REU Report: The LineMap Cartographer

by S. Charles Brubaker
Advisor: P.S. Krishnaprasad

U.G. 99-4

S. Charles Brubaker
8/1/99

The LineMap Cartographer

Abstract

This cartographer is designed for a two dimensional system, where sensors on the robot return the distance and angle of a detected object and where the position and orientation of the robot is known in some global frame. Based on these knowns, the coordinates of the object are calculated in the global frame and stored as points. The map consists of a series of boxes, whose length and orientation are determined by line segments formed by line fits of groups of these points. If lines grow together, they are linked, and if the environment changes, they are split or erased. The cartographer is best suited for mapping environments with straight boundaries.

Introduction

The LineMap cartographer was developed as part of the MDLe (Motion Description Language extended) project in the Intelligent Servosystems Lab at the Institute for Systems Research. The goal of the project is to put into practice the methodology laid out in V. Manikonda, P.S.Krishnaprasad, and J. Hendler's research report, Languages, Behaviors, Hybrid Architecture and Motion Control.[1] The emphasis of the project is twofold: to develop a motion description language that is nearly independent of the robot used and to develop a motion planner that does not rely on prior knowledge of the surroundings.

Accordingly the cartographer is designed to be easily adapted for any set of sensors returning a direction and a distance. Also, the cartographer has to be able to analyze and

[1]V. Manikonda,P.S.Krishnaprasad, and J. Hendler. Languages, Behaviors, Hybrid Architectures and Motion Control. Technical Research Report CDCSS T.R.98-3.

to incorporate new data into the map as it becomes available. To wait for the robot to complete its task and to then analyze the data would be useless for a robot trying to complete a task without prior knowledge of its surroundings. However, if the data can be assessed as the sensors observe it, then the robot can learn about its surroundings, recognize landmarks, and more easily achieve its goal.

Grouping the points detected by the sensors into lines takes a significant amount of time and is prone to some error; however, there is good reason to think that it will improve the ability of a robot to complete its task. For instance, it allows the planner to assess a trajectory by examining a set of lines which might contain hundreds of points, each of which would otherwise have to be examined individually. Similarly, in assessing changes to the environment, the cartographer need only check whether the line of vision intersects any currently existing lines rather than checking each point on the map. Lastly, grouping the points in lines provides an efficient way to identify and return the immediate surroundings to the motion planner.

Theory

As mentioned above, the cartographer assumes a knowledge of the robot's position and orientation , which might come from a GPS system or from another module in the architecture. The sensors must return an angle and a distance.

LineMap Algorithm

1. Get the distance returned by the sensor and the position and orientation of the robot.
2. Calculate the position in global coordinates of the point returned by the sensor. The coordinates are given by the equations:

$$x = x_{robot} + r*\cos(\theta_{robot} + \theta_{sensor})$$
$$y = y_{robot} + r*\sin(\theta_{robot} + \theta_{sensor})$$

where x_{robot} and y_{robot} are the coordinates of the robot in the global frame,r is the distance the from the center of the robot to the point, θ_{robot} is the angle of the robot's orientation in the global frame and θ_{sonar} is the sonar's orientation with respect to the axis that determines θ_{robot}.

3. To record changes in the environment, first construct a line segment formed by the robot's current position and the point detected by the sensor. Then search the map for the intersection of this line segment and a line segment in the map. On the map line segment, erase all points within one half of the link range to either side of the intersection, eliminating one or both sides if necessary. If the line has only one point, adjust its orientation, which is arbitrary anyway, to be perpendicular to the “robot to sensor point” line segment.

4. To add a point to the map, search all of the lines in the map to see if the point belongs any of them. The criteria for belonging to a line are as follows:

> Let v be the vector from one end of the line segment to the point. Let e1 be the unit vector indicating the direction of the line, and e2 be the unit vector perpendicular to e1.
>
> 1) $-lr < v \cdot e1 < a + lr$, where a is the length of the line segment.
> 2) $-ps < v \cdot e2 < ps$
>
> ps and lr are constants to be assigned by the user. ps should roughly correspond to the expected error in the sensor distances, and lr should roughly correspond to

the diameter of the robot. If the line has only one point, ps is changed to lr in the second condition because the direction is arbitrary.

Update all modified lines using the line fit. Search for links between each of the modified lines and all other lines. Lines may be linked if either end of the line segment meets the first condition and both meet the second. Note that this operation is not commutative, so it must be tested both ways. If it is true in either sequence the lines are linked.

<u>Explanation of Line Fit</u>

The method of the least squares line fit does not work for this cartographer. Consider the equation y = m*x + b. Notice that y is written as a function of x. However, the robot might encounter a line that is vertical, in which case y is independent of x, or a line that is close to vertical, where error makes y seemingly independent of x. In this case, the output is meaningless. Instead the following algorithm is used.

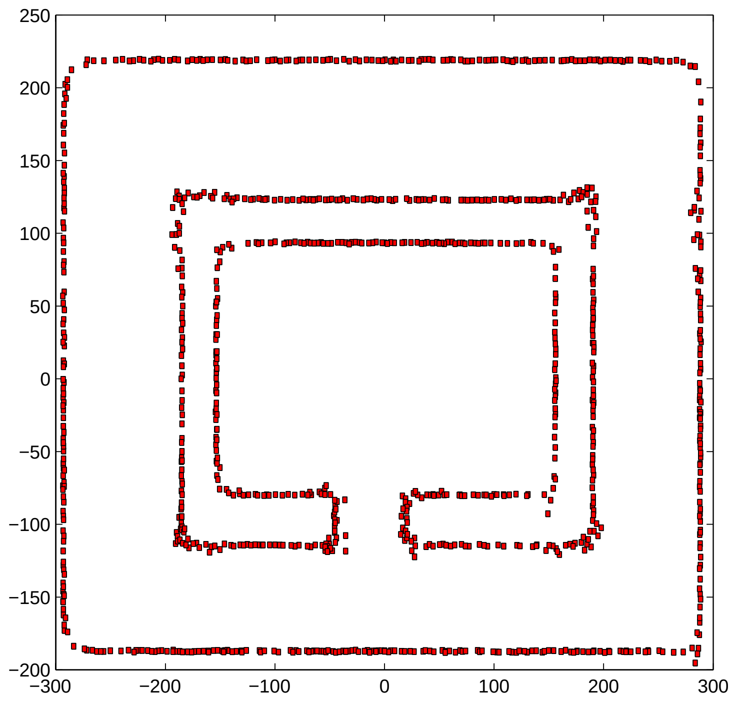
All of the points are assigned a weight of 1, and the center of mass is then calculated. The orientation is then determined by the equation

$$\alpha = \frac{\arctan\left(\dfrac{\sum_{i=1}^{n} 2x_i y_i}{\sum_{i=1}^{n} x_i^2 - y_i^2}\right)}{2}$$

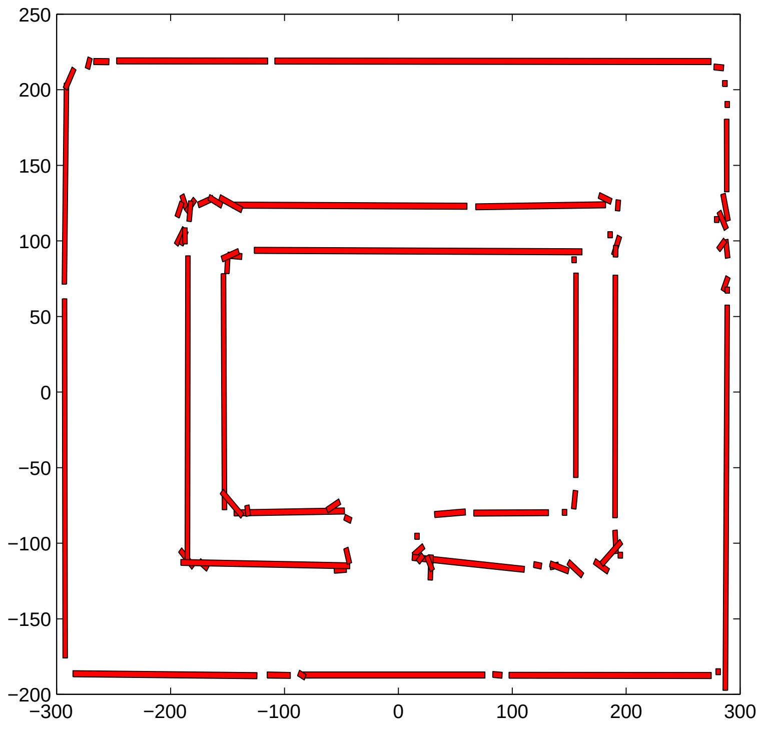
where alpha is the angle and x_i and y_i are the differences between the points and the center of mass. For those not familiar with this formula, it is derived in the appendix. To determine the ends of the line segment, each point (xi,yi) is projected onto the unit vector (cos(alpha),sin(alpha)). The greatest of these vectors in positive direction determines one end of the line segment and the greatest in the negative direction determines the other.

Data and Analysis

Unfortunately, the hardware necessary for testing the LineMap on the real robot has not been installed at the time of this report; only data from the simulator is available. The two graphs below show the data from a journey through a simulated environment intended to mimic a room with a hallway around it. Figure 1 shows the individual points detected. Figure 2 shows the LineMap.



250
200
150
100
50
0
−50
−100
−150
−200
−300
−200
−100
0
100
200
300



250
200
150
100
50
0
−50
−100
−150
−200
−300
−200
−100
0
100
200
300

The pictures show that the Linemap accurately summarizes the points where there are long, straight lines. Less clear are the regions such as the lower right-hand corner of the inner wall, where the data is more complicated. As noted above, grouping the points into lines decreases the number of elements in the map. In the case of the system above, there are 1010 points and 80 lines.

Conclusion

This cartographer would work well with good sensors in environments with straight boundaries, such as buildings. However, poor sensor data and quickly changing environments can create problems. The cartographer has not yet been used on a real robot, so no data from actual demonstrations is available.

Acknowledgements

Thanks to Dr. P.S. Krishnaprasad for allowing me to become involved with the project and to Sean Anderson and Fumin Zhang,his graduate students, for their assistance.

Appendix I: Derivation of Equation Used in Line Fit

Let S = {p_1,p_2,...$p_{j...}$,p_n} be the set of points to be fitted to a line. We assert that the line will pass will pass through the center of mass, whose x and y coordinates are given by the equations

$$x_{cm} = \frac{\Sigma_{j=1}^{n} x_j}{n} \qquad y_{cm} = \frac{\Sigma_{j=1}^{n} y_j}{n}$$

Then let us define x_i and y_i such that (x_i,y_i) = p_j - p_{cm}. If the orientation of the line is described by the vector (cos(θ),sin(θ)), then the goal is to minimize the projection of (x_i,y_i) onto the vector perpendicular to the direction of the line. That is, we want to minimize (x_i,y_i)*(-sin(θ),cosθ)). This is equivalent to minimizing the function

$$f(\theta) = \Sigma_{i=1}^{n} (-x*\sin(\theta)+y*\cos(\theta))^2$$

Setting the derivative equal to zero, we obtain

$$0 = \Sigma_{i=1}^{n} 2*(-x_i*\sin(\theta)+y_i*\cos(\theta))*(-x_i*\cos(\theta)-y_i*\sin(\theta))$$
$$0 = \Sigma_{i=1}^{n} 2*\sin(\theta)*\cos(\theta)(x_i^2-y_i^2)-(\cos^2(\theta)-\sin^2(\theta))*(2x_iy_i)$$

By the double angle formulas

$$0 = \Sigma_{i=1}^{n} \sin(2\theta)*(x_i^2-y_i^2)-\cos(2\theta)*(2x_iy_i)$$
$$\cos(2\theta)\Sigma_{i=1}^{n}(2x_iy_i) = \sin(2\theta)\Sigma_{i=1}^{n} (x_i^2-y_i^2)$$
$$\theta = \frac{\arctan\left(\frac{\Sigma_{i=1}^{n} (2x_iy_i)}{\Sigma_{i=1}^{n} (x_i^2-y_i^2)}\right)}{2}$$

A simple test of the set of points (0,0) and (1,1) shows that this result is in fact a minimum.

Appendix II: LineMap Manual

Introduction

The LineMap uses the position of the robot and the distances returned by the sensors to record the environment as series of points, which are then grouped and fitted to lines. The code is written in C++ and was designed for use with Nomadic Technologies' SuperScout,though it is easily adapted to other robots. This model is best suited for situations where the boundaries are straight, as in many office buildings.

How to Use the LineMap

1) Declare a map by: Map mapname;

2) To take a snapshot of the surroundings: mapname.UpdateMap();

3) To write the data to be plotted in matlab by the m-file pmap.m:mapname.Write_mfile();

4) To update the surroundings array, which contains pointers to lines which the sensors would detect at a given Point p with an orientation robotangle(in degrees): Updatesurround(p,robotangle);

5) To return the point where a change has occurred in the environment(the current position is returned if no changes have taken place):Chasecheck();

The Data Structure

Explanation of Point Class

The Point class stores x and y coordinates. There are two constructors: a simple memory allocation and one that takes all of the relevant information from the robot and calculates x and y. Get and set functions for x and y, and operator definitions for =, != and == are also provided.

Explanation of the Line Class

The Line class stores a singly linked list of points, a point p0, and real numbers dx,dy, and a. The Point p0 is one end of the line segment, (dx,dy)gives the direction, and a is the length of the line segment. Although this is not the most efficient was to store the necessary information, it is the most convenient for the task. A singly linked list was chosen as the data structure for the points, because points need to be removed and inserted from the middle. Public functions are provided for getting p0,a,dx, and dy. The Map class is declared as a friend because of the awkwardness of returning a singly linked list.

Explanation of the Map Class

The Map class stores a singly linked list of lines. As in the Line class, this seems to be the appropriate data structure because of the need to insert and remove elements from the middle. Most of the functions for organizing the data are contained in this class, along with all of the public functions to be called by the user.

Appendix III: The C++ Code

```
/*CMap.h **************************************************************/

#ifndef CMap
#define CMap
#include <math.h>
#include "stlinclude.h"
#include "Cpoint.h"
#include "CLine.h"
#include "GRobot.h"
#include <stdio.h>


class Map
{
 private:
  slist<Line> v; /* vector of lines of points */
  float ps, lr;
  float Getroboty (void);  /* will be replaced by better function from
GRobot*/
  float Getrobotx (void);
  float Getrobotangle (void);
  float Getsonardistance (int i);
  void Crosscheck(Point probot,Point p);
  int Linktwo(slist<Line>::iterator vit1, slist<Line>::iterator vit2);
  void LinkLook2(slist<Line>::iterator);
  void Breakline(slist<Line>::iterator vit1, float aprime);
  Point Singlecheck(Point probot,Point p,slist<Line>::iterator it);
 public:
  Map(void); /* constructor */
  Map(float sps, float slr);
  Line Whatline(Point p);
  void UpdateMap(void); /*puts a point into the map */
  void Write_mfile (void);
  void Write_mfile2 (void);
  void LinkLook (void);
  Point Chasecheck(void);
  slist<Line>::iterator surround[16];
  void UpdateSurround(Point probot,float robotangle);
  Line GetLine(slist<Line>::iterator it);
};

#endif


/*CMap.cpp***********************************************************/
#include "CMap.h"

#define LR 10.0
#define PS 2.0

Map::Map (void)
{
  ps = PS;
  lr = LR;
}

Map::Map (float sps, float slr)
{
  ps = sps;
  lr = slr;
}
```

```
void Map::UpdateMap(void)
{
  int i,flag,indic;
  float robotangle;
  Point probot;
  get_sn();
  get_rc();
  probot.changexy(Getrobotx(),Getroboty());
  robotangle = Getrobotangle();
  slist<Line>::iterator viti = v.begin();
  slist<Line>::iterator vits = v.end();
  float sn[16];
  for (i=0; i<16; i++){
    sn[i] = Getsonardistance(i);
  }
  for (i=0; i<16; i++){
    if (i != 9) {
      Point p(i,sn[i],probot.getx(),probot.gety(),robotangle);
      Crosscheck(probot,p);
      if (sn[i] < 100){
      flag = 0;
      viti = v.begin();
      vits = v.end();
      while (viti != v.end()){
        indic = (*viti).ExamineLine(p);
        if (indic==1){
            flag=1;
            LinkLook2(viti);
          }//if
        if (indic==2) flag = 1;
        advance(viti,1);
      }//while
      if (flag==0) v.insert(v.end(),Line(p,ps,lr));
      }//if
    }//if
  }//for
}



float Map::Getroboty (void){
  return(State[STATE_CONF_Y]*1.0/10);
}

float Map::Getrobotx (void){
  return (State[STATE_CONF_X]*1.0/10);
}

float Map::Getrobotangle (void){
  return (State[STATE_CONF_STEER]*1.0/10);
}

float  Map::Getsonardistance (int i){
  return(State[STATE_SONAR_0+i]);
}



void Map::Crosscheck (Point probot,Point p)
{
  if (v.empty()) return;
  float deltax,deltay,s,aprime,chx,chy,det,dot;
  deltax = p.getx() - probot.getx();
  deltay = p.gety() - probot.gety();
  slist<Line>::iterator vit1 = v.begin();
  while (vit1 != v.end())
    {
      if ((*vit1).L.size()==1){
      if (Singlecheck(probot,p,vit1) != probot) v.erase(vit1);
      }
      else{
      chx = probot.getx() - (*vit1).getp0().getx();
      chy = probot.gety() - (*vit1).getp0().gety();
      det = ((*vit1).getdy()*deltax - (*vit1).getdx()*deltay);
      s = ((*vit1).getdx()*chy - (*vit1).getdy()*chx)/det;
      aprime = (-deltay*(chx) + deltax*(chy))/det;
      dot = ((*vit1).getdx()*deltax
+(*vit1).getdy()*deltay)/sqrt(deltax*deltax+deltay*deltay);
      if (0 < s && s < 0.8 && 0 < aprime && aprime < (*vit1).geta() &&
(*vit1).abso(dot) < cos(PI/4))
        {
          Breakline(vit1,aprime);
      }//if
      }//else
      advance(vit1,1);
    }//while
}

Point Map::Singlecheck (Point probot,Point p,slist<Line>::iterator it)
{
  int flag;
  float deltax,deltay,step,chx,chy,s,det,aprime,ax,ay;
  deltax = p.getx() - probot.getx();
  deltay = p.gety() - probot.gety();
  s = sqrt(deltax*deltax+deltay*deltay);
  ax = -deltay/s;
  ay = deltax/s;
  chx = probot.getx() - (*it).getp0().getx();
  chy = probot.gety() - (*it).getp0().gety();
  det = ay*deltax - ax*deltay;
  s = (ax*chy - ay*chx)/det;
  aprime = (-deltay*chx + deltax*chy)/det;
  if (0 < s && s < 0.8 && -ps < aprime && aprime < ps)
    {
      p.changexy(s*deltax,s*deltay);
      return(p);
    }
  return(probot);
}


Point Map::Chasecheck(void)
{
  int i,flag;
  float robotangle;
  float deltax,deltay,s,aprime,chx,chy,det,dot;
  Point probot,dummy;
  get_sn();
  get_rc();
  probot.changexy(Getrobotx(),Getroboty());
  robotangle = Getrobotangle();
  slist<Line>::iterator viti = v.begin();
  float sn[16];
  for (i=0; i<16; i++){
    sn[i] = Getsonardistance(i);
  }
  for (i=0; i<16; i++){
    if (i != 9) {
      Point p(i,sn[i],probot.getx(),probot.gety(),robotangle);
      deltax = p.getx() - probot.getx();
      deltay = p.gety() - probot.gety();
      slist<Line>::iterator vit1 = v.begin();
      while (vit1 != v.end())
      {
        if ((*vit1).L.size()==1){
          dummy = Singlecheck(probot,p,vit1);
          if (dummy != probot) return(dummy);
        }
        else{
          chx = probot.getx() - (*vit1).getp0().getx();
          chy = probot.gety() - (*vit1).getp0().gety();
          det = ((*vit1).getdy()*deltax - (*vit1).getdx()*deltay);
          s = ((*vit1).getdx()*chy - (*vit1).getdy()*chx)/det;
          aprime = (-deltay*(chx) + deltax*(chy))/det;
          if (0 < s && s < 0.8 && 0 < aprime && aprime < (*vit1).geta())
            {
            dummy.changexy(s*deltax,s*deltay);
            return(dummy);
            }//if
        }//else
        advance(vit1,1);
      }//while
    }//if
  }//for
  return(probot);
}




int Map::Linktwo(slist<Line>::iterator vit1, slist<Line>::iterator vit2)
{
  printf("linktwo\n");
  float dot;
  dot = (*vit1).getdx()*(*vit2).getdx()+(*vit1).getdy()*(*vit2).getdy();
  if ((*vit1).abso(dot) < cos(PI/9) && (*vit1).L.size() > 1 &&
(*vit2).L.size() > 1) return(0);
  slist<Point>::iterator slit = (*vit2).L.begin();
  //slist<Point>::iterator slit2 = (*vit1).L.begin();
  while(slit != (*vit2).L.end()){
    (*vit1).Enterpoint(*slit);
    advance(slit,1);
  }
// while
  //slit2 = (*vit1).L.begin();
  //while (slit2 != (*vit1).L.end())
  // {
  //   printf(" %f \n",(*vit1).getastar(*slit2));
  //   advance(slit2,1);
  // }
  (*vit1).MyUpdateline();
  v.erase(vit2);
  return(1);
}


void Map::Breakline(slist<Line>::iterator vit1, float aprime){
  slist<Point>::iterator slit = (*vit1).L.begin();
  if ((*vit1).geta() < aprime + lr/2  && aprime -lr/2 < 0){
     v.erase(vit1);}
   if ((*vit1).geta() < aprime + lr/2  && aprime -lr/2 > 0){
     slit = (*vit1).L.begin();
     while (slit != (*vit1).L.end())
       {
       if ((*vit1).getastar(*slit) > aprime -lr/2){
         (*vit1).L.erase(slit);
       }
       advance(slit,1);
       }
     (*vit1).MyUpdateline();
   }//if
   if ((*vit1).geta() > aprime + lr/2  && aprime -lr/2 < 0){
     slit = (*vit1).L.begin();
     while (slit != (*vit1).L.end())
       {
       if ((*vit1).getastar(*slit) < aprime + lr/2){
         (*vit1).L.erase(slit);
       }
       advance(slit,1);
       }
     (*vit1).MyUpdateline();
   }//if
   if ((*vit1).geta() > aprime + lr/2 && aprime -lr/2 > 0){
     slit = (*vit1).L.begin();
     advance(slit,(*vit1).L.size() -1);
     v.insert(v.end(),1,Line(*slit,ps,lr/2));
     slist<Line>::iterator vit2 = v.begin();
     advance(vit2,v.size() -1);
     slit = (*vit1).L.begin();
     while (slit != (*vit1).L.end())
       {
       if ((*vit1).getastar(*slit) > aprime - lr/2){
         if ((*vit1).getastar(*slit) > aprime + lr/2)
(*vit2).Enterpoint(*slit);
         (*vit1).L.erase(slit);
       }
       advance(slit,1);
       }//while
     (*vit1).MyUpdateline();
     (*vit2).MyUpdateline();
   }//if
}

void Map::UpdateSurround(Point probot, float robotangle){
  int i;
  float chx,chy,det,aprime,s,tempd,deltax,deltay;
  tempd = 1;
  slist<Line>::iterator viti = v.begin();
  for (i=0; i<16; i++){
    Point p(i,255,probot.getx(),probot.gety(),robotangle);
    deltax = p.getx() - probot.getx();
    deltay = p.gety() - probot.gety();
    surround[i] = v.end();
    viti = v.begin();
    while (viti !=v.end()){
      chx = probot.getx() - (*viti).getp0().getx();
      chy = probot.gety() - (*viti).getp0().gety();
      det = ((*viti).getdy()*deltax - (*viti).getdx()*deltay);
      s = ((*viti).getdx()*chy - (*viti).getdy()*chx)/det;
      aprime = (-deltay*chx + deltax*chy)/det;
      if (0 < s && s < tempd && 0 < aprime && aprime < (*viti).geta()){
      surround[i] = viti;
      tempd = s;
      }
      advance(viti,1);
    }
  }
}
```

```
Line Map::GetLine(slist<Line>::iterator it){
  if (it == v.end()) printf("*(v.end()) is not a line");
  return (*it);
}


void Map::Write_mfile(void){ // will want user to be able to input filename
eventually
  int i;
  FILE *pcool;
  pcool = fopen("cool.m","w");
  //dx
  slist<Line>::iterator vit = v.begin();
  fprintf(pcool,"dx = [");
  while (vit != v.end()){
    fprintf (pcool," %f",(*vit).getdx());
    advance(vit,1);
  }
  fprintf (pcool,"];\n\n");
  //dy
  vit = v.begin();
  fprintf(pcool,"dy = [");
  while (vit != v.end()){
    fprintf (pcool," %f",(*vit).getdy());
    advance(vit,1);
  }
  fprintf (pcool,"];\n\n");
  //a
  vit = v.begin();
  fprintf(pcool,"a = [");
  while (vit != v.end()){
   fprintf (pcool," %f",(*vit).geta());
   advance(vit,1);
  }
  fprintf (pcool,"];\n\n");
  //x
  vit = v.begin();
  fprintf(pcool,"x = [");
  while (vit != v.end()){
    fprintf (pcool," %f",(*vit).getp0().getx());
    advance(vit,1);
  }
  fprintf (pcool,"];\n\n");
  //y
   vit = v.begin();
  fprintf(pcool,"y = [");
  while (vit != v.end()){
    fprintf (pcool," %f",(*vit).getp0().gety());
     advance(vit,1);
  }
  fprintf (pcool,"];\n\n");
  fclose(pcool);
}

void Map::LinkLook2(slist<Line>::iterator vit1){
  int restart;
  float X,Y,dot;
  Point dummy;
  slist<Line>::iterator vit2 = v.begin();
  slist<Point>::iterator slit = (*vit1).L.begin();
  restart = 0;
  while (vit2 != v.end())
    {
      if (vit1!=vit2)
      {
        X = (*vit2).getp0().getx()+(*vit2).getdx()*(*vit2).geta();
        Y = (*vit2).getp0().gety()+(*vit2).getdy()*(*vit2).geta();
        dummy.changexy(X,Y);
        if (((*vit1).WhereCheck((*vit2).getp0()) !=0 ||
(*vit1).WhereCheck(dummy)!=0))   restart = Linktwo(vit1,vit2);
      }//if
      advance(vit2,1);
      if (restart ==1){
      vit2 = v.begin();
      restart =0;
      }
    }//while vit2
}


void Map::LinkLook(){
  int restart;
  float X,Y,dot;
  Point dummy;
  slist<Line>::iterator vit1 = v.begin();
  slist<Line>::iterator vit2 = v.begin();
  slist<Point>::iterator slit = (*vit1).L.begin();
  restart = 0;
  while (vit1 !=v.end())
    {
    while (vit2 != v.end())
      {
      if (vit1!=vit2)
        {
          X = (*vit2).getp0().getx()+(*vit2).getdx()*(*vit2).geta();
          Y = (*vit2).getp0().gety()+(*vit2).getdy()*(*vit2).geta();
          dummy.changexy(X,Y);
          if (((*vit1).WhereCheck((*vit2).getp0()) !=0 ||
(*vit1).WhereCheck(dummy)!=0))   restart = Linktwo(vit1,vit2);
        }//if
      advance(vit2,1);
      if (restart ==1){
        vit2 = v.begin();
        restart =0;
      }
      }//while vit2
    advance(vit1,1);
    vit2 = v.begin();
    } //while vit1
}



void Map::Write_mfile2(void){ // will want user to be able to input filename
eventually
  int i;
  FILE *qpcool;
  qpcool = fopen("qcool.m","w");
  //dx
  slist<Line>::iterator vit = v.begin();
  fprintf(qpcool,"dx = [");
  while (vit != v.end()){
    fprintf (qpcool," %f",(*vit).getdx());
    advance(vit,1);
  }
  fprintf (qpcool,"];\n\n");
  //dy
  vit = v.begin();
  fprintf(qpcool,"dy = [");
  while (vit != v.end()){
    fprintf (qpcool," %f",(*vit).getdy());
    advance(vit,1);
  }
  fprintf (qpcool,"];\n\n");
  //a
  vit = v.begin();
  fprintf(qpcool,"a = [");
  while (vit != v.end()){
   fprintf (qpcool," %f",(*vit).geta());
   advance(vit,1);
  }
  fprintf (qpcool,"];\n\n");
  //x
  vit = v.begin();
  fprintf(qpcool,"x = [");
  while (vit != v.end()){
    fprintf (qpcool," %f",(*vit).getp0().getx());
    advance(vit,1);
  }
  fprintf (qpcool,"];\n\n");
  //y
   vit = v.begin();
  fprintf(qpcool,"y = [");
  while (vit != v.end()){
    fprintf (qpcool," %f",(*vit).getp0().gety());
     advance(vit,1);
  }
  fprintf (qpcool,"];\n\n");
  fclose(qpcool);
}


/* CLine.h **************************************************************/
#ifndef CLine
#define CLine
#include "stlinclude.h"
#include "CPoint.h"
#include <math.h>

class Line
{
  friend class Map;
 private:
  slist<Point> L;
  Point p0;
  float dx, dy,a,lr,ps;
  void MyUpdateline(void);
  int sign(float q);
  float abso(float q);
  float getastar (Point p);
  float getbstar (Point p);
  void SemiSort(void);
 public:
  Line (Point p,float sps,float slr);/* construction : makes line*/
  int TooCloseCheck(Point p);/* check for duplication*/
  int ExamineLine (Point P);  /* tests if the point belongs to the line*/
  float getdx(void);
  float getdy(void);
  float geta (void);
  Point getp0 (void);
  int WhereCheck(Point p); /* returns 1 if point should be linked */
  void Enterpoint(Point p);
};
```

```
#endif


/*CLine.cpp ***********************************************************/
#include "CLine.h"


Line::Line(Point p,float sps, float slr)
{
  L.insert(L.begin(),p);
  p0 = p;
  a = 0;
  dx = 1;
  dy = 0;
  lr = slr;
  ps = sps;
}
int Line::TooCloseCheck(Point p)
{
 float xwrtp0,ywrtp0,astar,bstar;
 slist<Point>::iterator it=L.begin();
 while(it != L.end())
   {
     xwrtp0 = p.getx() - (*it).getx();
     ywrtp0 = p.gety() - (*it).gety();
     if (xwrtp0*xwrtp0+ywrtp0*ywrtp0 < ps*ps) return(1);
     advance(it,1);
   }
 return(0);
}

int Line::ExamineLine (Point p)
{

  if (WhereCheck(p)==0)  return(0);
  if (TooCloseCheck(p) == 1) return(2);
  Enterpoint(p);
  MyUpdateline();
  return(1);
}

int Line::WhereCheck(Point p)
{
  float xwrtp0, ywrtp0, bstar,astar;
  xwrtp0 = p.getx() - p0.getx();// see TooCloseCheck
  ywrtp0 = p.gety() - p0.gety();
  bstar =(dx*ywrtp0 - dy*xwrtp0);
  astar = (dx*xwrtp0 + dy*ywrtp0);
  if (L.size() == 1){
    if ((-lr < bstar && bstar < lr) && (-lr < astar && astar < lr)) return
(2);
    else return (0);
  }//if
  else{
    if (-ps < bstar && bstar < ps){ // has to be good in both b and a
      if (-lr < astar && astar < -ps) return(1);
      if (a + ps <astar && astar < a + lr) return (2);
      if (- ps <astar && astar < a + ps) return (3);
      if (-lr > astar || astar > a + lr) return(0);
    }//if
    else  return (0);
  }//else
  printf ("you messed up in where check\n");
}
```

```
int Line::sign(float q)
{
  if (q < 0) return (-1);
  else return (1);
}
float Line::getdx(void)
{
  return(dx);
}
float Line::getdy(void)
{
  return (dy);
}
float Line::geta (void)
{
  return (a);
}

Point Line::getp0 (void)
{
  return(p0);
}

float Line::abso(float q)
{
  if (q < 0) return (-1*q);
  else return (q);
}

float Line::getastar(Point p)
{
  float xwrtp0,ywrtp0,astar;
  xwrtp0 = p.getx() - p0.getx();// calculates x w/respect to p0
  ywrtp0 = p.gety() - p0.gety();// calculates y w/respect to P0
  astar = (dx*xwrtp0 + dy*ywrtp0);
  return(astar);
}

float Line::getbstar(Point p)
{
  float xwrtp0,ywrtp0,bstar;
  xwrtp0 = p.getx() - p0.getx();// calculates x w/respect to p0
  ywrtp0 = p.gety() - p0.gety();// calculates y w/respect to P0
  bstar = (dx*ywrtp0 - dy*xwrtp0);
  return(bstar);
}
void Line::Enterpoint(Point p){
  float astar;
  astar = getastar(p);
  slist<Point>::iterator it=L.begin();
  advance(it,L.size()-1);
  if (astar < getastar(*(L.begin()))) L.insert(L.begin(),p);
  if (astar > getastar(*it)) L.insert(L.end(),p);
  if (getastar(*(L.begin())) < astar && astar < getastar(*it))
L.insert_after(L.begin(),p);
}

void Line::SemiSort(void)
{
  slist<Point>::iterator iti = L.begin();
  slist<Point>::iterator itl = L.begin();
  while (iti != L.end())
    {
      //if (getbstar(*iti) > ps){
      //L.erase(iti);
      // }
      //else{
      itl = L.begin();
      advance(itl,L.size()-1);
      if (getastar(*iti) < getastar(*(L.begin()))){
      L.insert(L.begin(),*iti);
      L.erase(iti);
      }
      if (getastar(*iti) > getastar(*itl)) {
      L.insert(L.end(),*iti);
      L.erase(iti);
      }
      //}
      advance(iti,1);
    }
}


void Line::MyUpdateline(void){
  float s,X,Y;
  slist<Point>::iterator it1 = L.begin();
  if (L.size()==1) return;
  if (L.size()==2)
    {
    float mag;
    p0 = *it1;
    advance(it1,1);
    dx = (*it1).getx() - p0.getx();
    dy = (*it1).gety() - p0.gety();
    mag = sqrt(dx*dx+dy*dy);
    dy = dy/mag;
    dx = dx/mag;
    }
  else{
    double sx,sy,mag,xwrtp0,ywrtp0,difsq,prod,theta,arg;
    int ndata = L.size();
    sx = 0; // summing x and y
    sy = 0;
    it1 = L.begin();
    while (it1 != L.end())
      {
      sx = sx + (*it1).getx();
      sy = sy + (*it1).gety();
      advance(it1,1);
      }
    p0.changexy(sx/ndata,sy/ndata);
    prod = 0;
    difsq = 0;
    it1 = L.begin();
    while (it1 != L.end())
      {
      xwrtp0 = (*it1).getx() - p0.getx();
      ywrtp0 = (*it1).gety() - p0.gety();
      difsq = difsq + xwrtp0*xwrtp0 - ywrtp0*ywrtp0;
      prod = prod + xwrtp0*ywrtp0;
      advance(it1,1);
      }
    arg = 2*prod/difsq;
    printf("arg = %f\n",arg);
    theta = atan2(2*prod,difsq)/2;
    printf("theta = %f\n",theta);
    dx = cos(theta);
    dy = sin(theta);
    SemiSort();
    it1 = L.begin();
    s = dy*((*it1).getx() - p0.getx()) - dx*((*it1).gety()-p0.gety());
    printf("s = %f\n",s);
    X = (*it1).getx() - dy*s;
    Y = (*it1).gety() + dx*s;
    p0.changexy(X,Y);
  }//else
  it1 = L.begin();
  advance(it1,L.size()-1);
  a = dx*((*it1).getx() - p0.getx()) + dy*((*it1).gety() - p0.gety());
}



/*CPoint.h**********************************************************/
#ifndef CPoint
#define CPoint
#include <math.h>
class Point
{
 private:
  float x,y;
  float  GetSonarAngle(int sn_num);
 public:
  /* construction: translates input to x and y coordinate */
  Point(int sn_num,float sn_dist,float Robotx,float Roboty, float Robotangle);
  Point(void);
  Point(Point const &other);
  Point Add(Point p);
  Point scalarmult(float a);
  void operator=(Point const &other);
  int operator==(Point const &other);
  int operator!=(Point const &other);
  float getx();
  float gety();
  void changexy(float X, float Y);
};

#endif



/*CPoint.cpp *******************************************/
#include "CPoint.h"


#define PI 3.14159

Point::Point(int sn_num,float sn_dist,float Robotx,float Roboty, float
Robotangle)
{
  float angle;
  angle = (Robotangle + GetSonarAngle(sn_num));
  if (angle > 360.0) angle = angle - 360.0; // angles >360 don’t make sense
  x = Robotx + (sn_dist+7.5)*cos(PI*angle/180.0);
  y = Roboty + (sn_dist+7.5)*sin(PI*angle/180.0);
}
Point::Point(void){}

Point::Point(Point const &other)
{
 x =other.x;
 y =other.y;
}

void Point::operator=(Point const &other)
```

```
{
  x = other.x;
  y = other.y;
}

int  Point::operator==(Point const &other)
{
  return (x==other.x && y ==other.y);
}


int  Point::operator!=(Point const &other)
{
  return (x!=other.x || y !=other.y);
}

float Point::GetSonarAngle (int sn_num)
{
  float sonarangle;
  sonarangle = 360.0/16 * sn_num;// assumes that 0 is in front and that they
proceed counter clockwise
  return(sonarangle);
}

float Point::getx(void)
{
  return (x);
}

float Point::gety (void)
{
  return (y);
}

void Point::changexy(float X,float Y)
{
  x = X;
  y = Y;
}

Point Point::Add(Point p)
{
  Point out;
  float X,Y;
  X = x + p.getx();
  Y = y + p.gety();
  out.changexy(X,Y);
  return (out);
}

Point Point::scalarmult(float a)
{
  Point out;
  float X,Y;
  X = a*x;
  Y = a*y;
  out.changexy(X,Y);
  return(out);
}
```