University of Florida

Department of Electrical Engineering

EEL 5666

Intelligent Machines Design Laboratory

Written Report 3

Magellan

by

Scott Spurlock

Instructor: Dr. Keith L. Doty

Table of Contents

Abstract

Magellan is an autonomous, mobile robot designed around a Motorola 68HC11EVBU board. Magellan's primary goal is mapping. To this end, the robot is equipped with infrared emitter-detectors for object detection, shaft encoders for determining the robot's relative position, and an electromagnetic compass as a redundant back-up to the shaft encoders. Magellan maps an area by moving randomly through it, avoiding obstacles, constructing a map in its memory. Each position of the map array contains an integer indicating the robot's relative confidence that an object exists in a given place. Further goals for Magellan include the ability to determine, once it has mapped an area, the shortest distance from its current position to some other given point in the area; and the ability to determine its position on the map when set down randomly in a previously mapped area.

Executive Summary

Magellan consists of a Motorola 68HC11 board, augmented by two expansion boards, mounted on an aluminum platform along with wheels, sensors, and motors. Magellan's purpose is mapping. The robot maps the presence or absence of obstacles in the area that it explores by incrementing or decrementing a confidence value reflecting its assurance that an object is or is not present in a particular map square. Two sensors allow the robot to detect obstacles: infrared and bump. The IR sensors serve to detect objects at a distance and allow the robot to avoid actually running into obstacles. The bump sensors serve as a back-up to the IR sensors, indicating physical contact with an obstacle. Map confidence values are incremented by a greater amount when a collision occurs than when the IR sensors indicate a proximate collision.

Magellan's strategy of mapping consists of random exploration. Its current heading and displacement relative to its former heading is calculated based on information from its shaft encoders, which provide thirty pulses per wheel revolution. This information allows the robot to continually update its current position on its map. An electromagnetic compass, realized with rare earth magnets and Hall Effect transistors, rounds out Magellan's sensors. The compass provides a back- up for the robot, so that, if the compass heading and the software-accumulated heading differ by too great a margin, the robot will stop mapping and embark upon an as-yet-unrealized "lost" behavior.

Introduction

Magellan, the robot-cartographer, represents an exploration of the feasibility of automated mapping in an indoor environment. Automated mapping has many potential uses. For example, in an industrial setting, where autonomous, mobile robots are used to load and unload machines, the robots could optimize their performance by mapping the locations of obstacles and machines. Although such information could be hardwired into the robots or environmentally indicated (by bar-codes on the factory floor, for example), an intelligent mapping routine would be a much less brittle solution, supporting changes more readily and cutting down on installation cost. Indeed, almost any application in which a mobile robot would operate for an extended time in a particular environment could benefit from the higher efficiency afforded by a robot with a working knowledge of its environment.

Magellan will emphasize robustness in mapping by following a "biological" approach. It will move through its environment randomly, avoiding collisions, rather than attempting to follow a particular strategy of exploration. It will map its environment by incrementing or decrementing “confidence values” corresponding to squares on the map, according to whether or not it senses an object in that square. This method of mapping produces a very fluid map, which changes according to new information, and should allow for a sizable margin for error. For the future, I would like to implement an intelligent algorithm to determine whether to update the map or the robot’s perceived position on it.

Integrated system

Magellan consists of an aluminum platform, which supports the 68HC11 board and the two expansion boards, the ME11 and the MTSX01. Eight AA Nicads power the system, and two dc motors driving model airplane wheels provide actuation (with an unpowered caster wheel for balance). A wide sensory array allows the robot to perceive its environment. Five infrared detectors alert Magellan to the presence of objects (for both mapping and obstacle avoidance), and nine bump switches around the periphery of the platform sense contact with an object. For purposes of mapping, two shaft encoders sense how far each wheel has turned, allowing the robot to calculate positional data, and four Hall Effect sensors, combined with five rare earth magnets, implement a compass, which serves as an absolute directional reference.

Mobile Platform

Magellan's platform is eighth inch aluminum cut according to the TALRIK pattern. The eighth inch aluminum should provide more than enough structural stability, while remaining light enough not to tax the motors unnecessarily. The simple circular shape enhances the robot's aesthetics, as well as its agility in tight spaces, and reduced the overall design time, because mechanical drawings were readily available.

Actuation

The two dc motors that drive Magellan constitute its only actuation mechanisms. They derive from model airplane servos. Once the stop on the servo and the circuit board have been removed to allow continuous rotation, the servo becomes a dc motor (with the gears already provided). The motors mount to the underside of the body with Novasoft servo mounts. Each motor screws into the plastic hub of a two and three-fourths inch model airplane wheel. A non-powered caster wheel provides balance at the back of the robot. The differential drive decreases mechanical complexity, allows the robot more mobility than other drives (e.g. car drive) and is also relatively inexpensive, as only two motors are necessary. The ME11 expansion board handles motor control through an H-bridge motor driver.

Sensors

Magellan's primary sensors are five Sharp IR sensors from Novasoft, which were easily modified to provide an analog signal instead of the original digital signal. Also, grounding the metal case of the sensor makes it effectively a Faraday cage, which cuts down on noise interference. These sensors rest around the front of the robot, pointed radially outward (see figure 1).

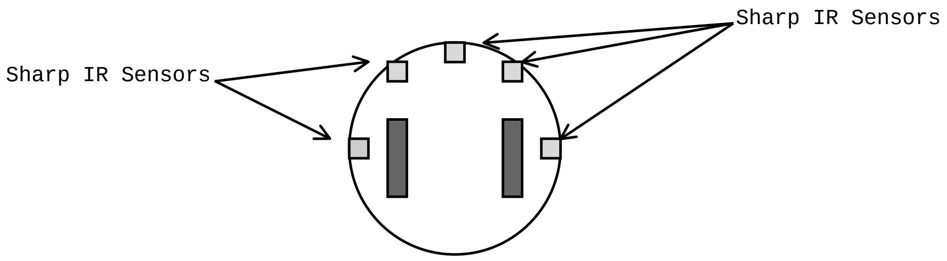

Fig 1: Placement of IR Sensors

Each IR sensor’s signal pin connects to an analog port, mapped onto the MTSX01 expansion board, so that its value can be easily read in software. These IR sensors play the primary role in object detection, for both avoidance and mapping routines (see the Behaviors section).

Magellan is also equipped with bump switches around the outside edge of its platform. These bump detectors are actually small microswitches of unknown manufacture, scavenged from discarded circuit boards. The five front bump switches are hot-glued to the aluminum body so that, with one in the front center, each of the others is between two IR detectors (see fig 1). The remaining four switches are spaced evenly around the back of the robot. The front switches share a common ground, but each is pulled up to power through a different sized resistor, so that a read of an analog port will return a different value for each switch or combination of switches pushed. Each pair of the rear switches is wired together so that if either one of a given pair is pushed, the analog pin to which it is wired will return the same reading. Thus, from the rear bump switches, only left or right side can be determined, while, for the front switches, the exact switch or switches closed can be determined. To further aid the robot in detection of bumps, I attached a quarter inch wide strip of acrylic in a circle around the switches so that, if an object impacts the robot between two switches, the acrylic bumper will press them both.

Magellan also has a shaft encoder attached to each wheel to keep track of the distance traveled by each wheel. The shaft encoders are actually part of an old three-button, IBM mouse. I hot-glued the central axis of each encoder into the center of the hub of each wheel. Then, when the wheel turns, so does the shaft encoder central axis. The encoder has two pins and acts like a switch. As the central axis is turned the pins are alternately connected and unconnected. With one pin grounded and the signal pin pulled up to five volts, the shaft encoder gives thirty pulses per revolution, with very little bounce. With each signal pin connected to an analog port, the software can poll to count the number of pulses. Magellan interprets the shaft encoder information by applying the algorithm $d = \frac{w_r + w_l}{2}$, where d is approximately the length of the vector the robot has traveled and w_i is the distance traveled by wheel i in shaft encoder pulses. Magellan can also determine the approximate angle, in radians, that it has turned by applying the algorithm, $\theta = \frac{w_r - w_l}{x}$, where Θ is the angle and x is the distance between the wheels. These equations give approximately polar coordinates, which are then converted to rectangular form for mapping use.

Magellan’s final sensor is a compass. My original design involved an encoder disc, a circular piece of transparency material, which was attached to a magnetic compass and rotated freely with the magnetic field between three emitter-detector pairs (see figure 2). The encoder disc was marked in such a way that the three emitter detector pairs would return a unique reading for each of eight different sectors (see figure 3).

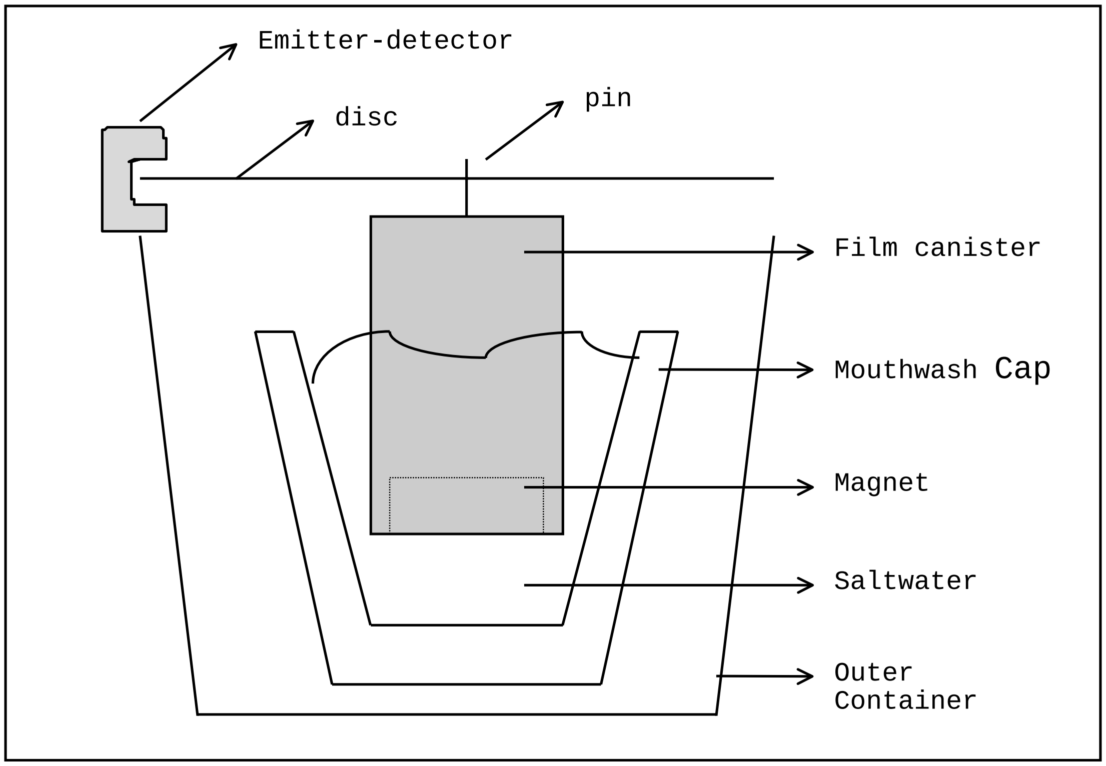

Figure 2: Compass System

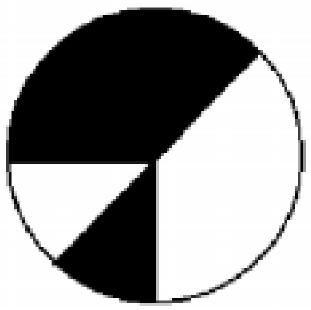

figure 3

Unfortunately, this design proved too brittle, requiring excessive perfection, and I was ultimately unable to make it work reliably. My new compass design centers around Hall Effect switches (HES), which I ordered from All Electronics (cat# HESW-2). Each HES has an input, output, and ground pin (see figure 4). When the front side of an HES is exposed to the magnetic field of a North pole, the output of the circuit of figure 4 is five volts, and it stays high until a South pole's magnetic field turns the HES off. The current implementation of the compass consists of a magnetic compass turning a disc with three magnets mounted on it (see figure 5). Currently I have four HESs around the disc in the cardinal directions (although I plan to increase the compasses accuracy by adding additional sensors). The three magnets on the disc are mounted so that the one in the middle has its North pole facing outwards, while the two on either side of it have their South poles facing out (see figure 6). This configuration ensures that only one HES will be on at a time, so that the software can read the four analog ports to which the HESs are connected and determine which direction the robot is facing.

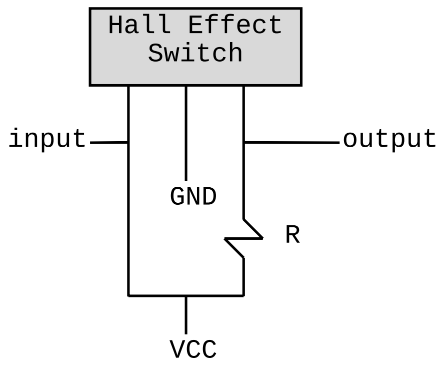

figure 4:
HES circuit

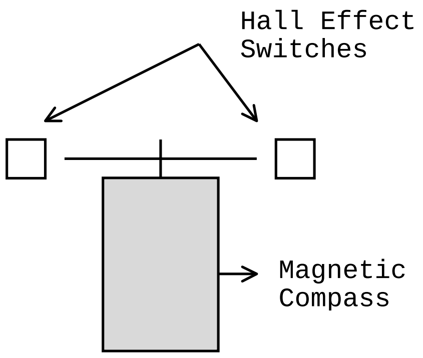

figure 5:
Compass side view

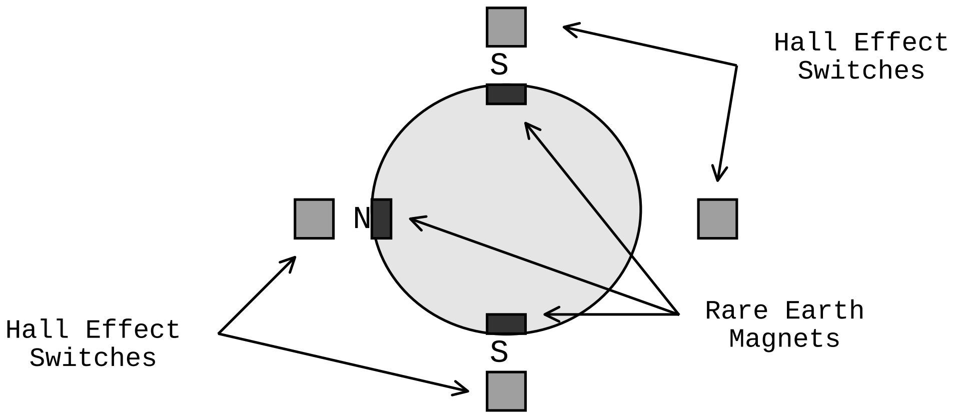

figure 6: Compass top view

Behaviors

Magellan currently has four main behaviors, although a fourth, “lost” behavior is planned. The first and highest priority is object avoidance. The avoidance algorithm is a refined, simplified version of the neural network idea. The arbitration process is winner take all among the five neurons (whose values are determined by the five IR cams). Object avoidance is primarily controlled by the front three sensors. The front left neuron advocates turning strong right, the front right neuron advocates turning strong left, and the front center neuron advocates backing up and turning in the same direction as the last turn. The left and right neurons advocate only a slight turn away from their respective sides. The motor control algorithm smoothly changes speeds from current to desired using the formula $speed_{next} = speed_{curr} + \frac{urgency}{100} \bullet (speed_{desired} - speed_{curr})$, where urgency is simply the value of the highest neuron. Thus, as an object nears the robot, the IR cam nearest it will return an increasingly high voltage, and the neuron associated with it will pass an increasingly large urgency to the motor control algorithm, which will then turn increasingly sharply. This architecture emphasizes biological-seeming turns, with smooth arcs when an obstacle is distant and sharper turns only when necessary.

Magellan’s second behavior is object collision. This behavior serves mainly as a backup to object avoidance and, ideally, never occurs. The bump switches alert the robot to a collision. If the robot has bumped an object in front of it, Magellan backs up slightly and turns sharply away from the side bumped (or in the direction of the last turn if only the front is bumped). If Magellan has bumped an object behind it, then it moves slightly forward and turns sharply away from the side bumped. I am still working on integrating the collision behaviors into the IR neuronal structure of object avoidance and, in particular, making sure that the robot is still mapping during the collision behavior. Magellan’s third behavior is mapping. This behavior is highly internal, and does not involve any external actuation. Thus, it appears extremely similar to object avoidance, because the only addition is that the robot “remembers” the readings from its sensors. At the software level, the robot will occupy the center position in a three-by-three array. Each of the positions in the array will be initialized to zero (which means that the robot has no opinion about whether or not an object occupies that position) except for the center position, which is initialized to -10 (the robot thinks it unlikely that there is anything in that position). It reads the values of the five IR neurons and adds them to the values in the corresponding positions in the three-by-three array. (A global variable normangle, which is the robot’s heading normalized to 45 degree intervals, helps Magellan decide which positions in the array correspond to which IR neurons at any given time.) The shaft encoders help Magellan keep track of its x and y coordinates within the center square of the array. Whenever Magellan moves out of this center square, all the information in the array is passed into the larger “world” map array. In other words, each entry in the three-by-three matrix is added to the corresponding entry in the main map. Also, the x and y coordinates of the larger map are updated. Then the three-by-three array is reinitialized and the process begins again. Confidence values can range from -1000 to 999. I also plan to integrate the bumpers so that a triggered bump would add a large negative to the corresponding position in the array.

Magellan also has three minor behaviors in various stages of planning. First, the “Edge of the World” behavior becomes active when the robot reaches the edge of the allocated map space. The software currently implements this behavior as a 90 degree turn, but I am planning on changing it so that the robot reacts as though it had bumped something (which would afford greater integration of the software). An as-yet-unrealized behavior is the “Lost” behavior which will occur when the compass heading differed too greatly from the software heading and will involve the robot stopping and beeping. Magellan’s final behavior is the “Show Map” behavior, which will go into effect when the rear bumper is pressed just after reset. Then Magellan outputs the map to the serial port at 9600 baud, scaling the confidence values to the range 0 to 9 so that the entire map can be displayed on the screen at once.

Conclusion

Magellan’s purpose is to map its environment in terms of the presence or absence of obstacles. A gray area between obstacle and no obstacle provides a measure of the robot’s confidence in the data. The Motorola 68HC11 forms the basis of the robot, along with two expansion boards, all physically mounted on an aluminum frame, along with two motors and wheels and a castor. IR sensors provide the main interface with the outside world, specifically indicating the presence or absence of obstacles. Bump switches act as a backup system to detect collisions. Shaft encoders provide information concerning the robot's relative motion, enabling it to map, and the compass provides a means of checking mapping validity.

To date, I have done only rudimentary testing of the integrated system. Therefore the feasibility of mapping in this manner requires additional investigation. Also, the rigid system of x and y coordinates seems brittle and strays from my original biological approach. I would like to experiment further with object recognition and categorization.

Appendix: Program Code

```
/* Motor control/object avoidance/mapping
   by Scott Spurlock

   Sensors:              4
                      3     5
                     2       6

   Motors:          0          1
   Shaft Encoders: cds4       cds1

*/

/* constants */

int maxconf = 999;
float pi = 3.141592654;
float width = 24.3;     /* distance between wheels in clicks */
int col = 30;       /* number of squares along a global map column */
int currcol = 3;    /* number of squares in a col of the local map */
int mapsize = 1000; /* total size of map array */

/* Global Variables */

int irsensor[7];
int thresh[7]={105,105,105,105,110,105,105};
int lmotor=0, rmotor=0; /* speed of motors */
int lcount=0, rcount=0; /* current count of shaft encoders */
int x=15, y=15;         /* current position on global map  */
float vector, angle=pi/2.0, normangle=pi/2.0, x1=0.0, y1=0.0;
int currmap[9];
persistent int map[1000];

/* Reads IR sensors and stores values in irsensor array */
void ir() {
  int i;

  for (i=2;i<11;i++) {
      if (i <= 7) irsensor[i] = (analog(i));
        else { poke(0x4000,(i-8)); irsensor[i] = (analog(0)); }
  }
}

/* Returns reading from CDS cell */
int cds(int detector) {
  if (detector == 1) { poke(0x4000,0x07); return(analog(0)); }
  else if (detector == 2) { poke(0x4000,0x00); return(analog(1)); }
  else if (detector == 3) { poke(0x4000,0x08); return(analog(1)); }
  else if (detector == 4) { poke(0x4000,0x10); return(analog(1)); }
  else if (detector == 5) { poke(0x4000,0x18); return(analog(1)); }
  else if (detector == 6) { poke(0x4000,0x20); return(analog(1)); }
  else return(0);
}

/* Returns reading from front bumper */
int f_bump() {
  poke(0x4000,0x30);
  return(analog(1));
}
```

```
/* Returns reading from rear bumper */
int r_bump() {
  poke(0x4000,0x28);
  return(analog(1));
}

/* Reads sensor data, scales it and assigns it to irsensor array.
   Returns high neuron number.                               */
int read() {
int i, high = 0;

 ir();
 for (i=2;i<7;i++) {
   irsensor[i] -= thresh[i];
   if (irsensor[i] > irsensor[high]) {high = i;}
 }
 return high;
}

/* changes speed to required values for lmotor and rmotor. Change is
   more gradual when urgency is smaller. */
void drivemotors(int left, int right, int urgency) {
  int i;

  lmotor += urgency*(left - lmotor)/100;
  rmotor += urgency*(right - rmotor)/100;
  motor(0,(float)lmotor);
  motor(1,(float)rmotor);
  sleep(0.05);

}

/* returns compass heading, where 1 is N, 2 is E,..., and 4 is W */
int compass() {
  int dir0,dir1,dir2,dir3;

  dir0 = irsensor[7];
  dir1 = irsensor[8];
  dir2 = irsensor[9];
  dir3 = irsensor[10];
  if (dir0) return 1;
  if (dir1) return 2;
  if (dir2) return 3;
  if (dir3) return 4;
}

/* Turns the robot the # of radians CCW with no obstacle avoidance */
void turn(float degrees) {
  int rflag=1, lflag=1, rtemp=0, ltemp=0;

  while((((float)(rtemp-ltemp))/width) < degrees) {
     drivemotors(-50,50,10);

     if ((lflag) && (cds(4) > 250)) {
      lflag=0;
      if (lmotor > 0) ltemp++;
      else ltemp--;
     }
     else if (cds(4) < 10) lflag = 1;
     if ((rflag) && (cds(1) > 250)) {
      rflag=0;
      if (rmotor > 0) rtemp++;
      else rtemp--;
     }
     else if (cds(1) < 10) rflag = 1;
  }
}

/* Updates global map based on currmap array, and resets currmap. If
   max confidence has been reached in a square, it is not incremented.
*/
void updatemap() {
  int i,j,index1,index2;

  for (i=0; i<3; i++) {
     for (j=0; j<3; j++) {
      index1 = (i*currcol)+j;
      index2 = ((x-1+j)*(col) + (y+1-i));
      if(currmap[index1] > 0) {
         if(map[index2] < maxconf) {
         map[index2] += currmap[index1];
         }
      }
      else if (map[index2] > -maxconf) map[index2] += currmap[index1];
      currmap[index1] = 0;
     }
  }
  currmap[4] = -10;  /* set center square to -10 */
}

/* Increments squares on local map where the robot has recorded IR hits.
   The variable normangle is the robot's global heading in 45 degree
   increments, which dictates which squares record hits.
 */
void updatecurrmap() {
    if (normangle == 0.0) {
      currmap[1] += (irsensor[2] >> 3);
      currmap[2] += (irsensor[3] >> 3);
      currmap[5] += (irsensor[4] >> 3);
      currmap[8] += (irsensor[5] >> 3);
      currmap[7] += (irsensor[6] >> 3);
   }
   else if (normangle == pi/4.0) {
      currmap[0] += (irsensor[2] >> 3);
      currmap[1] += (irsensor[3] >> 3);
      currmap[2] += (irsensor[4] >> 3);
      currmap[5] += (irsensor[5] >> 3);
      currmap[8] += (irsensor[6] >> 3);
   }
   else if (normangle == pi/2.0) {
      currmap[3] += (irsensor[2] >> 3);
      currmap[0] += (irsensor[3] >> 3);
      currmap[1] += (irsensor[4] >> 3);
      currmap[2] += (irsensor[5] >> 3);
      currmap[5] += (irsensor[6] >> 3);
   }
   else if (normangle == 3.0*pi/4.0) {
      currmap[6] += (irsensor[2] >> 3);
      currmap[3] += (irsensor[3] >> 3);
      currmap[0] += (irsensor[4] >> 3);
      currmap[1] += (irsensor[5] >> 3);
      currmap[2] += (irsensor[6] >> 3);
   }
   else if (normangle == pi) {
      currmap[7] += (irsensor[2] >> 3);
      currmap[6] += (irsensor[3] >> 3);
      currmap[3] += (irsensor[4] >> 3);
      currmap[0] += (irsensor[5] >> 3);
      currmap[1] += (irsensor[6] >> 3);
   }
   else if (normangle == 5.0*pi/4.0) {
      currmap[8] += (irsensor[2] >> 3);
      currmap[7] += (irsensor[3] >> 3);
      currmap[6] += (irsensor[4] >> 3);
      currmap[3] += (irsensor[5] >> 3);
      currmap[0] += (irsensor[6] >> 3);
   }
   else if (normangle == 3.0*pi/2.0) {
      currmap[5] += (irsensor[2] >> 3);
      currmap[8] += (irsensor[3] >> 3);
      currmap[7] += (irsensor[4] >> 3);
      currmap[6] += (irsensor[5] >> 3);
      currmap[3] += (irsensor[6] >> 3);
   }
   else if (normangle == 7.0*pi/4.0) {
      currmap[2] += (irsensor[2] >> 3);
      currmap[5] += (irsensor[3] >> 3);
      currmap[8] += (irsensor[4] >> 3);
      currmap[7] += (irsensor[5] >> 3);
      currmap[6] += (irsensor[6] >> 3);
   }
}

/* Calculates robot’s movements in polar coords. and changes them to
rect.
   Then calls updatecurrmap(). If robot has moved off of the currmap,
   it calls updatemap(). If Robot is moving off of the global map, it
   calls turn().
*/
void fmap() {
float theta, dx1, dy1;

theta = ((float)(rcount - lcount))/width;
angle += theta;

if (angle >= 2.0*pi) angle -= 2.0*pi;
else if (angle < 0.0) angle += 2.0*pi;

if ((angle-normangle) > (pi/8.0))  normangle += pi/4.0;
else if ((angle-normangle) < (-pi/8.0)) normangle -= pi/4.0;

vector = (float)(lcount + rcount)/2.0;
if (theta < 0.0) theta = -theta;

dx1 = vector*sin(theta);
dy1 = vector*cos(theta);
if ((angle > pi/2.0) && (angle < 3.0*pi/2.0)) dx1=-dx1;
if (angle > pi) dy1=-dy1;

x1 += dx1;
y1 += dy1;
```

```
lcount = 0;
rcount = 0;
updatecurrmap();

if (x1 > 30.0) {
   updatemap();
   if (x < col) {x++; x1 -= 30.0;}
   else turn(pi/2.0);
}
else if (y1 > 30.0) {
      updatemap();
      if (y < col) {y++; y1 -= 30.0;}
      else turn(pi/2.0);
     }
     else if (x1 < 0.0) {
           updatemap();
           if (x > 0) {x--; x1 += 30.0;}
           else turn(pi/2.0);
     }
        else if (y1 < 0.0) {
              updatemap();
              if (y > 0) {y--; y1 += 30.0;}
              else turn(pi/2.0);
        }
}

/* initializes global map to all zeros */
void init_map() {
   int i;
   for (i=0;i<mapsize;i++) map[i]=0;
}


void main() {
  int i, sensor, next=100;
  int rflag=1, lflag=1;
  poke(0x7000,0xff);
  charge(); /* uses music.c  */

  while(1) {
     if (sensor = read()) {
      if (sensor == 2) {drivemotors(100,75,irsensor[2]); next = 100;}
      if (sensor == 3) {drivemotors(100,-25,irsensor[3]); next = 100;}
      if (sensor == 4) {drivemotors(next,-next,(irsensor[4]));}
      if (sensor == 5) {drivemotors(-25,100,irsensor[5]); next = -100;}
      if (sensor == 6) {drivemotors(75,100,irsensor[6]); next = -100;}
     }
     else {drivemotors(100,100,10); }

     if (i=f_bump()) {
      drivemotors(-25,-25,100);
      if (i>70) drivemotors(0,-50,25);
      else drivemotors(-50,0,25);
     }

     if (i=r_bump()) {
      drivemotors(0,0,100);
      drivemotors(25,-25,25);
     }
/* shaft encoder code */
     if ((lflag) && (cds(4) > 250)) {
      lflag=0;
      if (lmotor > 0) lcount++;
      else lcount--;
     }
     else if (cds(4) < 10) lflag = 1;

     if ((rflag) && (cds(1) > 250)) {
      rflag=0;
      if (rmotor > 0) rcount++;
      else rcount--;
     }
     else if (cds(1) < 10) rflag = 1;
     lcount=2; rcount=1;
     fmap();
  }
}


/* scale map confidence levels to 0 to 9 and display them.  */
/* (uses serial.c)                                           */
void show_map() {
  int i,j,conf;

  init_serial();

    msleep(10L);
    for (i=0;i<30;i++) {
      for (j=0;j<30;j++) {
       conf = (map[(i*col + j)]) + 1000;
       conf = conf/200;
       put_int(conf);
       write(" ");
       if (j==29) write("\n");
      }
    }
}
```