Generalized Voronoi Diagram

At first, my algorithm gets the 2-dimensional configuration space $Q$ starting from the workspace $W$ and the robot shape $R$. Then it computates the Generalized Voronoi Diagram of the free configuration space $Q_{\text{free}}$.

Let us call the generalized Voronoi region as the closure of the set of points closest to any obstacle $QO_i$:

$$\mathcal{F}_i = \{ q \in Q_{\text{free}} | d_i(q) \leq d_h(q) \forall h \neq i \}$$

where $d_i(q)$ is the distance to an obstacle $QO_i$ from $q$.

In order to calculate the distance, I have used at first the saturated distance function:

$$\rho_R(q, \theta) = \begin{cases} 
\min_{\lambda \in [0, \infty)} d(q, q + \lambda [\cos \theta, \sin \theta]^T) & \text{s.t. } q + \lambda \cdot [\cos \theta, \sin \theta]^T \in \bigcup_i QO_i, \text{ if } \rho_R(q, \theta) < R \\
\infty & \text{otherwise}
\end{cases}$$

where the parameter $R$ is chosen bigger than the average diameter of the configuration space. I have implemented a Matlab function $\text{rho} = \text{vor_d}(Q, q, \text{step}, \text{x_max}, \text{y_max}, \text{j_max}, \text{i_max}, \text{grid}, R, R_{\text{inf}})$ in order to computate the saturated distance. The parameter $Q$ is the bitmap of the configuration space, the parameter $q$ is the given configuration and the parameter $R$ is the saturation limit.

Then I have computated for each obstacle to which the configuration $q$ has line-of-sight the minimum of the saturated distance function:
\[ d_i(q) = \min_{\vartheta \in [0, 2\pi]} \rho_R(q, \vartheta) \quad \text{s.t.} \quad q + \rho_R(q, \vartheta) \cdot [\cos \vartheta, \sin \vartheta]^T \in \mathbf{QO}_i \]

In order to find the minima, I have implemented the Matlab function \([O, m, \text{rho_min_vec}] = \text{vor_cont}(\rho, R_{\text{inf}})\). The matrix \text{rho_min_vec} contains for each obstacle the distance \(d_i\) and the angle \(\vartheta\).

Using the above functions I can define for each couple of obstacles a two-equidistant line in the configuration space:

\[ S_{ij} = \{ q \in Q_{\text{free}} \mid d_i(q) - d_j(q) = 0 \} \]

In order to refine the two-equidistant line, eliminating the portions with non-distinct gradient vectors we can define a two-equidistant surjective line:

\[ S_{ij} = \{ q \in S_{ij} \mid \nabla d_i(q) \neq \nabla d_j(q) \} \]

Moreover, in order to avoid that this line intersects other obstacles, I restrict \(S_{ij}\) to the set of points that are equidistant to the two obstacles \(\mathbf{QO}_i\) and \(\mathbf{QO}_j\) and have them as closest obstacles:

\[ F_{ij} = \{ q \in S_{ij} \mid d_i(q) \leq d_h(q) \forall h \neq i, j \} \]

Since the function \text{vor_cont} returns the distance to all the obstacles within line-of-sight of the configuration \(q\), I can easily computate \(S_{ij}\) and \(F_{ij}\). However, since the real bitmap is a discrete word in my implementation I have used the parameter \text{d_thd} as threshold to define when two distances are numerical “equal” or “different”.

Finally I can define the generalized Voronoi diagram (GVD):

\[ \text{GVD} = \bigcup_i \bigcup_j F_{ij} \]
Each edge terminates either at a meet point, which is a point equidistant to three obstacles, or at a boundary point, which is a point whose distance to the closest obstacle is zero (or less than the threshold $d_{thd}$).

In order to incrementally build the GVD, the algorithm at first moves the robot on the edge $F_{ij}$ closest to the start configuration. Then it moves the robot along the edges computing their tangent space: it passes a line through the two closest points on the two closest obstacles and takes the line perpendicular to it. Then it moves the robot along this direction by an incremental step. If the new position is no more in the equidistance interval given by the parameter $d_{thd}$ a small perpendicular motion is made in order to put the robot on the edge $F_{ij}$ again.

I have used the list $\text{vor\_meet}$ to memorize all the meet points and the list $\text{vor\_bound}$ to store all the boundary points. Then we have three different cases:

- when the robot arrives in a meet point for the first time the algorithm adds this point to the list $\text{vor\_meet}$;
• when the robot arrives in a boundary point, the algorithm takes the first element in the list `vor_meet` that is the start point of some edges not yet explored and explores one of this edges;
• when the robot arrives in a meet point that is already in the list `vor_meet`, if this meet point is connected to some edges not yet explored, the robot explores one of this edges. Otherwise the algorithm cancels this meet point from the list `vor_meet`, takes the first element in the list that is the start point of some edges not yet explored and explores one of this edges.

The algorithm terminates when the list `vor_meet` is empty.

When the program starts, push the button “Draw Obstacle” to draw an obstacle and set with the left mouse button the vertices of the obstacles within the white area (robot workspace \( W \)). To end the process click with the right mouse button somewhere. To draw the robot push the button “Draw Robot” and set with the left mouse button the vertices of the robot within the white area. To end the process, click somewhere with the right mouse button. A black point will be placed on the reference vertex of the robot. To calculate the configuration space, push the button “Compute Q”. To calculate the GVD push the button “Calculate Path”. In order to show an animation and to register a movie...
of the GVD building process, there is a check box “Movie” that has to be checked before pushing the button “Calculate Path”.