Department of Electrical Engineering

University of Florida

INTELLIGENT MACHINES DESIGN LABORATORY

EEL5934 Robotics

Instructor: Keith L. Doty

11

Bot'arina

Daniel Coggin

Mon, May 1, 1995

TABLE OF CONTENTS

Opening

Main Body

Closing

Abstract

The mobile agent, Bot'arina, described in this report was built to explore the possibilities of the ability of a mobile agent to follow sound using only one directional microphone. The agent also has infra-red emitters and detectors which are used to avoid obstacles while the agent moves toward a sound source. The performance of the agent was mediocre. Perhaps due to the clipping the signal from the microphone amplifier.

Executive Summary

The ability to hear sounds and determine from which direction they are coming from is a very useful function to have realized on a mobile agent. Using this function an agent could follow sounds for

example go towards machines "calling" them at a unique and audible frequency. In order to accomplish this, Bot'arina, the agent described in this report, uses a directional microphone. The microphone is constructed by placing a condenser microphone at the base of a tube. The tube is then wrapped with sound damping material. The microphone is mounted securely to the base of the agent. The agent determines the direction of the sound by rotating and monitoring the microphone signal. When the signal starts decreasing in magnitude the agent reverses the direction of for a short time and then moves forward for about a second. This process is repeated as the agent approaches the sound source. The agent is also equipped with infra-red emitters and detectors which allow the agent to avoid obstacles while following sound. Bot'arina was built to explore the possibilities of sound sensors.

Introduction

Sounds are everywhere in an entities life. Without capable receivers, ears, to hear them, humans would be lost. Our ears allow us to perceive a direction from which a sound emanates, determine if we are in a room or outside (spatial perception), and aids us in communication. These abilities would be very advantageous to a mobile agent. Whether the robot was to be a security guard or answer calls from machines in a work cell needing parts, determining the direction of a sound and heading in that direction would be a useful pursuit. Bot'arina was built to explore these ideas.

Integrated System

The robot consists of the following components:

* 68HC11 EVBU board with 68HC11A0 chip
* Memory expansion chips (32 kbytes)
* A voltage regulator
* Cake Pan
* Two Swiss motors
* One L293 motor driver chip
* Two shaft encoder sensors
* Three battery packs, 8 x AA = 12V and 2 x 9V
* One Panasonic WM-34CY195 microphones

* Two Sharp GP1U52X infrared detectors
* Two infrared LEDs
* Two model airplane wheels
* A 74LS390 dual decade counter

The microcontroller was upgraded to 32 kbytes of ram to satisfy the requirements of Interactive C (IC) [2]. A voltage regulator and its associated circuitry was added to make the robot autonomous. The shaft encoders report the speed of the wheels to the microcontroller which processes the data and pulse width modulates the motors. The 12V battery pack provides unregulated power to the motors, a nine volt alkaline cell provides power to the regulated logic on the 6811 board, another nine volt battery provides power to the shotgun microphone amplifier. The second nine volt supply was nesessary to prevent interference from the 40kHz IR oscillator. The two Sharp 55infrared detectors were interfaced to the pins PE0 and PE1 of the microcontrollers A/D port.

The Cake Pan Platform

The specifications for my mobile platform are as follows:

* Able to support 6811 EVBU board and future additions
* Round - to rotate out of tight corners
* Light weight
* Easy to work with (drill, cut, etc...)
* Cheap
* Avoid obstacles

The cake pan was purchased from Kash 'N Karry for $1.69 and was used as a base for the robot because it fit the specifications perfectly. Holes were cut for the wheels through the tin using a drill and a nibbler from Radio Shack. The airplane wheels were glued to the Swiss motors and mounted inside the cake pan using hot glue. Very soon it was discovered that the glue would not hold the wheels to the motor shaft. This is fixed by drilling a hole through the wheel shaft and motor shaft, then a cotter pin was placed through the holes to secure the wheel to the shaft. In the beginning a 64 segment encoder disc was used which was to fine for the sensor to distinguish. This was replaced by a 32 segment disc which worked great. Hot glue was used to attach the encoder sensor to the inside of the pan as close to the encoder disc as possible. The most embarrassing lesson learned was that the encoder sensors were originally installed backwards.

Mobility

The specifications for mobility were as follows:

* Able to maneuver on shag carpet
* Efficient
* Cheap
* Attain a speed of eight inches per second

The Swiss motors and airplane wheels fit the specifications perfectly. The 12V Swiss motors had a 2mV operating ripple and 200mA stall current. The motors worked fine without placing diodes on the

L293 motor driver chip. The Swiss motors were purchased from American Science and Surplus for $15 ea. The two airplane wheels were bought from Hobbyland for $4. A cotter pin holds the wheels to the motors. A simple straight line algorithm was written in IC and works fair [app. A]. There seems to be a small variation from a speed of 20% pwm and 100% pwm using the **motor(a,b)** command in IC. This maybe due to the highly efficient Swiss motors.

Sensors

The specifications for the sensors were as follows:

* One shotgun microphone to detect the direction of a sound
* Two IR LEDs to work with the IR detectors
* Two IR detectors to avoid obstacles

A system diagram is shown below in figure 1.

The microphone circuitry is given by figure 2 taken from [1].

The shotgun michrophone was constructed as follows. A condenser microphone was mounted in the bottom of a brass tube 1/3" in diameter and 8" long. Cotton balls were then glued to the tube. The cotton rod was then inserted into a PVC pipe 1" in diameter and 8" long. This was done to reduce echo.

The shotgun microphone construction is shown in figure 3.

The signal from the microphone and amplifier was connected to the analog port PE2. An example of the signal produced is shown below in figure 4.

The Follow Sound Behavior

The robot detects sound and discerns a direction by rotating its platform, which the microphone is mounted on, counter clockwise . Samples are taken to determine if the magnitude of the sound is increasing or decreasing. If the magnitude starts decreasing the platform stops and rotates clockwise for 100 miliseconds then it goes forward for one second. This process is then repeated producing "baby steps" toward the sound. The IR emitters and detectors work together to avoid obstacles while the robot searches for sound. A block diagram below (Fig. 5) shows the program flow for the follow sound routine. The code for this behavior is in appendix A.

The Avoid Obstacle Behavior

Another processes running concurrently with the follow sound process is the avoid obstacle process. This process uses the information from the IR detectors to avoid obstacles. If an object is detected to the left the robot rotates right until the object is no longer detected. The opposite is true for the right case. The code for this behavior is in appendix A.

Conclusion

Working with sound is difficult. The completed robot seemed to have trouble with echoes. Also as the robot approaches the sound source the magnitude increases to the extent that there is a clipping of

the signal into the microcontroller. When this occurs little information can be obtained form the signal. The robot gave a poor performance perhaps due to this. The future work includes the following:

* A better mobile platform using three wheels
* Improved sound following with a digitally attenuated amplifier
* A ISD1000 chip to record and playback sounds
* A speech recognition circuit to resond to voice commands
* Cds cells to allow hide in the dark or search for light behaviors

References

[1] Joseph Jones and Anita Flynn, *Mobile Robots: inspiration to implementation*, A K Peters, Wellesley, Massachusetts, 1993.

[2] Fred G. Martin, *The 6.270 Robot Builder's Guide*, MIT Media Lab, Cambridge, MA, 1992.

[3] Pattie Maes, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, MIT Press, Cambridge, MA, 1990.

Appendix A Program Code

```
/*  A simple program for straight line motion */

void straight()
{
                                                        /* initialize variables */
      int right_speed;
      int left_speed;
      int a;
      int b;

      a = 50;
      b = 50;
                                                        /*  initialize encoders */
      enable_encoder(0);
      enable_encoder(1);

      while(1) {                              /*  Loop forever  */

      right_speed=read_encoder(0);
      left_speed=read_encoder(1);

      if (left_speed > right_speed)
        {
          a=a-10;                             /* Slow down the left motor */
          move(b,a);                          /* If faster than the right */
        }
      else if (left_speed < right_speed)
        {
          a=a+10;                             /* Speed up the left motor  */
          move(b,a);                          /* If slower than the right */
        }
      if (a == 50)                            /* Left motor is at the cieling */
        {
          if (left_speed > right_speed)
            {
              b=b+10;                         /* Speed up the right motor */
              move(b,a);              /* If left is faster        */
            }
          else if (left_speed < right_speed)
```

```
            {
              b=b-10;                    /* Slow down the right motor  */
              move(b,a);          /* If left is slower          */
            }
          }
      }
}


void move(int right, int left)          /* One command for both motors */
{
      motor(0,right);
      motor(1,left);
}

void wait(int milli_seconds)                    /* Pass time and allow other */
                                                /* Processes to execute          */
{
      long timer_a;

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

/*  The demonstratioin program */
/*  Global variables declared  */

int MAX_SAMPLE;
int Avoid_LF;
int Avoid_RT;
int Avoid_Priority;
int Follow_LF;
int Follow_RT;
int Follow_Priority;
int LF_IR=0;
int RT_IR=1;
int NEW_LF;
int NEW_RT;
int LF_MOT_POW;
int RT_MOT_POW;

void main() {                   /* Start all processes */
  start_process(avoid());
  start_process(follow());
}

void avoid() {                  /* Declare local variables */
  int Threshold = 128;
  int LF_DAT;
  int RT_DAT;
```

```
 while(1) {                          /* Loop forever */

  LF_DAT = analog(LF_IR);          /* Read the sensors */
  RT_DAT = analog(RT_IR);

  if (LF_DAT > Threshold) {        /* If obstacle to the left */
    Avoid_RT = 0;                  /* Turn right              */
    Avoid_LF = 50;
    Avoid_Priority = 100; /* Set priority           */
    arb();                         /* Call for arbitration    */
    }

  else {                           /* If no obsacle then go forward */
    Avoid_LF = 50;
    Avoid_RT = 50;
    Avoid_Priority = 20;   /* Low priority */
    }

  if (RT_DAT > Threshold) {        /* If obstacle to the right */
    Avoid_RT = 50;                 /* Turn left                     */
    Avoid_LF = -50;
    Avoid_Priority = 100; /* Set priority                  */
    arb();                         /* Call for arbitration          */
    }

  else {                           /* If no obstacle go forward */
    Avoid_LF = 50;
    Avoid_RT = 50;
    Avoid_Priority = 20;   /* Low priority */
    }
 }

}
                                        /* Follow sound process */

void follow()
{                                       /* Declare local variables */
  int MAX_VAL;
  long Start_time;
  long Stop_time;

  while(1) {                       /* Loop forever */

    Follow_LF = 0;                 /* Don't move   */
    Follow_RT = 0;
    Follow_Priority = 0;    /* Low priority */
    arb();                         /* Call arbitration */

    sample();                      /* Take a maximum reading from the microphone */

      while(MAX_SAMPLE > 160) {

                                        /* Do while sound is detected */
        Follow_LF = 50;   /* Turn right */
```

```
        Follow_RT = 0;
        Follow_Priority = 50;
        arb();              /* Call for arbitration */

        sample();           /* Take a sample */

        if (MAX_SAMPLE > MAX_VAL) {
          MAX_VAL = MAX_SAMPLE;   /* If the sample is increasing */
        }                                   /* Keep turning right */

        else {                              /* If sample starts decreasing */

          reset_system_time();
          Start_time = mseconds();
          Stop_time = Start_time + 100l;

          while(mseconds() < Stop_time) {
            Follow_LF = -50;                /* Rotate left for 100 msec */
            Follow_RT = 50;                 /* To correct for overshoot */
            arb();
            }

          reset_system_time();
          Start_time = mseconds();
          Stop_time = Start_time + 1000l;

          while(mseconds() < Stop_time) {
            Follow_LF = 50;                 /* Move forward for 1 sec */
            Follow_RT = 50;
            arb();
            }

          MAX_VAL = 0;
        }                                   /* Repeat the process */
      }
    }
}



void sample()                               /* Finds the maximum sample   */
{                                           /* In 15 samples from the mic */
  int I=0;
  int val;
  MAX_SAMPLE = 0;

  while(I < 15) {
    val = analog(2);
    if (MAX_SAMPLE < val) {
      MAX_SAMPLE = val;
    }
  I++;
  }
}
```

```
/* Arbitraion Process */

void arb() {

    if (Avoid_Priority > Follow_Priority) {
        NEW_LF = Avoid_LF;      /* If Avoid process is higher priority */
        NEW_RT = Avoid_RT;      /* Move according to Avoid process     */
      }

    else {                                  /* If Follow process has higher priority*/
        NEW_LF = Follow_LF;     /* Move according to Follow process              */
        NEW_RT = Follow_RT;
      }

    move(NEW_RT, NEW_LF);               /* Activate the motors */

}
```