Robot Localization Simulator

March 27, 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 $O(\log^3 n)$ factor approximation algorithm as given by [1]. CGAL [2] has been used for the various computational geometry algorithms.

1 Robot Localization Algorithm

Input:
Map polygon P, the visibility polygon V.
Output:
The robot localizes to its actual position $h \in H$

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

2 Notation

- 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 polygon.
  An edge is of type 1 or type 2, if it belongs to exactly one of the translates.
- g_i: The 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. γ can be either a point or a region.

3 Computing Visibility Polygons

Visibility polygon is an indispensable component is 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.

3.1 Visibility Polygon of a Point Inside a Polygon

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. Besides these some additional spurious vertices will also be introduced. Each spurious vertex which is a part of the visibility polygon can be obtained by extending the line joining the point P to some reflex vertex. We do this in the if block and collect all the spurious vertices also. Finally we sort all the vertices obtained in an order so that they form the visibility polygon of P.

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.

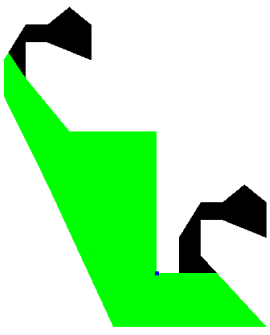

Figure 1: Visibility Polygon of Point

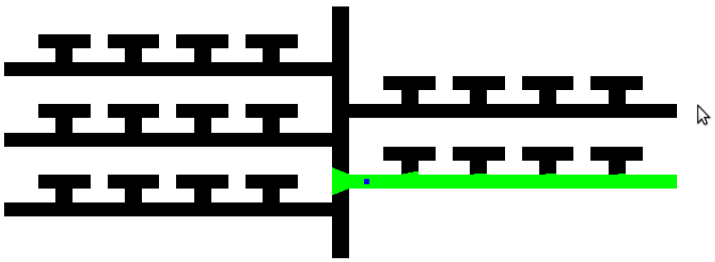

Figure 2: Visibility Polygon of Point

3.2 Visibility Polygon of an edge of the polygon

The algorithm for the visibility polygon of an edge has been taken from [3]. 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.

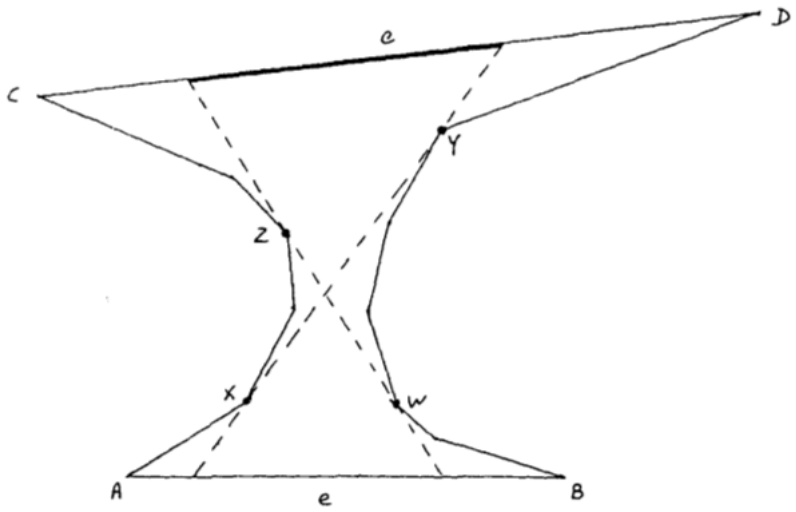

Figure 3: Visibility Polygon of Edge, Illustration taken from:[3]

$CalcVisibilityPolygonEdge()$ 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.

3.3 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 1 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.*

Utilizing these properties we construct a visibility graph for the polygon and use the normal dijikstra's single source shortest path algorithm 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[])`

Figure 4: Visibility Polygon of Edge

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

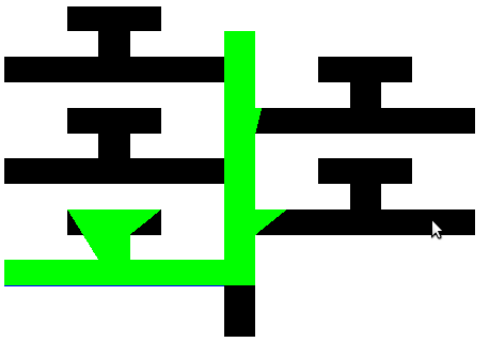

Figure 5: Visibility Polygon of Edge

4 Hypothesis Generation

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

Theorem 2 *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 3 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.*

Theorem 4 *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.

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.

Examples

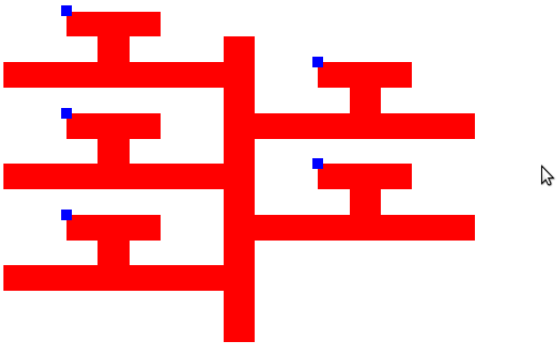

Figure 6: Hypothesis Generation

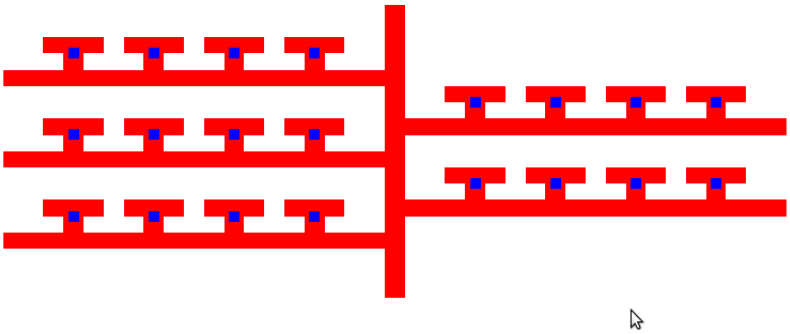

Figure 7: Hypothesis Generation

5 Computing the Majority Rule Map P_{maj}

5.1 Construction

The following example taken from [1] demonstrates the construction of a majority rule map.

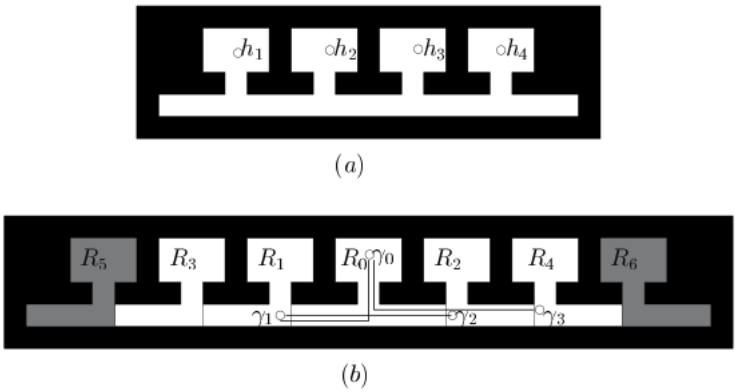

FIG. 2.1. (a) *A half-localization problem with grid graph G and $H = \{h_1, h_2, h_3, h_4\}$.* (b) *The majority-rule map for HALF-LOCALIZE(G, H) with two halving paths $(\gamma_0, \gamma_1, \gamma_2)$ and (γ_0, γ_3).*

Figure 8: 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 9: Majority Rule Map shown in green

5.2 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.

5.3 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(); ++fit)
  {
    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.

5.4 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.

Examples

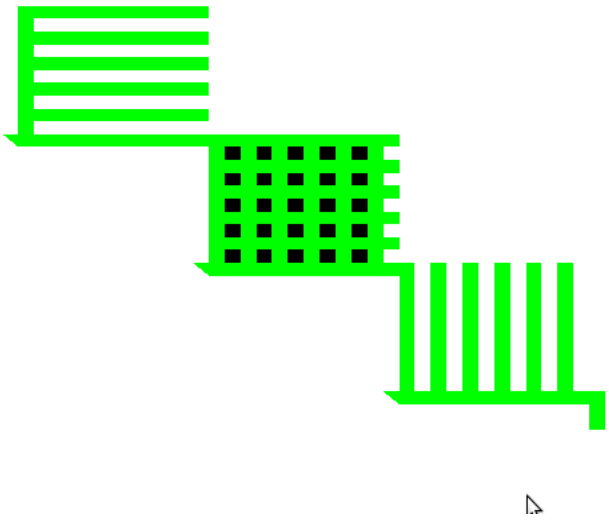

Figure 10: Majority Rule Map

6 Computing G_{ij}'s

References

[1] *A Near-Tight approximation algorithm for the robot localization problem*, Koenig, Sven and Mudgal, Proceedings of the Symposium on Discrete Algorithms SODA, 2006.

[2] <`http://www.cgal.org/`>

[3] *Linear time algorithms for visibility and shortest path problems inside simple polygons*, Guibas, Hershberger, Daniel Leven, Micha Sharir, Robert E. Tarjan