Robot Localization Simulator

A Project Report Submitted
in Partial Fulfillment of Requirements
for the Degree of

Bachelor of Technology

by

Apurv Verma(P2008CS1002)
Prateek Garg (P2008CS1022)
Kumar Ashwani (P2008CS1015)

Department of Computer Science & Engineering

Indian Institute of Technology Ropar

Rupnagar 140001, India

April 2012

Abstract

In this project we explore the various algorithms for the Robot Localization Problem and build a simulator to visualize the results on various 2D maps. Robot localization is an important problem in robotics. Simply put, the robot localization problem is as follows. A robot is placed at an unknown point inside a simple polygon P. The robot has a map of P and can compute visibility polygon from its current location. The robot must determine its correct location inside the polygon P at a minimum cost of travel distance. We implement an approximation algorithm as given by Apurva Mudgal [2006]. The paper gives an $O(\log^3 n)$ factor approximation algorithm however our main emphasis is to show the practicality of the algorithm. In this project we are simulating it on different maps without taking time complexity in consideration. Computational Geometry Algorithms Library CGA has been used for the various computational geometry algorithms.

Acknowledgements

We would like to acknowledege Assistant Professor Apurva Mudgal for helping us understand the difficult Robot Localization Algorithm. His supervision made it appear all very easy. We are also thankful to the ever enthusiastic CGAL community for helping us with the various CGAL issues.

Certificate

It is certified that the work contained in this report titled "Robot Localization Simulator" is the original work done by Apurv Verma(P2008CS1002), Prateek Garg (P2008CS1022), Kumar Ashwani (P2008CS1015) and has been carried out under my supervision.

Dr. Apurva Mudgal
Project Supervisor
Department of Computer Science & Engineering
Indian Institute of Technology Ropar
Rupnagar-140001

Contents

List of Figures

Nomenclature

Greek Symbols

γ A point inside the polygon with origin choosen at one of the hypotheses. γ can also sometimes denote a region.

Acronyms

H A list of points which denotes potential robot positions.

P_{maj} The majority rule map of all translates of the polygon. The translates of the polygon are obtained by choosing one hypothesis as the origin and translating all the remaining hypotheses to this chosen origin.

G_{ij} For the translates corresponding to pair of hypotheses h_i and h_j, G_{ij} is the origin containing region obtained by taking the lower envelope of visibility polygons of all type 1 and type 2 edges of the union-polygon.
An edge is of type 1 or type 2, if it belongs to exactly one of the translates.

g_i Set of all points at which h_i does not share the majority opinion about i.

K_i The majority rule map of G_{ij}'s

$Maj(\gamma)$ Set of hypothesis which share the majority opinion about γ.

Chapter 1

Robot Localization Algorithm

Input:

Map polygon P, the visibility polygon V.

Output:

The robot localizes to its actual position $h \in H$

Compute the set of hypotheses H.
while $|H| > 1$ **do**
    Compute the majority-rule map P_{maj}
    Compute the polygons G_{ij} for each pair of hypotheses, h_i and h_j
    Compute the majority rule map K_i of G_{ij}'s
    Find the edges on the boundary of K_i which are not on the boundary of P_{maj}
    Draw grids and compute the set of coordinates Q_H on these edges.
    Make instance $I_{P,H}$ of $\frac{1}{2}$ -Group Steiner Problem
    Solve $I_{P,H}$ to compute a half computing path $C \subset P_{maj}$
    Half-Localize by tracing C and making observations at coordinates Q_H
    Move back to the starting location.
end while

Chapter 2

Geometrical Algorithms

2.1 Computing Visibility Polygons

Visibility polygon is an indispensable component in the hypothesis generation step of the algorithm. Since CGAL had no inbuilt support for computing visibility polygons we implemented the following two routines for our purposes.

- Visibility Polygon of a point inside a polygon
- Visibility Polygon of an edge of the polygon.

2.1.1 Visibility Polygon of a Point Inside a Polygon

Definition 1 Visibility Polygon of Point: *p is the bounded polygonal region of all points of the polygon visible from p.*

The following is the C++ function in the file PolygonUtil.cpp. We pass the map polygon and the point P, whose visibility polygon is to be computed. The function returns the visibility polygon of P as another Polygon type. In $setVisiblePoints$ we collect all those points which are directly visible from P. Note that all these points will form a part of the visibility polygon of P since they are directly visible from P. These points can contain some reflex vertices as well. A reflex vertex occludes a portion of map resulting in a spurious vertex [Guibas and Raghavan [1995]]. Each spurious vertex which is a part of the visibility polygon can be obtained by extending the line joining the point P to a reflex vertex. We do this in the if block and collect all the spurious vertices also. Finally we sort all the vertices obtained by traversing along the boundary of the map in anticlockwise direction to get the visibility polygon of P.

```
Polygon PolygonUtil::CalcVisibilityPolygon(Polygon& map, Point& point)
{
  Polygon setVisiblePoints = VisiblePointSet(map,point);
  list<Point> listVisiblePoints;

  for (VertexIterator vi = setVisiblePoints.vertices_begin(); vi != setVisiblePo
  {
    listVisiblePoints.push_back(*vi);
    if(IsReflex(map,*vi))
    {
      Ray rayToCorner(point,*vi);
      std::list<Point> intPointList;
      std::list<Point>::iterator it;
      FindCandidatePoints(map,rayToCorner,intPointList);
      if(intPointList.size() > 0){
        intPointList.sort(CompareDistance1);
        Point spuriousVertex = *intPointList.begin();
        listVisiblePoints.push_back(spuriousVertex);
      }
    }
  }
  return sortPoints(listVisiblePoints);
}
```

Algorithm

1. Collect all the vertices of the polygon which are visible from the point P.

2. Iterate over the list of visible vertices and for each reflex vertex, compute

the spurious vertex introduced in the visibility polygon.

3. Finally sort all the vertices in an order so that they form a simple polygon.

Examples

The point P is shown in blue and the visibility polygon is shown in green.

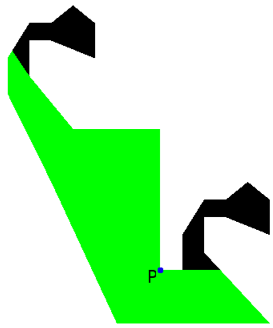

Figure 2.1: Visibility Polygon of Point

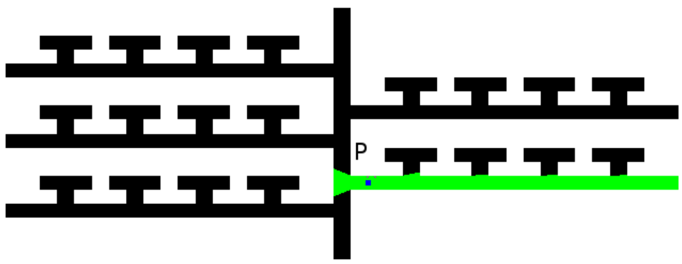

Figure 2.2: Visibility Polygon of Point

2.1.2 Visibility Polygon of an edge of the polygon

Definition 2 Visibility Polygon of Edge: *e is the bounded polygonal region of all points of the polygon visible from any point on the edge e.*

The algorithm for the visibility polygon of an edge has been taken from Guibas. Let E be the set of edges of the polygon. To find the visibility polygon of an edge AB, we compute, for each of the remaining edges of the polygon the portion of it which is weakly visible from the edge AB. Once we obtain these portions we join all of them to obtain the visibility polygon of the edge AB. Implementation of this algorithm requires computing shortest path between vertices of the polygon, the construction of which we describe in the next section. For now assume that we have at our disposal a routine which gives the shortest path between two vertices of the polygon as a list of Point type.

The main steps of computing the visible portion of an edge CD from another edge AB of the polygon can be enumerated as follows.

Algorithm

1. Compute the shortest path P_{AC}, from A to C and the shortest path P_{BD}, from B to D. Call this pair 1.

2. Similarly compute the shortest path P_{AD}, from A to D and the shortest path P_{BC}, from B to C. Call this pair 2.

3. Find out which of these pairs is outward convex. An outward convex pair implies an hourglass shape is formed by the two paths.

4. If none of the pairs is outward convex this means that no portion of edge CD is visible from any point on edge AB and we can completely ignore such an edge.

5. If one of the pairs is outward convex then without loss of generality, let pair 1 be the outward convex pair. Now compute the shortest paths P_{AD} and P_{BC}.

6. Let X be the point where path P_{AD} and P_{AC} split and let W be the point where path P_{BD} and P_{BC} split. Let Y be the next point on the path P_{AD} and Z be the next point on the path P_{BC}. Extending XY we get one extreme point of the portion of CD visible from AB. We repeat this on other side to get the other extreme point.

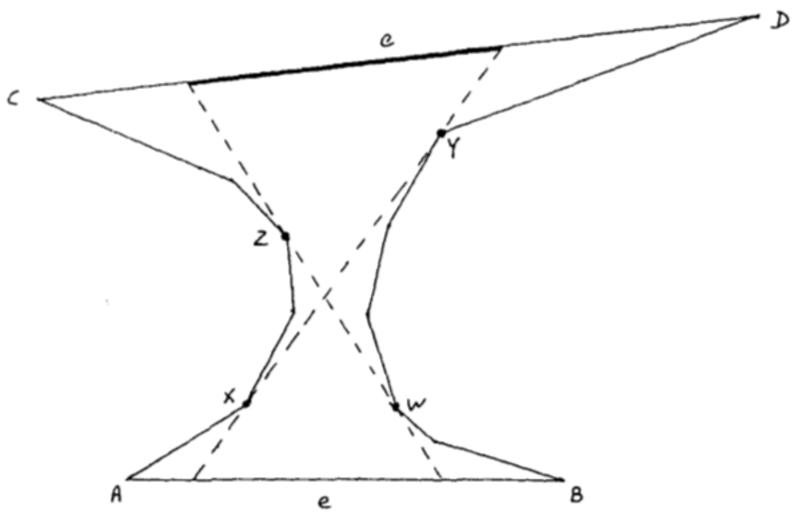

Figure 2.3: Visibility Polygon of Edge, Illustration taken from:Guibas

CalcVisibilityPolygonEdge() calculates the visibility polygon of an edge in PolygonUtil.cpp

Note: This routine computes the visibility polygon of the edge excluding its endpoints. To obtain the visibility polygon of the edge where endpoints are inclusive one could compute the visibility polygon of point at the two endpoints and take a union with the visibility polygon returned by this routine.

2.1.3 Shortest Path Calculation

For the calculation of shortest path between any two vertices of the polygon the following property was exploited.

- The shortest path must turn only at vertices of the polygon.
- It is possible to move from one vertex to the another only if they are visible to each other.

Definition 3 Visibility Graph *The visibility graph of a polygon can be formed as follows. Draw a vertex corresponding to each vertex in the polygon. Draw an edge between two vertices if the line joining the corresponding vertices in the polygon lies completely inside the polygon.*

For Example

Error in inserting figure babaji

Utilizing these properties we construct a visibility graph for the polygon and use the normal dijikstra's single source shortest path algorithm of Boost Graph Libraray BOO on the visibility graph obtained. The following two functions do the above mentioned tasks.

- `PolygonUtil::PrepareVisibilityGraph(Polygon& map, Point vertex[])`

- `PolygonUtil::CalcShortestPath(int source,graph_t& g,Point vertex[])`

Examples

The edge is shown in blue and its visibility polygon is shown in green.

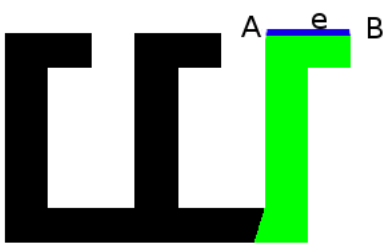

Figure 2.4: Visibility Polygon of Edge

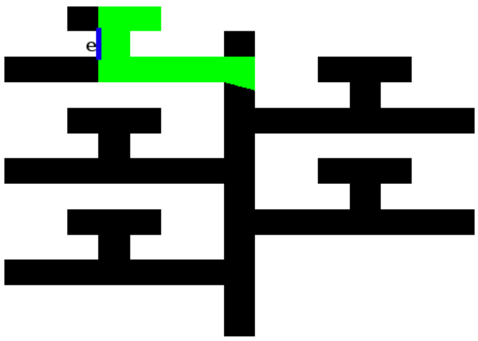

Figure 2.5: Visibility Polygon of Edge

Chapter 3

Hypothesis Generation

3.1 Hypothesis Generation

Hypothesis Generation phase is based on the following conjecture which we try to prove in the next subsection.

Theorem 4 *A point, P inside a simple polygon sees atleast one edge of the polygon completely.*

The proof of the above theorem comes from the following two simple facts.

- An edge is partially visible from a point inside the polygon only if it is occluded partially by another reflex vertex of the polygon not belonging to that edge.
- A reflex vertex can occlude one and only one edge of the polygon.

To prove the theorem for any arbitrary polygon we obtain the visibility polygon of point P and show that atleast one edge of this visibility polygon, which is also an edge of the original polygon, is completely visible from point P. Alternatively Theorem 1 can be restated as follows.

Definition 5 Spurious Edge: *In the visibility polygon of a point, an edge is called a spurious edge if it is obtained by extending the line joining the point P and a reflex vertex till it meets the polygon.*

Additional details about spurious edges and vertices can be obtained from Guibas and Raghavan [1995]

Theorem 6 *The visibility polygon of a point P has atleast one edge which completely overlaps with an edge of the original polygon.*

Proof Let the visibility polygon be V. Let the visibility polygon have n non-spurious edges and r spurious edges. Each of the spurious edge is due to a reflex vertex, so the polygon would have r reflex vertices at least. According to the theorem, one of the n edges must overlap completely with an edge of the polygon. We prove it by contradiction. Assume to the contrary that all of the n edges are partially visible from P. Thus each of the n edges must be occluded by a reflex vertex v not lying on the edge. And since a reflex vertex cannot occlude multiple edges, therefore a total of n reflex vertices will be required to occlude the n edges. So such a polygon, if it exists, should have $n + r$ reflex vertices, but it is not possible to construct a closed polygon with all vertices as reflex vertex. Hence our original assumption was wrong.

3.2 Algorithm

1. Iterate over the edges of the polygon and the edges of the map. and find an edge in the map which has the same length and orientation as an edge in the polygon.

2. Translate the visibility polygon such that the matching edge of the map polygon and the visibility polygon coincide.

3. For each of the remaining edges of the visibility polygon, check whether a complete match exists or not. If all the remaining edges match, the point where the origin was translated is added to the set of hypotheses.

3.3 Examples

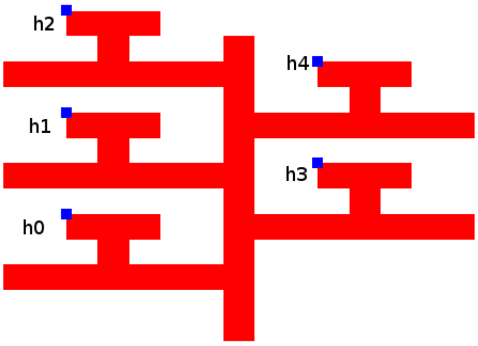

Figure 3.1: Hypothesis Generation

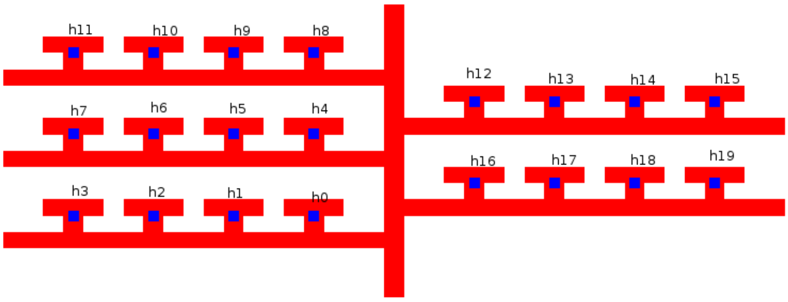

Figure 3.2: Hypothesis Generation

Chapter 4

Majority Rule Map

4.1 Computing the Majority Rule Map P_{maj}

4.1.1 Construction

The following example taken from Apurva Mudgal [2006] demonstrates the construction of a majority rule map.

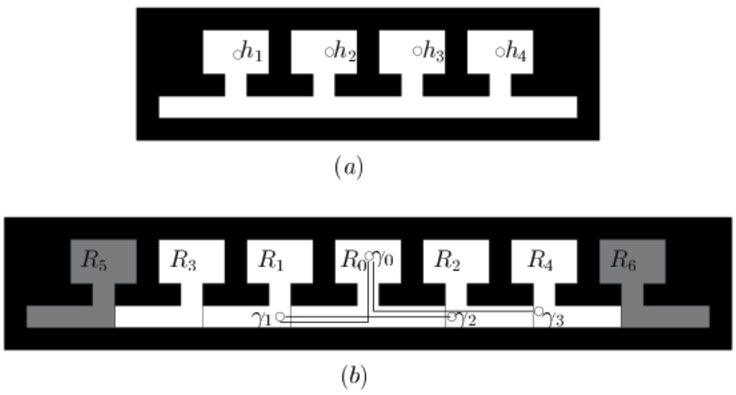

FIG. 2.1. (a) *A half-localization problem with grid graph G and $H = \{h_1, h_2, h_3, h_4\}$.* (b) *The majority-rule map for HALF-LOCALIZE(G, H) with two halving paths $(\gamma_0, \gamma_1, \gamma_2)$ and (γ_0, γ_3).*

Figure 4.1: Majority Rule Map Construction

h_1, h_2, h_3, h_4 form the set of hypotheses. Arbitrarily we choose h_1 as the origin. Next we translate all the remaining hypotheses to h_1 to obtain the overlay arrangement. The overlay arrangement contains the following faces $R_0, R_1, R_2, R_3, R_4, R_5, R_6$. Recall from the definition of $Maj(\gamma)$

$Maj(R_0) = h_1, h_2, h_3, h_4$, $Maj(R_1) = h_2, h_3, h_4$, $Maj(R_2) = h_1, h_2, h_3$, $Maj(R_3) = h_3, h_4$ and $Maj(R_4) = h_1, h_2$

In the majority rule map the region R_5 and R_6 are blocked because less than half the hypothesis said that they were traversable. They have been shown in gray.

Figure 4.2: Majority Rule Map shown in green

4.1.2 Majority Rule Map Type

The Majority Rule Map is represented as a class. The following code demonstrates the values stored along with a majority rule map and the functions applicable on it.

```
class Majoritymap {
public:
  list<Faces> listMmapFaces;
  Polygon map;
  int noOfHypothesis;
  Point *hypothesis;
  Point center;
  list<Polygon> listTanslatedPolygons;
  Arrangement mmapArrangement;
  PolygonUtil pUtil; //For using polygon util functions.
  Majoritymap();
  Majoritymap(int n, Point H[],Point c,Polygon P);
  Majoritymap(int n,std::list<Polygon> PolygonList);
  void PrintMajorityMap();
  void GenerateMajorityMap();
  Polygon GetTranslatePolygon(Transformation& translate, Polygon& polygon);
  Polygon ConvertFaceToPolygon(Arrangement::Ccb_halfedge_const_circulator circ);
  bool IsContainedIn(Polygon outer,Polygon inner);
  bool CheckPartOfMajorityMap(int agree, int noOfHypothesis);
  list<Polygon> findRegionContaningOrigin();
  bool areAdjacent(Polygon& poly1, Polygon& poly2);
  Polygon OverlayContaningOrigin(Point &center);
  void GenerateOverlay(list<Polygon> polygonList);
  void partMajority();
  virtual ~Majoritymap();
};
```

Each majority rule map is basically a collection of faces. Faces is another type that encapsulates information about the opinion of each of the hypotheses about that face. Here is the Faces class.

```
class Faces {
public:
  Polygon face;
  int noOfHypothesis;
  bool *containedIn;
  bool partOfMajorityMap;
  Faces();
  Faces(int n,Polygon p, bool *A,bool partMmap);
  Faces(Polygon p);
  void PrintDescription();
  virtual ~Faces();
};
```

Each Face has a bool flag partOfMajorityMap, which is true if this face is a part of the majority map i.e. atleast half of the hypotheses say that this face is traversable and false otherwise. It also has an bool array to specifically store the opinion of each of the hypothesis about this face.

4.1.3 CGAL's Arrangement class to generate Overlay

```
void Majoritymap::GenerateOverlay(list<Polygon> polygonList)
{
  list<Polygon>::iterator pi;

  for(pi=polygonList.begin();pi!=polygonList.end();++pi)
  {
    for (EdgeIterator ei = pi->edges_begin(); ei != pi->edges_end(); ++ei)
    {
      Point s=ei->start();
      Point d=ei->end();
      Point_2 source(s.cartesian(0),s.cartesian(1));
      Point_2 destination(d.cartesian(0),d.cartesian(1));
      Segment_2 seg(source, destination);
      CGAL::insert (mmapArrangement,seg);
    }
  }

  Arrangement::Face_const_iterator fit;
  for (fit = mmapArrangement.faces_begin(); fit != mmapArrangement.faces_end();
  {
    if (!fit->is_unbounded())
    {
      Polygon p=ConvertFaceToPolygon(fit->outer_ccb());
      Faces f(p);
      //constructing the overlay Arrangement by adding the faces as polygons.
      listMmapFaces.push_back(f);
    }
  }
}
```

First we insert all edges in the region using CGAL::insert (mmapArrangement,seg). The Arrangement class which is inbuilt in CGAL automatically generates all faces that can be formed by the intersection of the edges.

4.1.4 Algorithm

1. The overlay arrangement can be easily constructed using CGAL's inbuilt Arrangement class. Obtain the translates of the polygon by choosing one hypothesis as the origin and shifting other hypothesis to it.

2. Insert all these translates in CGAL's inbuilt arrangement to obtain all the faces in the overlay arrangement.

3. Faces which belong to atleast half the hypothesis are marked as part of the majority rule map.

4.1.5 Examples

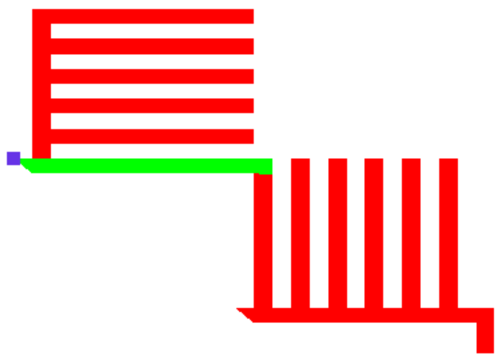

Figure 4.3: Map Polygon with robot position and Visibility Polygon

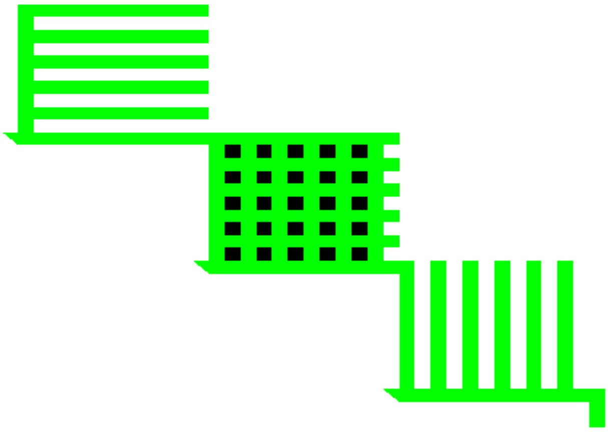

Figure 4.4: Majority Rule Map

4.2 Connected Component containing Origin in P_{maj}

We need to calculate connected component containing origin in P_{maj} as the robot can only move in this area.

For this purpose we need to make a dual graph [CGA] of all the faces of P_{maj}. Dual graph of a given planar graph G is a graph which has a vertex corresponding to each face of G and an edge joining two neighbouring faces for each edge in G.We also have a vertex for the unbounded face which is connected to all the faces sharing boundary with unbounded face.

4.2.1 Algorithm

1. Prepare a dual graph G of the traversible faces in the P_{maj}.
2. Find the vertex(v_0) corresponding to the face containing origin in dual graph G.

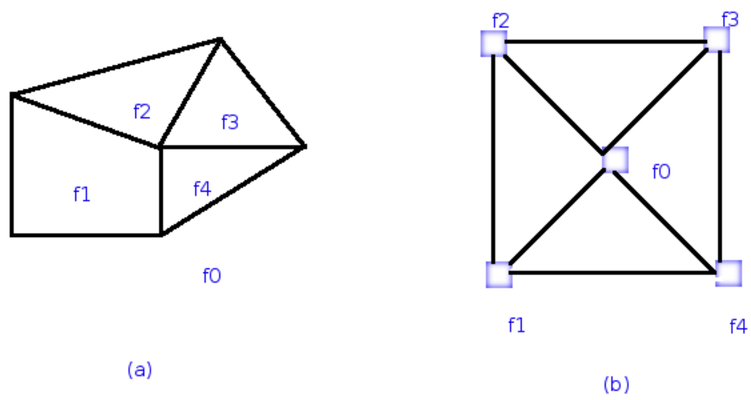

Figure 4.5: (a) Graph G (b) Dual Graph for G

3. Perform Depth First Search [BOO] from v_0 to obtain all the connected vertices.

4. Output the union of the faces corresponding to the connected vertices obtained above.

4.2.2 Examples

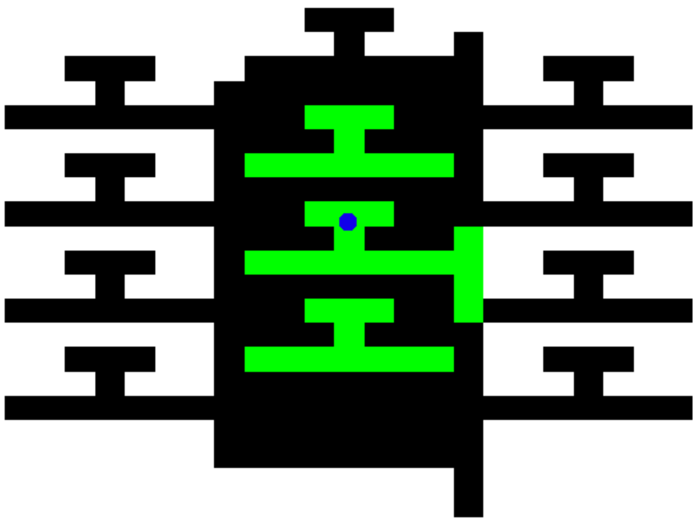

Figure 4.6: A Majority Map(blue point represents origin)

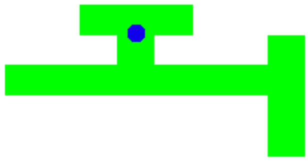

Figure 4.7: Connected Component Containing Origin

Chapter 5

Computing G_{ij}'s and K_i

5.1 G_{ij}

To compute G_{ij} we first compute F_{ij}. F_{ij} is the face containing origin in the overlay of polygons P_i and P_j. The following diagram taken from Apurva Mudgal [2006] demonstrates the notation and the construction.

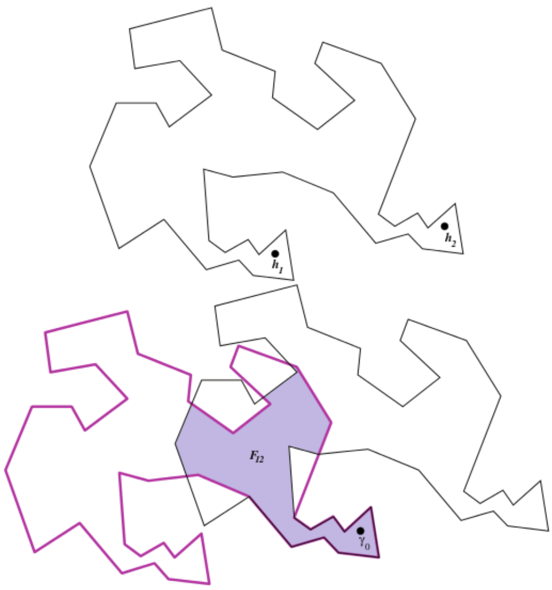

Figure 5.1: F_{12}

We state without proof the following lemma. For proof please refer Apurva Mudgal [2006]

Lemma 7 *The face F_{ij} has at most $2n$ edges.*

Each of the $O(n)$ edges on the boundary of F_{ij} can be one of three types.

1. e lies on the boundary of P_j but not on P_i
2. e lies on the boundary of P_i but not on P_j
3. e lies on the boundary of both P_j and P_i

To obtain G_{ij} from F_{ij} we draw visibility polygon of type 1 and type 2 edges and take their lower envelope.

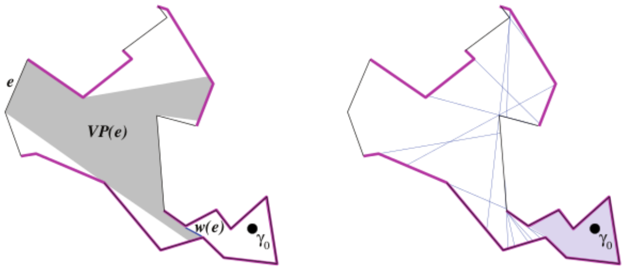

Figure 5.2: Lower Envelope

Finding the lower envelope is easy using the CGAL's inbuilt Arrangement class. We find the visibility polygon of each edge of type 1 or type 2 and insert it into an arrangement. Later we check the face which contains the point γ_0. This face is nothing else but G_{ij} The above algorithm is repeated to obtain G_{i1}, G_{i2}, G_{i3}, G_{i4}, ... G_{ik}.

5.1.1 Examples

For The below map

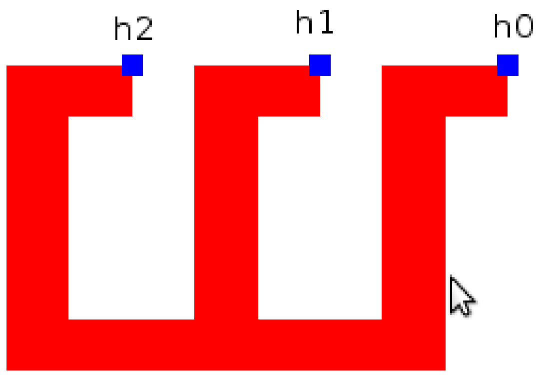

Figure 5.3: Lower Envelope

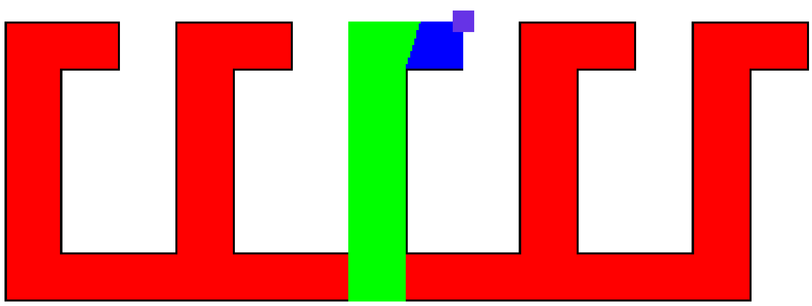

Figure 5.4: Lower Envelope

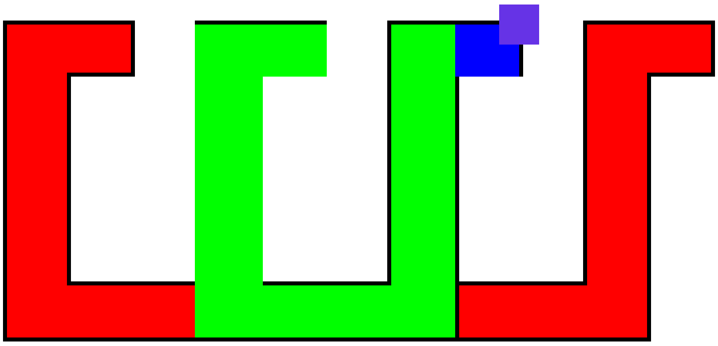

Figure 5.5: Lower Envelope

5.2 K_i

To obtain K_i we construct the majority rule map of all G_{ij}'s. K_i is a region of special interest because of the special following special property. For proof of it, please refer Apurva Mudgal [2006]

Remark 8 *A robot initially located at h_i half localizes if it crosses the boundary of K_i.*

Chapter 6

Computing Reference Points (Q_H)

6.1 Introduction to Reference Points

Definition 9 Reference Points: *are the discrete set of points on the edges of* $\partial K_i \cap \partial G_i$ *which are used to determine the half localization path of robot.*

Definition 10 Half Localization Path: *is the path travelling along which the robot can eliminate half of the hypotheses by making observations at the reference points.*

For example,

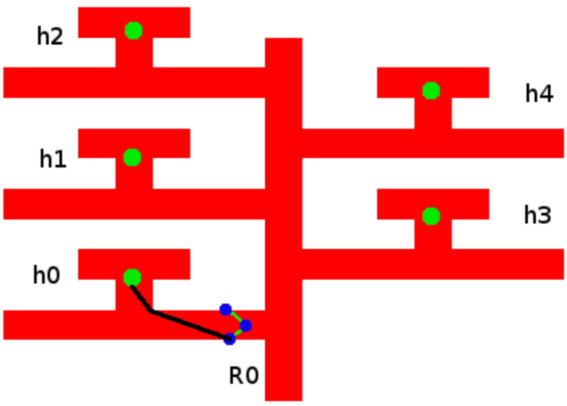

Figure 6.1: Reference Points in blue color and hypotheses in green color

The reference points shown in the figure are for the hypothesis h_0. The robot on following the path from h_0 to R_0 can differentiate between hypotheses $\{h_0, h_1, h_2\}$ and $\{h_3, h_4\}$ based on its observation at R_0. The path shown in black is the half computing path.

6.2 Algorithm

1. For every i find those edges in K_i which are not part of the boundary of majority map. These are important edges on which we will find out reference points. Let L_i contains all these edges for a particular i.

2. Calculate r_0(geodesic) radius of the smallest geodesic disk centered on γ_0 that intersects at least half of $L_i's$.

   (a) For all i calculate the minimum distance between γ_0 and any of the line segments of L_i.

   (b) Take the median of all the distances calculated above as the geodesic radius(r_0).

3. Let k be the number of hypotheses and R be a sequence of radii $r_0, 2*r_0, 4*r_0 \ldots, 2^{\lceil \log_2 k \rceil}$.

4. For every hypothesis i perform the following steps

   (a) Place each line segment σ in L_i on an axis aligned square centered at γ_0 of side length $2*2^j*r_0$ (where j from $(0..\lceil \log_2 k \rceil)$)

   (b) Decompose the square into kxk grid using $k-1$ horizontal and vertical lines.

   (c) Calculate the intersection of the line segment with the grid line. These intersection points are the reference points. Also include end points of the line segment(even if they do not lie on grid) in the set of reference points.

6.3 Examples

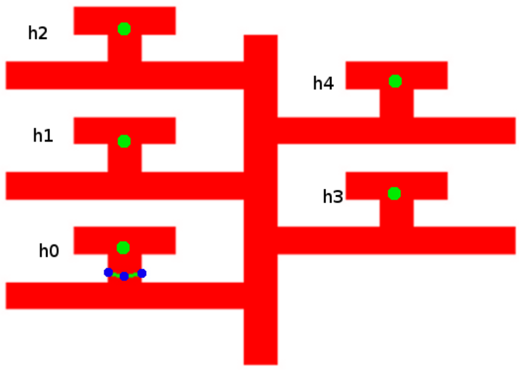

Figure 6.2: Reference Points in blue color and hypotheses in green color

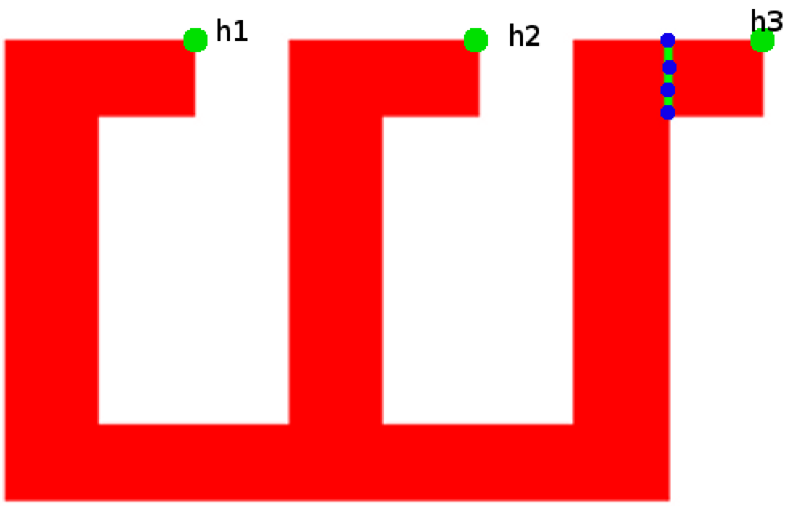

Figure 6.3: Reference Points in blue color and hypotheses in green color

Appendix1

.1 Input Format of the Map

A map is inputted in form of a scenario file. Here is a sample scenario file.

```
26
-3 0
0 -8
7 -8
4 -4
4 -2
5 -2
7 -3
7 -1
6 0
5 -1
4 -1
3 -3
3 -5
2 -5
2 3
-2 3
-4 6
-4 8
-3 8

-1 7
-1 9

-2 10
-3 9
```

```
-4 9
-5 7
-5 5
8
-4 8
-3 8
-1 7
-1 9
-2 10
-3 9
-4 9
-4.5 8



-2 8
```

The first line, 26 denotes the number of vertices in the polygon.
The next 26 lines contains 2 tuples representing the (x,y) coordinates of each point.
The next line containing 8, denotes the number of vertices in the visibility polygon.
Then 8 lines follow containing the coordinates of the visibility polygon. There is no absolute meaning to these coordinates, in the sense they are just a means to depict the shape of the visibility polygon.
The last line denotes the position of the robot within the visibility polygon.

Further Examples

and here I put some more postamble ...

References

http://www.boost.org/. 7, 19

http://www.cgal.org/. i, 18

Joseph S. B. Mitchell Craig Tovey Apurva Mudgal, Sven Koenig. A Near-Tight approximation algorithm for the robot localization problem. *SIAM Journal on Computing*, 2006. i, 12, 21, 24

Daniel Leven Micha Sharir Robert E. Tarjan Guibas, Hershberger. Linear time algorithms for visibility and shortest path problems inside simple polygons. vi, 5, 6

Motwani Guibas and Raghavan. The Robot Localization Problem. 1995. 2, 10