EEL 5666: Intelligent Machines Design Laboratory
Professor A. Antonio Arroyo

Tracker

Final Report by Todd Martin
Spring, 1999

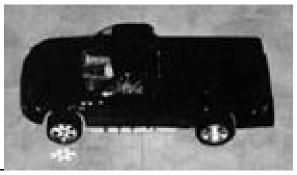
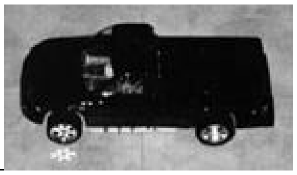
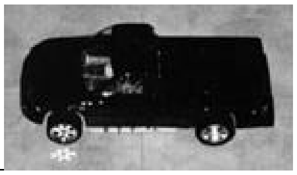


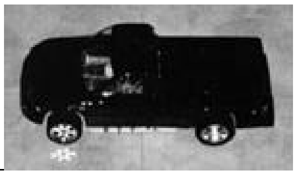
Table of Contents





Abstract

Tracker is a robot car designed to travel on a preprogrammed path. If an obstacle is blocking the path, Tracker will maneuver around the obstacle and rejoin the original path.

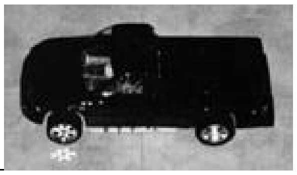


Introduction

The original idea for this project came from reading about the development of an Automated Highway System (AHS). While a real car traveling on an AHS will use radar, magnets, and vision as its sensors, I determined that these were too complicated and expensive for the scope of this project. Instead, I will use IR and sonar to track objects and a compass and odometer to follow the road.

The project's objective is to travel through a preprogrammed path that simulates a real highway. An autonomous vehicle must be able to react to obstacles, like a real driver would, and maneuver around them. But, maneuvering around the obstacle is not enough, if the robot is not able to resume its course then it is not useful for any practical functions. The purpose of a real, full size, automated vehicle is to carry passengers. Tracker will not carry anything to simulate passengers, but the objective is still the same.

If all that was needed to accomplish this was to create a road and use IR to follow it, then this project would be simple. However, in real life, nothing is simple. A car could follow a road by monitoring the white lane lines, but what happens when the car enters a construction zone where multiple lines have been painted. Sometimes even humans have trouble determining where the lanes go. These problems are still being solved by professionals who are building full size "robots."

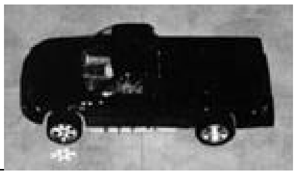


Tracker is not designed to address all of these problems. Instead, Tracker will be able to read a path from memory and follow it to reach a destination. If started from the same location Tracker should follow the path identically each time.

Tracker will be able to travel a course even if there is no road to tell it where to go.

The only IMDL project I have seen that uses some of the same principles is the Valet Robot, Val. Val allowed a user to walk it through a path and Val could recreate the path. Val was not able to avoid an obstacle and resume the path. Tracker uses different methods to solve a similar problem, because of the restrictions I placed on the platform. Tracker was designed to look like a real car once the top was placed over it. Val used two potentiometer wheels to monitor distance, but this was not feasible with Tracker's platform. Instead a custom made odometer and a compass were used to monitor distance traveled and degrees turned.

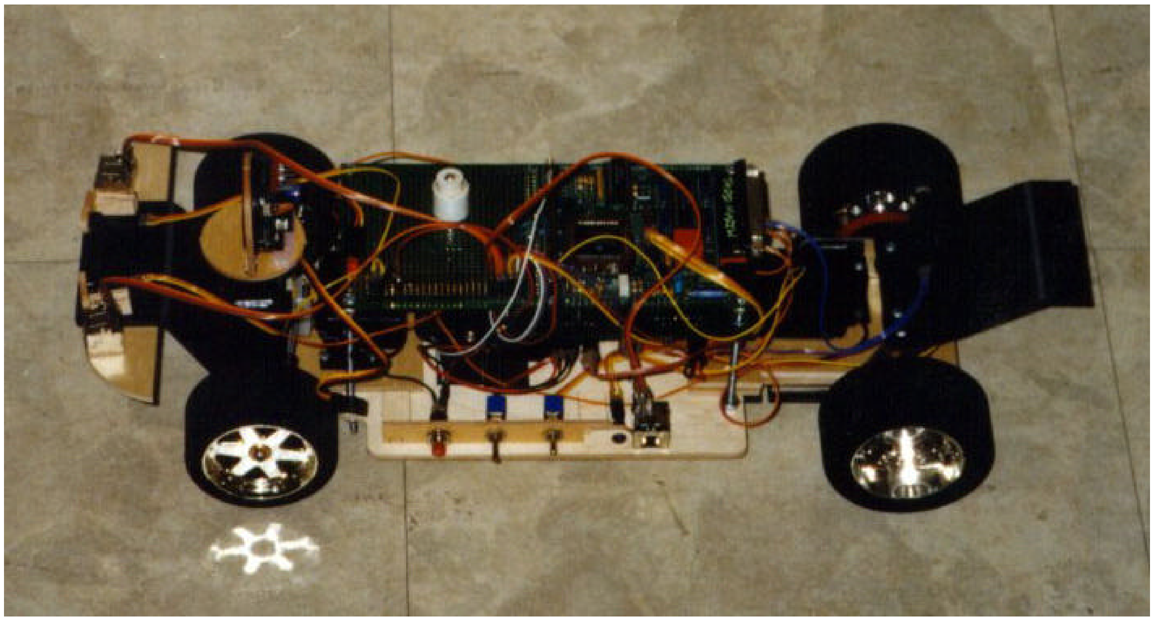
The following sections will detail the design of Tracker and list the results.

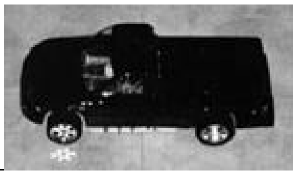


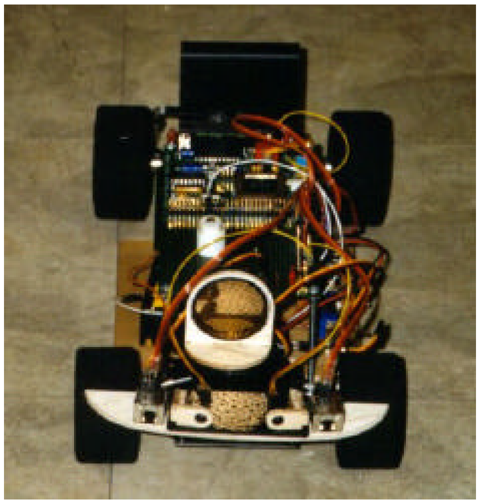
Integrated System

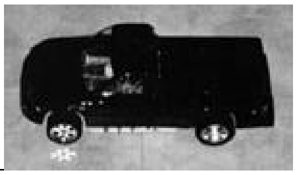
Tracker uses various sensors to achieve its goals. Ideally sonar would have been able to cover for IR if environmental conditions in the room saturated the sensors, but this has not yet been implemented. The hardest part of the programming and one that I have not perfected is using the odometer and compass together. My move_fwd() function takes parameters for distance, direction, and steering angle, but I did not fully explore how these parameters should relate to each other. Currently the function stops when either the distance or direction parameter is met. It does not use one parameter to check that the other is consistent.

The following pictures show the completed robot. The large sonar sensor up front is not actually being used by the software program.







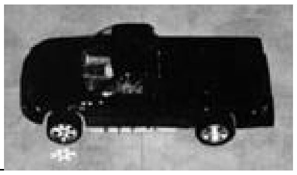


Mobile Platform

Tracker will use a Remote Control (RC) car kit of a Dodge Ram truck as its base. The kit is rear wheel drive. It was chosen because a front wheel steering system was included. It turned out that the rear axle provided an unanticipated bonus, because the axle made it easy for Tracker to travel in a straight line. If individual motors were used to power the rear wheels then it would have been very hard to synchronize them to make Tracker go in a straight line. Another good feature of the kit was its foam like wheels. They provided a large amount of grip so wheel slippage was not a problem.

The 68HC11 EVBU board is mounted on a wooden platform behind the steering servo. This wooden platform will be attached to the kit's plastic base by screws.

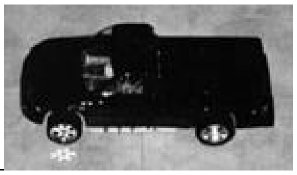
IR sensors were mounted in the fog lights of the cover. The sonar sensor was placed under the hood atop a partially hacked servo.



Actuation

A fully hacked servo powered the rear axle, providing forward and reverse movement. Steering was controlled by a servo. A third servo was placed in front of the steering servo to direct a sonar sensor, but it was not used in the final demo.

A faster motor could have been used, but a fully hacked servo was chosen because it could be bought from the lab.

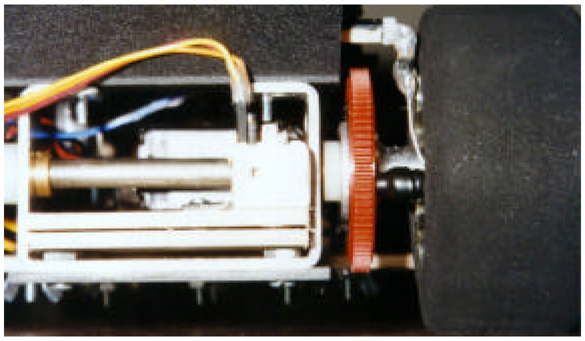


Sensors

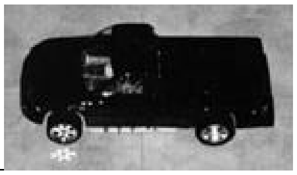
INFRARED (IR)

IR sensors were placed in the front of Tracker, in the fog lights, and on the side for wall following. Rear sensors were going to be implemented, but were left off due to time constraints.

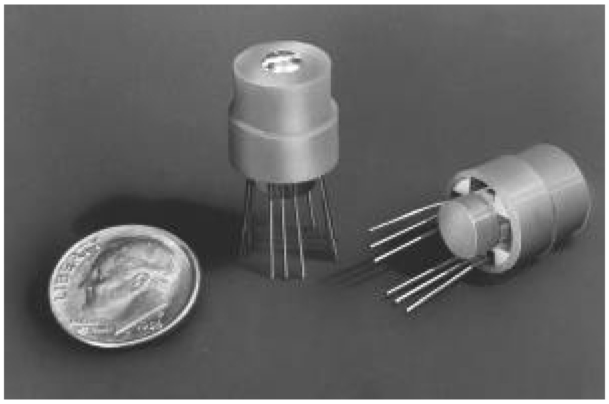
ODOMETER



An IR emitter and detector combination was used to measure distance traveled in order to assist with path following. The sensors were placed on either side of a large gear that transfers power from the motor to the axle. Slots in the gear were used to block the IR so pulses could be counted. Because the IR went right through the gear I had to plug the open areas with black elastic. This stopped the IR long enough for the program to count the pulses. There were six pulses for each revolution of the wheels.



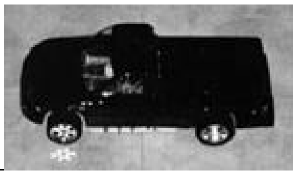
ANALOG COMPASS SENSOR



The sensor on the left is a digital compass with 45° of resolution. The right sensor is the 1655 analog compass.

The 1655 compass sensor from Dinsmore Instrument Company sends two analog signals to the 68HC11. These signals are two sinusoidal waves with their phases offset 90 degrees. After being converted to digital form the signals are compared against each other to determine the robots direction. Accuracy greater than one degree is possible if the output signals are amplified or if the analog to digital converter can restrict its samples to a 1.2 volt range. The current implementation uses un-amplified outputs running directly into the 68HC11's analog to digital converter that is sampling over a five volt range. This results in a resolution of just under two degrees.

In the future this sensor will be as easy to use as IR. Once the sensor is mounted onto a board it's leads must be connected to power, ground, and the analog to digital converter. Then the user can copy and paste the IC code (functions dir_angle, average_angle(), and average_sensor()) that calculates the absolute angle into their program. Overall implementation time could take a little as half an hour.

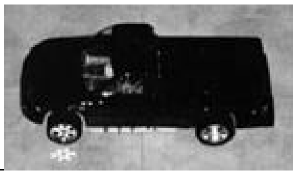


Behaviors

FOLLOW A PREPROGRAMMED PATH

Tracker's first behavior was following a preprogrammed path. I tried to interactively program the path instead of integrating it into the code, but I ran into problems which I eventually attributed to IC's continuous use of the serial port. I attempted to use a hyperterminal like program on a portable computer to steer Tracker through a path. The portable computer and the 68HC11 were connected through the serial port. After a random period of time the serial port would lock up.

An ideal system of recording a path using Tracker's sensors would have been to sample the compass heading, odometer distance, and amount the steering servo is turned, every second. Depending on how long the robot is running this could take up a large amount of memory. Each value would be placed in its own array like the function sample(), on page 36, did. The compass heading and steering servo data should be smooth and constant, but the compass data could fluctuate. Even if Tracker is going straight the data could read a couple of degrees different for each sample and Tracker's steering would become unstable when trying to follow the path. A smoothing algorithm could be implemented to make Tracker turn smoothly when following the path. Then running the record_path() function backwards would recreate the path.



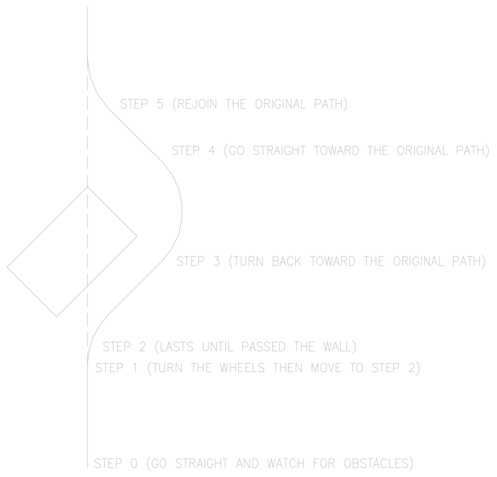
For the final demo Tracker's path was programmed through software. Both the odometer and compass were used as Tracker was told to go forward for a certain distance and turn for a specified number of degrees.

AVOID OBSTACLES

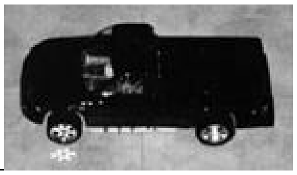
Front mounted IR sensors were used to perform basic obstacle avoidance.

RESUME THE PATH

The next goal was to program Tracker to resume the original path, but this has not yet been finished. To reduce the complexity of this problem I decided to make the path a straight line and place a rectangular object in the path. Using a straight path should make it easier to locate the original path once Tracker veers off course.


STEP 5 (REJOIN THE ORIGINAL PATH)
STEP 4 (GO STRAIGHT TOWARD THE ORIGINAL PATH)
STEP 3 (TURN BACK TOWARD THE ORIGINAL PATH)
STEP 2 (LASTS UNTIL PASSED THE WALL)
STEP 1 (TURN THE WHEELS THEN MOVE TO STEP 2)
STEP 0 (GO STRAIGHT AND WATCH FOR OBSTACLES)

This situation was broken up into 6 steps. An overview of these steps can be seen in the figure above. For more detail see the function resume(), on page 35, in Appendix B.



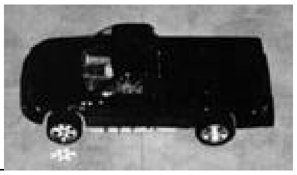
Experimental Layout and Results

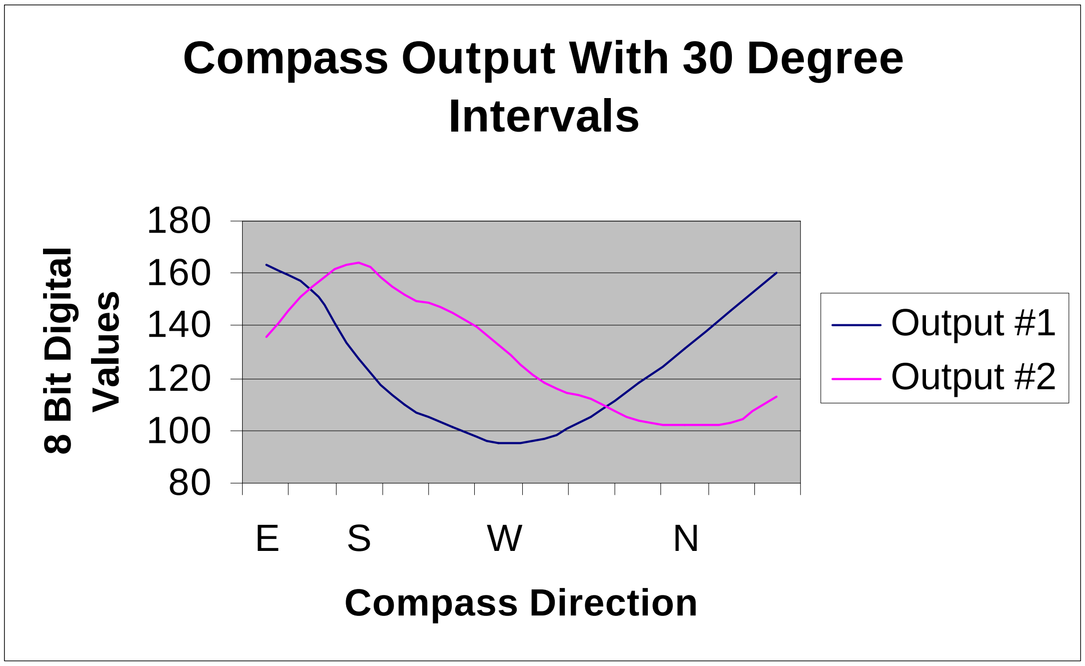
SENSORS

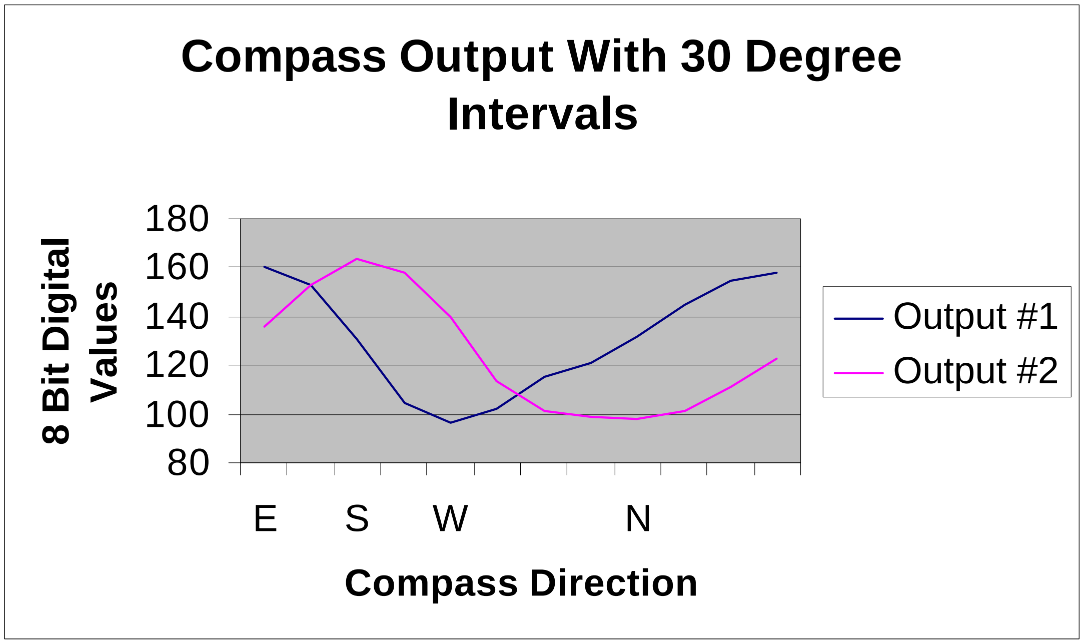
To test the accuracy of the 1655 Analog Compass Sensor from Dinsmore Instrument Company I sampled the outputs at 30 degree intervals. Graphing these outputs resulted in two offset sine waves as indicated on the data sheet. The results on the left correspond to the first graph. The sine wave from output #2 of this first graph was not as smooth as the first output, but the area between the crossing lines was fairly linear, as was anticipated. The results of both tests were comparable. All testing was performed in the IMDL lab.

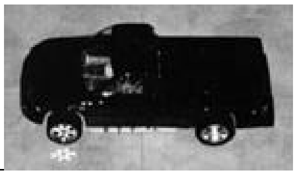
Output #1	Output #2	
163	136	E
153	155	
127	164	S
110	152	
101	145	
95	133	W
97	118	
105	112	
118	104	
131	102	N
146	103	
160	113	

Output #1	Output #2	
160	136	E
153	153	
131	164	S
105	158	
96	140	W
102	114	
115	101	
121	99	
132	98	N
145	101	
155	111	
158	123	



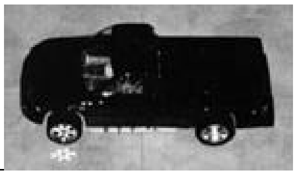

Compass Output With 30 Degree Intervals
8 Bit Digital Values
180
160
140
120
100
80
E
S
W
N
Compass Direction
Output #1
Output #2


Compass Output With 30 Degree Intervals
8 Bit Digital Values
180
160
140
120
100
80
E
S
W
N
Compass Direction
Output #1
Output #2



IR distance values

Analog Value	Distance to Object
85	no object
129	6"
109	1'-0"
95	1'-6"



Conclusion

Not all of my original goals were accomplished, but I feel that my platform and sensors were good enough to achieve those goals. I ran out of time developing and testing my navigation algorithms.

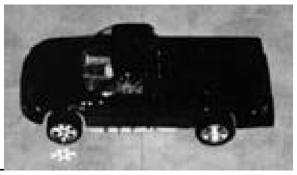
SENSORS

The custom made odometer was very accurate. Consistently achieving errors of less than one inch. This was helped by the foam tires which had very little slippage. Increasing the number of breaks on the gear would increase the odometer's resolution below one inch.

The compass gave consistent results although they varied depending on the magnetic field in the room. If I tried to turn Tracker 30 degrees it wouldn't always turn exactly 30 degrees, but it would always turn the same amount if I started from the same location. I'm not sure if there is any way to make the compass more resistant to magnetic fields generated by objects in the room. The compass' degree of resolution can be increased by amplifying the output signals so they have a range of five volts. Currently the range is 1.2 volts.

I would have liked to utilize sonar to detect objects up to five feet away. Possibly even trying to determine the objects width, but time constraints prevented me from doing so.

Bump sensors to aid with object avoidance would also have been helpful.

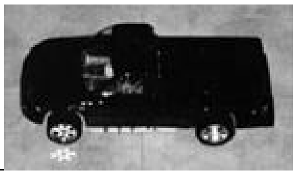


OBSERVATIONS

I was able to test the part of my code that records the path and it was recording data correctly. However, the serial port would lock up before I had a chance to figure out what was wrong with the function that followed the path. For each sample I recorded values into three separate arrays. For some reason IC was limiting the size of the arrays to less than 25, so instead of sampling data every second, I sampled based on how much Tracker turned. It is possible that ICC11 and assembly language would not have this same size restriction.

The part of my project that was a success was utilizing the compass and odometer sensors to continuously guide Tracker to the same location. If I had Tracker go forward through a preprogrammed number of turns and straightaways and then reversed the motor and traveled backwards through the path, Tracker ended up in almost the exact starting location.

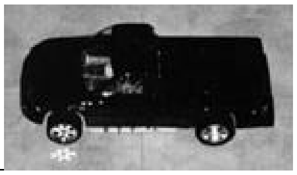
I determined as the semester progressed that it would be very hard to write a program that could allow a robot to find its way back onto a path. In the end I decided to simplify things and make the original path be a straight line. Then if I placed a rectangular object in Tracker's way, Tracker should have been able to negotiate its way around the obstacle and back onto the path. I have not completed this yet, but I feel that I am very close to getting it to work. The behaviors section of the report discussed this approach.



ADVICE

It seems that IC interferes with a programs ability to receive data through the serial port. When I finally determined that IC was probably causing the serial port communication to be unreliable it was too late to do anything about it. I would have liked to try porting the code to ICC11, but I did not have time. If you are planning to use the serial port to receive data, write this portion of your code early on and test it before progressing too far into the semester. Also, do not let yourself get hung up on one problem for too long, as I did.

I recommend that future robot builders use foam tires if slippage is a problem and one motor to power the robot. This will allow the robot to travel in a straight line easily. If a circular platform is needed for maneuverability then a single motor, might not make sense, but in general it seems to be a good idea.



Appendix A - Vendors

COMPASS SENSOR

Dinsmore Instrument Company. http://www.dismoregroup.com/dico

MAIN PLATFORM

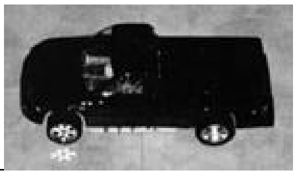
Dodge Ram Truck, 1/10 scale radio control kit by Parma International. Bought through Tower Hobbies. http://www.towerhobbies.com

SONAR

SonaSwitch Mini-A sonar sensor from EDP company. http://www.edpcompany.com

OTHER

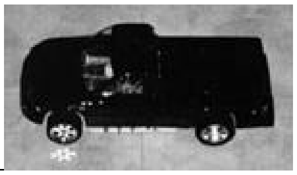
All other parts were purchased through IMDL.



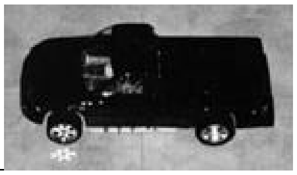
Appendix B - Code

The following program is the work in progress portion of the resume path program. Some functions like avoid() have parts commented which would need to be uncommented to work properly for certain situations. Almost all of my variables are global because earlier in the semester IC returned errors when I used local variables.

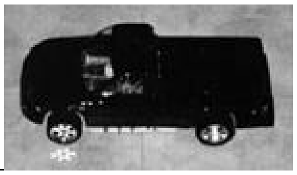
```
/************************************************************************************************
 * Title:      Control6.c
 *
 * Programmer:  Todd Martin
 *
 * Date:       April, 1999
 *
 * Version:     6
 *
 * Description: Develop algorithms for following a path.  Written in IC.
 *
 ***********************************************************************************************/

/**************************************** Constants *******************************************/
int COMPASS1 = 0;           /* analog port number for the cos curve*/
int COMPASS2 = 1;           /* analog port number for the sin curve*/
int MOTOR_JMPR = 0;         /* set to 1 for J4 and set to 0 for J5 */
float STRAIGHT = 40.0;   /* number of degrees to go straight */
float RIGHT = 60.0;         /* 70 max amount wheels can turn right */
float LEFT = 20.0;       /* 15 max amount wheels can turn left */
int IR_THRESHOLD = 100;     /* the value used in determining if an object has been detected */
int ODOM_THRESHOLD = 120;/* the value used in determining if the wheel is turning */
int ODOMETER = 2;           /* analog port number for the wheel encoder ir sensor */
int IR_FR_LFT = 3;       /* analog port number for the front left ir sensor */
int IR_FR_RGHT = 4;         /* analog port number for the front right ir sensor */
int IR_BK_LFT = 5;       /* analog port number for the back left ir sensor */
int IR_BK_RGHT = 6;         /* analog port number for the back right ir sensor */
int SONAR = 7;              /* analog port number for the sonar sensor */
float DEG1_CHG = 1.0;    /* amount deg1 is changed in turn() */
/************************************* End of Constants ***************************************/

/****************************************** Globals *******************************************/
int angle;                     /* the angle from 0 to 360 showing the direction */
int avg_angle;                 /* an average of multiple angle readings */
int new_angle;
int last_angle;
int total_dist;             /* keep track of the wheel encoder counts */
int output1;
int output2;
int range1;
int range2;
int upper_line = 155;       /* when output1 and output2 are equal high */
int lower_line = 108;       /* when output1 and output2 are equal low */
float angle_resolution;
float angle_res;
int forward;
int reverse;
int turn_right;
int turn_left;
int record;
int rst;
int straight;
float deg1;                    /* keeps track of the current setting for servo_deg1() */
float j;                       /* used to turn in small increments */
```



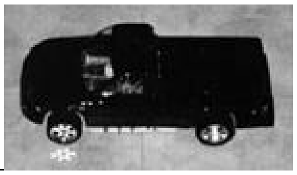
```
int record_pid;             /* pid for record_mode() */
int follow_active;          /* set to 1 when follow_path() is running as a process */
int odom;                   /* keeps track of pulse levels for the odometer */
int obj_fwd_left;
int obj_fwd_right;
int reverse_dist;
int straight_lvl;
int detect;                    /* set to 1 when an obstacle is detected */
long timer;
float servo_angle;          /* set to 1 when the wheels have turned the specified amount */
int ref_angle;
int turn_angle;
int turn_dist;
int straight_dist;
float turn_direction;
int step;
int ref_dist;
int wall;
int angle_chg;
int follow;
int follow_pid;             /* pid for follow_path() */
int dist[21];                  /* allocate space for record path */
int dir[21];                /* allocate space for record path */
float deg[21];                 /* allocate space for record path */
int path_index;
int mem_test;                  /* used in test_memory() */
/*************************************** End of Globals *****************************************/

/******************************************* Main ***********************************************/
void main()
{
   initialize();
   start_process(poll_distance());
   start_process(resume());
}
/**************************************** End of Main ******************************************/

/************************************ Program Functions ***************************************/
void average_sensor(int port, int *result)
{
   /* I could try dropping the highest and lowest values then dividing by 8 */
   int sum = 0;
   int i;

   for (i = 0; i < 3; i++) sum += analog(port);

   *result = sum / 3;
}

void average_angle(int *result)    /* runs as a process in record_mode(), but is called
otherwise */
{
   int sum = 0;
   int i;

   for (i = 0; i < 2; i++)
   {
      dir_angle();
      sum += angle;
   }

   *result = sum / 2;
}
```



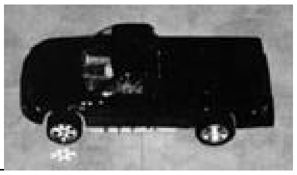
```
void avoid()    /* reads data from the forward IR and sonar and says if an obstacle is detected
*/
{
   obj_fwd_left = 0;
   obj_fwd_right = 0;
   detect = 0;
   if (analog(IR_FR_LFT) > IR_THRESHOLD)       /* an obstacle has been detected to the left */
   {
      obj_fwd_left = 1;     /* robot should turn right */
      /* tell the sonar look right to warn of future obstacles */
   }
   if (analog(IR_FR_RGHT) > IR_THRESHOLD)      /* an obstacle has been detected to the right */
   {
      obj_fwd_right = 1;    /* robot should turn left */
      /* tell the sonar look left to warn of future obstacles */
   }
/* if ((obj_fwd_left == 1) && (obj_fwd_right == 1))
   { */
      /* stop, or slow, the motor and confirm with sonar and judge the distance */
      /* confirm(1); */
      /* once confirmed stop and call back_up() */
/*    motor(MOTOR_JMPR, 0.0);  */    /* stop before backing up */
/*    detect = 1; */
/* set up a timer. if this happens milliseconds after one sensor returns true then straighten */
/*    if ((mseconds() - timer) < (long) 100)
      {
         straighten();
      }
      reverse = 1;
      back_up();
   } */
/* return this to else if, if I can generate a random direction to turn in the statement above
*/
   if (obj_fwd_left == 1)
   {
      detect = 1;
      motor(MOTOR_JMPR, 100.0);
/*    servo_deg1(RIGHT); */
      while (deg1 < RIGHT)
      {
         turn(-10, RIGHT);           /* the negative angle tells it to turn right */
         if (analog(IR_FR_LFT) < IR_THRESHOLD)
         {
            break;
         }
      }
      turn_direction = RIGHT;
      timer = mseconds();
   }
   else if (obj_fwd_right == 1)
   {
      detect = 1;
      motor(MOTOR_JMPR, 100.0);
/*    servo_deg1(LEFT); */
      while (deg1 > LEFT)
      {
         turn(10, LEFT);
         if (analog(IR_FR_RGHT) < IR_THRESHOLD)
         {
            break;
         }
      }
      turn_direction = LEFT;
      timer = mseconds();
```



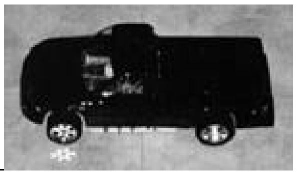
```
   }
}

void back_up()     /* reads data from the back IR and sonar and tells the motors an obstacle is
detected */
{
   obj_fwd_left = 0;
   obj_fwd_right = 0;
   motor(MOTOR_JMPR, -100.0);
   while (reverse == 1)
   {
      if ((analog(IR_FR_LFT) < IR_THRESHOLD) && (analog(IR_FR_RGHT) < IR_THRESHOLD))
      {
         /* continue in reverse for 6 inches or until an object is detected */
         distance();
         reverse_dist = total_dist + 6;  /* this statement backs up about 6 inches */
         while ((total_dist < reverse_dist) && (reverse == 1))
         {
            motor(MOTOR_JMPR, -100.0);      /* keep reversing */
            distance();
         }
         reverse = 0;
      }
      /* if (analog(IR_BK_LFT) > IR_THRESHOLD) an obstacle has been detected to the left, so
turn right */
         /* try to have the sonar look right to warn of future obstacles */
      /* if (analog(IR_BK_RGHT) > IR_THRESHOLD) an obstacle has been detected to the right, so
turn left */
         /* try to have the sonar look left to warn of future obstacles */
      /* if both the above are detected, stop, or slow, the motor and confirm with sonar */
         /* confirm(0) */
         /* once confirmed move forward */
         /* straighten();
         motor(MOTOR_JMPR, 100.0);    */
   }
}

void bump()        /* simulates a bump sensor */
{
   /* if the motor is moving, but the shaft encoder says Tracker is not moving */
   /* stop and move in the opposite direction. */
}

void confirm(int direction)     /* check to make sure obstacles are where ir detects them */
{
   /* direction is 1 for forward and 0 for reverse */
   /* average_angle(&avg_angle); */
   /* based on value of avg_angle calculate where to look for objects */
}

void dir_angle()
{
   average_sensor(COMPASS1, &output1);   /* cos curve */
   average_sensor(COMPASS2, &output2);   /* sin curve */

   angle_res = 1.915;          /* this changes with upper_line and lower_line */

   if ((output1 >= upper_line) && (output2 >= upper_line))
   {
      angle = 45;
   }
   else if ((output1 <= lower_line) && (output2 <= lower_line))
   {
      angle = 225;
```



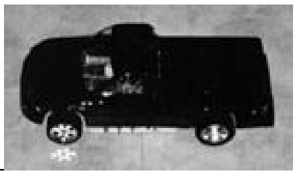
```
   }
   else if (output2 >= upper_line)
   {
      /* output1 measures from 45 to 135 degrees */
      range1 = (upper_line - output1);
      angle_resolution = ((float)range1 * angle_res);
      angle = ((int)angle_resolution + 45);
   }
   else if (output1 <= lower_line)
   {
      /* output2 measures from 135 to 225 degrees */
      range2 = (upper_line - output2);
      angle_resolution = ((float)range2 * angle_res);
      angle = ((int)angle_resolution + 135);
   }
   else if (output2 <= lower_line)
   {
      /* output1 measures from 225 to 315 degrees */
      range1 = (output1 - lower_line);
      angle_resolution = ((float)range1 * angle_res);
      angle = ((int)angle_resolution + 225);
   }
   else if (output1 >= upper_line)
   {
      /* output2 measures from 315 to 45 degrees */
      range2 = (output2 - lower_line);
      angle_resolution = ((float)range2 * angle_res);
      angle = ((int)angle_resolution + 315);
   }
   else
   {
      angle = angle;
   }
   if (angle >= 360)
   {
      angle = angle - 360;     /* keep the values of angle between 0 and 360 */
   }
   /* the write functions are used to observe the outputs with hyperterminal */
/* write("angle = ");
   write_int(angle);
   write("output1 = ");
   write_int(output1);
   write("output2 = ");
   write_int(output2); */
}

void distance()             /* measure the distance traveled */
{
   /* there are 6 counts per wheel revolution */
   if (analog(ODOMETER) < ODOM_THRESHOLD)
   {
      if (odom != 1)
      {
         total_dist++;               /* add to the count */
      }
      odom = 1;
/*    write("odom = ");
      write_int(odom);
      write_int(total_dist); */
   }
   else
   {
      odom = 0;
/*    write("odom = ");
```



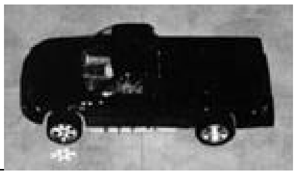
```
      write_int(odom);
      write_int(total_dist); */
   }
   /* divide by 6 to get the number of inches traveled */
   /* if integers aren't accurate enough use float */
}

void follow_path()    /* recreates the recorded path */
{                        /* receive() and object avoidance should still be running */
   int last_index;
   int follow_dist;
   int follow_dir;
   float follow_deg1;
/* write("Entering follow_path()\n\r"); */
   last_index = path_index;
   path_index = 0;
   rst = 0;

   while ((path_index < (last_index - 1)) && (follow == 1))
   {
      follow_dist = dist[path_index + 1] - dist[path_index];
      follow_dir = dir[path_index + 1] - dir[path_index];
      follow_deg1 = deg[path_index + 1] - deg[path_index];
/*    write("follow_dist = ");
      write_int(follow_dist); */
      if ((follow_dir > -5) || (follow_dir < 5))
      {
         follow_dir = 0;
      }
/*    write("follow_dir = ");
      write_int(follow_dir); */
      move_fwd(follow_dist, follow_dir, follow_deg1);

      path_index++;
   }
   path_index++;
   follow = 0;
   motor(MOTOR_JMPR, 0.0);        /* stop when finished following the path */
   straighten();
}

void initialize()
{
   init_servos();
   init_serial();                 /* initialize the serial port */
   motor(MOTOR_JMPR, 0.0);        /* stop motor */
   total_dist = 0;             /* clear the distance counter */
   odom = 0;                   /* set it so distance() triggers on a rising edge */
   rst = 1;                       /* tells arbitrate the board has just been reset */
   straight_lvl = 0;              /* this makes it so the first location is recorded */
   follow_active = 0;
   average_angle(&avg_angle);     /* returns the current angle */
                                  /* scan the surroundings */
   poke(0x7000, 0xFF);            /* turn on the infared LED's */
   detect = 0;
   step = 0;
}

void init_servos()
{
   servo_on();
   straighten();      /* straighten the steering servo */
}
```



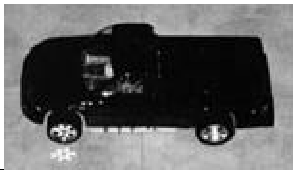
```
/* controls how far the robot moves based on the wheel encoder */
/* a travel_dist of 36 is 51 inches, 14 is 1.5 ft */
/* this seemed to be repeatable with a 1 inch accuracy */
/* turns until the robot reaches the specified angle change */
void move_back(int travel_dist, int angle_chg, float angle_deg1)
{
   int end_dist;

/* write("Entering move_back()\n\r"); */
   distance();
   end_dist = total_dist + travel_dist;

   /* monitor compass to see when I reach the desired angle */
   j = DEG1_CHG;

   /* calculate the current angle
   average_angle(&avg_angle);        /* returns avg_angle */

   /* calculate what the new angle should be */
   new_angle = avg_angle + angle_chg;

   /* keep the value between 0 and 359 */
   if (new_angle >= 360)
   {
      new_angle = new_angle - 360;
   }
   if (new_angle < 0)
   {
      new_angle = new_angle + 360;
   }

   /* make sure Tracker turns in the most efficient direction */
   if (angle_chg > 180)
   {
      angle_chg = angle_chg - 360;
   }
   if (angle_chg < -180)
   {
      angle_chg = angle_chg + 360;
   }

   if (new_angle > avg_angle)
   {
      if (angle_chg >= 0)
      {
         while ((avg_angle < new_angle) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn_back(angle_chg, angle_deg1);     /* turn */
               motor(MOTOR_JMPR, -100.0);               /* move robot backwards */
/*          }
            else {break;} */
            distance();
         }
      }
      else
      {
         while ((avg_angle > 0) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
```



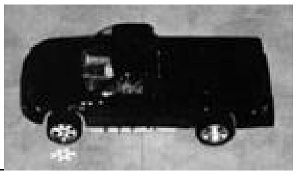
```
               turn_back(angle_chg, angle_deg1);     /* turn */
               motor(MOTOR_JMPR, -100.0);               /* move robot backwards */
/*          }
            else {break;} */
            distance();
         }
         while ((avg_angle > new_angle) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn_back(angle_chg, angle_deg1);     /* turn */
               motor(MOTOR_JMPR, -100.0);               /* move robot backwards */
/*          }
            else {break;} */
            distance();
         }
      }
   }
   else if (new_angle < avg_angle)
   {
      if (angle_chg < 0)
      {
         while ((new_angle < avg_angle) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn_back(angle_chg, angle_deg1);     /* turn */
               motor(MOTOR_JMPR, -100.0);               /* move robot backwards */
/*          }
            else {break;} */
            distance();
         }
      }
      else
      {
         while ((avg_angle < 360) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn_back(angle_chg, angle_deg1);     /* turn */
               motor(MOTOR_JMPR, -100.0);               /* move robot backwards */
/*          }
            else {break;} */
            distance();
         }
         while ((avg_angle < new_angle) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn_back(angle_chg, angle_deg1);     /* turn */
               motor(MOTOR_JMPR, -100.0);               /* move robot backwards */
/*          }
            else {break;} */
            distance();
         }
      }
   }
   else
   {
      servo_deg1(deg1);                      /* keep going in the same direction */
```



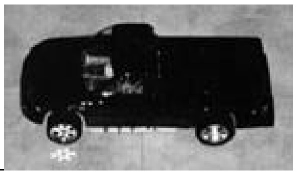
```
      while (total_dist < end_dist)
      {
/*          avoid();
            if (detect == 0)
            { */
               motor(MOTOR_JMPR, -100.0);        /* move robot backwards */
/*          }
            else {break;} */
            distance();
      }
   }
}

/* controls how far the robot moves based on the wheel encoder */
/* a travel_dist of 36 is 51 inches, 14 is 1.5 ft */
/* this seemed to be repeatable with a 1 inch accuracy */
/* turns until the robot reaches the specified angle change */
/* how can travel_dist and angle_chg be compared to make sure the data is valid? */
void move_fwd(int travel_dist, int angle_chg, float angle_deg1)
{
   int end_dist;

/* write("Entering move_fwd()\n\r"); */
   distance();
   end_dist = total_dist + travel_dist;

      /* call avoid(); to check for objects */
      /* or have arbitrate determine whether to exit the function and call avoid() */
      /* if (obj_fwd_left or obj_fwd_right = 1) record current distance */
         /* avoid_dist = total_dist; to save the state when robot veers of the path */
      /* check to see if any objects in the grid are too close */

   /* monitor compass to see when I reach the desired angle */
   j = DEG1_CHG;

   /* calculate the current angle
   average_angle(&avg_angle);        /* returns avg_angle */

   /* calculate what the new angle should be */
   new_angle = avg_angle + angle_chg;

   /* keep the value between 0 and 359 */
   if (new_angle >= 360)
   {
      new_angle = new_angle - 360;
   }
   if (new_angle < 0)
   {
      new_angle = new_angle + 360;
   }

   /* make sure Tracker turns in the most efficient direction */
   if (angle_chg > 180)
   {
      angle_chg = angle_chg - 360;
   }
   if (angle_chg < -180)
   {
      angle_chg = angle_chg + 360;
   }

   if (new_angle > avg_angle)
   {
      if (angle_chg >= 0)
```



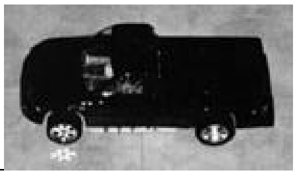
```
      {
         while ((avg_angle < new_angle) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn(angle_chg, angle_deg1);    /* turn */
               motor(MOTOR_JMPR, 100.0);       /* move robot forward */
/*          }
            else {break;} */
            distance();
         }
      }
      else
      {
         while ((avg_angle > 0) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn(angle_chg, angle_deg1);    /* turn */
               motor(MOTOR_JMPR, 100.0);       /* move robot forward */
/*          }
            else {break;} */
            distance();
         }
         while ((avg_angle > new_angle) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn(angle_chg, angle_deg1);    /* turn */
               motor(MOTOR_JMPR, 100.0);       /* move robot forward */
/*          }
            else {break;} */
            distance();
         }
      }
   }
   else if (new_angle < avg_angle)
   {
      if (angle_chg < 0)
      {
         while ((new_angle < avg_angle) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn(angle_chg, angle_deg1);    /* turn */
               motor(MOTOR_JMPR, 100.0);       /* move robot forward */
/*          }
            else {break;} */
            distance();
         }
      }
      else
      {
         while ((avg_angle < 360) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn(angle_chg, angle_deg1);    /* turn */
               motor(MOTOR_JMPR, 100.0);       /* move robot forward */
```



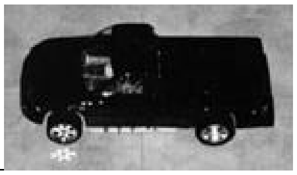
```
/*          }
            else {break;} */
            distance();
         }
         while ((avg_angle < new_angle) && (total_dist < end_dist))
         {
/*          avoid();
            if (detect == 0)
            { */
               turn(angle_chg, angle_deg1);    /* turn */
               motor(MOTOR_JMPR, 100.0);       /* move robot forward */
/*          }
            else {break;} */
            distance();
         }
      }
   }
   else
   {
      servo_deg1(deg1);                    /* keep going in the same direction */
      while (total_dist < end_dist)
      {
/*          avoid();
            if (detect == 0)
            { */
               motor(MOTOR_JMPR, 100.0);       /* move robot forward */
/*          }
            else {break;} */
            distance();
      }
   }
}

void obstacle_path()     /* keeps track of distances and angles traveled when avoiding obstacles
*/
{
   /* should be similar to record_path() */
   /* this function or another one will then guide the robot back onto the original path */
}

void poll_distance()
{
   while (1)
   {
      distance();
/*    wait(100); */
   }
}

void receive()     /* receives characters from the serial port */
{
   char data;
   forward = 0;
   reverse = 0;
   turn_right = 0;
   turn_left = 0;
   record = 0;
   straight = 0;

   while (1)
   {
      data = get_char();
      if ((data == 'B') || (data == 'b'))      /* apply the brakes */
      {
```



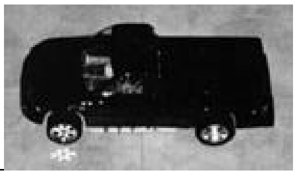
```
         motor(MOTOR_JMPR, 0.0);
         /* set every other serial port variable, except record, to 0 */
            forward = 0;
            reverse = 0;
            turn_right = 0;
            turn_left = 0;
            straight = 0;
            follow = 0;
      }
      else if (((data == 'F') || (data == 'f')) && (record == 0))
      {
/*       write("follow = 1, rst = 0\n\r"); */
         follow = 1;
         rst = 0;              /* make sure the path is not followed until the board is reset
*/
/*       write("Spawning follow_path()\n\r"); */
         follow_pid = start_process(follow_path());
         follow_active = 1;
      }
      else if ((data == 'R') || (data == 'r'))    /* toggle record_mode on and off */
      {
         if (record == 1)
         {
            record = 0;     /* turn off record mode */
            kill_process(record_pid);
/*          write("Killing record_mode()\n\r"); */
            sample();    /* sample the last data point */
            /* stop the robot when exiting record mode */
               forward = 0;
               reverse = 0;
               turn_right = 0;
               turn_left = 0;
               straight = 0;
               follow = 0;
               motor(MOTOR_JMPR, 0.0);
/*             write("Recorded data\n\r");
               for (i = 0; i < path_index; i++)
               {
                  write_int(dist[i]);
                  write_int(dir[i]);
               } */
         }
         else
         {
            record = 1;     /* put in record mode */
            /* stop and straighten the robot when entering record mode */
               forward = 0;
               reverse = 0;
               turn_right = 0;
               turn_left = 0;
               straight = 1;
               follow = 0;
               motor(MOTOR_JMPR, 0.0);
               straighten();
            record_pid = start_process(record_mode());
         }
      }
/*    else if (data == '&') */
      else if ((data == 'I') || (data == 'i'))
      {
         if (reverse == 1)
         {
            motor(MOTOR_JMPR, 0.0);
            reverse = 0;
```



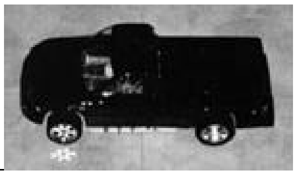
```
            follow = 0;
         }
         else
         {
            forward = 1; /* go forward */
            reverse = 0; /* turn off reverse */
            motor(MOTOR_JMPR, 100.0);
            follow = 0;
         }
      }
/*    else if (data == '(') */
      else if ((data == 'K') || (data == 'k'))
      {
         if (forward == 1)
         {
            motor(MOTOR_JMPR, 0.0);         /* stop before reversing */
            forward = 0;
            follow = 0;
         }
         else
         {
            reverse = 1; /* go in reverse */
            forward = 0; /* turn off forward */
            motor(MOTOR_JMPR, -100.0);
            follow = 0;
         }
      }
/*    else if (data == ''') */
      else if ((data == 'L') || (data == 'l'))
      {
         if (turn_left == 1)
         {
            turn_right = 0;
            turn_left = 0;
            straight = 1;
            servo_deg1(STRAIGHT);
            follow = 0;
         }
         else
         {
            turn_right = 1; /* turn right */
            turn_left = 0;
            straight = 0;
            servo_deg1(RIGHT);
            follow = 0;
         }
      }
/*    else if (data == '%') */
      else if ((data == 'J') || (data == 'j'))
      {
         if (turn_right == 1)
         {
            turn_right = 0;
            turn_left = 0;
            straight = 1;
            servo_deg1(STRAIGHT);
            follow = 0;
         }
         else
         {
            turn_left = 1;     /* turn left */
            turn_right = 0;
            straight = 0;
            servo_deg1(LEFT);
```



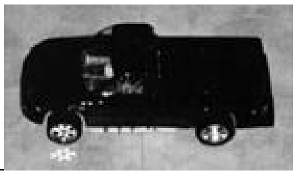
```
            follow = 0;
         }
      }
/*    else if ((data == 'S') || (data == 's'))
      {
         straight = 1;
         turn_right = 0;
         turn_left = 0;
         straighten();
         follow = 0;
      } */
      else
      {
         /* continue as previously instructed */
/*          forward = forward;
            reverse = reverse;
            turn_right = turn_right;
            turn_left = turn_left;
            straight = straight;
            follow = follow;
            record = record; */
      }
      if ((follow == 0) && (follow_active == 1))
      {
/*       write("Killing follow_path()\n\r"); */
         kill_process(follow_pid);
         follow_active = 0;
      }

/*    if ((follow == 1) && (rst == 1))   */    /* wait for the board to be reset to begin */
/*    {
         write("Spawning follow_path()\n\r");
         follow_pid = start_process(follow_path());
         follow_active = 1;
      } */
   }
}

void record_mode()    /* control the robot through the serial port */
{
/* write("Entering record_mode()\n\r"); */
   /* watch out for conflicts between user control and autonomous mode */
   /* add an led that lights up when in record mode */
   total_dist = 0;                /* clear the distance counter */
   odom = 0;                      /* set it so distance() triggers on a rising edge */
   rst = 0;                          /* a reset must occur before follow_path() */
   average_angle(&avg_angle);        /* keeps the angle updated for record_path() */
   distance();                       /* keeps the distance traveled updated */
   last_angle = avg_angle;
   path_index = 0;

   while (1)
   {
      record_path();                    /* keeps track of the path */
      average_angle(&avg_angle);        /* keeps the angle updated for record_path() */
      distance();                       /* keeps the distance traveled updated */
   }
}

void record_path()    /* keeps track of distances and angles traveled when being programmed */
{                       /* does not control the robots actual motion */
   /* take a sample when deg1 changes from straight */
   if ((straight_lvl == 1) && (straight == 0))
   {
```



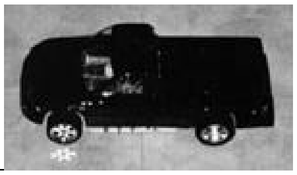
```
      sample();
      straight_lvl = 0;
   }
   if ((straight_lvl == 0) && (straight == 1))
   {
      sample();
      straight_lvl = 1;
   }
   /* currently only record while going forward */
   /* record any changes in avg_angle of greater than 9 degrees, this will remove small
imperfections in steering. */
   if ((avg_angle > (last_angle + 9)) || (avg_angle < (last_angle - 9)))
   {
      sample();
   }
}

void resume()
{
   motor(MOTOR_JMPR, 100.0);
   while (1)
   {
      if (step == 0)
      {
         write("step = 0\n\r");
         avoid();
         if ((detect == 1))
         {
            step = 1;
            ref_dist = total_dist;
            average_angle(&avg_angle);
            ref_angle = avg_angle;
         }
      }
      else if (step == 1)
      {
         write("step = 1\n\r");
         while ((analog(IR_FR_LFT) > IR_THRESHOLD) || (analog(IR_FR_RGHT) > IR_THRESHOLD))
         {
            /* stay at this point until object is no longer being detected */
         }
         step = 2;
      }
      else if (step == 2)
      {
         write("step = 2\n\r");
         while (wall == 0)        /* front and rear sensors should wall follow so I know when
Tracker is parallel with the object */
         {
            if (analog(5) > (116))
            {
               wall = 1;
            }
         }
         turn_dist = total_dist - ref_dist;
         average_angle(&avg_angle);
         turn_angle = avg_angle - ref_angle;
         while ((wall == 1))
         {
            straighten();         /* continue going straight until passed the object */
            motor(MOTOR_JMPR, 100.0);
            if (analog(5) < (116))
            {
               wall = 0;
```



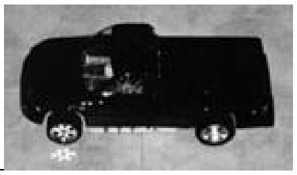
```
            }
         }
         straight_dist = total_dist - turn_dist;
         step = 3;
      }
      else if (step == 3)
      {
         write("step = 3\n\r");
         write_int(turn_dist);
         write_int(turn_angle);
         if (turn_direction == RIGHT)
         {
            turn_direction = LEFT;
         }
         else if (turn_direction == LEFT)
         {
            turn_direction = RIGHT;
         }
         turn_dist = (2 * turn_dist);
         write_int(turn_dist);
         turn_angle = -(2 * turn_angle);
         write_int(turn_angle);
         move_fwd(turn_dist, turn_angle, turn_direction);
         step = 4;             /* go straight again */
      }
      else if (step == 4)
      {
         write("step = 4\n\r");
         straighten();
         move_fwd(straight_dist, 0, STRAIGHT);
         step = 5;             /* turn back to the original angle */
      }
      else if (step == 5)
      {
         write("step = 5\n\r");
         average_angle(&avg_angle);
         angle_chg = ref_angle - avg_angle;
         if (turn_direction == RIGHT)
         {
            turn_direction = LEFT;
         }
         else if (turn_direction == LEFT)
         {
            turn_direction = RIGHT;
         }
         move_fwd(50, angle_chg, turn_direction);
         straighten();
         motor(MOTOR_JMPR, 0.0);
      }
   }
}

void sample()      /* add the current avg_angle and total_dist values to their arrays */
{
/* write("Entering sample()\n\r"); */
   dist[path_index] = total_dist;
   dir[path_index] = avg_angle;
   deg[path_index] = deg1;           /* record the current servo_deg1 setting */

   path_index++;
   last_angle = avg_angle;
}

void straighten()
```



```
{
   /* stop turning by smoothly straightening the wheels */
   /* implement logic to turn smoothly like turn() has */
   servo_deg1(STRAIGHT);
   deg1 = STRAIGHT;
}

void track() /* tracks objects with sonar and adds them to the grid until confirm is called */
{

}

void turn(int angle_chg, float angle_deg1)     /* turns the robot and updates the angle */
{
   /* logic that determines turning smoothness, speed, and direction */

   if (angle_chg < 0)                    /* turn right */
   {
      if (deg1 < angle_deg1)          /* keeps robot from turning past RIGHT */
      {
         deg1 = deg1 + j;
         servo_deg1(deg1);
         j += DEG1_CHG;
      }
   }
   else     /* turn left */
   {
      if (deg1 > angle_deg1)          /* keeps robot from turning past LEFT */
      {
         deg1 = deg1 - j;
         servo_deg1(deg1);
         j += DEG1_CHG;
      }
   }
   average_angle(&avg_angle);            /* check the current angle */
}

void turn_back(int angle_chg, float angle_deg1)    /* turns the robot and updates the angle */
{
   /* logic that determines turning smoothness, speed, and direction */

   if (angle_chg > 0)                    /* turn right */
   {
      if (deg1 < angle_deg1)          /* keeps robot from turning past RIGHT */
      {
         deg1 = deg1 + j;
         servo_deg1(deg1);
         j += DEG1_CHG;
      }
   }
   else     /* turn left */
   {
      if (deg1 > angle_deg1)          /* keeps robot from turning past LEFT */
      {
         deg1 = deg1 - j;
         servo_deg1(deg1);
         j += DEG1_CHG;
      }
   }
   average_angle(&avg_angle);            /* check the current angle */
}

void wait(int milli_seconds)
{
```



```
   long timer_a;

   timer_a = mseconds() + (long) milli_seconds;
   while (timer_a > mseconds())
   {
      defer();
   }
}
/******************************* End of Program Functions ************************************/

/*********************************** Testing Functions **************************************/
void monitor_angle()
{
   while (1)
   {
      average_angle(&avg_angle);
      write_int(avg_angle);
      wait(1000);
   }
}

void pattern()     /* Go in a pattern using the compass as a guide */
{
   move_fwd(9, 0, STRAIGHT);

   move_fwd(50, 30, LEFT);

   straighten();
   wait(1000);

   move_fwd(50, -60, RIGHT);

   straighten();
   wait(1000);

   move_fwd(50, 30, LEFT);

   straighten();
   wait(1000);

   motor(MOTOR_JMPR, 0.0);
   wait(3000);

   motor(MOTOR_JMPR, -100.0);
   wait(1000);

   move_back(50, -30, LEFT);

   straighten();
   wait(1000);

   move_back(50, 60, RIGHT);

   straighten();
   wait(1000);

   move_back(50, -30, LEFT);

   straighten();
   move_back(9, 0, STRAIGHT);

   motor(MOTOR_JMPR, 0.0);
}
```



```
void speed_test()        /* Tracker moves about 3.6 inches/second, varies with battery power */
{                        /* this speed is not valid since I changed the gear ratio */
   straighten();
   motor(MOTOR_JMPR, 100.0);
   wait(5000);
}

void test_odometer()
{
   /* uncomment serial outputs in distance() to verify that the odometer is working */
   move_fwd(36, 0, 0.0); /* moves Tracker in a straight line for 50 to 51 inches */
}

void test_memory()       /* see if variable values are retained when a reset occurs */
{                           /* if the variable is persistent it does not change on a reset */
   wait(2000);
   write_int(mem_test);
   if (mem_test == 71)
   {
      servo_deg1(RIGHT);
   }
   else
   {
      servo_deg1(LEFT);
   }
   mem_test = 71;
   write_int(mem_test);
}

void test_serial()
{
   char tst;
   while (1)
   {
      tst = get_char();
      write("echo: ");
      put_char(tst);
      write("\n");
   }
}
/******************************** End of Testing Functions ***********************************/
```