Robot Localization Simulator

February 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. Stated simply, 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 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.

1.1 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.

```
Polygon PolygonUtil::CalcVisibilityPolygon(Polygon& map, Point& point)
{
  Polygon setVisiblePoints = VisiblePointSet(map,point);
  Polygon setVisiblePointsCopy = clonePolygon(setVisiblePoints);

  VertexIterator viCopy = setVisiblePointsCopy.vertices_begin();

  for (VertexIterator vi = setVisiblePoints.vertices_begin();
                  vi != setVisiblePoints.vertices_end(); ++vi)
  {
    if(IsReflex(map,*vi))
    {
      Ray rayToCorner(point,*vi);
      std::list<Point> intPointList;
      std::list<Point>::iterator it;

      Point p1=*vi;
      Point p2;
      FindCandidateIntrPoints(map,rayToCorner,intPointList);

      if(intPointList.size() > 0){
        //Sort the points in intersectionPolygon according to distance
        intPointList.sort(CompareDistance);
        intPointList.unique();
        Point spuriousVertex = *intPointList.begin();
```

```
        p2=spuriousVertex;

        insertBefore(setVisiblePointsCopy, viCopy, spuriousVertex);
      }
    }

    viCopy++;
  }

  setVisiblePointsCopy=SortPolygon(setVisiblePointsCopy,map);
  return setVisiblePointsCopy;
}
```

1.2 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.

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.

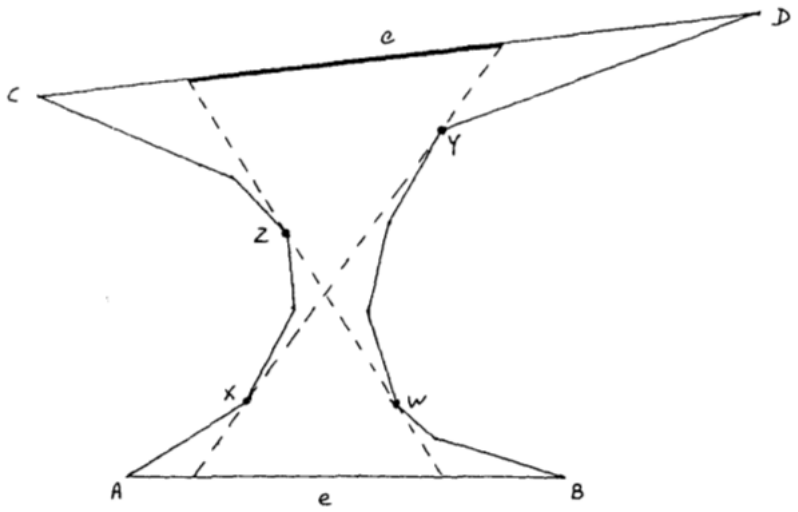

Figure 1: Visibility Polygon of Edge, Illustration taken from:[3]

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