JAMES HOFFACKER

FINAL ROBOTICS REPORT
TUCKER/BUDDY

University of Florida

12/12/97

EEL5666

TABLE OF CONTENTS

ABSTRACT

Teamwork is the main theme of my robotics project. A smaller “child” robot (Buddy) with a limited sensor array asks for help from a larger, well-equipped “mother” robot (Tucker) when it gets in trouble. This communication is performed using 40KHz and three piezo buzzers with a tone decoder circuit. This circuit is the key element of my project and allows robots to be controlled using various tones. These tones allow a whole new medium for communications and do not interfere with each other like IR.

EXECUTIVE SUMMARY

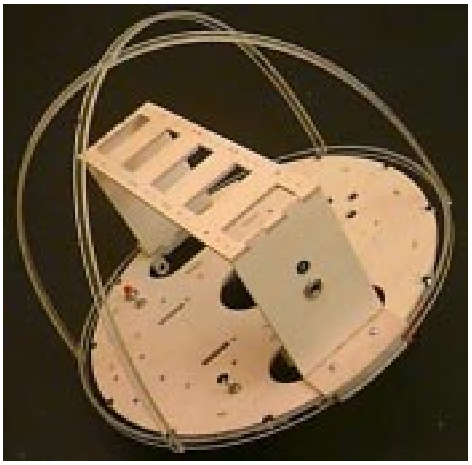
To create Tucker and Buddy, I used two known platforms to allow for easier integration. I first integrated basic object avoidance and collision avoidance on each robot to allow for flexibility in my software. I then integrated the more advanced systems such as the tone detection and tone generation. To perform these behaviors, each robot has two motors mounted on a round platform. Using this type of platform allows for easy obstacle avoidance. To gather sensory information from the environment, both robots use Sharp IR detectors. The TJ also integrates bump switches as an additional behavior.

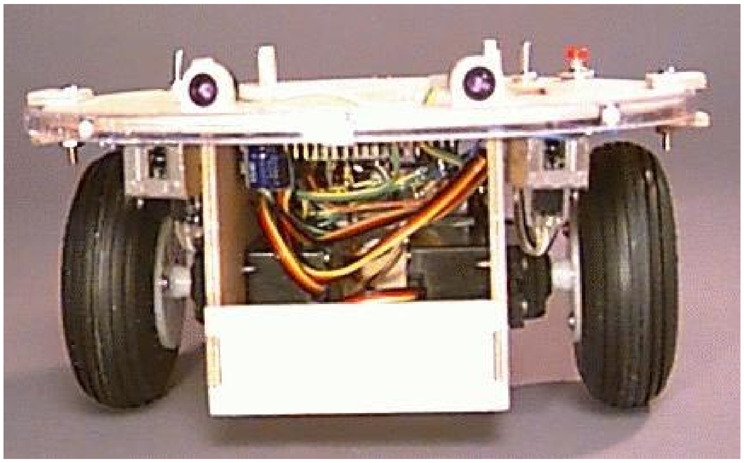
Tucker also uses transistors to switch battery voltage to the piezo buzzers allowing for higher decibel signals than are available from the board’s 5 volt Output Ports. Buddy uses a high gain amplifier stage and a condenser microphone to receive and amplify the signals for input into his board.

The brain of the two robots is the Motorola 68HC11 microprocessor. Buddy uses a single-chip board with 2K of EEPROM and Tucker uses an EVBU board with an expanded 32K of Ram.

INTRODUCTION

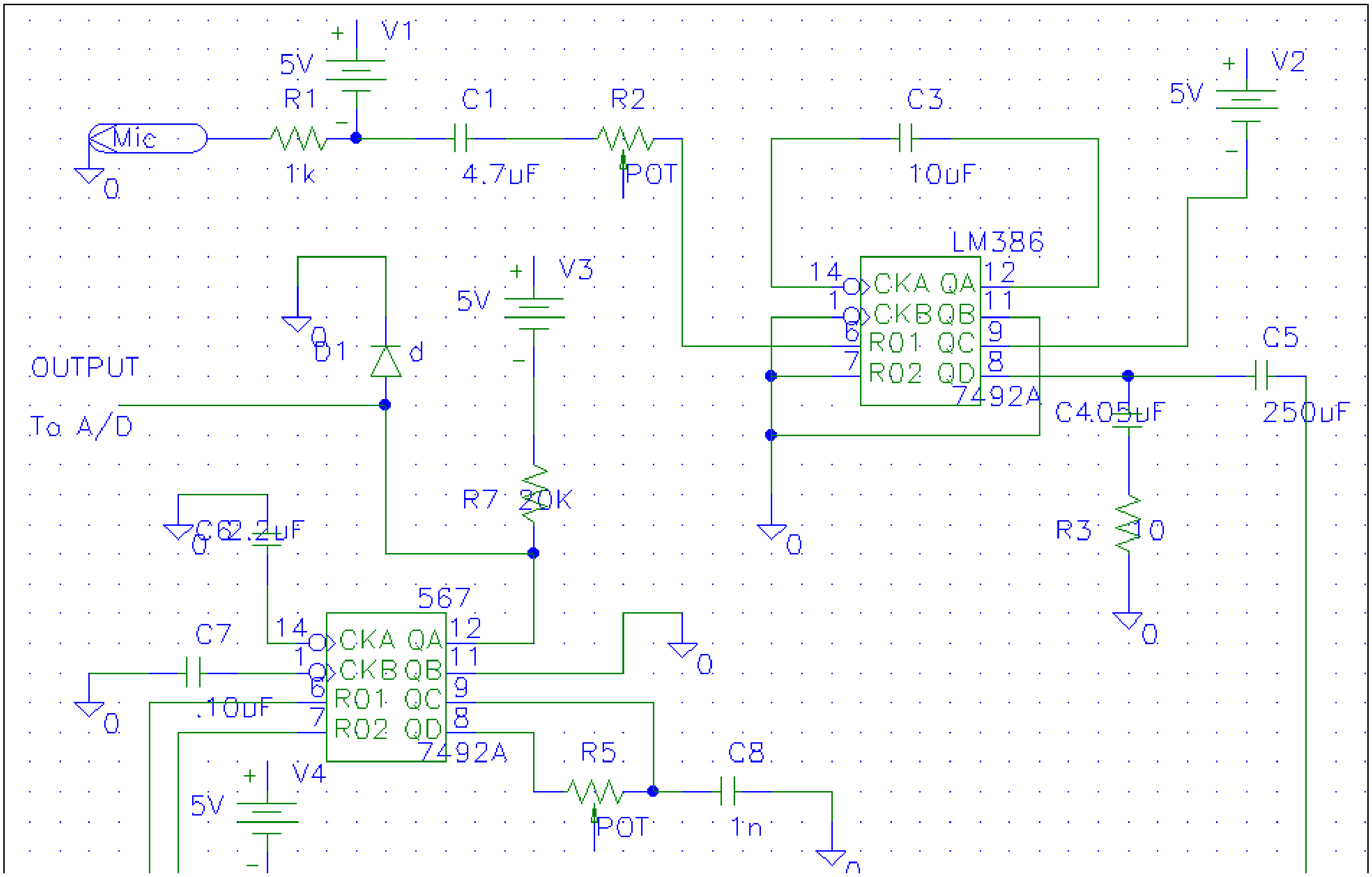
The goal of using a mother and child robot was to mimic human social behavior. I wanted to see if the two robots could effectively communicate with each other and use each other’s different strengths to complete a common goal. With this idea in mind I went ahead with the production of Tucker and Buddy.

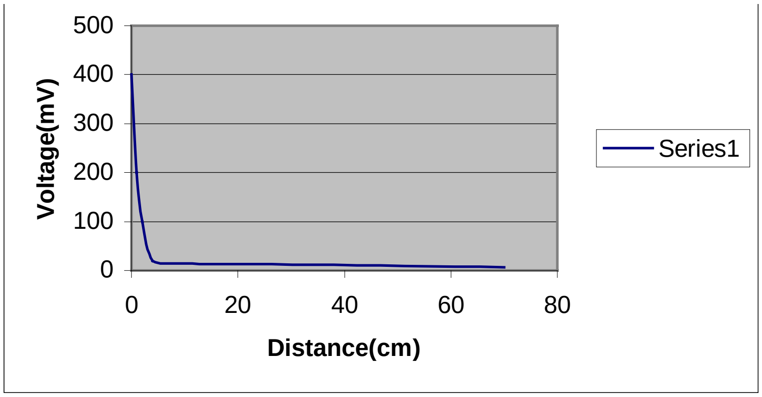




Actuation and Sensor Arrays

The smaller child robot will be based on the TJ platform and will incorporate an IR emitter and two eyes that can sense light via hacked CDS cells. This child will also have bumpers but will not have IR detectors. This lack of IR detection will render the child blind to walls and similar obstructions. The child will also have a bright light on its top as well as a relatively loud buzzer, which will emit a whining sound.





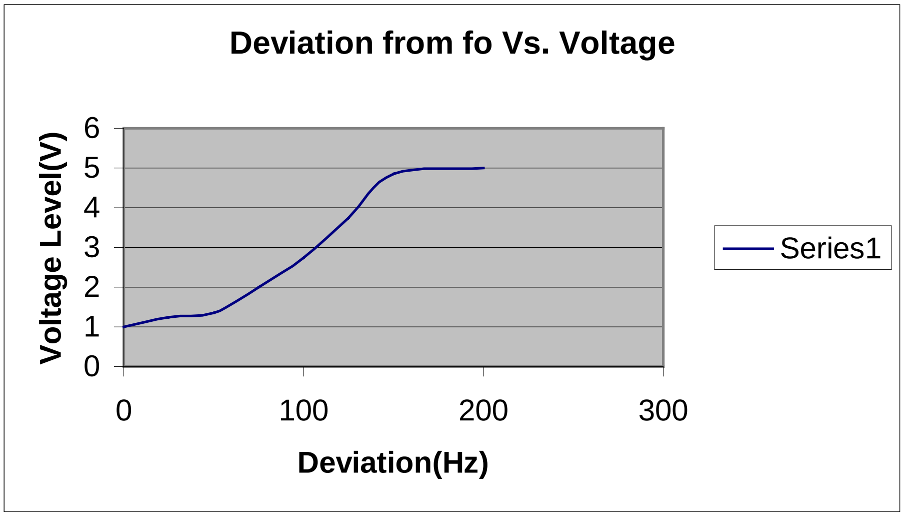
The larger robot will be the child’s mother and will have a more impressive array of sensors. Based on the Talrik, it will have CDS cells for light sensing as well as IR emitters and sensors for object avoidance. A listening device will be fitted to the top, as well as a bright light on the its back for the child to follow.

Detailed Tone Decoder Description

The three frequencies that the child robot will produce are 2.5Khz, 3.5KHz and 4.5Khz. These frequencies were chosen because they are readily available in piezo buzzers and are not normally observed in everyday sound. They are also reproducible at high levels with the relatively low voltage levels that are available using NiCad batteries.

To detect these tones, I am using a condenser microphone available from Radio Shack. Its Signal/Noise Ration is 60dB, which is higher than needed for frequency detection of this type. Once the tones enter the microphone they are filtered by a capacitor to eliminate any DC voltage that may be associated with the signal. Once the signal has been filtered, it is sent to a LM386 audio amplifier. This amplifier has an adjustable gain using external resistors and capacitors. The gain is set to 200V/V to amplify the signal received by the microphone. The reason such a high gain is needed will be shown later. After the gain stage, the signal is fed into a 567 tone detector. This tone detector is set using external components and will only detect frequencies within ~14% of the center frequency. So for example, my signal of 3.5KHz has an error range of +/-500Hz. This range is in agreement with the rating of the piezo buzzers which are typically +/-300Hz. Once the tone detector detects the frequency, the output voltage level drops to zero. This will set a flag on one of the mother’s ports letting it know that a message has been sent. Using these tone detectors eliminates external noise signals from triggering the flag and virtually eliminates error.

The greatest advantage of using multiple tone detectors and multiple piezo buzzers is the ability to send multiple messages. Using simple AND Gates or a MUX, up to seven different messages can be sent using all three piezo buzzers. This opens up



many different possibilities, such as the child robot giving the mother directions via tones in addition to calling for help.

Integrated System

As stated in the introduction, the basic goal of this project was teamwork. The child/mother system allows the TJ(child) to call for help and be rescued when it gets into trouble. Trouble will be defined as running into an obstacle and setting off the TJ's bumpers. The scenario will unfold as follows:

Both robots will begin in the same area. The child robot(TJ) will be active at the start, and as children are naturally curious, this child will seek an unknown part of the room. Invariably, the child will get itself into trouble and run into an object setting off its bumper. When this bumper is hit the child will stop, its IR will go on and it will emit a modulated IR to alert the mother robot(idle thus far) that it is in trouble. This whine will trigger the mother robot's microphone and put it in search mode. This will cause the mother to go begin a scan of the room for the child's IR. The mother will then track the child robot down using Sharp IR sensors. Eventually the mother will find the child and give it a home tone. This will change the mode of both robots to lead/follow respectively; the mother will turn its rear light turns on and the child will follow by activating its IR sensors and following the mother's light. The mother's will then safely lead the child back to their original position. The original position will be identified by 3KHz, with a timing delay so it does not interfere with the initial search of the child.

CONCLUSION

The scenario described above is what my project has accomplished. Because there is such a great deal of hardware on my system, there are other endless possibilities for behaviors. I had some trouble with my tone decoder circuit, and this did not allow me to use its full range of about 2.5 meters. I was still able to complete the scenario, it was just under tighter quarters.

CODE

BUDDY

```
/*
  10.28.96
  This program was written for a Talrik Junior platform in order to
  achieve collision avoidance. Ivan Zapata
 */
/*  MODIFIED 12/12/97  BY M. JONES AND J. HOFFACKER */
#include <servotj.h>
#include <hc11.h>
#include <mil.h>
#include <irtj.h>
/*#include <serial.h>*/
#include <analog.h>
#include <vectors.h>

#define ZEROL  3000   /* Servo signals */
#define ZEROR  3000
#define FORWL  2400
#define FORWR  3600
#define BACKL  3600
#define BACKR  2400
#define SENSORMIN 84
```

```
#define SENSORMAX 116
#define LEFT_SERVO 0
#define RIGHT_SERVO 1
#define SELF  0
#define OTHER 1

/* These two variables will be set and cleared by the servo
   interrupt handlers, interrupts must be allowed to occur when at
   least one of these is set                              */

int home,play,hold;


extern servos_busy;

void wait();
void turn(void);
void follow(void);

int main(void)
{
  int rval, lval, rlast, llast, rdes, ldes, rcurr, lcurr;
  int vnorm, offset, localflag, followflag;
  int i, j;
  int rspeed[21], lspeed[21];
  char c;

/*  init_serial();*/
  init_servos();
  init_analog();
  init_ir();
  ir_mode(SELF);

  hold=home=0;
  play=1;

  DDRC = 0xDF;          /* 1101 1111 bit 5 of PORTC is input */

/* The full range of speeds for each motor will be stored
   in rspeed[] and lspeed[] */
  followflag = 0;

  rval = FORWR;
  lval = FORWL;
  for (i = 20; i > -1; i--)
  {
    rspeed[i] = rval;
    lspeed[i] = lval;
    rval -= (FORWR-BACKR)/20;
    lval += (BACKL-FORWL)/20;
  }
  rspeed[19] = rspeed[20];
  lspeed[19] = lspeed[20];

  vnorm = (2000/(SENSORMIN - SENSORMAX));
  offset = 10 - ((SENSORMIN * vnorm)/100);
  rval = lval = SENSORMIN;
  rcurr = rlast = rdes = rspeed[20];
  lcurr = llast = ldes = lspeed[20];

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

  while(1)
  {

/* servo0 = left, servo1 = right */
/* PE6 is on the right, PE7 is on the left */

    if((analog(1) < 75)||(home))
    {
      ir_mode(OTHER);
      play = 0;
      home = 1;
      follow();
    }
    else
    {
     if((analog(2) < 75)||(play))
      {
       ir_mode(SELF);
       play = 1;
       home = 0;
      }
/************************************/
    rval = ir_value(6);             /* Get sensor readings */
    lval = ir_value(7);

    rval = ((rval*vnorm)/100) + offset;    /* Calculate indexes */
    lval = ((lval*vnorm)/100) + offset;


    rdes = rspeed[10+rval];             /* Get duty cycle from table */
    ldes = lspeed[10+lval];


    rcurr = rlast + ((rdes - rlast) >> 1);   /* Approximate desired speed */
    lcurr = llast + ((ldes - llast) >> 1);

    servo(RIGHT_SERVO, rcurr);                     /* Set actual servo speed */
    servo(LEFT_SERVO, lcurr);

    rlast = rcurr;
    llast = lcurr;



    }
    if(PORTC & 0x20)
    {
      TMSK1 = TMSK1 | 0x20;
      ir_mode(OTHER);
      servo(LEFT_SERVO, BACKL);
      servo(RIGHT_SERVO, BACKR);
      for(i = 0; i < 5000; i++);
      turn();
      wait();
      ir_mode(SELF);
    }
```

```
  }
}

void follow(void)
{
  int folr, foll;
  folr = analog(7);
  foll = analog(6);
  ir_mode(OTHER);
  if (analog(2) < 75)
   {
    play = 1;
    home = 0;
   }
  if (foll + 5 < folr)
    {
      servo(LEFT_SERVO, 2300);
      servo(RIGHT_SERVO, 3200);

    }
  else if(folr + 5 < foll)
    {
      servo(LEFT_SERVO, 2800);
      servo(RIGHT_SERVO, 3500);
    }
  else
    {
      servo(LEFT_SERVO, 3100);
      servo(RIGHT_SERVO, 3150);
    }
}
/**********************************/

void wait()
 {
   int tone1,tone2,tone3;
   int i;
   i = 0;
   servo(LEFT_SERVO, 3000);
   servo(RIGHT_SERVO, 3000);

   while((i < 20000) || (hold))
     {

        tone1 = analog(1);
        tone2 = analog(2);
        tone3 = analog(3);
        if(tone1 < 50)
          {
            TMSK1 = TMSK1 & 0xDF;
            home = 1;
            play = 0;
            break;
          }

        else if(tone2 < 50)
          {
            TMSK1 = TMSK1 & 0xDF;
            play = 1;
            home = 0;
            break;
          }
/*         else if(tone3 < 50)
          {
            TMSK1 = TMSK1 | 0x20;
            play=home=0;
            hold = 1;
          }             */
        i += 1;
     }
     if((!hold) && (!home))
     {
        TMSK1 = TMSK1 | 0x20;
        play = 1;
     }
 }
void turn()
{
 int i;
 unsigned rand;

 rand = TCNT;

 if (rand & 0x0001)
 {
   servo(RIGHT_SERVO, FORWR);
   servo(LEFT_SERVO, BACKL);
 }
 else
 {
   servo(RIGHT_SERVO, BACKR);
   servo(LEFT_SERVO, FORWL);
 }
 for (i = 0; i < rand; i++);


}
```

TUCKER

```
/*  FOR TALRIK PLATFORM:
CODE WRITTEN BY J. HOFFACKER/M. JONES FOR BASIC
OBSTICLE AVOID AND 32KHz AND 40KHz FOLLOWING:
EASILY ADAPTABLE, 12/12/97*/

#include <analog.h>
#include <motor.h>

#define IR_LEDS *(unsigned char *)(0x7000)

#define FAST 90
#define MED 60
#define SLOW 30
#define CREEP 10
#define STOP 0
```

```
int threshold, thres2;
int ir_FL, ir_FR, folflag;



void go_forward(int speed)
{motor(0, speed);
 motor(1, speed);
}

void go_left(int speed)
{motor(0, -speed);
 motor(1, speed);
}

void go_right(int speed)
{motor(0, speed);
 motor(1, -speed);
}

void stop()
   {
     go_left(STOP);
     go_right(STOP);
   }


void follow()
{
  ir_FR=analog(2);
  ir_FL=analog(3);
  if ((ir_FL > 95) || (ir_FR > 95))
    {
     if (ir_FL < ir_FR)
     {
      motor(1 , MED);
      motor(0, CREEP);
     }
     if (ir_FR < ir_FL)
     {
     motor(0 , MED);
     motor(1 , CREEP);
     }
    }
     else
  {
      go_left(CREEP);
  }
}

void follow32()
{
  ir_FR=analog(6);
  ir_FL=analog(7);
  if ((ir_FL > 95) || (ir_FR > 95))
    {
     if (ir_FL < ir_FR)
     {
      motor(1 , MED);
      motor(0, CREEP);
     }
     if (ir_FR < ir_FL)
     {
     motor(0 , MED);
     motor(1 , CREEP);
     }
    }
     else
  {
      go_left(CREEP);
  }
}


void homet()
 {
   int i;
   IR_LEDS = IR_LEDS | (0x01);
   for (i=0; i < 5000; ++i)
   {
      if(i < 1250)
      {
         go_left(MED);
      }
      else if(i < 3750)
      {
         go_right(MED);
      }
      else
      {
         go_left(MED);
      }
   }
   stop();
   IR_LEDS = IR_LEDS & 0xFE;
 }

void playt()
 {
   int i;
   IR_LEDS = IR_LEDS | (0x04);
   for (i=0; i < 5000; ++i)
   {
      if(i < 1250)
      {
         go_left(MED);
      }
      else if(i < 3750)
      {
         go_right(MED);
      }
      else
      {
         go_left(MED);
      }
   }
   stop();
   IR_LEDS = IR_LEDS & 0xFB;
 }
```

```
void obsavoid()
   {
     ir_FR=analog(2);
     ir_FL=analog(3);
     if ((ir_FL>threshold) || (ir_FR>threshold))
     {
        if (ir_FL>threshold) {go_right(MED);}
        if (ir_FR>threshold) {go_left(MED);}
     }
     else
     {
        go_forward(MED);
     }
   }


void main()
{
    int i,j;
    init_motors();
    init_analog();
    threshold=123;
    thres2 = 128;
    IR_LEDS = 0x00;
    folflag = 0;
    for(i=0; i < 30000; ++i)
     for(j=0; j < 4; j++);


    playt();

    while(1)
    {
      while((ir_FL < 110) && (ir_FR < 110))
      follow();

      IR_LEDS = 0x40;
      homet();
      IR_LEDS = (IR_LEDS | 0x40);
      while(1)
```

```
            follow32();



        }
    }
```