Zaphod: An Autonomous Agent

EEL 4914: Senior Design
Intelligent Machines Design Laboratory

Department of Electrical Engineering
University of Florida

by

Anita Nariani

TABLE OF CONTENTS

Introduction

Implemented as a true design class, the EEL 5934 course allows students to define a semester project involving autonomous agents and build it. As noted in the Mobile Robots book, the design and construction of mobile robots is as much an art as science. Not only does this class allow the student to explore the complexities of human behavior, but it also allows for creativity towards an interactive, but autonomous agent. The primary reference for the Zaphod project are Professor Doty, Scott Jantz, and Erik Del Iglesia. In addition, conceptual ideas about the design have been implemented from the *Mobile Robots* book, by Joseph L. Jones and Anita M. Flynn and *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back,* edited by Pattie Maes. On the other hand, the technical questions have been answered by the *6.270 Robots Builder's Guide*, by Fred Martin.

The objective of Zaphod is to be an autonomous agent that will have features such as obstacle avoidance, automatic transmission, a "pseudo-bumper," a land's end characteristic, a heat detection system, and also an override control. This paper describes the acutation methods for the motors, the characterization of the sensors used and also discusses the highlights of the algorithms for individual behaviors.

Overall Behavior Integration

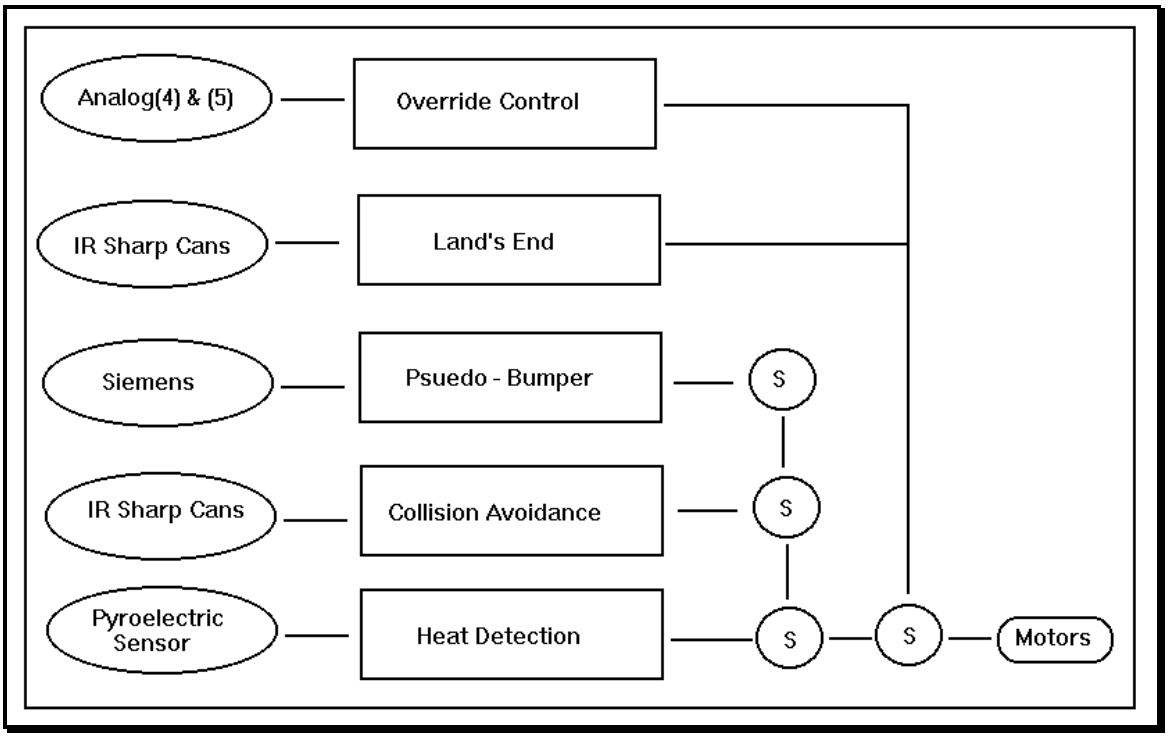


Figure 1: Overall Structure of Algorithms

Robotics had been controlled by the "world modeling and planning" (Flynn, 243) approach that decomposes a robot program into an ordered sequence of functional components. Such a traditional approach of collecting data, processing it, and then building a solution has quite a few drawbacks. On the other hand, the "subsumption architecture," advocates a method that combines distributed real-time control with sensor-triggered behaviors. Such a system allows all behaviors to be simply layers of control systems that all run in parallel whenever appropriate sensors respond. The most challenging task thus remains in devising an arbitration scheme that would fuse

any conflicting data. Although, all the behaviors run in parallel, there is some form or hierarchical nature to this architecture. In other words, the most dominant behavior (which is at a higher level) may suppress a low-level behavior. This could be characterized by the fact that the collision avoidance feature takes precedence over the "pseudo-bumper" mechanism. First, Zaphod will try to use the normal collision avoidance scheme and after a set number of failures, it could pass control to a lower level, such as the "pseudo-bumper," or heat detection mechanism.

Zaphod integrates its behaviors: collision avoidance, "pseudo-bumper," override control, heat detection, and land's end, by devising a hierarchical structure. A higher level layer first takes control of Zaphod and if it is successful, it stays in that module, else it passes control to a higher layer. Then depending on the output of the sensors corresponding to that layer, Zaphod will make the necessary procedures. The above figure illustrates Zaphod's mental structure.

Zaphod's Body Structure

Having bought the remote control car from Toys' R Us, the algorithm techniques to implement are quite different from the conventional one used in the Machine Intelligence Laboratory. This specifically refers to the "square robot" (Flynn, 146) shape of Zaphod. The body of the car houses the two motors required for steering and forward/backward motion. In addition to the RC car circuitry and the standard M68HC11 with expansion, two additional boards were mounted. One board houses the switching circuitry for the two motors and voltage dividers for readings to the A/D for the override control feature. The second board contains circuitry required for multiplexing an A/D port (analog(7)) and also the multiplexing of the Pulse Accumulator for shaft encoders. Lastly, it also has the single-chip board for controlling 16 servos.

Motor Realization

Zaphod carries a total of four motors of which two are used for driving the car and the other two are mounted on a column for a pan-tilt feature. The designers spent a significant amount of time realizing the motor controls. The motor controls were finally actuated by using nFET's as a switching device that allows CPU to control the motors. Appendix A shows the circuitry for the motor driver control.

The second set of motors were used for a pan-tilt feature. The pan-tilt servo head mounted on Zaphod uses standard Futaba servos to derive its actuation. The 190^{o} of servo movement allows accurate positioning of the servo head in any direction in the upper hemisphere. The position of the servo is controlled by Erik Del Iglesia's 16 servo motor routine.

Since the two standard motor driver controls available were already used for realizing the car's drive, I decided to implement the single-chip board that would serially communicate with the M68HC11. Erik's 16 servo motor routine was modified to have a software switch. For instance, any command to the servos had to be initiated with a string of four "SSSS." Not only did this allow for an easier implementation, but it also eliminated any additional hardware.

The transmit pin on the M68HC11 (Pin J9) was connected to the receive pin on the singe-chip board. Since "serial.c," a library file that establishes a serial link between two boards. The file contains functions that initialize the SCI port on the M68HC11 to operate at 9600 baud and write an ASCII character to the serial port. Therefore, it was interesting to notice that Interactive C could not be used during the debugging procedure due to this link. In addition, after every run the "ie9" batch file had to be reloaded. As a result, the designer replaced the standard M68HC11 chip with the M68HC11A0FN chip that loaded the Pcode directly through an "init" batch file. Also, Kermit was used to get a visual picture of what was being transmitted or received. The designer found a strange occurrence of the output format at the

receiving end. When the computer was connected to the single-chip board, only 7 bytes of an 8 byte command were printed on the same line and then an automatic carriage return line feed was generated. The line following contained the eighth byte of the command. Such was not the case on the transmitting end. Although the cause of this feature did not affect the output, it would be interesting to evaluate this problem.

Collision Avoidance Feature

Collision avoidance and land's end features are realized by using infrared detectors. The basic principle of the Sharp detectors is to detect the reflected IR light emitted by the robot's LED's. The closer an object the more intense the reflected light and greater are the readings of the A/D ports. As shown by Erik, the detectors were modified to change their output from a digital one to an analog one. Also, the Sharp GP1U58X modulated detectors had a sampling rate of about 100ms, which would drastically affect the smoothness of Zaphod's behavior and also affect the processing time. Therefore, Erik showed the designers how to replace the capacitor on-board with a 10nF capacitor. This hack will allow the detectors to sample every 1ms instead of 100ms. After the hack, the Sharp detector has been well established as the predominant distance measuring technology.

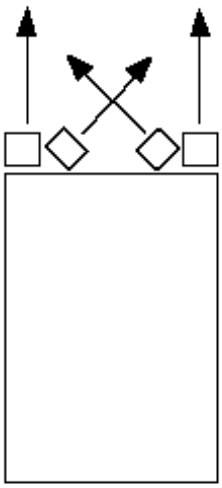


Figure 2: Cross-Eye Configuration

The sensor arrangement shown in the adjoining figure uses a “cross-eye” configuration of four IR detectors to realize the collision avoidance feature and was promoted by Professor Doty. Such a configuration allows the robot’s line of sight to be more widespread and thus results in the robot being more aware of its environment and thus have a better collision avoidance behavior. The range of the voltage for the four sensors were from 85-125 (A/D units) on a scale of 0-255 for the 5V reference on the A/D converters.

In this phase of the project, the designer spent much of the time testing the sensors and developing a proper range for the obstacle avoidance algorithm to be successful. By using 3 1kΩ pots that fit into an IC socket, the resistance values were varied until the desired ranges were reached. The detectors begin to see an obstacle at a distance of 42 inches. Also, the minimum turning radius of the car was determined to be 30 inches.

In addition, two Sharp cans were used to develop a land’s end behavior that detected an edge and guided the robot to turn away from the edge. In order to realize this, the A/D port (analog(7)) was multiplexed and a bit select pattern chose the appropriate Sharp can.

Shaft Encoders

Shaft encoders work on the principle that the IR reflects off white surfaces and does not reflect off black surfaces. Therefore, as shown in class, a disk made of black and

white segments attached to the spinning wheel will alternate between black and white as seen by a stationary sensor positioned in front of the segments. This periodic change is detected by an IR emitter/detector sensor. The Siemens SFH900 IR emitter/detector, Erik de La Iglesia's idea to the students, is ideal for this application. Since Interactive C runs as interrupt driven multi-tasking shell, the implementation of the shaft encoders provides a straightforward mechanism to realize a closed loop control feedback for speed. After testing the car in several positions, the designer determined to place the shaft encoders on the two back wheels (that are driven by the forward/backward motor) because of a differential drive system. The shaft encoders implement a "pseudo-bumper" mechanism. Due to a differential drive, the designers have tested that if the car does bump into an object, one of the two rear wheels stops turning. By this observation, the robot determines that the bumper behavior must take over and resolves the situation. The Siemens sensor's signals were translated into Pulse Accumulator (PA7) counts. Since the designers had mounted the two Siemens sensors, one of either side of the car, PA7 was multiplexed. Appendix B shows the circuitry involved in developing the Siemens sensors.

Originally, the designers had intended to implement the "automatic transmission" feature using shaft encoders. Having the car propped up (i.e.: no surface friction) allowed satisfactory speed control. In the final product, this feature was eliminated due to the weight of the car. The normal speed of the car was greater due to additional boards and the pyroelectric sensor mount which required a pulse width modulated signal of greater than a 100. Therefore, the designer may modify the PWM

code to further work for the “automatic transmission” feature. On the other hand, the “pseudo-bumper” mechanism was successfully implemented.

Heat Detection

The heat detection feature is realized by using a pyroelectric sensor. The basic principle of a pyroelectric sensor is to detect a transition to a heat source and thus interact with humans. The output of a pyroelectric sensor is the essential component in certain types of motion-detecting alarms. Therefore, the designer chose to utilize a commercially available backyard motion detector due to its economical price. The output of a pyroelectric sensor changes when small changes in the temperature of the sensor occur over time. The active element in such a sensor is typically a lithium tantalate crystal and charge is induced as the crystal is heated. The development of this sensor mainly required reengineering the circuitry on the detector board that would provide useful information about the surroundings.

First, the designer tapped the input voltage on an LM324 op amp on the board by using 120V AC power supply. There were two boards in the detector mount, one concerned with relays, voltage regulator, and AC power supply, and the other concentrated on the actual pyroelectric sensor requiring DC voltages. The feedback wires between the two boards was cut and the signals from the DC board were analyzed using a 5V DC voltage supply to the board. The pyroelectric sensor was

now responding to a heat source. Thus the first phase of the pyroelectric sensor development was successfully accomplished.

The next phase of the sensor required a characterization of the sensor with regards to a changing heat source. Using the Digital Visual Oscilloscope software and hardware in the Machine Intelligence Lab, traces of varying the distance from the heat source and the orientation were generated. Such a characterization allowed a graphic understanding of how the sensor reacted to the environment. Appendix C shows sample traces achieved at vayring distances and illustrates the time-varying output signal of a pyroelectric sensor. It was interesting to notice that a when a person walks in front of the sensor moving left to right, the signal will rise above 2.5 volts by about one volt and then fall below it, finally returning to the steady state value. Should a person walk from right to left, the reverse will happen.

The final phase in implementing this sensor was to build a robust mount for the sensor and also use a pan tilt head for heat source detection. Two servo motors were used to allow a pan-tilt feature. Having used the motor routines available through Interactive C, the designer had to incorporate Erik Del Iglesia's 16 servo motor routine. Therefore, the single chip board, as previously discussed, was used to pan the surrounding for a heat source.

Developing the Override Control Feature

In this phase of the Zaphod project, the main task involved using the already available circuitry on the remote control car to be incorporated with the M68HC11 EVBU control board. After having a complete understanding of the output signals of the ASIC chip on the car circuitry, a voltage divider was wired and one branch was fed into the A/D ports that checked whether the user was trying to take control of the car. Appendix A shows the part of the RC car circuitry that was used for this operation.

Light Sensors

CdS cells, variable resistors controlled by ambient light, are used for corner detection. For application to the agent, the cells are collimated and used to give a digital response. The CdS cells were mounted and wired by my partner, Wayne DeVoe. The range of the CdS cells depend greatly on the ambient room light. Therefore, the data from these cells account for a variety of possible environments. The panning ability of the sensor head should make wall following very simple. The reader may refer to Wayne DeVoe's report for details of this sensor development.

Zaphod's Sense of the World

The second phase of this project involved a simple but rather elegant algorithm for obstacle avoidance. The program was written in the Interactive C language and was implemented. Since the cross-eye configuration was used for the four Sharp detectors, the algorithm (Appendix D) accounts for all possibilities of an object confronting Zaphod. The obstacle avoidance algorithm provides Zaphod with a fundamental view of the world. It follows the basic philosophy that it must avoid anything it sees (detects). The designer first implemented a "sensitive" type of obstacle avoidance routine that guided Zaphod to turn away from an object such that it would only have to forward turn right or left and not have to back up. In other words, as long as the A/D reading interpreted into a distance greater than the minimum turning radius, Zaphod would start turning away from the obstacle. The second more practical and realistic approach was to make Zaphod "bold" in the sense that it would back up from an object and depending on the angled detectors' readings determine which way it should back up and make a 3 point turn. Zaphod

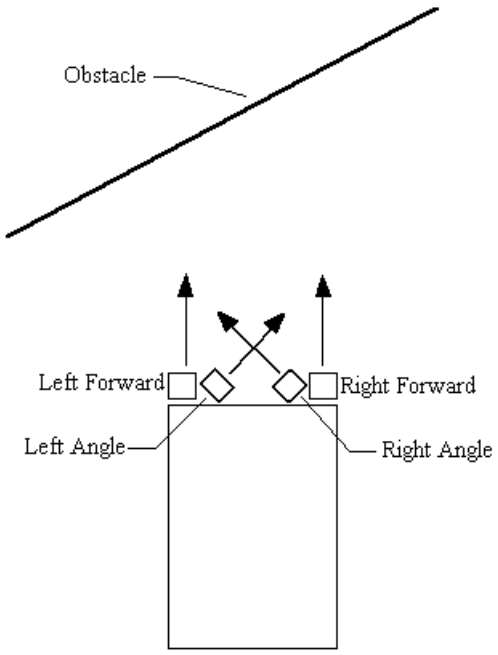


Figure 3: Scenario for Obstacle Avoidance

would then back up until its right-forward and left-forward detectors saw an opening. Then it would start moving forward and turn in the opposite direction while making sure the detectors do not saturate (i.e.: it does not hit an object). Figure 3 shows one of the numerous scenarios for obstacle avoidance. Although rather simple, this design accounts for most of the possibilities. The designer made some modifications to the algorithm as part of the testing. After designing both the behaviors, the designers decided to only implement the "bold" aspect of Zaphod since it handled a wider range of possibilities in a rather elegant manner. It was interesting to note how mutually exclusive events played a vital role in the collision avoidance algorithm.

Zaphod's Savior: The Land's End Feature

The land's end feature, utilizing the Sharp can sensors, allows Zaphod to detect an edge of the surface it is driving on. Although rather simple, this behavior enhances Zaphod's ability to be "bold." In other words, it is now less restrictive to a particular environment. Since the Sharp cans used for this feature were placed on the pyroelectric mount, the readings were in a mid range. When in its normal mode of operation, Zaphod roams around freely (i.e. in its normal mode of collision avoidance) but when the readings on the Sharp cans fall below a particular threshold (97 A/D units), the land's end feature takes control. Zaphod would then back up and turn and try to head its way (i.e. resume its normal operation). Thus this algorithm accounts for most of the possibilities for land's end detection and the function code is loacted in Appendix E.

Zaphod's "Pseudo-Bumper" Mechanism

In this phase of the Zaphod project, the main task involved a system that determines whether Zaphod has bumped into an object. As discussed in class, the shaft encoders concept is utilized for this algorithm. The concept of developing a "pseudo-bumper" mechanism is closely related to having a hardware switching mechanism, but uses a software switching mechanism. The algorithm (Appendix F) is based on the readings received from both the Siemens sensors. In order to begin checking the shaft encoder readings from PA7, the appropriate shaft encoder is selected through the multiplexer. Although simple, this algorithm checks if wither of the two wheels are not turning. Based on this information, it decides to either make a three point turn or exit the function. Although, the individual shaft encoders are checked sequentially, the processing time does not affect Zaphod's performance.

Zaphod's Social Personality

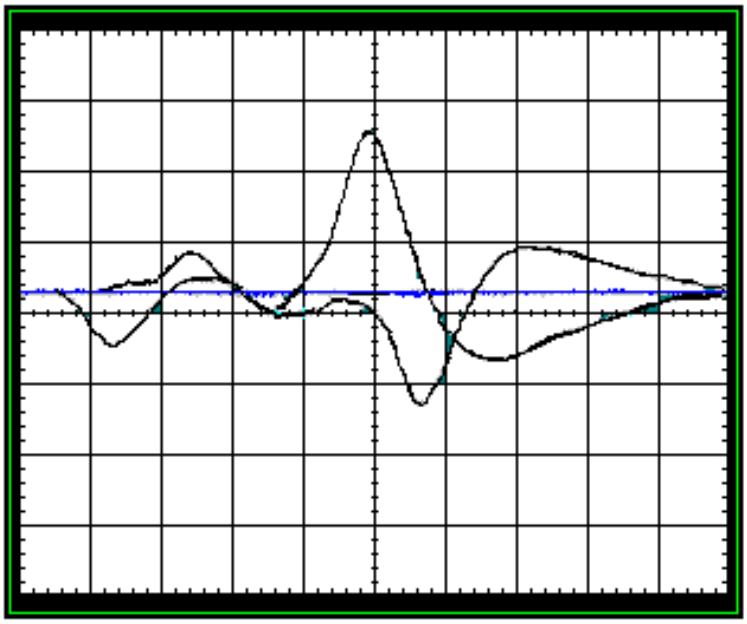
Although not exclusive to people detection, the heat detection behavior (Appendix G) allows Zaphod to approach a heat source. Using passive IR, the pyroelectric sensor detects a transition when sweeping the surrounding for a heat source. Using the second board and the serial communication link, the algorithm begins to pan the 190 degree range in steps of approximately 20 degrees. If a transition is detected in a particular position, Zaphod sets the servo position to the 90 degree position (looking straight ahead) and begins to back up and turn to the side of the heat source (i.e. a

person). If the pyroelectric sensor detects another transition, it stops backing up and turning and now heads straight for the source. With such simplicity, the problem of steering and making appropriate turns was totally eliminated. After heading for the object, the Sharp cans in cross-eye configuration are used to let Zaphod have knowledge of the distance from the destination. Thus, when the readings from the Sharp cans reaches a 100, Zaphod knows it has made it to its target and then slows down. The only drawback of this algorithm was that it did respond to the Interactive C's multi-tasking "start_process" function which may be due to the serial link. Although the sampling was sequential to the servo head movement, the time required for the servo to turn to a new position was much greater than the sampling time which provided good results. Thus, the people sensing feature results in a more interactive Zaphod!

Conclusion

In view of a realistic summary, the Zaphod project has definitely helped me realize that testing the sensors and determining all the specifications to the problem are the most important aspects. Since the last presentation, the designer has made considerable progress. First. In addition, a collision avoidance, pseudo bumper, land's end, override control, and a heat detection mechanism have been completed. The scope of the project has met the end results and the designers are content with the progress made. Further work will be continued next semester and the DSP board will be used on Zaphod.

Appendix C: Characterization of the Pyroelectric Sensor



Using a Digital Visual Oscilloscope, it was found that there was a negative to positive swing when a person walked in front of the pyroelectric sensor from left to right and vice versa.

Appendix D: Algorithm Exhibiting Obstacle Avoidance Behavior

```
/* globals */

int frd_bck = 1;
int steering = 0;
int OFF = 0;
int slow = 60, med = 60, fast = 70;
int right = 50, left = -50;
int override = 0;
int count = 5;
int gamma = 25;
int sec = 1000;
int speed;
int test_frd, test_bck;

int bk_wait = 1000, turn_wait = 2000;

/*  to put in turning radius or distance... */
int min_distance = 90;
int object, ctr, trial;
int drn;

int clear = 88;
int min = 100;
int saturation = 120;
int bk_min = 95;

int ctr1=0, ctr2=0, ctr3=0, ctr4=0, ctr5=0, ctr6=0;

int fwd=1, back=0, backup_left=0, backup_right=0;

void main ()
{
start_process(sensor_module());
start_process(backup_flag());
start_process(Override_Test());
start_process(Timer());
start_process(Motor_Driver());
}

/*  Sensory registers */
int left_forward, left_angle, right_angle, right_forward;

void sensor_module()
{
poke(0x7000,0xff);
  while(1)   {
      left_angle = analog(0) - 4;
      left_forward = analog(1);
      right_forward = analog(2) + 3;
      right_angle = analog(3);
      test_frd = analog(4);
      test_bck = analog(5);
      wait(1);                  /* waits 100msec */
      }
}

void backup_flag()
{
while(1)
{
while ((left_forward > min || right_forward > min) && (left_angle < right_angle))
   {
     backup_left=1;
     backup_right=0;
     fwd=0;
     back=0;
     }
```

```
while ((left_forward > min || right_forward > min) && (left_angle > right_angle))
     {
     backup_right= 1;
     backup_left=0;
     fwd=0;
     back=0;
     }

while ((left_forward > min || right_forward > min) && (left_angle == right_angle))

     {
     backup_left=1;
     backup_right=0;
     fwd=0;
     }

 while (left_forward < min && right_forward < min)
  {
    fwd=1;
    backup_left=0;
    backup_right=0;
    back=0;
  }
 }
}

void Motor_Driver()  {
 while(1)  {
   while(!override)  {
     while (backup_left == 1 && !override)  {
       while ((left_forward > clear && right_forward >clear) && !override)  {
         Right();
         Back(fast);
       }
       while((clear < left_forward && left_forward < saturation) && !override)  {
         Left();
         Forward(med);
       }
       Straight();
     }

     while (backup_right == 1 && !override)  {
       while ((right_forward > clear && left_forward > clear) && !override)  {
         Left();
         Back(fast);
       }
       while (right_forward > clear && right_forward < saturation && !override)  {
         Right();
         Forward(slow);
       }
       Straight();
     }

     while (fwd==1 && !override)  {
       Straight();
       Forward(med);
     }
   }
   Straight();
   while(override)  {
     while(test_frd < gamma)  {
       Forward(fast);
     }
     while(test_bck < gamma)  {
       Back(fast);
     }
     Stop();
   }
 }
```

```
}


void Override_Test()  {
  while(1)  {
    while((test_frd < gamma) || (test_bck < gamma))  {
      count = 5;
      override = 1;
    }
    while((test_frd > gamma) && (test_bck > gamma) && (count != 0))  {
      override = 1;
    }
    while((test_frd > gamma) && (test_bck > gamma) && (count == 0))  {
      override = 0;
    }
  }
}

void Straight()
{
  motor(steering,0);
}

void Stop()
{
motor(frd_bck,0);
}

void Left()
{
motor(steering, left);
}

void Right()
{
motor(steering, right);
}

void Forward(int speed)
{
motor(frd_bck,speed);
}

void Back(int speed)
{
motor(frd_bck, -speed);
}

void wait (int milli_seconds)  {
  long timer_a;
  timer_a = mseconds() + (long) milli_seconds;
  while (timer_a > mseconds())  {
    defer();
  }
}

void Timer()  {
  while(1)  {
    while (count != 0)  {
      wait(sec);
      count--;
    }
  }
}
```

Appendix E: Algorithm describing the Land's End Feature

```
int end=0;
int landend=113;
int cntr = 0, turn_left = 0;
int frd_bck = 1;
int steering = 0;
int OFF = 0;
int speed=80, bck_speed=50;
int right = 50, left = -50;

void main()
{
motor(1,80);

start_process(sensor());
start_process(land());
}

void sensor()
{
  while(1)
  {
   poke(0x7000,0x03);
   poke(0x6000,0x06);
   end = analog(7);
   wait(200);
  }
}

void land()
{
   while(1)
   {
   while(end < landend)
   {
     Stop();
     wait(10);
     Right();
     Back(bck_speed);
     turn_left = 1;
     wait(2000);
   }
   while(end > landend && turn_left && cntr < 10)
   {
   Stop();
   wait(10);
   Left();
   Forward(speed);
   cntr++;
   }
   cntr = 0;
   while(end > landend)
   {
   Forward(speed);
   }

  }
}

void wait(int m_second)
  {
   long stop_time;
   stop_time = mseconds() + (long)m_second;
   while(stop_time > mseconds())
     defer();
  }
```

Appendix F: Algortihm Describing the “Pseudo-Bumper” Behavior

```
/* globals */

int frd_bck = 1;
int steering = 0;
int OFF = 0;
int speed = 100;
int sec = 1000;
int  medium =80;
int val;
int left=50, right=-50;

int not_yet = 0;
int bump;

void main ()
{
Forward(medium);
wait(5000);
start_process(sensor_module());

}

/*  Sensory registers */

void sensor_module()
{
   poke(0x1026,0x50);
   while(1)
   {
   enable_encoder(1);
   wait(200);
   check();
   enable_encoder(2);
   wait(200);
   check();
   }
}

void check()
{
    val = peek(0x1027);
    if ((val == 0) && (not_yet == 0))
    {
    bump = 1;
    Right();
    Back(medium);
    wait(2000);
    Straight();
    Forward(medium);
    not_yet = 20;
    }
    if (not_yet > 0) not_yet--;
}

void enable_encoder(int number)
{
   if (number == 1) bit_set(0x6000,0x80);
   if (number == 2) bit_clear(0x6000,0x80);
   poke(0x1027,0);
}
```

Appendix G: Algorithm describing the Heat Detection behavior

```
int frd_bck = 1;
int steering = 0;
int OFF = 0;
int slow = 100, med = 100, fast = 100;
int right = 50, left = -50;
int min=90;

int heat=0,oldheat=0, heat1=0;
int max_thresh=0;
int i=0;
int left_forward=0, right_forward = 0;

void main()
{
  poke(0x6000,0x07);
  turn();
  while(1)
  {
  region();
  }

}

void turn()
{
    init();
    put_char('C');
    wait(50);
    put_char(0);
    wait(50);
}

void region()
{   int found=0;
    int ctr=0;
    int times=0;
    int do =0;
    int sharps =0;

    for (i=0; i<=140; i += 9)
    {
    set_servo(0,i);
    wait(50);
    heat1 = analog(7);

      if ((heat1-oldheat) > 0)
      {
      set_servo(0,72);
      wait(50);
      found = 1;
      do = 1;

      while (do == 1)
      {
       if (found == 1)
       {
          if (i < 75) Right();
          if (i >= 75) Left();
       Back(70);
       wait(1500);

        if (analog(7) > 0)
        {
        poke(0x7000,0xff);
        left_forward = analog(0);
        right_forward = analog(3);
        sharps = 1;
```

```
        while(sharps ==1)
        {
          while(left_forward < min || right_forward < min)
          {
          Straight();
          Stop();
          Forward(70);
          wait(10);
          }

          while (left_forward >= min || right_forward >= min)
          {
          Stop();
          wait(1500);

          set_servo(0,0);
          wait(50);
          set_servo(0,140);
          wait(50);
          set_servo(0,72);
          wait(50);
          found = 0;
          sharps = 0;
          do = 0;
          }
        }

        }
       }
      }
      }
      ctr++;
    }
}

void set_servo(int servo_num, int position)
{
  init();
  put_char('A');
  wait(50);
  single_digit_hex(servo_num);
  wait(50);
  hex_to_ascii_hex(position);
  wait(50);
}

void init()
{
  put_char('S');
  wait(50);
  put_char('S');
  wait(50);
  put_char('S');
  wait(50);
  put_char('S');
  wait(50);
}

int hex_to_ascii_hex(int value)
{
  int i=0;
  while(value>15)
    {
      value=value-16;
      i++;
    }
  put_char(single_digit_hex(i));
  wait(50);
  put_char(single_digit_hex(value));
```

```
  wait(50);
}

int single_digit_hex(int value)
{
  if (value<10) put_char(value+48);
     else put_char(value+55);
   wait(50);  }
```