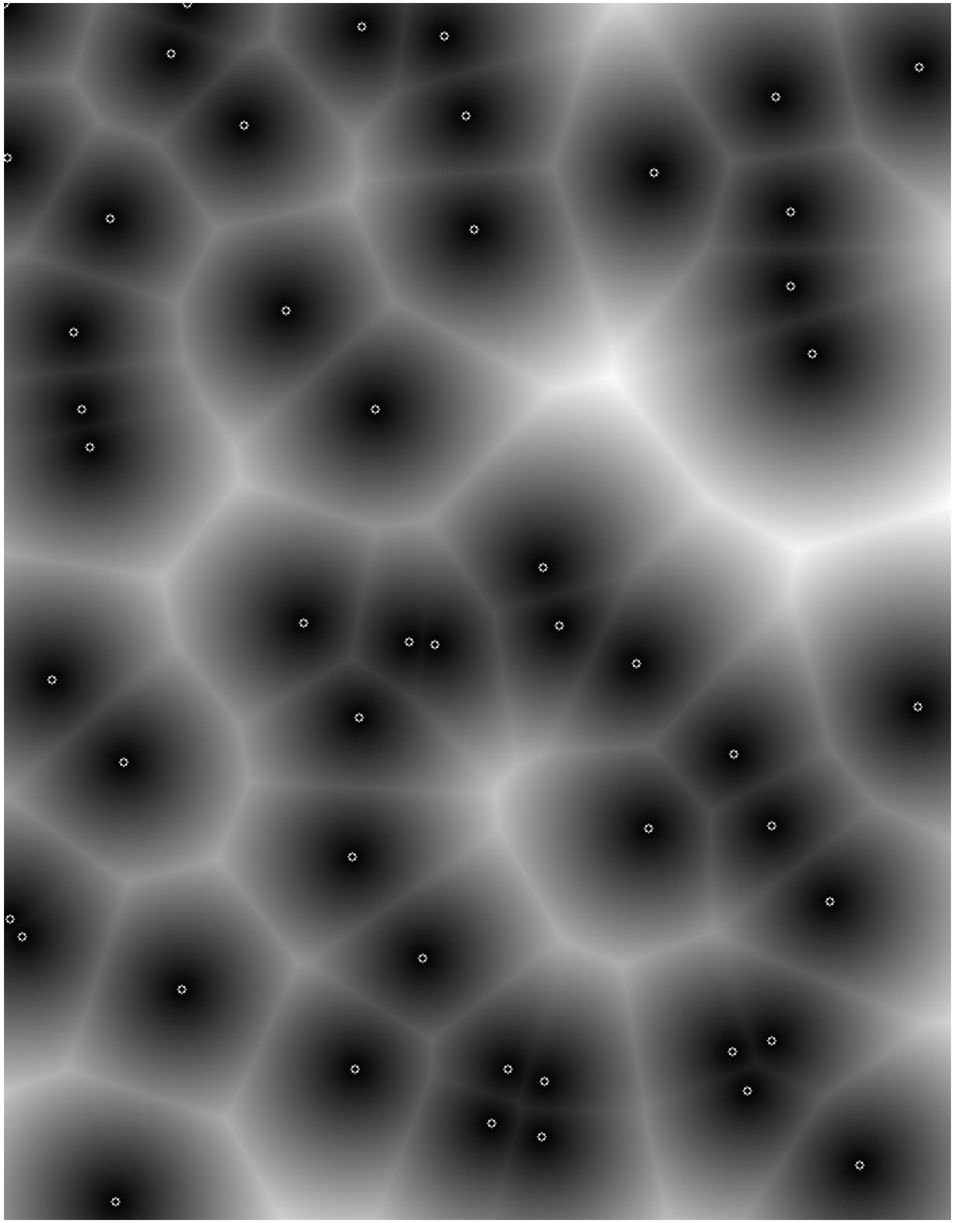
- Model it as a 8 connected graph where each cell is connected with its 8 neighbors
- The cost of moving from a cell to one of its neighbors is a (decreasing) function of the distance and the length of the motion (diagonal moves cost more)
- Use your favorite search algorithm



DMap for Path Planning

~~Use your favorite search algorithm~~

- If the environment is static, use once Dijkstra to compute the cost map over the area you want to plan on
- This gives you a policy on the closest cell to follow
- This policy is the optimal heuristic for A*
- Use A* and add obstacles to the (local map)



DMap for Localization

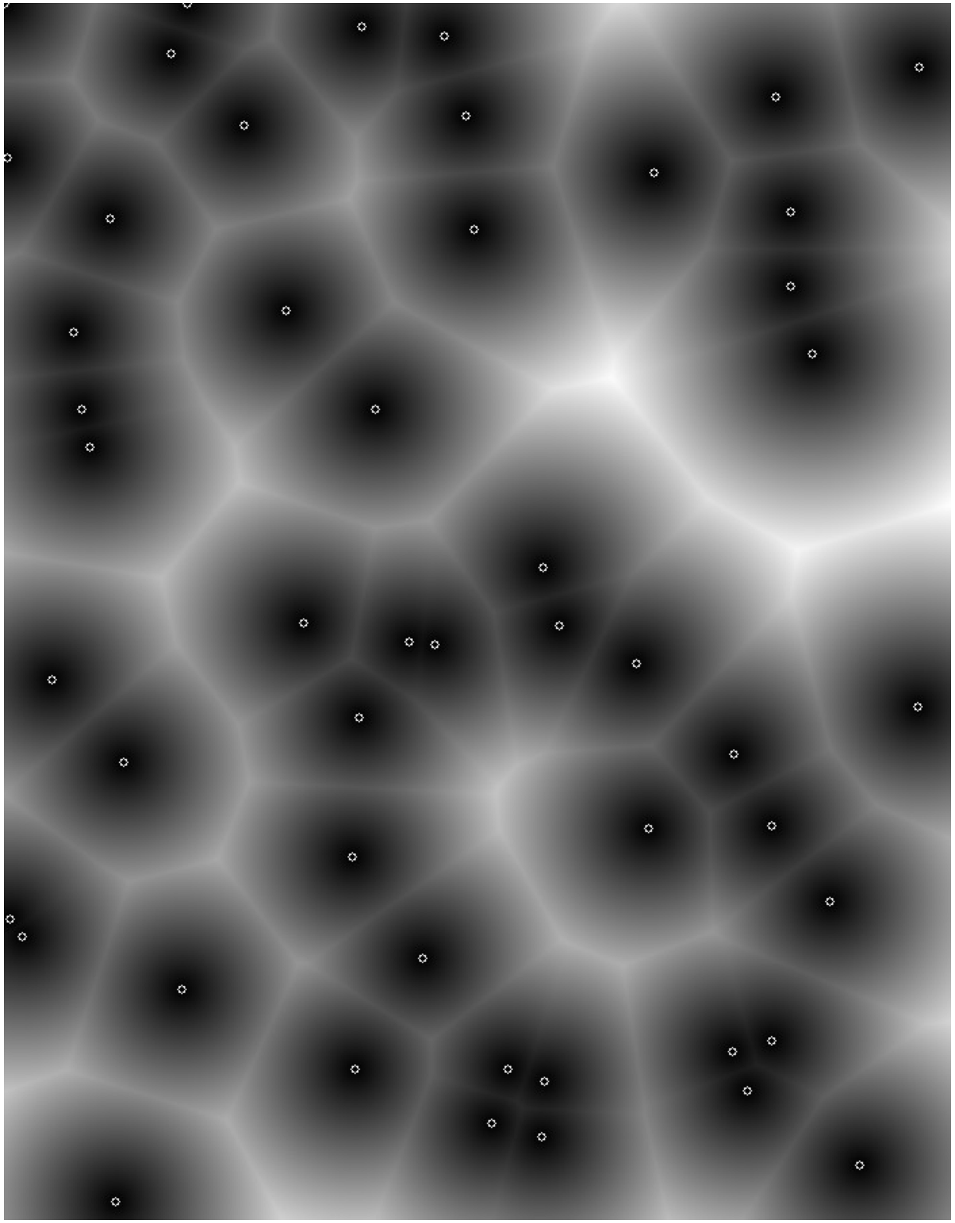
Let our reference be the set of points shown to the right

Formulate localization as a minimization problem

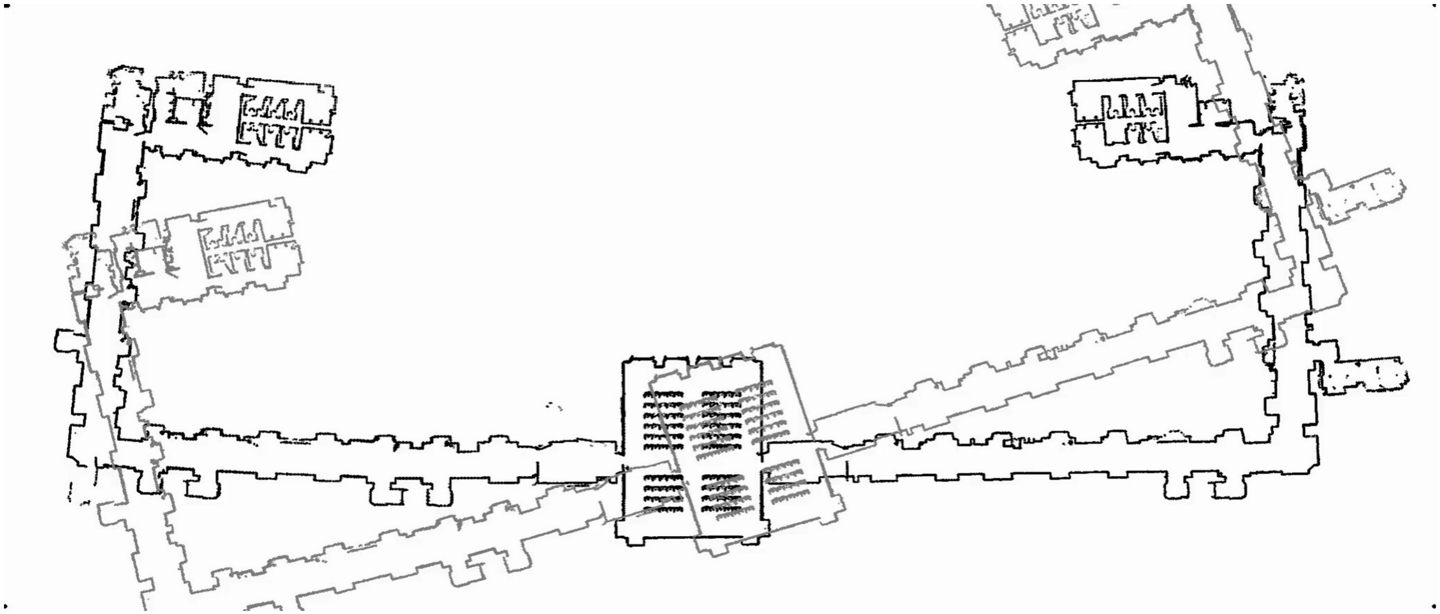
Endpoint in robot frame

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

Robot pose



DMap for Localization



In the next weeks

- We will see concrete examples on these subjects