S.I.R. Bot

“Sector Information and Retrieval Bot”

Ruste Enderby

04-28-97

EEL 5666

Table of Contents

Abstract

S.I.R. bot is designed to gather sector information and retrieve the data for later viewing. In order to accomplish this the bot needs many sensors and behaviors working together to report a sectors activity. S.I.R. currently is equipped with a heavy duty chassis including a tank style tread for extra traction. It also includes two high speed motors, five IR transmitter and receiver units, two high speed motor drivers, Dallas SRAM, two internal IR shaft encoders, two terrain sensors, and a external audio speaker. The bot also includes various behaviors, including high speed/standard obstacle avoidance, terrain detection, multi-level stall detection, wall following, and odometer recording for distance. The incorporation of these behaviors working together with software allows S.I.R. to begin to function the basics of a information gather and retriever.

Introduction

The ability to obtain room or area information using a self propelled vehicle has always been of interest to many people. Although just being able to find obstacles and dimensions may not be enough information. It may be required to know if people are in the room, or audio is heard maybe for the use of remote recording. Features like heat sources may be able to detect location of fireplaces and other heat emitting sources. Motion will reveal animals or another verification of human movement or any other motion within the sector. To get the best description of a area, many sensors must be used to gather the most characteristics possible about the given sector, this is what S.I.R. bot is designed extract from a sector. S.I.R. Bot is designed to be a sector informational retrieval bot. The bot is able determine various types of occurrences within a sector. This includes perimeter distances, and terrain detection.

Bot Specifications

In order to gather information about many different phenomena, S.I.R. needed to be equipped with multiple sensors. S.I.R. is powered with high-speed motors to run tank treads providing maximum traction and control. The motors are driven by high speed motor drivers which are built with 12V, 10A relays, and darlington and normal NPN transistors. The two motors can draw up to 5A each, thus the high-speed drivers are sufficient to power these motors. The vehicle has the ability to use the high speed, if needed, given by the high output motors. The vehicle also has retained the ability to be remotely controlled if needed, since the chassis started as a R.C. vehicle. The front mounted IR transmitter cannons use automobile brake line tubing to provide two 7" mid range cannons. Two additional side rear IR transmitter/receiver cannons are also used to aid in the wall following behavior. An additional side front mounted IR transmitter/receiver have been installed on the right hand side to complete its side sensor array for wall following. S.I.R. also incorporates two terrain sensors, one for each motor to monitor the current drawn at any given time. This circuit was built with a 741 OP-Amp with a gain of 10 across a low in-line resistance under 1 ohm (see Appendix A). Finally shaft two shaft encoders were installed in the gear train of the vehicle, using IR transmitter/receiver pairs. They used 470, and 10k ohm resistances respectively. To incorporate the encoders, four equidistant holes were drilled in the main gear for each motor. The IR pair was glued on both sides of the walls parallel to the gear for both motors. The bot includes a external mounted speaker to play audio music pre-recorded at run-time. To keep the bot from losing all its programming when power is lost, a Dallas 32K SRAM is installed to keep battery backed up RAM. In order to provide sufficient power for all sensors, multiple motors, and weight of the vehicle, the bot runs off of a rechargeable 9.6 volt power unit, although a second additional power unit has been wired up and ready in case power consumption becomes a problem.

Behaviors

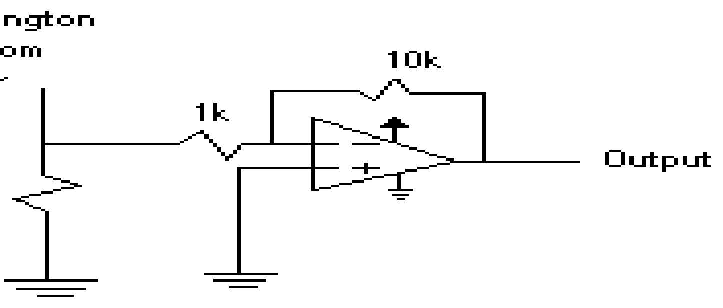
Various behaviors have been incorporated into S.I.R. bot in order to perform the basics of information retrieval. In order to accurately define a sector's size, accurate perimeter distancing needs to be calculated. This is done using the bots shaft encoders. While the bot is in wall following mode the software routine records total distance traveled by both wheels. Another behavior incorporated is high speed and standard obstacle avoidance. This is achieved using the two front mounted IR transmitter cannons and only two sharp IR 40K detectors. By increasing the power to the transmitters by removing the in-line resistors and lowering the software threshold on the receivers, it was possible to increase the vehicle's speed for obstacle avoidance. In addition, a terrain sensing behavior was added in addition to software to sample the sensor and report the average of a reading over time to detect various terrain encountered. While the bot is in high speed obstacle avoidance mode it is looking to encounter different terrain. Using the terrain sensors, it is possible to look for a medium or heavy load on the motors to determine harsher terrain to navigate over and record the total encounters. Another behavior incorporated is multi-level stall sensing. This behavior is very robust and should be the very basis for any other robot design. The behavior relies on the terrain sensor and the shaft encoders extensively. It out performs any bumper, or collision detector and is more universal for other types of motor monitoring by using the terrain sensor. A stall is detected when both of the motors report a very heavy terrain and do not report and distance traveled over a given period of time. This behavior is very useful, as it is quite easy to trap a robot by having it caught above the bumper, or if the robot runs into a heavy terrain and the motors slowly stop moving thus not reporting a 'collision' for a collision detector. Finally the last behavior incorporated in S.I.R. is wall following. In this behavior the bot uses the side front IR cannon, along with the side rear, and two front cannons to successfully navigate wall following. If the side front cannon receives a low reading then it will turn toward the wall, and if it gets too much of a reading then it turns away and continues this until the front detectors pick something up or a 90 degree angle has been found by using the side read IR cannon. A 90 degree angle is found by the instantaneous loss of signal on the side front IR while still getting a reading on the side rear IR. If a 90 degree angle is found the bot continues forward then turns right and slightly forward again thus performing a 90 degree turn to keep itself close to the wall it is following.

Conclusion

In conclusion S.I.R. bot currently has overcome excellent chassis mounting, superior movement ability, motor torque and weight technicalities while keeping an aesthetically pleasing design in mind. Some difficulties have included mounting problems, and circuit construction. S.I.R.'s programming currently is setup to first play a song 'Dixie' from Dukes of Hazzard and then performs a ½ second wheelie to show off the vehicles potential speed. The bot then proceeds into high speed obstacle avoidance for 30 seconds. Following the obstacle avoidance, the bot enters the wall following behavior for one minute and then stops allowing for the retrieval of the data collected during the high speed avoidance, and wall following. I am very pleased at the overall project as a whole. S.I.R. not only looks terrific, but it has much more potential that it is currently being used for. The terrain sensor has many other uses that can be easily incorporated, as well as the high speed motors. I feel S.I.R. has become a excellent start of a superb information sector and retrieval bot, incorporating all of the basic software and hardware needed for information gathering. In the future if more time permits, S.I.R. will be equipped with a long rage IR cannon to determine distances up to 20 feet using already purchased 2A IR transmitters.

Appendix A

Schematic of Terrain Sensor



Appendix B

```
/*  S.I.R. bot final DEMO code ran on 04-25-97            */
/*                                                          */
/* Written by: Ruste Enderby                               */
/*                                                          */
/* Sings 'dixie' from Dukes of Hazzard                     */
/* Performs a ½ second wheelie                             */
/* Enters high speed obstacle avoidance for 30 secs        */
/* Enters wall following mode for 60 secs                  */
/* Records all terrain found, and distance during          */
/* obstacle aviodance, and wall following                  */

/* IR Sensors and movement */

int nav=0;                          /* heading 0=off,1=slight right        */
                                    /*  2=slight left, 3=forward           */
int terrain=0;                      /* current terrain 0=light,1=medium    */
int total=0;                        /* total different terrains detected   */
int mo_l=1,mo_r=0;                  /* init motor left and right           */
int stall=0;                        /* detect global stall                 */
int left_ir,right_ir;               /* left and right IR sensors           */
int r_sr_ir,r_sf_ir;                /* right side front and rear ir's      */
int h_throttle=205,l_throttle=0;    /* current high and low throttle speeds     */
int terr_l=205,terr_r=205;          /* terrain reading for left and right wheels/
int r_ticks=0,l_ticks=0;            /* total wheel ticks from shaft encoders    */
int l_flipped=0,r_flipped=0;        /* total ticks flipped passed 32767         */
int sr_thresh=117,sf_thresh=120;    /* threshold values for side_ir's           */
int f_thresh;                       /* threshold value for front ir's           */
float l_offset=0.75,r_offset=0.0;   /* left and right motor speed offsets       */
float cruise=25.0;                  /* constant cruising speed of vehicle       */


void main()
  {
   /* start up IR-Transmitters */
   poke(0x7000,0xFF);
   /*init_serial();*/
   /* kick off dukes of hazzard! */
   dixie();
   start_process(get_ir_sensors());
   start_process(terrain_sensor());
   start_process(shaft_encoders());
   start_process(determine_stall());
   arbitrator();
  }

/* the big cheese, main kernel loop */
void arbitrator()
  {
   int id;

   /* fire off behaviors in order and time each */

   id=start_process(high_speed_aviodance());
   sleep(30.0);
   kill_process(id);
   /* cut motors */
   cruise=0.0;
   straight();
   sleep(1.0);
   /* clear out odometer to measure aproximate room perimeter */
   r_ticks=l_ticks=0;
   /* fire of reverse charge to announce new behavior      */
   rev_charge();
   /* reset cruise speed */
   cruise=20.0;
   id=start_process(follow_wall());
   sleep(60.0);
   kill_process(id);
   /* kill motors and shut-down */
   cruise=0.0;
   straight();
   /* announce finish blips */
   rev_charge();
  }

void high_speed_aviodance()
  {
   /* do wheelie! manually juice the motors to compensate
      for off_set adjustments                                         */
   motor(mo_l,100.0);
   motor(mo_r,100.0);
   sleep(0.5);
   /* cut motors */
   cruise=0.0;
   straight();
   sleep(1.0);
   cruise=25.0;
   /* using both motors */
   nav=3;
   while (1)
     {
      /* set front threshold */
      f_thresh=117;
      /* check for obsticle */
      if (detect_obsticle() && !stall)
            {
             while(detect_obsticle() && !stall)
               turn_left();
            }
      else if (stall)
                handle_stall(0.75);
      /* continue straight */
      straight();
     }
  }

/* wall following behavior */
void follow_wall()
  {
   /* initally only using left motor */
   nav=1;

   /* set front threshold */
   f_thresh=129;

   while (1)
     {
      /* have it look for below then if the r_sr_ir goes diminishes within
      few seconds to 90 degree turn */

      /* check for 90 degree turn */
      if ((r_sf_ir < sf_thresh-6) && (r_sr_ir > 126) && (!detect_obsticle())
            && !stall)
            {
             straight();
             sleep(0.40);
             turn_right();
             sleep(0.65);
             straight();
             sleep(0.25);
            }

      if ((r_sf_ir < (sf_thresh-1)) && (!detect_obsticle()) && !stall)
            {
             /* too far from the wall turn towards it slightly */
```

```
                 slight_turn_right();
                 nav=1;
                }
      else if ((r_sf_ir > (sf_thresh+3)) && (!detect_obsticle()) && !stall)
                     {
                      /* too close to wall turn away from it slightly */
                      slight_turn_left();
                      nav=2;
                     }
                   else if (stall)
                                   handle_stall(0.50);
                                 else if (detect_obsticle())
                                        {
                                                /* detected obsticle turn away from wall sharply */
                                                nav=0;
                                                turn_left();
                                                sleep(0.35);
                                        }
                                      /* otherwise continue current heading */
                                      else if (nav==2)
                                                    slight_turn_left();
                                                  else slight_turn_right();
     }
   }

/* if stall occurs deal with stall given response time */
void handle_stall(float response)
  {
   /* full reverse for 'response' seconds */
   reverse();
   sleep(response);

   /* left turn for 'response' seconds */
   turn_left();
   sleep(response*1.5);
   stall=0;
  }

/* look for obsticles */
/* 0 = no obsticles   */
/* 1 = left obsticle  */
/* 2 = right obsticle */
int detect_obsticle()
  {
   /* NOTE: Light can pick up to 115! */

   if (left_ir > f_thresh)
     return(1);
   else if (right_ir > f_thresh)
                   return(2);
                 else return(0);
  }


void get_ir_sensors()
  {
   while (1)
     {
      left_ir   = analog(1);
      right_ir  = analog(0);
      r_sr_ir   = analog(6);
      r_sf_ir   = analog(7);
      wait(200);   /* wait 400 ms */
     }
  }


void shaft_encoders()
```

```
 {
  int l_hit=0,r_hit=0;

  while (1)
    {
     /* left shaft encoder,150*/
     if (analog(5)<150)
            l_hit=1;
     if (l_hit && (analog(5)>250))
               {
                if (l_ticks++>32766)
                  {
                   l_flipped++;
                   l_ticks=0;
                  }
                l_hit=0;
               }
     /* right shaft encoder */
     if (analog(2)<185)
            r_hit=1;
     if (r_hit && (analog(2)>250))
               {
                if (r_ticks++>32766)
                  {
                   r_flipped++;
                   r_ticks=0;
                  }
                r_hit=0;
               }

     wait(200);
    }
 }


void terrain_sensor()
 {
  int c,t,lavg,ravg,avgl,avgr;
  int cntr=0;

  /* NOTE: approx 205 is motor OFF and FULL SPEED */

  while (1)
    {
     lavg=ravg=0;
     /* sample 50 data samples */
     for (t=0;t<50;t++)
            {
             lavg+=analog(3);
             ravg+=analog(4);
             wait(200);
            }

     /* update global terrain indicators to aid stall sensing */
     terr_l=lavg/50;
     terr_r=ravg/50;

     /* check for medium terrain */
     if (!terrain)
            if ((terr_l<200) && (terr_l>185))
              if ((terr_r<200) && (terr_r>185))
                {
                 terrain=1;
                 total++;
                }

     /* check for light terrain */
     if (terrain)
```

```
                 if ((terr_l<205) && (terr_l>195))
                   if ((terr_r<205) && (terr_r>195))
                     {
                      terrain=0;
                      total++;
                     }
     }
   }


void determine_stall()
  {
   int l_turns,r_turns;

   while (1)
     {
      r_turns=r_ticks;
      l_turns=l_ticks;
      sleep(1.5);
      /* if both motors are powered */
      if (nav==3)
                 {
                  if (!(r_ticks-r_turns) && !(l_ticks-l_turns) && (terr_l<190)
                    && (terr_r<190))
                    stall=1;
                 }
      /* only left motor powered */
      else if (nav==1)
                      {
                       if (!(r_ticks-r_turns) && !(l_ticks-l_turns) && (terr_l<190))
                                   stall=1;
                      }
                    /* only right motor powered */
                    else if (nav==2)
                                    if (!(r_ticks-r_turns) && !(l_ticks-l_turns) && (terr_r<190))
                                      stall=1;
     }
  }


void wait(int milli_seconds)
  {
   long timer_a;

   timer_a = mseconds() - (long) milli_seconds;
   while (timer_a > mseconds())
     defer();
  }

/* moderate turn to the right */
void mod_turn_right()
  {
   motor(mo_l,(cruise+l_offset)*1.50);
   motor(mo_r,(cruise+r_offset)*.20);
  }

/* slight turn to the right */
void slight_turn_right()
  {
   /* artifical increase of left motor due to differences between the 2 */

   motor(mo_l,(cruise+l_offset)*1.20);
   motor(mo_r,(cruise+r_offset)*.20);
  }

/* slight turn to the left */
void slight_turn_left()
  {
   motor(mo_l,(cruise+l_offset)*.20);
   motor(mo_r,(cruise+r_offset)*1.20);
  }

/* turn to the right */
void turn_right()
  {
   motor(mo_l,(cruise+l_offset)*1.20);
   motor(mo_r,-((cruise+r_offset)*1.20));
  }

/* turn to left */
void turn_left()
  {
   motor(mo_l,-((cruise+l_offset)*1.20));
   motor(mo_r,(cruise+r_offset)*1.20);
  }

/* just go straight */
void straight()
  {
   motor(mo_l,cruise+l_offset);
   motor(mo_r,cruise+r_offset);
  }

/* full reverse */
void reverse()
  {
   motor(mo_l,-(cruise+l_offset));
   motor(mo_r,-(cruise+r_offset));
  }


/* MUSIC CODE ADDED AND MODIFIED TO ADD IN DIXIE */

  char pp_song[]= "1#d 4e3r1#f4g3r1#d 3e1#f3g1c3bD1e3g1b 8&b2b2a2g2e2d10e 7r1#d 4e3r1#f4g3r1#d 3e1#f3g1c3b1g3b1e
28&e D3r1#d 4e3r1#f4g3r1#d 3e1#f3g1c3bD1e3g1b 8&b2b2a2g2e2d10e 12r U3e1d3b1a3g1#f 1&b3a1&b3a1&b3a1&b3a
2g2e2d20e";

  char w_song[]="1c2f1f2f1g2a1a2a1f2g1g2g1e4f1r 1f2a1a2a1&b2c1c2c1c2d1c2&b1a4g1r 1gU2d1d2d1c2d1e2f1d2c1d2c1&b4a1r
D1c2f1f2f1g2a1a2a1f2g1g2f1e5f";

  char classics_song[]="4g2c2d4c4g4a2g2f2e2c2d2e2f2g2a2b10c2r
4c4b4c2d2b2c2d2e2#f4g2#f2e4d2g2#f4g4d4e2d2c2b2g2a2b2c2b2c2d2e2d2e2#f12g";

  char charge_song[]="1c1e1g2c1r1g4c";

  char rev_charge_song[]="4c1g1r2c1g1e1c";

  char dukes[]="1e1d2c2c1c1d1e1f2g2g2g2e";

  char looney_tune_song[]= "U3e1d2c2d2e2d2e2c2d2d2d6d2r 3d1c2b2c2d2#c2d2b2c2c2c6c";

/* "dixie- dukes of hazzard" by Ruste Enderby and Mary Jo Black 04/97 */
/* YEE-HAW!                                                           */
void dixie()
{ play(dukes);}


/* reverse charge for various uses by Ruste Enderby and karsten 04/97 */
void rev_charge()
{ play(rev_charge_song);}


/* randy.  "pink panther" */

void pp() {
    play(pp_song);
}

/* added to by karsten ulland 10/92 */

void wate() {
  play(w_song);
}

void classics() {
  play(classics_song);
}


/* more from karsten 1/93 */

void charge() {
  play(charge_song);
}

/* randy 4/94 */

void looney_tune() {
    tempo=8;
    play(looney_tune_song);
    tempo=12;
}

/*********************************************************************/
/* music player     Randy Sargent */

/* # of milliseconds per 16th note, divided by 8 */
int tempo= 12;
long time, newtime;

/*  command letter to motor control  */
int music_current_command= 'o';

/*
Example using play:

    Songs should be GLOBALS so they can be longer - MD 7/93

    char song[]= "1#d 4e3r1#f4g3r1#d 3e1#f3g1c3bD1e3g1b 8&b2b2a2g2e2d10e 7r1#d 4e3r1#f4g3r1#d 3e1#f3g1c3b1g3b1e 28&e
D3r1#d 4e3r1#f4g3r1#d 3e1#f3g1c3bD1e3g1b 8&b2b2a2g2e2d10e 12r U3e1d3b1a3g1#f 1&b3a1&b3a1&b3a1&b3a 2g2e2d2e18e
8r 2g2e2d2e18e 8r 2g2e2d2e18e

void pp() {
    play(song);
}

Also see tequila.c for an example of music interspersed with commands
Also see tunes.c for other fun tunes.

*/

int music_next_command= 0;

void play(char song[])
{
    int i, duration, accidental, delta, note, rest;
    int notes[]= {0,2,3,5,7,8,10};
    int old_note= 30;
    play_reset();

    i= 0;
    while (song[i]) {
        while (1) {
          while (song[i] == ' ') ++i;
          if (song[i] == 'X') {++i; music_next_command= song[i]; ++i;}
          else break;
        }

        if (!song[i]) break;

        while (song[i] == 'D') {
          old_note -= 12;
          ++i;
        }
        while (song[i] == 'U') {
          old_note += 12;
          ++i;
        }
        if ('0' <= song[i] && song[i] <= '9') {
          duration= 0;
          while ('0' <= song[i] && song[i] <= '9') {
                duration = duration * 10 + song[i]-'0';
                ++i;
          }
        }
        if (song[i] == '#') {
          accidental= 1;
          ++i;
        } else if (song[i] == '&') {
          accidental= -1;
          ++i;
        } else {
          accidental= 0;
        }
/*    printf("dur %d a %d i %d\n", duration, accidental, i);*/
/*    wait();*/
        if (song[i] == 'r') rest= 1;
        else {
          if (song[i] < 'a' || song[i] > 'g') {
                printf("\nBad note:%c\n", song[i]);
                beep();
                beep();
                beep();
                sleep(5.0);
                return;
          }
          note= notes[song[i]-'a'] + accidental;
          rest= 0;
        }
        i++;

        if (rest) {
          play_note(0, duration);
        } else {
          delta= note - (old_note % 12);
          old_note += delta;
          if (delta > 5) old_note -= 12;
          if (delta < -5) old_note += 12;

          play_note(old_note, duration);
        }
  }
  play_note(0, 1);
  music_command('o');
}

void music_command(int c)
{
  music_current_command= c;
}
```

```
void play_reset()
{
    time= mseconds();
    newtime= time+100L;
}

void play_note(int note, int duration)
{
   play_note_2(note, duration*7);
   play_note_2(0, duration);
}

void play_note_2(int note, int duration)
{
    float freq;
    int   period;

    if (note) {
                freq= 55.0 * (2. ^ (((float) note) / 12.));
    }

    while (mseconds() < newtime);
    if (note) {
                set_beeper_pitch(freq);
                beeper_on();
    }
    else {
                beeper_off();
    }
    if (music_next_command &&
                music_current_command != music_next_command){
                music_command(music_next_command);
                music_next_command= 0;
    }

    newtime += (long)(duration * tempo);
}
```