Ziggy: Final Report

By

Reynaldo A. Molina

Machine Intelligence Laboratory

Prof: Keith L. Doty

12 / 4 / 95

Table of Contents

Abstract

"Ziggy" is a mobile robot created to behave as a prey to defend itself from "Mazo;" its counter part. To simulate a predator/prey relation between these two creatures, Ziggy and Mazo must display basic predator/prey behaviors. Ziggy's reaction to Mazo's actions will determine how well it copes with its environment.

Executive Summary

This project consists in the creation of an autonomous agent that is able to display specific behaviors programmed in "Interactive C" and stored in 32k of Random Access Memory. The Brain of this agent is a MC68HC11 microprocessor board with a circuit to control two motors and a circuit to regulate voltage. The board is screwed to a plywood platform that provides support for the rest of the components. Ziggy uses two servo motors placed under the plywood board and attached to two model airplane wheels to give this agent mobility. The polarity in the motors determines the direction in which they rotate. To interact with the environment, Ziggy uses three kinds of sensors. These are infrared, ultrasonic, and sound detection sensors.

All these sensors combined give Ziggy the ability to interact with the environment in the way it was intended. Ziggy'y program takes information from each sensor and determines whether it is relevant or not based on a series of condition statements. Although some sensors can provide relevant information at the same time, Ziggy's programming can only evaluate one at a time and make a decision.

Introduction

In this project we intend to use two autonomous agents to simulate behaviors that predators and prey display when they meet. Although, at this point, it is impossible to simulate complex behaviors, we are capable of simulating some of the most basic ones as locating the prey, following it, and coming in contact with it.

On the other hand, the prey also displays behaviors that it uses to survive and that may or may not save it from a predator. Our intentions are to provide Ziggy with means to defend itself. Among these means we have: searching for possible predators in the area, knowing when the predator is following, running away from it, and hiding from it. This report describes some of the means we are going to use to simulate these behaviors.

Integrated Systems

To achieve the goal of this project Ziggy must have the means to move and interact with the environment that surrounds it. These means are realized in the integrated system of this unit.

Mobile Platform

Ziggy's body consist of a very flexible plywood 12in in diameter painted in green to maintain consistency with the color of its brain. This round shaped platform makes Ziggy able to turn on its axis helping it avoid objects easily. In addition, a circular platform requires fewer wheels and actuators to control the robot helping it save power. The platform supports the EVBU board and the battery pack.

The platform rests on three wheels. Two of these are model airplane wheels with 4.75in in diameter parallel to each other. Their main function is to move the body in any direction that the behaviors indicate. Unlike the main wheels, the third wheel is screwed to the platform. It provides balance to the hole structure. See Figure 1 for visual representaion.

Servo Motors

Ziggy uses two servo motors to drive the main wheels. These servos are adequate for the purpose of this simulation because they provide sufficient torque-speed for Ziggy to maintain a reasonable distance from Mazo that also uses servos. In addition, the noises the servo make, simulate the noises prey make as they move. With this in mind the predator will be looking for noises Ziggy produces helping the simulation to be more realistic.

A thin sheet of metal, screwed to the platform, keeps the servos in a fix position. The ground and voltage wires connect to the EVBU board through a hole in the center of the platform 0.4in in diameter.

Ziggy's Sensors

For its eyes Ziggy uses infrared sensor. With six of these sensors Ziggy is able to avoid walls, detect other robots, and small objects in the area. These sensors are necessary for Ziggy, as a prey, to survive and defend itself in case of attack. See Figure 2 for visual representation.

For its ears Ziggy uses microphones to detect noises and determine the direction of their source. The microphones have plastic covers to enhance the acoustics. This capability, in conjunction with a sonar sensor, helps Ziggy determine if a potential predator is in the area. Ziggy uses four microphones for a more accurate approximation in terms of direction. Figure 3 provides a diagram of a microphone circuit.

Ziggy's ultrasonic motion detector helps it identify movement. With this sensor Ziggy is able to recognize the predator because it is the only moving object in the simulation. This sonar has a transmitter that sends out a steady ultrasonic tone at 40khz. Any reflected sound is detected by the receiver. If no movement is detected, the sampling time between emission and reception of the pulse is constant; however, if there is movement, then the sampling time is not constant and the output signal of the circuit changes, setting the alarm. For this process to be effective Ziggy has to stay still while the sampling is taking place.

Behaviors

Besides the obvious behaviors as avoiding object, Ziggy identifies potential predators in the area. Once the predator detects Ziggy's movement, it tries to pursuit and attack. Ziggy senses the movement as Mazo approaches and tries to escape by running in the opposite direction. For a more realistic simulation, Ziggy turns off its IR sensors and stay immobile to make the predator believe it is not around. Unfortunately, Ziggy is not able to maintain this state for long because it is programmed to have the urge to run.

Experimental Layout and Results

According to the reading obtained from the A/D channels, the output integer values from the infrared sensors vary from sensor to sensor. In general, if there is no object present in a range of three to four feet, the sensors return a value of 85-90. On the other hand, if there is an object present within this range all six sensors return a value of 100 to 128, depending on how close the object is to the emitter.

The ultrasonic motion detector returns a value of 255 if there is no movement present, and a value of less if there is. Similarly, the microphone return values of 150 if there is no sound detected , and values of 180 if there is. This range is not sufficient to determine which of the microphones is closer to the source.

It is necessary to set boundaries in the programs that determine when Ziggy has to change direction or take some action. These boundaries are variables of type integer named “Thresholds.” The threshold for the infrared sensors must be 126 before the robot considers the information valuable.

Conclusion

At this point in time Ziggy is able to display object avoidance behavior, detect moving objects and sounds. The combination of all the sensors give Ziggy the ability to respond better to the surrounding environment. We are giving Ziggy a limited set of behaviors with intentions to incorporate new ones and improve on the old ones. The success of this simulation depends upon how well Ziggy can play the game of the predator and the prey.

Documentation

J. L. Jones and A. M. Flynn, “Mobile Robots: Inspiration to implementation,” A. K. Peters, ltd., Wellesley Ma, 1993.

F. G. Martin, “The 6.270 Robots Builder’s Guide,” F. G. Martin, 1992.

Motorola, MC68HC11 EVBU User’s Manual, Motorola, Inc., 1992.

Appendices

```
/* Right motor and Left motor */
int     Right_Motor = 1;
int     Left_Motor  = 0;

/* boundaries for LEDs */
int     T_Hold0 = 110;
int     T_Hold1 = 110;
int     T_Hold2 = 128;

int     T_Hold4 = 111;
int     T_Hold5 = 110;
int     T_Hold6 = 118;
```

```
/* Speeds  */
int     Full_Speed = 100;
int     Norm_Speed =  80;
int     Half_Speed =  50;
int     Stop       =   0;

/* Infrared sensor variables */

int     Left;
int     Right;

int     Center_Left;
int     Center_Right;

int     Left2;
int     Right2;

int     Center_Left2;
int     Center_Right2;

int     Back_Left;
int     Back_Right;

/* Sonar sensor variable & boundaries */
int     Sonar;
int     Sonar_T_Hold = 255;

/* microphones variables and boundaries */
int     M1;
int     M1_T_Hold = 165;

int     M2;
int     M2_T_Hold = 139;

int     M3;
int     M3_T_Hold = 150;

int     M4;
int     M4_T_Hold = 159;

/* Time variables */
int     Time = 517;
int     Time2 = 50;
int     Time3 = 70;
int     Time4 = 375;
int     TimeRandom;

float   Execution_Time1 = 17.0;
float   Execution_Time2 = 4.0;
float   Execution_Time3 = 2.0;

/* Go to the right */
void Go_Right (int Left_Speed, int Right_Speed) {
   motor (Left_Motor,Left_Speed);
```

```
   motor (Right_Motor,Right_Speed  *  -1);
}

/* Go to the Left */
void Go_Left (int Left_Speed, int Right_Speed) {
   motor (Left_Motor,Left_Speed * -1);
   motor (Right_Motor,Right_Speed);
}

/* Go back */
void Go_Back (int Left_Speed, int Right_Speed) {
   motor (Left_Motor, Left_Speed * -1);
   motor (Right_Motor, Right_Speed * -1);
}

/* Go forward */
void Go_Forward (int Left_Speed, int Right_Speed) {
   motor (Left_Motor, Left_Speed);
   motor (Right_Motor, Right_Speed);
}

/* stay still */
void Stay_Still () {
   motor (Left_Motor, Stop);
   motor (Right_Motor, Stop);
}

/* This function delays for n milliseconds */
void wait(int milli_seconds)     {
  long timer_a;
  timer_a = mseconds() + (long) milli_seconds;
  while(timer_a > mseconds())
    defer();
}

/* random number generator */
void RG () {
  if ((TimeRandom > 7)||(TimeRandom < 0))
    TimeRandom = 0;
  else
   TimeRandom = TimeRandom + 1;
}

/* This function reads continuosly
   the infrared sensors */
void Infrared_Emitting () {
  while (1) {

   poke(0x7000,0b00010000);
   wait(Time2);
   Center_Left = analog(4);

   poke(0x7000,0b00001000);
   wait(Time2);
```

```
   Center_Right = analog(0);

   poke(0x7000,0b00100000);
   wait(Time2);
   Left = analog(5);

   poke(0x7000,0b00000100);
   wait(Time2);
   Right = analog(1);
  }
}

/* This function reads IRs
   when infrared is off*/
void Read_Receivers () {
   while (1) {
    poke (0x7000,0b00000000);
    wait(Time2);

    Center_Right2 = analog(0);
    Right2 = analog(1);
    Back_Right = analog(2);

    Center_Left2 = analog(4);
    Left2 = analog(5);
    Back_Left = analog(6);
   }
}

/* This function reads sonar and mics */
void Read_Sonic_Sensors () {
   while (1) {

    wait(Time3);
    Sonar = analog(7);

    poke(0x6000,0x00);
    wait(Time3);
    M2 = analog(3);

    poke(0x6000,0x06);
    wait(Time3);
    M3 = analog(3);

    poke(0x6000,0x02);
    wait(Time3);
    M1 = analog(3);

    poke(0x6000,0x04);
    wait(Time3);
    M4 = analog(3);
   }
}
```

```
/* This function checks for sounds
   in the surroundings */
int Sound_Checking () {

  int M1_Diff;
  int M2_Diff;
  int M3_Diff;
  int M4_Diff;
  int M_Sound;
  int Direction;

  float Current_Time;
  float Duration;

  Current_Time = seconds();
  Duration = Current_Time + Execution_Time3;

  Stay_Still ();
  wait(Time4 * 2);

  while ((Current_Time = seconds()) < Duration) {

   M_Sound = 0;

   if ((M1 > M1_T_Hold) || (M4 > M4_T_Hold)) {

    M1_Diff = (M1 - M1_T_Hold);
    M4_Diff = (M4 - M4_T_Hold);

    if ((M1_Diff > M4_Diff) && (M1_Diff > 0)) {
     Go_Forward (Half_Speed,Half_Speed);
     wait(Time4);
     return 1;
    }
    else if ((M4_Diff > M1_Diff) && (M4_Diff > 0)){
     if (TimeRandom > 3) {
      Go_Right(Full_Speed,Norm_Speed);
      wait(1500);
      return 1;
     }
     else {
      Go_Left(Norm_Speed,Full_Speed);
      wait(1500);
      return 1;
     }
    }
   }

   if ((M2 > M2_T_Hold) || (M3 > M3_T_Hold)) {

    M2_Diff = (M2 - M2_T_Hold);
    M3_Diff = (M3 - M3_T_Hold);

    if ((M2_Diff > M3_Diff) && (M2_Diff > 0)) {
```

```
     Go_Left (Norm_Speed,Full_Speed);
     wait(Time4 * 2);
     return 1;
    }
    else if ((M3_Diff > M2_Diff) && (M3_Diff > 0)) {
     Go_Right(Full_Speed,Norm_Speed);
     wait(Time4 * 2);
     return 1;
    }
   }
  }
  return 0;
}

/* This function detects motion */
int Motion_Detection () {

 int Direction_Flag;

 wait (Time * 3);

 if ((Center_Left2 > T_Hold5) || (Left2 > T_Hold4)) {
  Stay_Still();
  Direction_Flag = 1;
 }
 else if ((Center_Right2 > T_Hold0) || (Right2 > T_Hold1)) {
  Stay_Still();
  Direction_Flag = 2;
 }
 else if (Back_Left > T_Hold6) {
  Stay_Still();
  Direction_Flag = 3;
 }
 else if (Back_Right > T_Hold2) {
  Stay_Still();
  Direction_Flag = 4;
 }
 else if (Sonar < Sonar_T_Hold) {
  Stay_Still();
  Direction_Flag = 5;
 }
 else
  Direction_Flag = 0;

 return (Direction_Flag);
}

/* This function does object avoidance */
void Object_Avoidance () {

  int Detected;
  int Action;
  int Priority;
```

```
  float Duration;
  float Current_Time;

  while (1) {

   Priority = 0;
   Current_Time = seconds();
   Duration = Current_Time + Execution_Time1;

   while ((Current_Time = seconds()) < Duration) {
    if ((Center_Right > T_Hold0) || (Right > T_Hold1)) {
     Go_Left (Norm_Speed,Full_Speed);
     wait(TimeRandom * Time4);
    }
    else if ((Center_Left > T_Hold4) || (Left > T_Hold5)) {
     Go_Right (Full_Speed,Norm_Speed);
     wait(TimeRandom * Time4);
    }
    else {
     Go_Forward (Norm_Speed,Norm_Speed);
    }
   }

   Current_Time = seconds();
   Duration = Current_Time + Execution_Time2;

   Stay_Still();
   wait(Time4);

   while ((Current_Time = seconds()) < Duration) {
    Detected = Sound_Checking();
    if (Detected == 1) {
     Action = Motion_Detection();
     if (Action > Priority)
      Priority = Action;
    }
   }

   if ((Priority == 1) || (Priority == 3)){
     Go_Right (Full_Speed,Norm_Speed);
     wait(Time4);
   }
   else if ((Priority == 2) || (Priority == 4)) {
     Go_Left (Norm_Speed,Full_Speed);
     wait(Time4);
   }
   else if (Priority == 5) {
    Go_Back (Full_Speed,Full_Speed);
    wait(Time4);
    if (TimeRandom <= 3) {
     Go_Right(Full_Speed,Norm_Speed);
     wait(1500);
    }
    else {
```

```
     Go_Left(Norm_Speed,Full_Speed);
     wait(1500);
    }
   }
  }
}

/* This function executes all processes */
void main() {
 start_process(RG());
 start_process(Infrared_Emitting());
 start_process(Read_Receivers());
 start_process(Read_Sonic_Sensors());
 start_process(Object_Avoidance());
}
```