Dino: A Prey-Based Mobile Robot

Final Report

Designer: Timothy J. Wiant

University of Florida
EEL5666 - Intelligent Machines Design Laboratory
Spring 1997

Instructor: Keith L. Doty
T.A.'s: Scott Jantz
Charles Barker
Kevin Harrelson

April 28, 1997

Table of Contents

Abstract

This report is based on the construction of a mobile robot built for EEL 5666: Intelligent Machines Design Laboratory. The robot was based upon a prey for a predator/prey combination. So, the robot is supposed to be able to detect and flee the predator. The construction of the robot took place in phases. The first phase was the development of the platform. The second phase was the development of the sensor suite which the robot will use to gather information about its environment. The last phase was concerned with the development of a set of well integrated behavior algorithms. When completed, the robot should be able to gather data from the environment, process this data, and determine a course of action based upon this data.

Executive Summary

The robot designed in this project is a prey built for a predator/prey combination. The main intent of the robot was for it to be able to hide from a predator and detect when predators are near through the use of sound.

The robot was built on a circular, two-wheeled platform for speed and ease of maneuverability. Since the robot is circular, it does not have any problems with turning on its center axis to avoid obstacles.

The sensor suite consists of 10 microswitches, 4 Sharp IR detectors, 2 Cadmium Sulfide cells, and 2 microphone-based sound detection circuits. The microswitches allow the robot to determine when it has come in contact with an object. The IR detectors allow the robot to try to avoid obstacles by detecting them before the robot bumps into them. The Cadmium Sulfide cells are used in light detection.

The source code for the robot was written using IC. The current behaviors integrated are bump and object detection, light detection, sound detection, IR threshold modification, and trap avoidance.

Introduction

The basis for this project was a prey robot. It was determined that the robot would be based upon typical prey characteristics. It's main defense against predators is its ability to detect sound and avoid the object making the noise. This means the robot must have some way of determining when a sound is produced and the direction of the source of the sound. To make it more interesting, the robot is intended to be able to detect sound while moving so it can flee from predators constantly until it is safe. The robot also has the ability to detect light intensity so it can hide in shady areas. This allows the robot to find a suitable place to hide.

Integrated System

The hardware components of the robot are integrated as shown in Figure 1. The inputs to the processor are provided by the sensors. These sensory inputs are then used to determine which direction the robot should move. Each sensory input has its own control process which feeds directions to an arbitrator. This arbitrator takes all the inputs from the sensor control processes and computes values to feed to a motor control system, that controls the robot's movement.

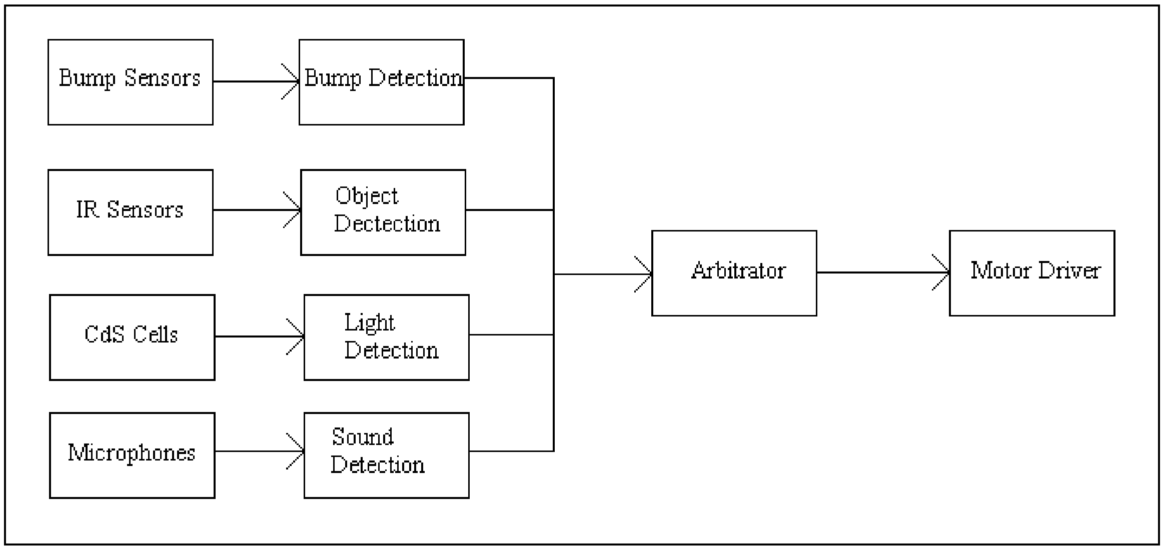


Figure 1: Integrated System Flow Chart

Mobile Platform

<u>Microprocessor</u>

The microprocessor used to control the robot's behaviors is the Motorola MC68HC11E9. This microprocessor was used primarily because it is a robust processor, giving the programmer many different options, for its size and cost. The MC68HC11E9 is relatively inexpensive and easily obtainable which made it a desirable choice for this project. Experience working with this specific microprocessor made it easier to adapt the processor to perform certain tasks, i.e. multiplexing the A/D pins, with little effort.

The memory of the microprocessor was expanded using the ME11 memory expansion board from Novasoft. This expansion board automatically adds 32K RAM, 2 motor drivers, 4 memory-mapped input enables, 4 memory-mapped output enables, and a memory-mapped digital output port.

Chassis

The chassis chosen as the basis for this robot is the TALRIK base obtainable from Novasoft. This chassis is a 1 foot diameter board with two holes cut for wheels, a hole cut in the middle for feeding wires through, and mounting holes cut for the MC68HC11 board. This particular configuration was chosen because it is more forgiving in its mobility and allows the programmer to worry more about the behaviors than the details of maneuvering in an environment.

Motor Actuation

The robot is actuated using two Futaba FP-S148 servos. These servos have been hacked to change them into DC gearhead motors. The gearing in the motors allows the robot to pull more weight, thereby eliminating some of the weight constraints instilled by no gear train. The motors are driven using pulse-width modulated signals that are produced by the microprocessor and a 293 motor driver chip.

Sensors

Bump Sensor

The bump sensor consists of 10 microswitches evenly spaced around the outside of the robot platform. These switches are wired together using the resistor ladder shown in Figure 2.

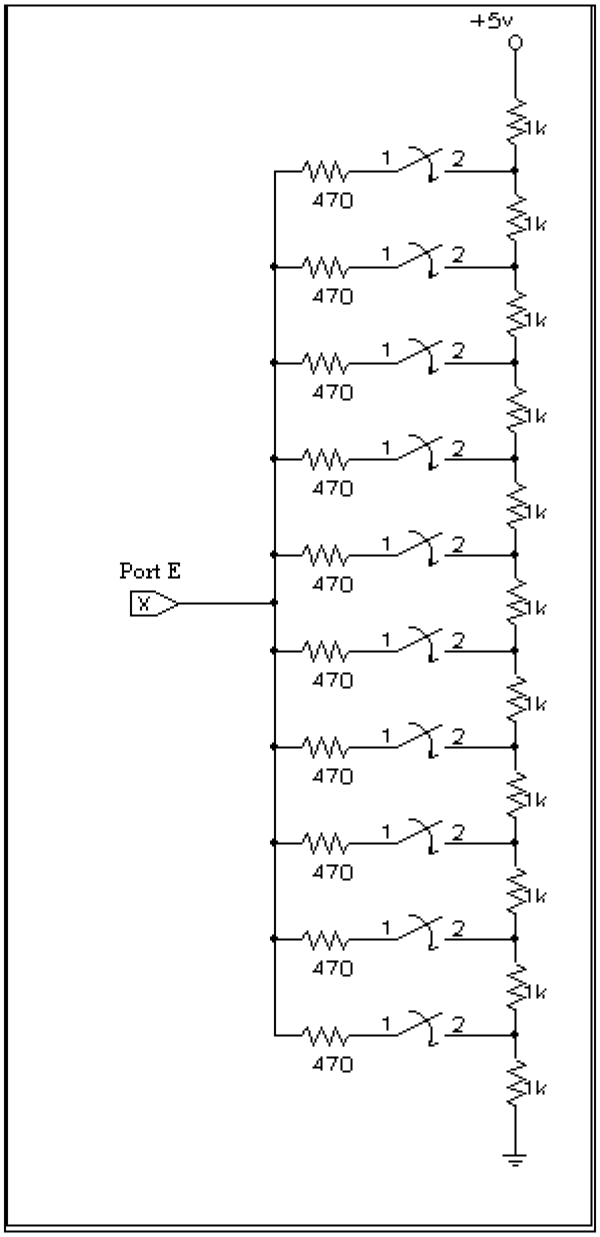


Figure 2: Bump Detection Circuitry

When any of the switches are pressed, the input voltage to the A/D port changes thereby allowing the robot to determine which switch was pressed and the location of the obstacle. This data can then be used to determine an appropriate course of action to maneuver around the obstacle.

Sharp IR Detectors

There are 4 Sharp IR Detectors located on the front of the robot as shown in Appendix A. The two detectors located towards the middle are arranged in a cross-eyed formation. This formation allows a wider range of object detection by giving the robot the ability to detect objects that are near the middle of the robot's field of vision as well as objects that are farther off to the sides. The IR detectors are mainly used for object detection and avoidance.

CdS Light Sensors

There are two Cadmium Sulfide cells located near the wheels of the robot. These cells are variable resistors whose resistance decreases with the amount of light. By wiring up the cells in series with another resistor (Figure 3), a voltage reading can be taken to determine the intensity of the ambient light. These cells are used to allow the robot to find and hide in shady areas.

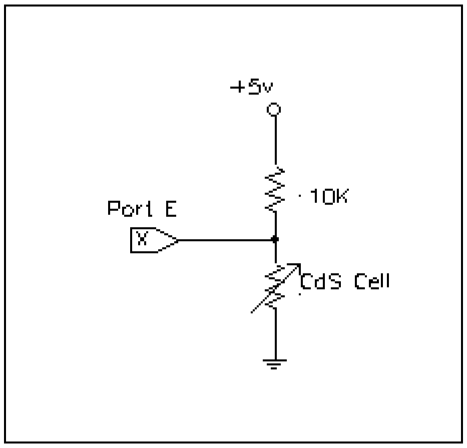


Figure 3: Light Detection Circuitry

Sound Detector

The sound detection circuitry of the robot consists of two differential amplifiers. The main layout of the differential amplifier circuitry is shown in Figure 4.

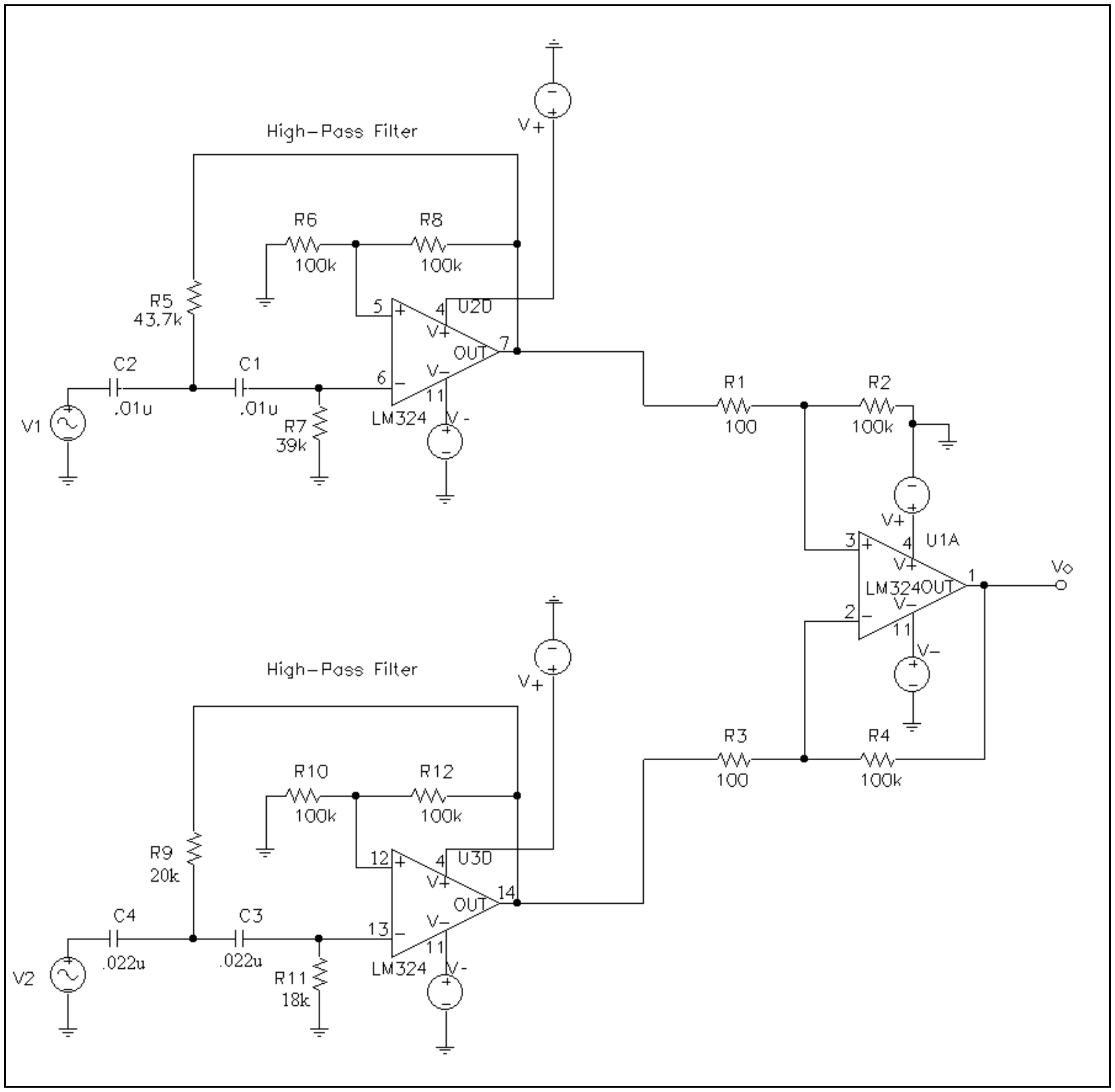


Figure 4: Differential Amplifier used for Sound Detection

Each differential amplifier gets its inputs from a microphone pointed out into the environment and a microphone attached directly to the side of the motor. The microphone inputs are passed through high-pass filters to eliminate the DC voltage necessary to carry the microphone signal. These filtered signals are then used as inputs to the differential amplifier. The differential amplifier takes the input from the surroundings of the robot, subtracts out the motor noise, amplifies the difference, and outputs the amplified signal. This circuit has a gain of approximately 2000 which is enough to make the signal visible (to the microprocessor) for relatively faint noises. The main problem with this circuit is the difficulty in determining where a sound is located since the amplifier is peaked easily by loud noises.

Behaviors

The behaviors for Dino were written in pieces. As new sensors were added to the robot, new sensor routines and behaviors were also added. The code was designed to be mostly modular which allows simple modification and testing without having to write entirely separate test programs. Instead, a portion of the program can be commented out, while a new routine or behavior is being tested.

The behavior arbitrator was written to take suggestions from the various sensor control routines to determine the direction in which the robot should move. This allows the arbitrator to combine the outputs of several sensor control routines and determine a course of action. In various routines, random variables are used to help eliminate the possibility of being trapped or going in circles.

Bump Detection

As with most robots, one of the first behaviors implemented was bump detection. Since IR has limited capabilities in detecting objects, like chair legs, windows, or black objects, bump detection is a necessity to prevent the robot from trapping itself against a wall.

The bump detection algorithm used computes the switch pressed and submits an appropriate suggestion to avoid the obstacle detected. When a bump is detected, the robot should move in the opposite direction, either forward or backward, from its original path. The robot should then turn according to which switch was pushed to try to avoid the obstacle. If the switch in the front of the robot is triggered, a random number is used to determine whether the robot should make a hard left or a hard right. This randomness helps keep the robot from getting trapped or going in circles.

Object Avoidance

The IR detectors are used to help the robot try to avoid bumping into objects. By taking readings off of each of the IR sensors and comparing them to thresholds and each other, it is possible to tell if there is an obstacle in the vicinity of the robot. The main problem with using IR for obstacle detection is its range and reliability.

IR Threshold Adjustment

After bumping into objects, the robot periodically takes a reading of the IR sensors and determines a new threshold for the IR sensors to be compared to. This allows the robot to adjust itself to its surroundings. So, instead of having to program a certain threshold value for each different environment, one threshold can be programmed and the variable can be adjusted as needed.

Trap Avoidance

A few lines of code were added to the arbitrator to help the robot avoid getting stuck in a corner. By adding a counter and a timer, if the robot starts rotating back and forth fairly often, it will eventually back out of the area, turn and then continue forward.

Hide in Shadows

The readings off of the Cadmium Sulfide cells are used to determine when the robot is in a shady area. If the robot runs under a shady spot with one side, it will rotate until the whole robot is under the shade. At that point, the robot will remain still and hide.

Run from Sound

If the robot detects sound, it will try to determine the direction from which the sound came and run the opposite direction. The design of the sound detection circuit allows the robot to detect sound while it is moving and determine an appropriate course to try to evade the source of the noise. The main problem with the algorithm used is that it cannot tell the difference between front and back, only left and right. Therefore, it is possible that the robot will head straight toward the source of the noise. However, the imbalance in the wheels makes the robot head to the left, so the robot will eventually get to a point where it detects that the sound is on one side and turn away from it.

Conclusion

The robot comes close to performing the actions that I had designed for it to do. Sound detection and direction was a little more difficult than originally anticipated, and, in fact, did not really work in the demonstration. The robot does detect sound relatively well, but the direction determination algorithm needs to be modified so that the robot will actually turn away from sound. The arbitration of the control processes combines the inputs efficiently and gives a realistic idea of which way the robot should head.

In the future, I hope to implement some more complicated algorithms on this robot. Besides modifying the sound detection process, I would also like to make the arbitrator more compact and neural net based. I also hope to implement more learning and self-adjustment algorithms to avoid continuously manipulating parameters and thresholds. This would allow the robot to be completely autonomous, without needing tampering to get it started.

Documentation

Jones, Joseph L. and Anita M. Flynn. Mobile Robots: Inspiration to Implementation. A. K. Peters, Wellesley: 1993.

Maes, Pattie. Designing Autonomous Agents. MIT Press/Elsevier Science Publishers: 1990.

Martin, Fred. The 6.270 LEGO Robot Design Competition Course Notes. Electrical Engineering and Computer Science Department, MIT: 1992.

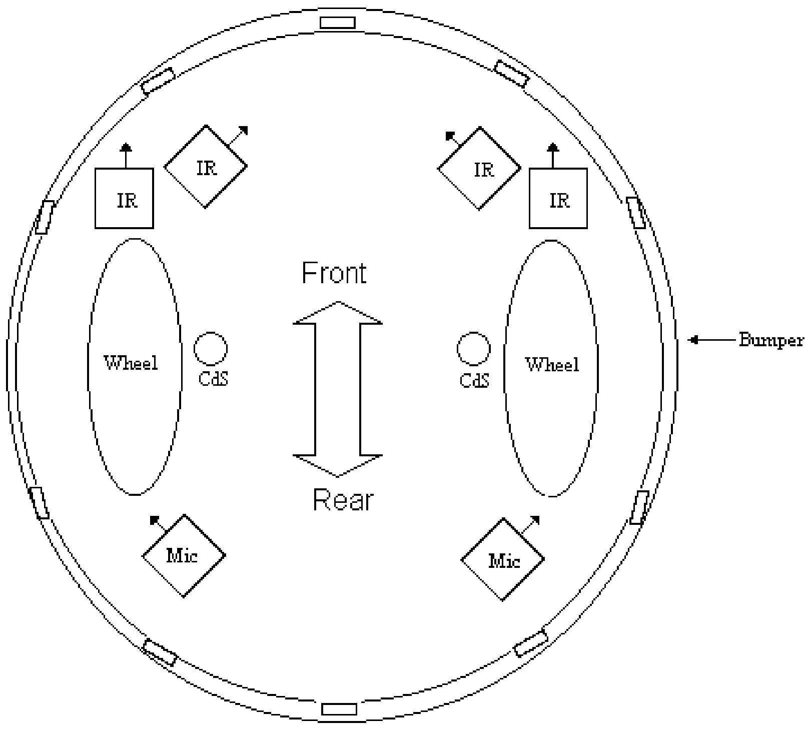
M68HC11 E Series Programming Reference Guide. Motorola: 1991.

M68HC11 Reference Manual. Motorola: 1991.

MC68HC11E9 Technical Data Manual. Motorola: 1991.

MC68HC11EVBU Universal Evaluation Board User’s Manual. Motorola: 1992.

Appendix A: Robot Layout



Appendix B: Source Code

```
/* GLOBALS */

/* Constants */
int left_motor = 0;
int right_motor = 1;
int left_ir_pin = 7;
int midleft_ir_pin = 6;
int midright_ir_pin = 5;
int right_ir_pin = 4;
int left_ir = 0;
int midleft_ir = 1;
int midright_ir = 2;
int right_ir = 3;
int bump_pin = 0;
int bump_lb_min = 20;
int bump_max = 255;
int bump_mid_value = 125;
int bump_count_mod = 0b1111111;
int left_cds_pin = 1;
int right_cds_pin = 1;
int left_sound_pin = 3;
int right_sound_pin = 2;
int right_sound_mult = 7;
int left_cds_amask = 0b10000000;  /* bit to be cleared to read CdS cells */
int left_cds_dmask = 0b00000100;  /* bit to be set to read left CdS cell */
int right_cds_dmask = 0b00001000; /* bit to be set to read right CdS cell */
int motor_damping_constant = 9;
int port_a = 0x1000;
int port_d = 0x1008;
int port_ddrd = 0x1009;
int port_pactl = 0x1026;
int ddrd_mux_mask = 0b00001100;
int pactl_mux_mask = 0b10000000;
int trapped = 10;
int trap_time = 10;

/* Sensory Registers */

int ir_readings[4];
int bump_reading = 0;
int left_light_value = 0, right_light_value = 0;
```

```
/* Bumper Variables */

int bump_detected = 0;                    /* bump sensor triggered flag */
int bump_count = 0;              /* counter to determine when to adjust ir */
int bump_value, bump_div;
int bump_motor_direction, bump_motor_factor, bump_priority;

/* IR Variables */

int ir_threshold = 110;         /* ir threshold value */
int in_thresh = 10, out_thresh = 10;   /* differential thresholds for inner/outer pairs */
int ir_motor_direction, ir_motor_factor, ir_priority;

/* Light Variables */
int light_threshold = 160;           /* CdS cell threshold value */
int light_motor_direction, light_motor_factor, light_priority;

/* Sound Variables */
int sound_left, sound_right;
int sound_timer = 0;
int run_time = 5;
int sound_left_threshold = 40;
int sound_right_threshold = 5;
int sound_detected = 0;        /* flag to determine if there was a sound in the environment
*/
int sound_motor_direction, sound_motor_factor, sound_priority;

/* Arbitrate Variables */
int motor_factor = 0, motor_direction = 0;
int current_left = 0, current_right = 0;           /* current motor values for robot */
int desired_left, desired_right;           /* arbitrator values */
int next_left, next_right;      /* temp storage for bump arbitration */
int motor_working = 0;                    /* enables (0) and disables (1) arbitrator */
int hide_value = 1;                 /* determines whether robot should run (0) or hide (1) */
int turn_timer = 0;
int last_factor = 0;
int turn_count = 0;


/* Function to tell a process to wait a certain amount of time */
void wait(int milli_seconds)
{
   long timer_a;
```

```
    timer_a = mseconds() + (long) milli_seconds;
    while(timer_a > mseconds()) {
            defer();
     }
}

/* Random number generator, returns a 0 or 1 */
int rand()
{
  return (peek(0x100f) & 1);
}

/* Returns sign of value */
int sign(int passed_int)
{
    if (passed_int > 0)
            return (1);
    else if (passed_int < 0)
            return (-1);
    else
            return (passed_int);
}

/* Returns the absolute value of the value */
int abs(int value)
{
    if (value < 0)
        return(- value);
    else
        return (value);
}

/* Adjust the threshold values used by the ir sensors */
void adjust_ir_thresh()
{
    int i;

    for (i = 0; i <= 3; i++) {
        if ((ir_readings[i] < ir_threshold) && (ir_threshold > 90))
                ir_threshold = ir_readings[i];

        if (ir_threshold < 90)
                ir_threshold = 90;
    }
}
```

```
/* Read sensor values into global variables at 10 Hz */
void sensor_module()
{
   while (1) {
      sound_left = analog(left_sound_pin);
      sound_right = analog(right_sound_pin);
      bump_reading = analog(bump_pin);
      bit_set(port_d, left_cds_dmask);
      msleep((long) 0.03);
         left_light_value = analog(left_cds_pin);
      bit_clear(port_d, left_cds_dmask);
      bit_set(port_d, right_cds_dmask);
      msleep((long) 0.03);
      right_light_value = analog(right_cds_pin);
      bit_clear(port_d, right_cds_dmask);
         ir_readings[left_ir] = analog(left_ir_pin);
         ir_readings[midleft_ir] = analog(midleft_ir_pin);
         ir_readings[midright_ir] = analog(midright_ir_pin);
         ir_readings[right_ir] = analog(right_ir_pin);
         wait(100);    /* Wait 100 msec */
   }
}

/* Collision detection using bumper */
void bump_sense()
{
   while(1) {
      bump_detected = 0;
      bump_value = 0;
         if (bump_reading >= bump_lb_min){
           bump_detected = 1;
           bump_count += 1;
           bump_value = bump_reading - bump_mid_value;
           if (bump_value < 0) {
          bump_motor_direction = 1;
          bump_motor_factor = -1*(((bump_value * 5)/bump_div) + 2);
           }
           else {
          bump_motor_direction = -1;
          bump_value = (bump_value*5/(bump_max - bump_mid_value)) + 1;
             if (bump_value == 3)
                  bump_motor_factor = rand()*4 - 2;
             else if (bump_value >= 4)
                  bump_motor_factor = bump_value - 6;
```

```
                else
                    bump_motor_factor = bump_value;
            }
            bump_priority = 5;
        }
        else {
            bump_priority = 1;
            bump_motor_direction = 1;
            bump_motor_factor = 0;
        }
        bump_count &= bump_count_mod;
        defer();
    }
}

/* Object avoidance using IR sensors */
void ir_sense()
{
   int out_dif;
   int in_dif;

   while(1) {
      if (((ir_readings[midleft_ir] > ir_threshold)&&
                 (ir_readings[midright_ir] > ir_threshold))||
                 ((ir_readings[left_ir] > ir_threshold)&&
                 (ir_readings[right_ir] > ir_threshold))) {
            ir_motor_factor = rand()*4 - 2;
            ir_motor_direction = -1;
            ir_priority = 5;
         }
         else {
            out_dif = ir_readings[right_ir] - ir_readings[left_ir];
            in_dif = ir_readings[midleft_ir] - ir_readings[midright_ir];
            if ((in_dif > in_thresh) || (in_dif < (-1*in_thresh))) {
             ir_motor_direction = -1;
             ir_motor_factor = sign(in_dif);
                 ir_priority = 2;
            }
            else if ((out_dif > out_thresh) || (out_dif < (-1*out_thresh))) {
             ir_motor_direction = -1;
             ir_motor_factor = sign(out_dif) * 2;
             ir_priority = 4;
            }
            else {
                 ir_motor_direction = 1;
                 ir_motor_factor = 0;
                 ir_priority = 1;
             }
         }

         defer();
     }
}

/* Look for hiding place using light intensity detection */
void light_sense()
{
     while(1) {
       light_motor_direction = 1;
         light_motor_factor = 0;

         if (left_light_value > light_threshold) {
                 light_motor_direction += 1;
                 light_motor_factor -= 2;
         }

         if (right_light_value > light_threshold) {
         light_motor_direction += (2 * (1 + light_motor_factor));
                 light_motor_factor += 2;
         }

       light_motor_direction = sign(light_motor_direction);

       if (hide_value == 1)
         light_priority = 3;
       else
         light_priority = 2;

         defer();
     }
}

/* Predator avoidance using sound detection */
void sound_sense()
{
         while (1) {
         if ((sound_left > sound_left_threshold)&&
             (sound_right > sound_right_threshold)) {
             sound_timer = (int) seconds() + run_time;
```

```
                hide_value = 0;
                sound_motor_direction = 1;
                if (sound_left > (sound_right * (right_sound_mult + rand())))
                    sound_motor_factor = 2;
                else
                    sound_motor_factor = -2;
                sound_priority = 6;
            }
            else if (sound_left > sound_left_threshold) {
                sound_timer = (int) seconds() + run_time;
                hide_value = 0;
                sound_motor_direction = 1;
                sound_motor_factor = 2;
                sound_priority = 6;
            }
            else if (sound_right > sound_right_threshold) {
                hide_value =0;
                sound_timer = (int) seconds() + run_time;
                sound_motor_direction = 1;
                sound_motor_factor = -2;
                sound_priority = 6;
            }
            else {
                sound_motor_direction = 1;
                sound_motor_factor = 0;
                sound_priority = 0;
                if (seconds() >= (float) sound_timer)
                    hide_value = 1;
            }

            defer();
            }

}

/* Decide what values to send to the motors */
void arbitrate()
{
    while (1) {
        if (motor_working == 0) {
            if (bump_detected == 1) {
                turn_count += 1;
                last_factor = sign(motor_factor);
                turn_timer = (int) seconds() + trap_time;
                desired_left = 100 * bump_motor_direction;
                desired_right = 100 * bump_motor_direction;
                next_left = (bump_motor_factor + sound_motor_factor * 2) * (-50);
                next_right = (bump_motor_factor + sound_motor_factor * 2) * 50;
                /* Backup */
                motor_control(1000);
                /* Adjust IR thresholds if necessary */
                if (bump_count == 1)
                    adjust_ir_thresh();
                desired_left = next_left;
                desired_right = next_right;
                /* Turn to avoid obstacle */
                motor_control(500);
            }
            else {
                /* Determine motor_direction and
                  motor_factor values to be used by arbitrator */
                if (hide_value == 1) {
                    if ((light_motor_direction == 0)&&(light_motor_factor == 0)) {
                       motor_factor = 0;
                       motor_direction = 0;
                    }
                    else {
                       motor_direction = sign(ir_motor_direction * ir_priority +
light_motor_direction * light_priority);
                       if (ir_motor_factor == 0)
                          motor_factor = light_motor_factor;
                       else
                          motor_factor = (light_motor_factor * light_priority + ir_motor_factor *
ir_priority)/
                        (light_priority + ir_priority);
                    }
                }
                else {
                    motor_direction = sign(ir_motor_direction * ir_priority
                       + sound_motor_direction * sound_priority);
                    motor_factor = (ir_motor_factor * ir_priority + sound_motor_factor *
sound_priority)/
                       (ir_priority + sound_priority);
                }

                if ((motor_factor != 0)&&(sign(motor_factor) != last_factor)) {
                    turn_count += 1;
                    last_factor = sign(motor_factor);
                    turn_timer = (int) seconds() + trap_time;
                }
```

```
/* Convert motor_direction and motor_factor to actual
   motor speed values */
if (motor_direction == 0){
     desired_left = 0;
     desired_right = 0;
     if ((int) seconds() > turn_timer) {
         last_factor = 0;
         turn_count = 0;
     }
}
else if (motor_factor < 0) {
     desired_right = 100 * motor_direction;
     desired_left = (100 + 50 * motor_factor) * motor_direction;
}
else if ((motor_factor == 0) && (motor_direction != 0)) {
     desired_left = 100;
     desired_right = 100;
     if ((int) seconds() > turn_timer) {
         last_factor = 0;
         turn_count = 0;
     }
}
else if (motor_factor > 0) {
     desired_left = 100 * motor_direction;
     desired_right = (100 - 50 * motor_factor) * motor_direction;
}

if ((turn_count > trapped)&&(turn_timer > (int) seconds())) {
     next_left = desired_left;
     next_right = desired_right;
     desired_left = -100;
     desired_right = -100;
     motor_control(1000);
     desired_left = next_left;
     desired_right = next_right;
     turn_count = 0;
     turn_timer = 0;
}

if (abs(desired_left) > 100)
     desired_left = sign(desired_left) * 100;
if (abs(desired_right) > 100)
     desired_right = sign(desired_right) * 100;
```

```
                motor_control(300);
            }
            }

        else
            defer();
    }
}

/* Drive the motors */
void motor_control(int motor_timer)
{
    long timer_b;

    motor_working = 1;                /* prevent arbitrator from changing motor values */

    timer_b = mseconds() + (long) motor_timer;
    while (timer_b > mseconds()) {
        current_left = (motor_damping_constant * current_left + desired_left)/
            (motor_damping_constant + 1);
        current_right = (motor_damping_constant * current_right + desired_right)/
            (motor_damping_constant + 1);
        motor(left_motor, (float) current_left);
        motor(right_motor, (float) current_right);
        defer();
    }

    motor_working = 0;                /* enable arbitrator */
}

/* Main program to start the individual processes */
void main()
{
      /* Initialize data direction for control pins of Analog Mux */
      bit_set(port_ddrd, ddrd_mux_mask);
      bit_set(port_pactl, pactl_mux_mask);

      /* Initialize port A for control of Analog Mux */
      bit_clear(port_a, left_cds_amask);

      /* initialize constant values that depend on other constants */
      bump_div = bump_mid_value - bump_lb_min;

      /* Spawn control processes */
      start_process(sensor_module());
    start_process(bump_sense());
    start_process(ir_sense());
    start_process(light_sense());
    start_process(sound_sense());
    start_process(arbitrate());
}
```