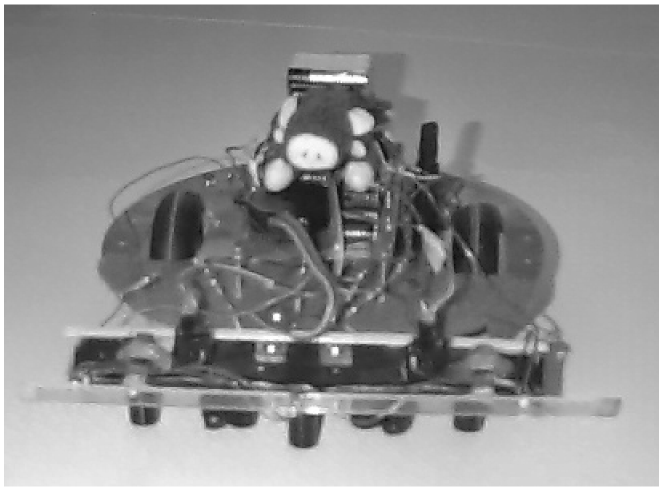
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF FLORIDA
INTELLIGENT MACHINES DESIGN LABORATORY
EEL 5666 (ROBOTICS)



FINAL REPORT ON AUTONOMOBILE

INSTRUCTOR: DR. KEITH L. DOTY

PREPARED BY: KALPESH GALA
APRIL 26, 1997

TABLE OF CONTENTS

INDEX OF APPENDICES

INDEX OF FIGURES

ABSTRACT

The development of an autonomous agent's behavior set depends on its perception of stimuli provided from the environment. Sensory information available to the robot is its only way to react to its environment.

Autonomobile is a mobile robot that performs a variety of behaviors in order to carry out a specific goal. That goal is to serve as an autonomous car. Its behaviors are self-calibration, outward spiral search, obstacle avoidance, find road, road follow, and visibility. The purpose of the visibility behavior is to vary Autonomobile's speed to adapt to changing levels of light (i.e. day, night, or fog.) The other behaviors are responsible for calibrating Autonomobile's sensors for its environment and then actual navigation. The integration of these behaviors is used to simulate a human's ability to drive.

INTRODUCTION

The scope of the Autonomobile project was chosen to allow completion by the end of the semester. Today’s autonomous mobile robots roam with no sense of guidance. Basic obstacle avoidance algorithms direct robots randomly. Autonomobile’s programming has object avoidance, road following, self-calibration, and visibility algorithms. The resulting behavior set consists of basic yet useful routines that allow Autonomobile to drive safely around a city.

This paper is composed of seven major sections. The Integrated System section describes the relationship between the behaviors, sensors, and platform. The Mobile Platform section relates the platform structure with the controlling microprocessor. The Sensor section describes the three sensor types and indicates their use relative to the behaviors. The Behavior section examines implemented behaviors in detail. The Experimental Results and Conclusions sections summarize the verification and success of each behavior/sensor combination. Appendix A contains all the "C" source code Autonomobile uses to implement its behaviors.

INTEGRATED SYSTEMS

Autonomobile follows general structural guidelines given by Brooks [1]. Multiple procedures constitute behaviors based on sensory input. The competing behaviors feed information into an arbitration network, which determines Autonomobile's resulting action.

BEHAVIOR SIGNAL FLOW

Autonomobile uses three sensors and seven behaviors to navigate a city. Autonomobile has a sense of vision. In order to establish the correct readings a calibration period is used to orient Autonomobile to its environment. It also dodges obstacles using infrared (IR) light reflection and a front fender for bump detection. The arbitration network realizes a simple priority scheme to choose between behaviors. Figure 1 shows the signal flow through the behavior modules [4].

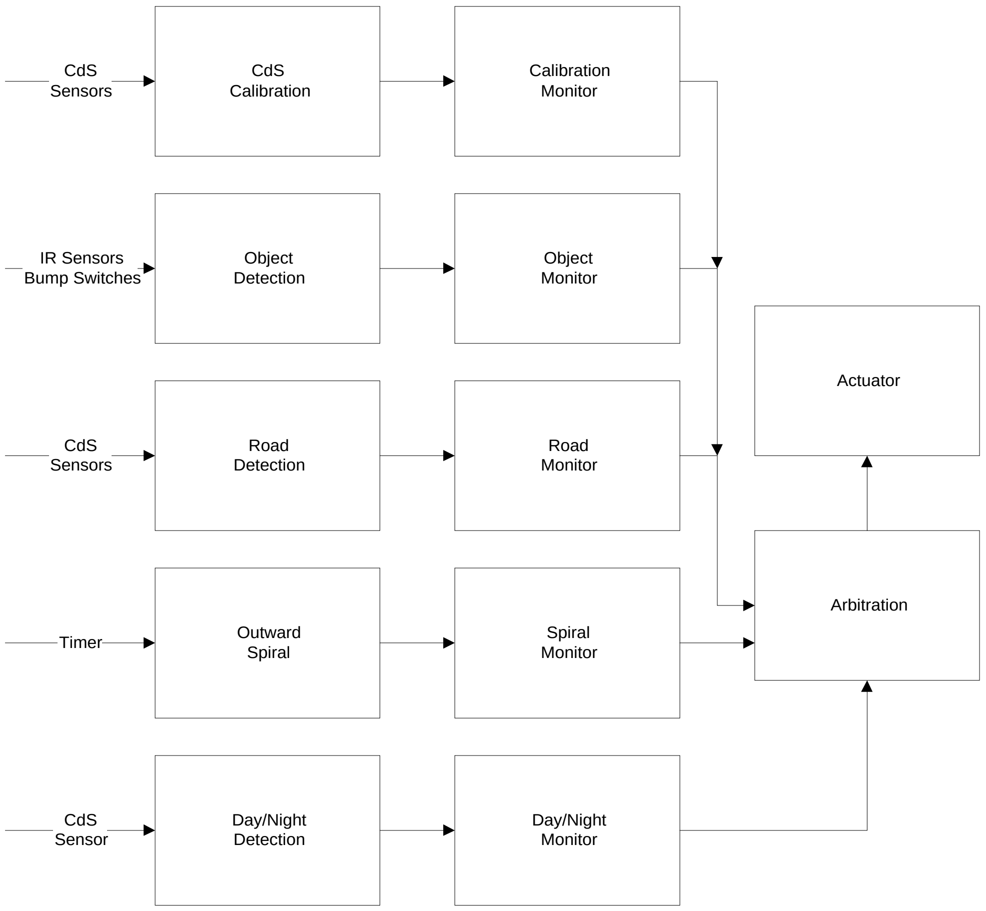


Figure 1: Behavior Signal Flow.

BEHAVIOR RELATIONSHIPS

Initial self-calibration is performed to orient Autonomobile to its current environment. The object avoidance routine works in conjunction with the road follow behavior, driving Autonomobile within the confines of a road lane when no obstacles are present. The visibility behavior works much like a speed control behavior. Depending on the light level, the relative speed of the car is varied to assist in object avoidance and road following. For more detailed discussion see the Behavior Section page 11.

MOBILE PLATFORM

CENTRAL PROCESSING UNIT

The Motorola MC68HC11 Evaluation Board (EVBU) mated with the ME11 Expansion board is responsible for Autonomobile's actions. The ME11 daughter-board [3] provides additional memory, motor and sensor control when integrated with the EVBU. Interactive C (IC), used in the MIT 6.270 notes [2], is used to program Autonomobile's behaviors. IC manages the multitasking environment necessary for multiple behaviors to coexist. Additionally, it includes routines that drive the expansion circuitry.

CAR BASE

Autonomobile uses a prefabricated Talrik wood base. This simple round platform has holes for switches and motors, EVBU mounting, and wheel wells. Autonomobile is essentially an abstract car. Therefore, a traditional auto body style does not prove advantageous.

All sensors will be located on the mobile platform. Object avoidance sensors are attached to the platform using Velcro in a variety of places. The Road Follow CdS sensors are mounted under the front fender of the robot.

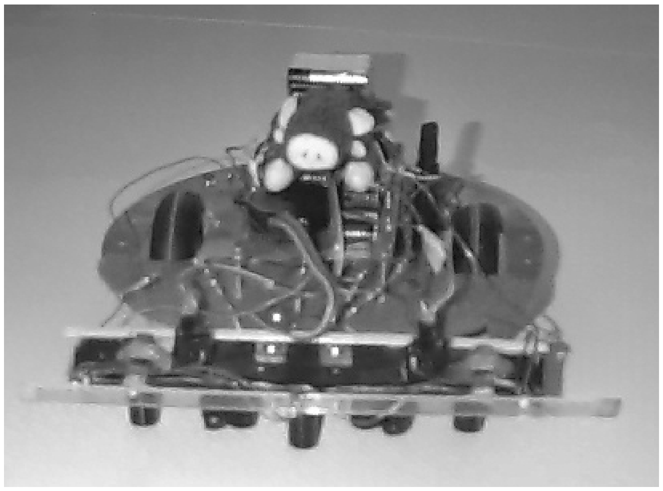


Figure 2: Front View.

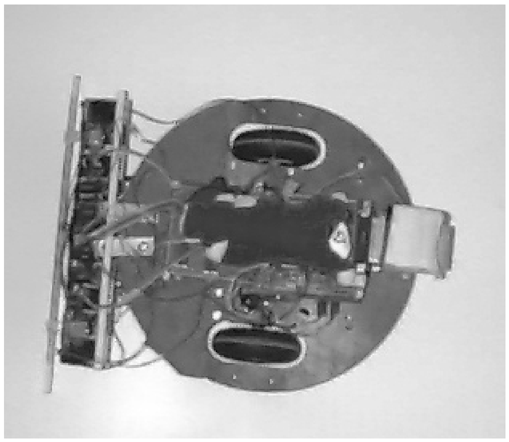


Figure 3: Top View.

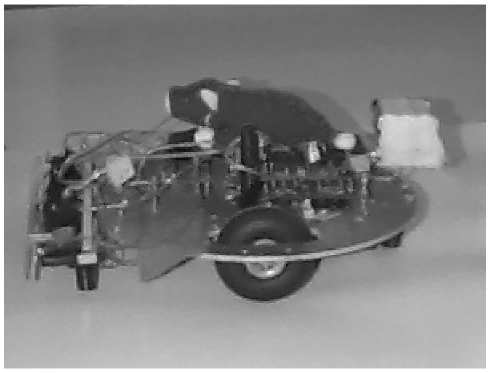


Figure 4: Side View.

Figures 2,3, and 4 are actual images of Autonomobile. The base wood platform of the Talrik was modified to accommodate the front road find/follow sensor array.

SENSOR SUITE

The sensor suite contains three different types of sensors: bump, IR, and light sensors. These sensory arrays provide Autonomobile with its environmental inputs. Autonomobile's need for ten sensory inputs forced an expansion of the analog port. To do this two integrated circuit chips, a latch [D1] and analog mux [D2], were wire-wrapped to the EVBU. Figure 5 shows the necessary connections between the EVBU, 573 latch, and 4051 analog mux to add eight additional analog inputs.

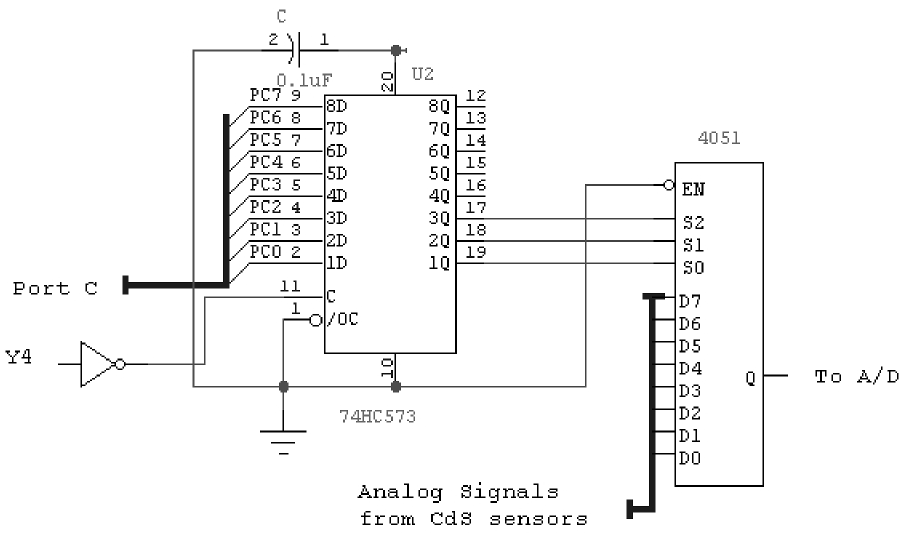


Figure 5: Analog Expansion Circuit.

BUMP SENSORS

"Fenders" for the car are implemented through a simple micro-switch network. On the front fender are three microswitches that are connected to an analog input port. The pressing of the bumper will trigger a collision avoidance behavior. The option to add a similar bumper to the rear is left for future work. The actual front bumper is shown in Figure 6.

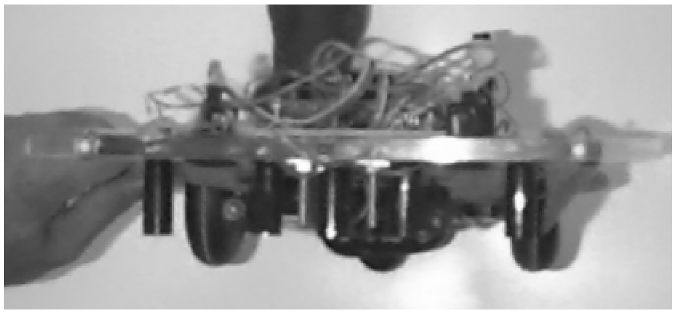


Figure 6: Front Bumper.

CADMIUM SULFIDE (CDS) CELLS

The road follow and visibility sensors are made by collimating the CdS cells. A single upright collimated CdS cell provides visibility information, while six CdS cells mounted downwards in the configuration shown in Figure 7 provide road following input.

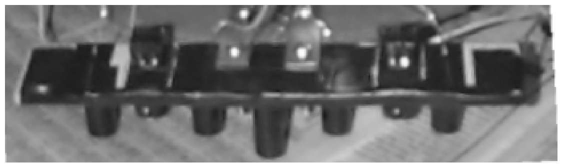


Figure 7: Road Follow

SHARP IR DETECTORS

Sharp 40KHz IR Receivers as shown in Figure 7, meet the first requirement of any autonomous mobile agent--object avoidance. They receive 40KHz modulated IR light reflected from objects in the area. Autonomobile supports two IR sensors in a cross-eyed configuration for object avoidance.

BEHAVIORS

Autonomobile has seven main behaviors. These behaviors make Autonomobile autonomous. The entire "C" source code to realize the following behaviors can be found in Appendix A. It should be noted that all of the behaviors are based on dynamic coding strategies with no discrete thresholds related to sensor characteristics.

CALIBRATION

Autonomobile's main goal is to follow a road. In order to achieve this its road following sensors must be attune with its environment. A thirty second calibration sequence is performed upon Autonomobile's startup. Once the data acquisition is completed an average is computed to characterize each CdS cell. In addition, the maximum CdS cell average is normalized against in order to remove any sensory bias, which may cause a behavior to function incorrectly.

OUTWARD SPIRAL

This behavior is used to cover a large amount of surface area efficiently. During the calibration phase of Autonomobile's CdS sensors, the Outward Spiral behavior serves as a method of data acquisition. The spiraling path is achieved by keeping the left motor at a fixed 100% while incrementing the right motor speed from 10% to 75%.

OBJECT AVOIDANCE

Traditional object avoidance uses discrete values, however, Autonomobile's avoidance routine is based on dynamic scheme similar to DYNAM11. A cross-eyed IR configuration mounted on the front fender, Figure 7, provides sensory input into a second order differential equation. This equation, based on current and old sensor readings as well as old motor speeds, dynamically sets new motor values for avoiding objects.

Collision Avoidance and Detection

In the event the Object Avoidance behavior fails to recognize an object the front bumper will be pressed and trigger this behavior. In doing so, Autonomobile will reverse for a second and then rotate a few degrees and drive forward again. This behavior is typically used in conjunction with the Object Avoidance behavior.

Road Following

A complex CdS sensor array, Figure 6, is responsible for providing the necessary data to Autonomobile's brain for its road following routine. The algorithm is similar to Critter's, which simply compares left, right, and center CdS readings to determine whether to turn right, left or stay straight. This behavior is heavily coupled with the collision avoidance and detection behavior.

Visibility

Visibility is important to driving. A topside surface mounted CdS cell provides the necessary visibility reading. Autonomobile will drive at different speeds according to the ambient light. The relationship of light directly proportional to speed was derived from the nature of the CdS cell. Therefore, as the light level decreases so does the speed.

EXPERIMENTAL LAYOUT AND RESULTS

Testing the robot throughout its stages of evolution has proved to be essential. Modular coding practices and implementation of behaviors provided the foundation for the integration multiple behaviors. Behaviors were coded and tested in a stand-alone environment. Each singular behavior was tested thoroughly before integration into the larger behavior set. This made for easier debugging. Also, behaviors implemented as modules can be turned off and on easily for the dynamic process scheduling implemented with Autonomobile's arbitrator.

BEHAVIORAL TESTING

All behaviors are have been demonstrated to cooperate with each other. Autonomobile was able to calibrate itself in day and night settings, spiral outward to cover a large surface area, avoid obstacles and collisions, find the road, follow the road, and react to varying lumination levels during D-Day.

CURRENT STATUS

Calibration, outward spiral, object avoidance, collision detection and avoidance, and visibility are all running smoothly and in concert. Road find and follow algorithms are working crudely, and need to be revised to improve robustness.

CONCLUSION

The past semester has taught me lessons about basic robot construction and programming, and more importantly--patience. A robot's behaviors depend directly on the capabilities of perceiving its environment. Input sensors perform differently than expected in most every instance, resulting is a plethora of design iterations. Robots are machines without programmed behaviors. Based on my experience, more time should be spent on learning coding practices before platform construction to ensure better functionality in autonomous agents.

REFERENCES

[1] Elephants Don't Play Chess. Brooks, Rodney. From Designing Autonomous Agents, ed. by Pattie Maes. The MIT Press, c. 1990.

[2] Fred Martin, The 6.270 Robot Builders' Guide. MIT Media Lab, Cambridge, MA. 1992.

[3] Doty, Keith L. et. al. EVBU Expansion Schematic. Machine Intelligence Laboratory. University of Florida.

[4] Doty, Keith L. Class lectures in EEL 5666. Intelligent Machines Design Laboratory. University of Florida, Spring 1996.

REFERENCED DATA SHEETS

[D1] Motorola. Specification sheet for the 573 integrated circuit.

[D2] Motorola. Specification sheet for the 4051 integrated circuit.

APPENDIX A: BEHAVIOR SOURCE CODE

```
/*  Autonomobile is an autonomous agent who is designed to model a car.
    His currently implemented behaviors are outward spiral, object avoidance,
    and visibility.

    Written by :  Kalpesh Keshavji Gala
           on :  April 23, 1997
           for:  EEL 5666

    This program is used to control Autonomobile.                           */

/***************************************************************************/

/*  Constants used throughout the program.                                  */

int TRUE  = 1;                      /* Boolean constants */
int FALSE = 0;

int ir_address = 0x7000;
int cds_address = 0x6000;

float PASS_SPEED  = 1.0;            /* Reserved for Passing */

/***************************************************************************/

/*  Variables and constants related to the Obstacle Avoidance behavior.     */

float ca = 0.015;
float cb = 0.5;                     /* Constants */
float ce = 50.0;

int lo = 0;                         /* Temporary Variables */
int ro = 0;

int left_ir;                        /* Left IR Sensor value */
int right_ir;                       /* Right IR Sensor value */

float abs_right;                    /* Absolute valued speeds  */
float abs_left;

float ld;                           /* Left IR differential */
float rd;                           /* Right IR differential */
float abs_ld;
float abs_rd;                       /* Absolute valued differentials  */

float oa_left;                      /* Object Avoidance motor speeds */
float oa_right;
int oa_priority;                    /* Object Avoidance priority */

/***************************************************************************/

/*  Variables and constants related to the Visibility behavior.             */

int initial_bump_sensor;            /* Initial bumper reading */
int bump_sensor;                    /* Bump Detection Sensor value */
float bd_left;                      /* Bump Detection motor speeds */
float bd_right;
int bd_priority;                    /* Bump Detection priority */

/***************************************************************************/

/*  Variables and constants related to the Visibility behavior.             */

int vis_cds;                        /* CdS Sensor value */
int initial_vis_cds;                /* Initial CdS reading */
float motor_speed;                  /* Dynamic speed factor */

/***************************************************************************/
```

```
/*  Variables related to motor control.                                    */

int LEFT  = 0;                       /* Motor constants */
int RIGHT = 1;
int speed_const;

float direction_left;                /* Raw left motor value */
float direction_right;               /* Raw right motor value */

/***************************************************************************/

/*  Variables for calibrating CdS cells.                                   */

int done;
int delay_time;

int number_of_samples;

int r_int_cds_sum;
int r_road_cds_sum;
int edge_cds_sum;
int c_road_cds_sum;
int l_road_cds_sum;
int l_int_cds_sum;

/***************************************************************************/

/*  Variables and constants related to the Outward Spiral behavior.        */

float os_left;                       /* Outward Spiral motor speeds */
float os_right;
int os_priority;                     /* Outward Spiral priority */

/***************************************************************************/
/*  Variables and constants related to the Road Follow behavior.           */

int initial_edge_cds;
int initial_left_road_cds;
int initial_center_road_cds;         /* Initial CdS readings */
int initial_right_road_cds;
int initial_left_int_cds;
int initial_right_int_cds;

int edge_cds;                        /* Edge of World CdS Sensors */
int left_road_cds;
int center_road_cds;                 /* Road Following CdS Sensors */
int old_center_cds;
int right_road_cds;
int left_int_cds;                    /* Intersection Detection CdS Sensors */
int right_int_cds;
int c1, c2, c3;
int rco, lco, cco;
int rcd = 0;
int lcd = 0;
int ccd = 0;

float rf_speed;                      /* Road Following speed */
float rf_left;                       /* Road Follow motor speeds */
float rf_right;
int rf_priority;                     /* Road Follow priority */

/***************************************************************************/

/*  Variables for process identification necessary for dynamic spawning and
    killing.                                                               */

int arbitrate_pid;                   /* Arbitrator process id */
int bd_pid;                          /* Bump Detection process id */
int ca_pid;                          /* Calibration Arbitrator process id */
int cc_pid;                          /* Calibrate CdS process id */
int delay_pid;                      /* Delay process id */
int ep_pid;                         /* End Process process id */
int motors_pid;                     /* Motors process id */
int oa_pid;                         /* Object Avoidnace process id */
int os_pid;                         /* Outward Spiral process id */
int rf_pid;                         /* Road Following process id */
int rfa_pid;                        /* Road Following Arbitrate process id */
int rfb_pid;
int s1_pid;                         /* Sensor1 process id */
int s2_pid;                         /* Sensor2 process id */
int vis_pid;                        /* Visibility process id */
int bb_pid;                         /* Backup Beep process id */
/***************************************************************************/
/***************************************************************************/
/*  This is the body --"driver"-- of the main program.                     */

void main()

{
  beep();
  initialize_variables();
  wait(1000);
  s1_pid = start_process(sensor1());
  s2_pid = start_process(sensor2());
  oa_pid = start_process(object_avoidance());
  os_pid = start_process(outward_spiral());
  cc_pid = start_process(calibrate_cds());
  bd_pid = start_process(bump_detection());
  bb_pid = start_process(backup_beep());
  motors_pid = start_process(motors());
  vis_pid = start_process(visibility());
  delay_pid = start_process(delay());
  ca_pid = start_process (calibration_arbitrate());
  ep_pid = start_process (end_procs());
}

/*  End of function main.                                                  */
/***************************************************************************/

/*  This function is used to produce a tone everytime the robot backsup.    */

void backup_beep()

{
  while(1)
    {
      if ((direction_left < 0.0) && (direction_right < 0.0))
        {
          tone(1000.0, 0.4);
          wait(300);
          defer();
        }
    }
}

/*  This is the end of function backup_beep.                               */

/***************************************************************************/

/*  This function is used to change from Calibration mode to Road Find mode.*/

void end_procs()

{
  while (1)
    {
      if (done == 1)
        {
          compute_average_cds();
          direction_left = 0.0;
          direction_right = 0.0;
          kill_process(delay_pid);
          kill_process(ca_pid);
          kill_process(cc_pid);
          kill_process(s2_pid);
          beep();
          wait(3000);
          beep();
          bd_priority = 0;             /* Return priorities to normal */
          oa_priority = 0;
          rf_priority = 20;
          os_priority = 0;
          rfa_pid = start_process(road_find_arbitrate());
          rfb_pid = start_process(road_find());
          rf_pid = start_process(road_follow());
          done = 0;
        }
      defer();
    }
  kill_process(ep_pid);
}

/*  This is the end of function end_procs. */

/***************************************************************************/

/*  This function handles a bump.  If the front fender is pressed
    Autonomobile will backup, turn around, and go off in another direction. */

void bump_detection()

{
  bd_priority = 0;
  while (done == 0)
    {
      if (bump_sensor > 100)
        {
          bd_priority = 100;          /* Give highest priority */
          bd_left = -100.0;
          bd_right = -100.0;         /* Backup */
          wait(500);
          bd_left = -100.0;
          bd_right = 100.0;          /* Turn around */
          wait(500);
          oa_priority = 10;
          bd_priority = 0;
        }
      defer();
    }
}

/*  End of function bump_detection.                                         */

/***************************************************************************/

/*  This function arbitrates between initial behaviors while the robot
    calibrates its CdS cells.                                               */

void calibration_arbitrate()

{
  while (1)
    {
      if ((bd_priority > oa_priority) && (bd_priority > os_priority))
        {
          direction_left = bd_left;
          direction_right = bd_right;
        }
      else
        if ((oa_priority > bd_priority) && (oa_priority > os_priority))
          {
            direction_left = oa_left;
            direction_right = oa_right;
          }
        else
          {
            direction_left = os_left;
            direction_right = os_right;
          }
      if (done == 1)
        {
          break;
        }
      defer();
    }
}

/*  End of function calibration_arbitrate.                              */

/***********************************************************************/

/*  This function is used to calibrate the CdS cells.                   */

void calibrate_cds()

{
  while (1)
    {
      poke(cds_address, 0x00);
      r_int_cds_sum = r_int_cds_sum + analog(0);
      poke(cds_address, 0x01);
      r_road_cds_sum = r_road_cds_sum + analog(0);
      poke(cds_address, 0x02);
      edge_cds_sum = edge_cds_sum + analog(0);
      poke(cds_address, 0x03);
      c_road_cds_sum = c_road_cds_sum + analog(0);
      poke(cds_address, 0x04);
      l_road_cds_sum = l_road_cds_sum + analog(0);
      poke(cds_address, 0x05);
      l_int_cds_sum = l_int_cds_sum + analog(0);
      number_of_samples = number_of_samples + 1;
      defer();
    }
}

/*  End of function calibrate.                                          */

/***********************************************************************/

/*  This function is used to compute the average of the */

void compute_average_cds()

{
  int max;

  initial_right_road_cds = (int)(r_road_cds_sum / number_of_samples);
  initial_center_road_cds = (int)(c_road_cds_sum / number_of_samples);
  initial_left_road_cds = (int)(l_road_cds_sum / number_of_samples);
  initial_right_int_cds = (int)(r_int_cds_sum / number_of_samples);
  initial_left_int_cds = (int)(l_int_cds_sum / number_of_samples);
  initial_edge_cds = (int)(edge_cds_sum / number_of_samples);

  max = initial_right_road_cds;
  if (initial_left_road_cds > initial_center_road_cds && initial_left_road_cds >
initial_right_road_cds)
    max = initial_left_road_cds;
  if (initial_center_road_cds > initial_left_road_cds && initial_center_road_cds >
initial_right_road_cds)
    max = initial_center_road_cds;

  c1 = max - initial_right_road_cds;
  c2 = max - initial_center_road_cds;
  c3 = max - initial_left_road_cds;

  rco = initial_right_road_cds;
  cco = initial_center_road_cds;
  lco = initial_left_road_cds;
}

/*  End of function compute_average_cds.                                 */

/************************************************************************/

/*  This function is used to delay for approximately thirty seconds.        */

void delay()

{
  while (1)
    {
      wait(delay_time);           /* Delay for delay_time */
      done = 1;
      delay_time = 0;
      defer();
    }
}

/*  This is the end of function delay.                                   */

/************************************************************************/

/*  This function is used to initialize all global varibles.               */

void initialize_variables()

{
  ir_address = 0x7000;
  cds_address = 0x6000;
  direction_left = 0.0;
  direction_right = 0.0;

  bd_right = 0.0;
  bd_left = 0.0;
  bd_priority = 0;

  oa_right = 80.0;
  oa_left = 80.0;
  oa_priority = 0;

  os_right = 0.0;
  os_left = 0.0;
  os_priority = 5;

  speed_const = 1;

  rf_right = 0.0;
  rf_left = 0.0;
  rf_speed = 30.0;
  rf_priority = 0;

  initial_vis_cds = analog(1);
  initial_bump_sensor = analog(4);
  initial_right_int_cds = 0;
  initial_right_road_cds = 0;
  initial_edge_cds = 0;
  initial_center_road_cds = 0;
  initial_left_road_cds = 0;
  initial_left_int_cds = 0;

  done = 0;
  delay_time = 30000;
  number_of_samples = 0;
  r_int_cds_sum = 0;
  r_road_cds_sum = 0;
  edge_cds_sum = 0;
  c_road_cds_sum = 0;
  l_road_cds_sum = 0;
  l_int_cds_sum = 0;

  poke(ir_address, 0xff);            /* Enable all IR LEDs. */
}

/*  End of function intialize intialize_variables.                        */

/*************************************************************************/

/*  This function is used to control the motors.  Values set for motor speed
    and direction are determined in various behaviors.  The motor ouput is
    smoothed via an averaging function.                                   */

void motors()

{
  while (1)
    {
      if (motor_speed > 100.0)
       motor_speed = 100.0;
      if (motor_speed < 0.0)
       motor_speed = 0.0;

      direction_left = (direction_left - motor_speed) / (float)speed_const;
      direction_right = (direction_right - motor_speed) / (float)speed_const;

      if (direction_left > 100.)
       direction_left = 100.0;
      if (direction_left < -100.0)
       direction_left = -100.0;

      if (direction_right > 100.)
       direction_right = 100.0;
      if (direction_right < -100.0)
       direction_right = -100.0;

      motor(LEFT,direction_left);
      motor(RIGHT,direction_right);
      defer();
    }
}

/*  End of function motors.                                               */

/*************************************************************************/

/*  This function is used to avoid objects.  Sensory input is collected by
    IR sensors.  Based upon IR values Autonomobile will either turn left,
    right, go forward or backwards (if cornered.)                         */

void object_avoidance()

{
  while (1)
    {
      if ((left_ir >= 115) || (right_ir >= 115))
       oa_priority = 10;

      rd = (float)(ro - right_ir);   /* Sensor reading differential */
      ld = (float)(lo - left_ir);

      abs_ld = absf(ld);             /* Absolute value of */
      abs_rd = absf(rd);             /* IR differential readings */

      abs_left = absf(oa_left);      /* Absolute value of motor speeds */
      abs_right = absf(oa_right);
      /*control equation for collision avoidance*/
      oa_left = oa_left + ca*abs_left + cb*abs_ld + ce*ld;
      oa_right = oa_right + ca*abs_right - cb*abs_rd + ce*rd;

      if (oa_left > 100.0)
       oa_left = 100.0;
      if (oa_left < -100.0)
       oa_left = -100.0;

      if (oa_right > 100.0)
       oa_right = 100.0;
      if (oa_right < -100.0)
       oa_right = -100.0;

      lo = left_ir;
      ro = right_ir;
      defer();
    }
}

/*  End of function object_avoidance.                                      */

/***********************************************************************/

/*  This function drives Autonomobile in an outward spiral searching for the
    road.                                                                  */

void outward_spiral ()

{
  while (1)
    {
      if (os_priority == 5)
       {
         os_right = 20.0;            /* Reset right motor speed */
         os_priority = 7;
       }
      else
       {
         if (os_right < 75.0)       /* Increment right motor, max 75% */
           {
             os_right = os_right + 1.0;
             wait (1000);
           }
       }
      os_left = 100.0;               /* Constant for spiral effect */
      os_priority = 5;
      defer();
    }
}

/*  End of function outward_spiral.                                        */

/***********************************************************************/

/*  This function is the heart of the robot.  Depending on sensory input the
    logic will decide what to do.                                          */

void road_find_arbitrate()

{
  while (1)
    {
      if ((bd_priority > oa_priority) && (bd_priority > os_priority) && (bd_priority >
rf_priority))
       {
         direction_left = bd_left;
         direction_right = bd_right;
       }
      else
```

```
        if ((oa_priority < os_priority) && (rf_priority < os_priority))
          {
            direction_left = os_left;
            direction_right = os_right;
          }
        else
          if ((os_priority < oa_priority) && (rf_priority < oa_priority))
            {
              direction_left = oa_left;
              direction_right = oa_right;
            }
          else
            {
              direction_left = rf_left;
              direction_right = rf_right;
            }
      defer();
    }
}

/*  End of function road_find_arbitrate.                                  */

/*************************************************************************/


void road_find()

{
  bd_pid = start_process(bump_detection());

  beep();
  wait(1000);
  beep();
  wait(1000);
  beep();
  wait(10000);
  rf_priority = 25;
  beep();
  kill_process(rfb_pid);
}

/*************************************************************************/

/*  This function is used to follow a road.  Sensory input is collected by
    CdS Cells.  Based upon CdS values Autonomobile will follow a black line
    painted on a road.                                                    */

void road_follow()

{
  int l;
  int c;
  int r;

  while (1)
    {
       poke(cds_address, 0x01);
       r = analog(0) + c1;
       poke(cds_address, 0x03);
       c = analog(0) + c2;
       poke(cds_address, 0x04);
       l = analog(0) + c3;

       if (l > c && l > r)
         {
           rf_left = 0.0;             /*  Turn Left.  */
           rf_right = rf_speed;
         }
       else
         if (r > c && r > l)
           {
            rf_left = rf_speed;         /*  Turn Right.  */
            rf_right = 0.0;
          }
        else
          if (c > r && c > l)
            {
              rf_left = rf_speed;
              rf_right = rf_speed;        /*  Go Forward.  */
            }
       /*         else
            {
              rf_left = 0.0;
              rf_right = 0.0;
            } */
    }
  defer();
}

/*  End of function road_follow.                                          */

/************************************************************************/

/*  This function is responsible for getting non-CdS sensor inputs and
    storing them into the appropriate variables.                          */

void sensor1()

{
  int bump_diff;

  while (1)
    {
      vis_cds = analog(1);
      left_ir = analog(2);
      right_ir = analog(3);
      bump_diff = analog(4) - initial_bump_sensor;
      bump_sensor = (int)absf((float)bump_diff);
      defer();
    }
}

/*  End of function sensor1.                                              */

/************************************************************************/

/*  This function is responsible for getting CdS sensor inputs and store them
    into the appropriate variables.                                      */

void sensor2()

{
  while (1)
    {
      poke(cds_address, 0x00);
      right_int_cds = analog(0) - initial_right_int_cds;
      poke(cds_address, 0x01);
      right_road_cds = analog(0) - initial_right_road_cds;
      poke(cds_address, 0x02);
      edge_cds = analog(0) - initial_edge_cds;
      poke(cds_address, 0x03);
      center_road_cds = analog(0) - initial_center_road_cds;
      poke(cds_address, 0x04);
      left_road_cds = analog(0) - initial_left_road_cds;
      poke(cds_address, 0x05);
      left_int_cds = analog(0) - initial_left_int_cds;
      defer();
    }
}

/*  End of function sensor2.                                              */
```

```
/***************************************************************************/

/*  This function is used to determine visibility.  Sensory input is
    collected by a CdS Cell.  Based upon CdS value Autonomobile will adjust
    his speed for adverse conditions.                                     */

void visibility()

{
  while (1)                           /* Visibility differential */
    {
      motor_speed = (float)vis_cds - (float)initial_vis_cds;
      defer();
    }
}

/*  End of function visibility.                                           */

/***************************************************************************/

/*  This function waits for the inputted number of milliseconds.          */

void wait(int m_second)

{
  long stop_time;

  stop_time = mseconds() + (long)m_second;
  while (stop_time > mseconds())
    defer();
}

/*  End of function wait.                                                 */

/***************************************************************************/

/*  This function returns the absolute value of an inputed float.         */

float absf(float number)

{
  if (number < 0.0)
    number = number * -1.0;          /* Negate number if negative */
  return (number);
}

/*  End of function absf.                                                 */

/***************************************************************************/

/*  This function returns the absolute value of an inputed integer.       */

int absi(int number)

{
  if (number < 0)
    number = number * -1;            /* Negate number if negative */
  return (number);
}

/*  End of function absi.                                                 */

/***************************************************************************/
```