University of Florida

Dept. of Electrical and Computer Engineering

EEL 5666C

Intelligent Machines Design Laboratory

Final Paper

December 4, 2001

Lesley Boylston

Table of Contents

A. Abstract

A.J. is an autonomous mobile robot that will move around a room detecting the color of the objects it finds. It will use Cadmium Sulfide photocells as color detectors. This robot does not have any purpose other than entertainment.

Executive Summary

Albert Junior, named after the mascot of the University of Florida, is a mobile, autonomous, robot. He is made of wood and is in the shape of an alligator. He has three main parts to his body: head, torso, and tail. A.J. moves around a room seeking out objects to attack. If they are orange or blue, he wags his tail happily and moves on looking for red and yellow FSU fans to chew on. He has no practical purpose; he is for entertainment only.

He uses CdS cells to detect color and IR detectors to detect objects. He has three IR detectors on his head so he can see on his left and right as well as ahead. He has bump switches so signals can be sent to A.J. to trigger behaviors. His behaviors include: wagging his tail, moving his head, backing up, moving forward, and detecting color. He moves around using two servos hacked to be motors. A.J. operates using eight rechargeable NiCad batteries.

Introduction

This purpose of this project was to build an autonomous robot in the shape of our school mascot – the Alligator. The objective is to have the robot, Albert Junior (A.J.) move around looking for objects. If the object he finds is orange or blue it is deemed friendly and he wags his tail. If the robot finds a red or yellow object, A.J. determines it to be an FSU fan and moves towards it aggressively. Otherwise, he backs away from the object and continues searching. This paper will describe the requirements for my robot and how they were met.

Integrated System

A.J.'s brain is an HC11 microprocessor on the MRC11 board. His behaviors are based on the following software structure:

- Routines to initialize all systems
- A calibration routine to calibrate the color sensor
- Routines to move him forward and move his head and tail
- A routine to check the sensors
- A routine to check color of object detected

A.J. determines which behavior to follow based information received from his sensors that include infrared emitters/detectors, bump switches, and CdS cells. A.J. communicates to the world using action and LED lights.

Please see figure marked Flowchart for a flowchart of his behaviors and responses.

Mobile Platform

A.J.'s body was designed to resemble an alligator. It is made of four parts including a head, tail, body, and mounting platform. Please see the figure marked Body.

The only requirement for the body was that all 3 pieces be able to fit in a 17 inch x 11-inch rectangle.

The body was designed keeping in mind that the wheels, motors, servos, and IR detectors would have to be mounted. It did not take into account the color sensor, LED panel, bump switches or the various other switches needed. It would have been better to design for the switch holes instead of drilling them as an afterthought.

Please see figure marked Schematic for the platform schematic.

Figure: FLOWCHART

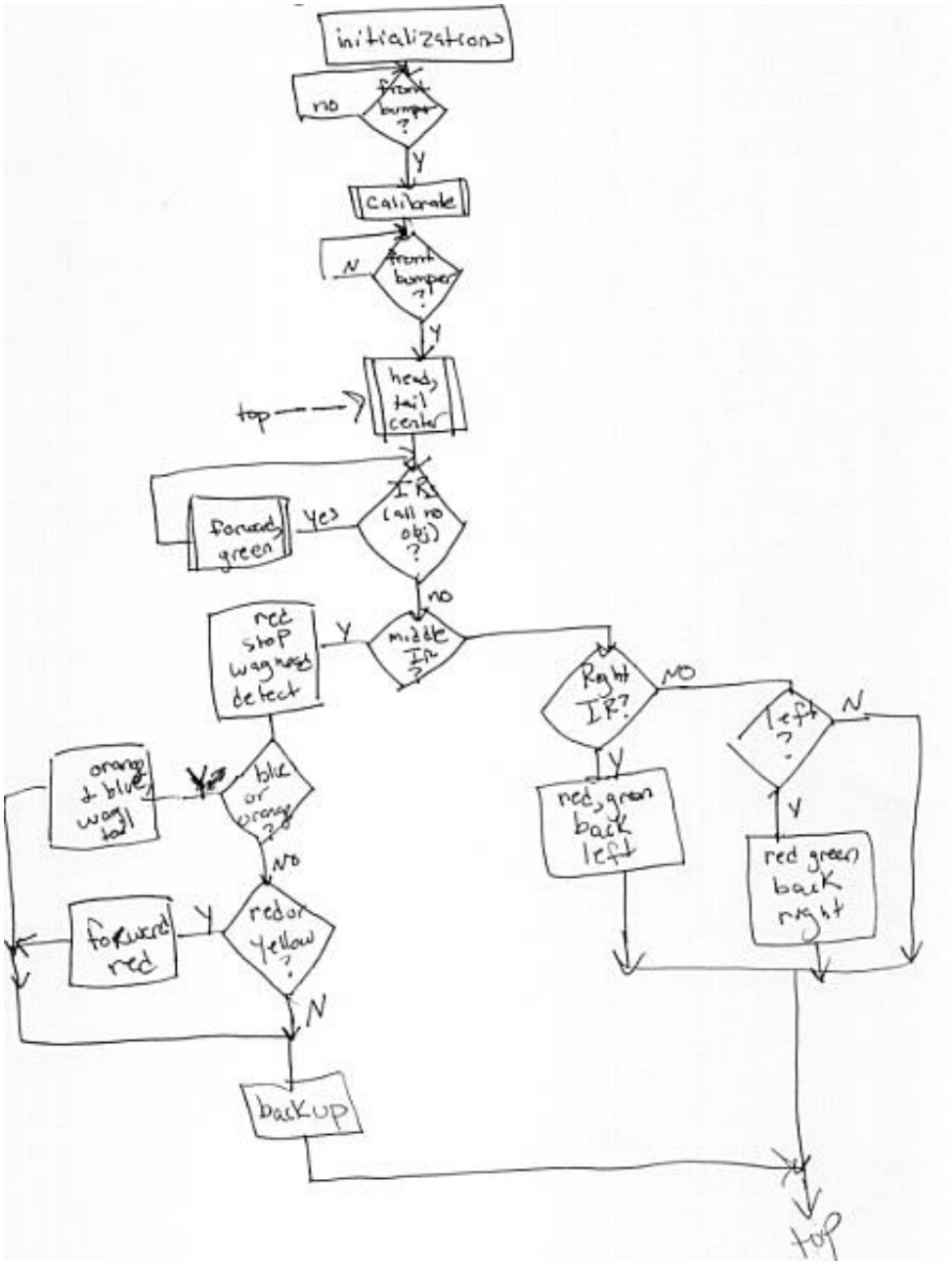


Figure: BODY

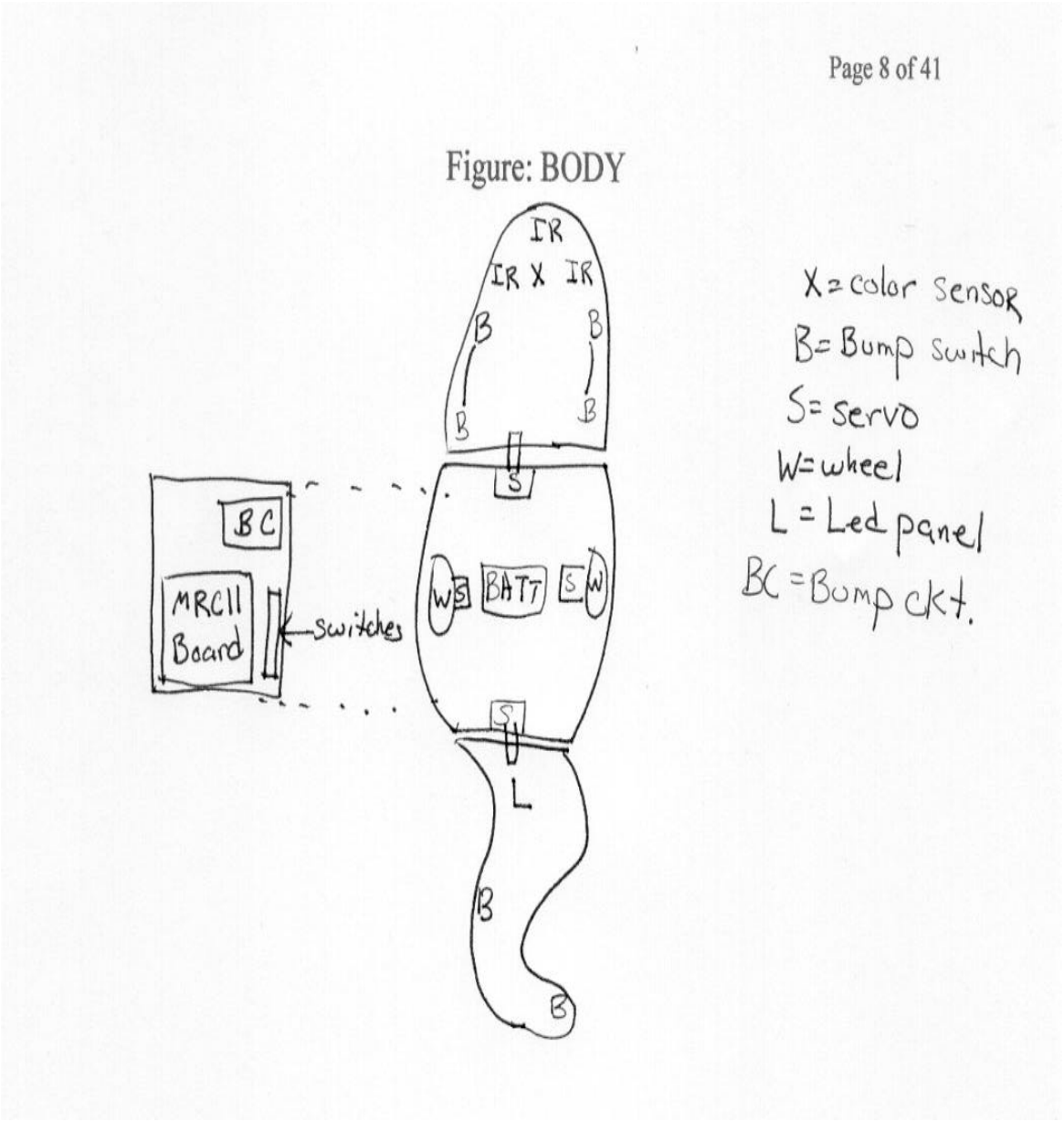
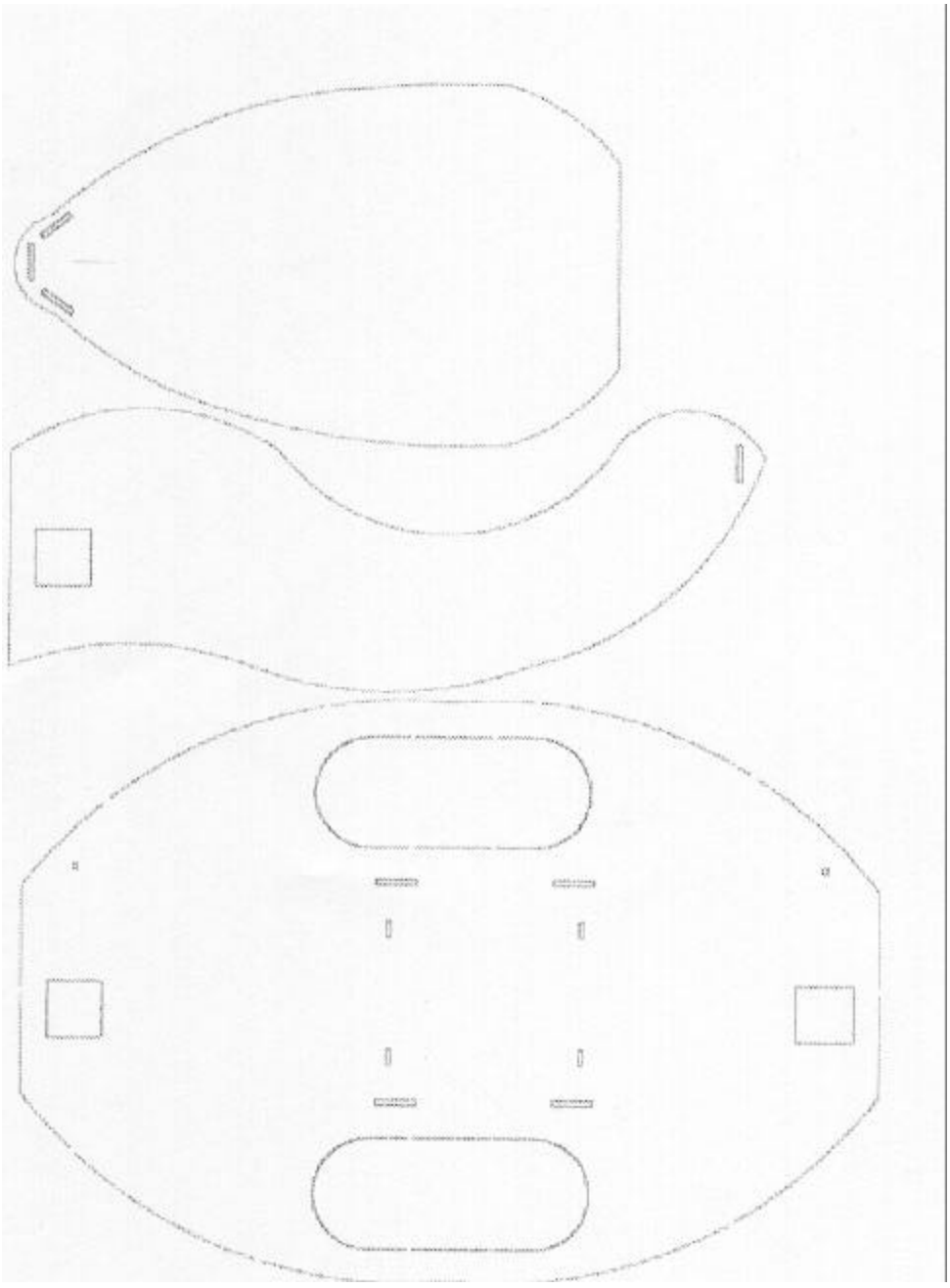

Page 8 of 41
Figure: BODY
IR
IR X IR
B
B
B
B
S
BC
MRCII Board
switches
WS
BATT
S
W
S
L
B
B
X = color sensor
B = Bump switch
S = servo
W = wheel
L = Led panel
BC = Bump ckt.

Figure: SCHEMATIC



Actuation

A.J. is mobilized by using two hacked servos for motors purchased from Mekatronix. This involved removing the stop from the servo and grinding out the inside of the shaft so that it is not connected to the potentiometer. The servos were then calibrated by sending it a pulse width modulated signal of approximately 10 percent of its period, which is about 20 milliseconds. The potentiometer is then turned until the motion of the servo stops. This worked for the right motor. The left motor would occasionally move backward when it should be moved forward. It was recalibrated and it worked better being calibrated at 15 percent. This did not solve the entire problem. The function to move the left servo forward had to be put directly in the same file as the main program in order to work properly all of the time. The motion of A.J. is programmed using several functions. Each function assigns a pre-determined percentage of the duty cycle to the appropriate variables. Please the code titled “motor.c”.

The head and tail are moved using two unhacked servos. The functions to move them involve assigning the appropriate variables the number of E-clocks for which the output compare should be high. Please see the code title “servo.c”.

The servos (both hacked and unhacked) require three signals. Power, ground, and input signal. The power for the servos must come from the battery pack because the voltage

regulator on the board cannot source enough current for the servos. Please see the figure titled Bump Circuit for the wiring of the servos.

The motors and servos are controlled using the output compare system of the HC11. The motors and servos are interfaced with the board as follows in table 1:

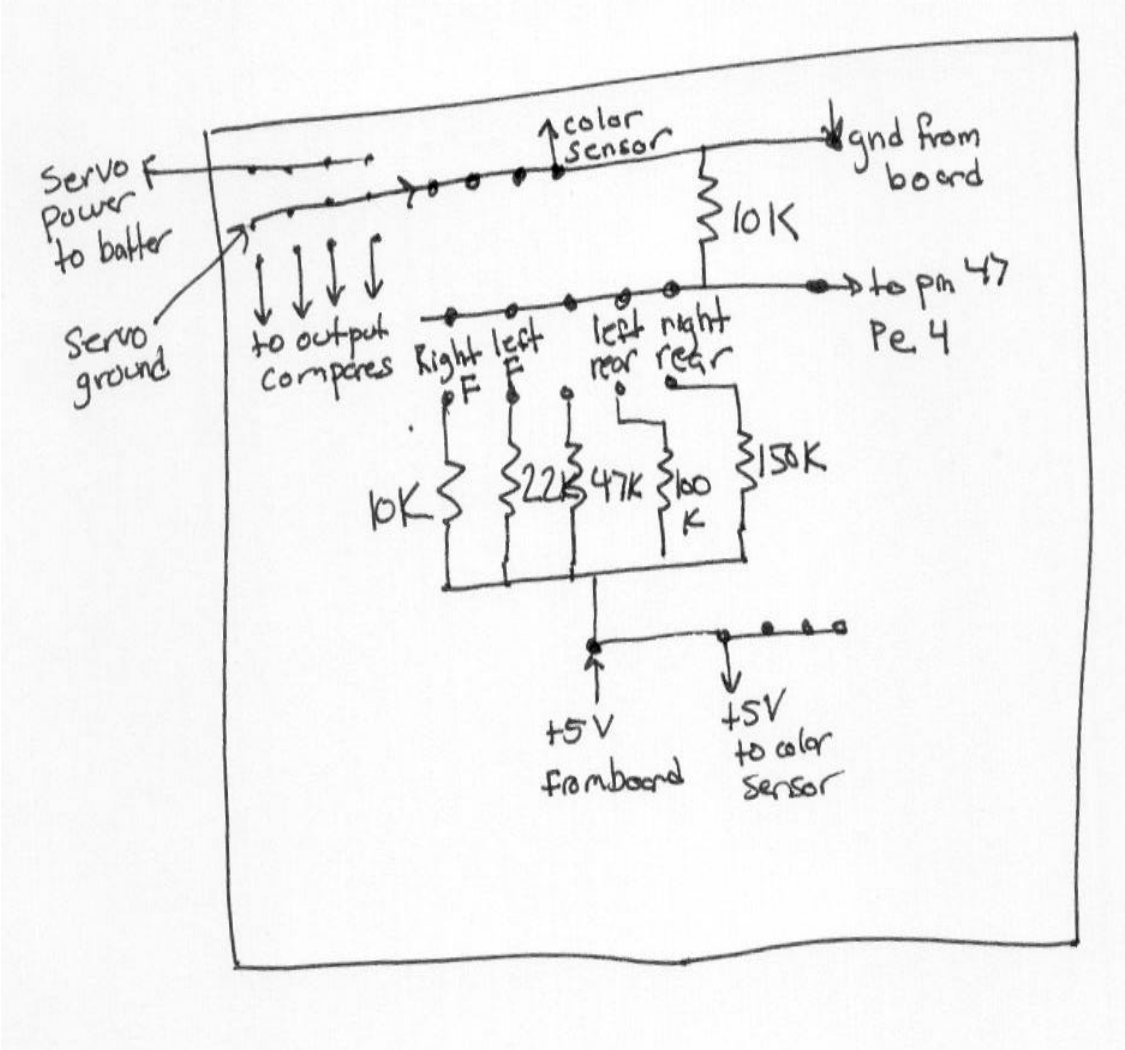
	Left Motor	Right Motor	Front Servo	Back Servo
Pin number	29	28	30	31
Output Compare	3	2	4	5
Port A number	5	6	4	3

Table 1

Sensors

IR detectors. A.J.'s eyes are comprised of three IR detectors. They are used on A.J.'s head in order to detect the presence of objects nearby. The left and right IR detectors are used to detect objects on either side. If sensed A.J. backs in the appropriate direction so he can move towards the object. The middle IR is used to detect when A.J. is directly in front of an object so that it's color can be read. The object is approximately 3 inches

Figure: BUMP CIRCUIT



away when A.J. senses it because that is the closest range to which the sensors are accurate. A.J. receives information from the IR's using the analog to digital port on the HC11. Please see the code titled "ir.c" for the code used to check the IR sensors. They are interfaced as follows in table 2:

Position	Analog Port	Port E	Pin #
Left	1	1	44
Middle	2	2	45
Right	3	3	46

Table 2

Bump Switches. The bump switches are used in case of failure of the IR's and to send "go" signals to A.J. The bumper for them never was made. This was due in part to the fact that the body of A.J. has many curves and the bumper needs to be flexible enough to follow these curves but sturdy enough to be a good bumper. Two bump switches are placed on the either side of the head. The bump switches on each side are wired together. There are 2 bump switches on the tail. All of the bump switches are wired together so that only one analog port needs to be used. Please see the figure marked Bump Circuit for a schematic of this wiring.

Color Sensor (Cadmium Sulfide (CdS) cells). Three CdS cells were used to make the color sensor. CdS cells are resistors whose resistivity changes based on the amount of light seen on the surface of the cell. This property of CdS cells can be used to detect color because objects of different color reflect different amounts of light.

Certain textures or finishes on objects can affect the light reflected as well, resulting in inaccurate readings. Color lenses are used to filter the light coming into the cells to reduce this error. A pack of color slides was purchased from Edmund's Scientifics (see references), each placed in front of a test cell, and the measure of the resistance of the cell was taken. This resistance is translated into a voltage which can be measured by the analog –to-digital port on the HC11 when the CdS cells are wired as in the figure marked CdS configuration. The original idea was that each cell would be used to detect a specific color: blue, orange, or black. (The black was originally intended to be used as the walls of the stadium he would move around in). The color filters were chosen for each cell so that each one would give a unique reading for each color. The following table shows the final choice of lenses and the expected resistance of the CdS cells for an object of each color.

CdS cell #	lens number	orange	blue	red	yellow	black	color detected:
1	818	99	166	72	63	172	blue
2	856	71	47	47	36	52	orange
3	823	249	291	154	158	340	black

Table 3: all readings in kilohms

For example, it can be seen that for a blue object the first cell gives a reading considerably more than the second does and less than the third. The reading is also very different from any of the readings for that cell for other color objects. Cell two would not be good to detect blue because it gives a reading closely equal to that of a red object but it is good for orange. The original theory was that white LED's would be used to provide the object, whose color is being detected with consistent lighting, thereby minimizing the effects of ambient light on the color sensor. As previously stated, this was the original plan.

In actuality, CdS cells are more sensitive than originally thought. Each cell was surrounded by heat shrink and the color lens placed across the front of the cell but this was not enough to prevent ambient light from affecting the resistance of the cells. The only way that I found to fix this problem was to use a skirt around the cells and place the object directly against the skirt. This was not acceptable to me because the skirt would stick out the front of the robot due to the placement of the color sensor and would not be able to be hidden in any way. It was decided that a slightly different approach would be better.

I wrote a calibration routine. This allows for changes in the lighting to be factored into the readings. Since the readings taken by A.J. are compared to readings taken in the same room as opposed to fixed values, the results are much more accurate. This method leads to determining the color in a slightly different way. Instead of assigning each cell to detecting a specific color, each cell is now used to get readings different from the other

cells so that comparisons can be made. In other words, each cell is used to detect all colors. Each lens filters the same light from the same object differently giving different readings from each cell. By using a test program, it can be observed that the readings of each color fall in a certain numeric order for each cell. For cells one and two, an orange object gives a reading less than yellow which is less than red; then blue followed by black. For cell two the order is: yellow, orange, blue, red, black. A.J. detects objects by taking the reading from cell 1 and testing to see where it falls in the numeric order of colors. Cell 2 is then checked follow by cell three. A.J. then has at most 6 possibilities to choose from; each cell reports that its reading is between 2 colors. This is narrowed down by choosing which of each pair is closest to the value gotten from the calibration routine. The final color is chosen using an average of these three colors. In order for A.J. to select the colors mathematically, each one is assigned a number.

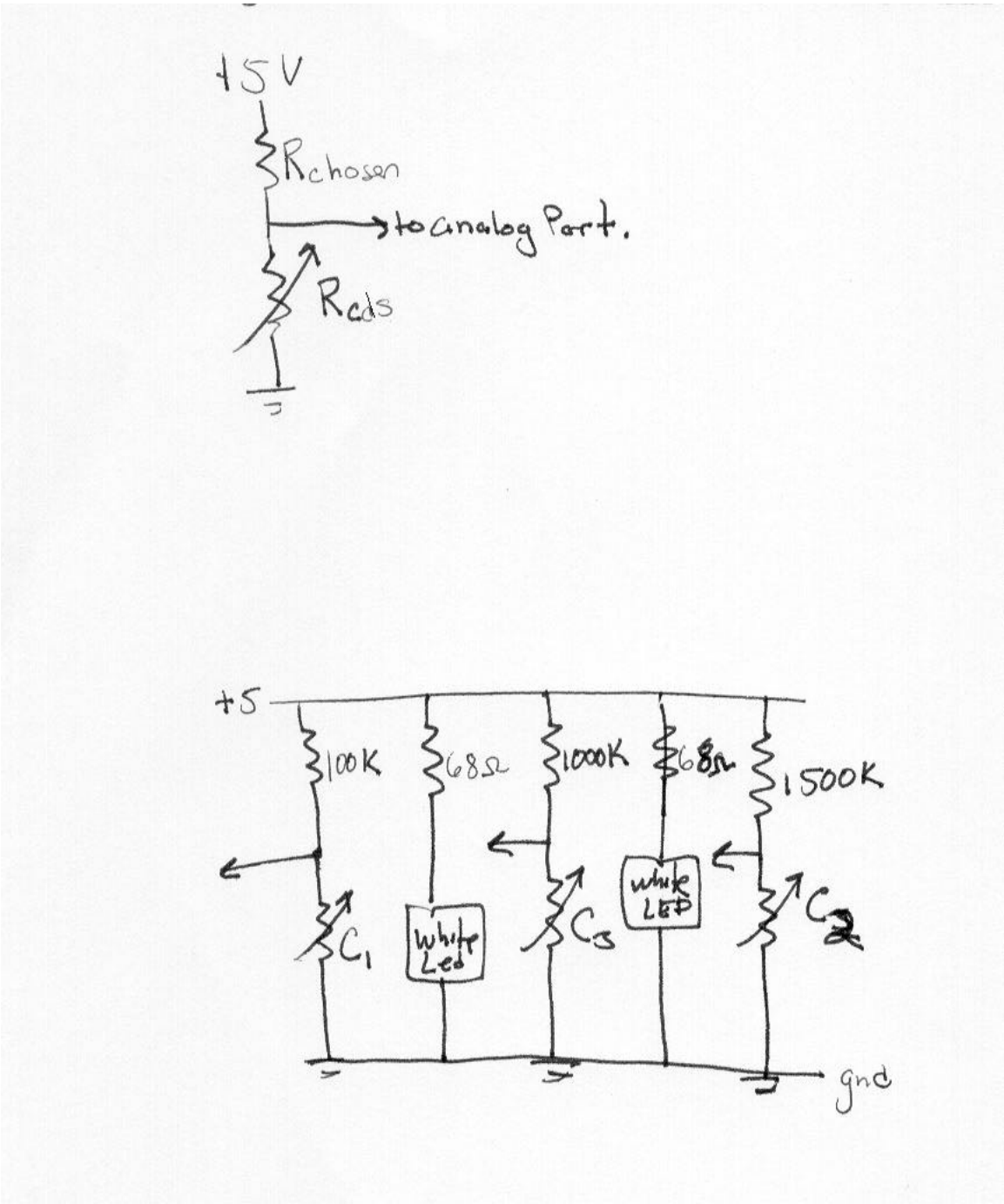
The second method of detecting color is considerably more accurate than the first. It is not 100% reliable but if correctly calibrated I would guess it is about 30% accurate. (I did not record the results each time A.J. set out to find objects so that is not a scientific guess.) A correct calibration consists of not placing the piece of paper that shows the distance away from the robot hat the object needs to be placed directly underneath the sensor. It is best to place it to the side because the floor reflects light and needs to be taken into account in the calibration. A.J. was not completely reliable because within a room, especially one with windows, the lighting from one area of the room to another can be different. The shadows of people and objects can also affect A.J.'s reliability.

Please see the test code titled "calibration.c" for the calibration routine.

Please see the page titled “output.txt” for sample readings and detection.

See the figure titled Color Sensor Circuit

Figures: CdS CONFIGURATION & COLOR SENSOR CIRCUIT



Behaviors

Object seeking/Color detection/Reactions. A.J. receives readings from all three of his IR sensors. If none of them detect an object nearby, he continues forward while lighting his green LED. The side IR's are set to detect objects far away. If either of those detect an object, A.J. backs up for a set amount of time in the direction opposite the side that the object is on, resulting in him facing the direction of the object. He lights the green and red LED's while doing this. He then goes back to heading forward. The middle IR is set to detect objects as close as the IR will accurately read. This is because I wanted A.J. to be as close to the object as possible for the best color readings. When A.J. senses and object he stops, turns on a red light, wags his head so it appears he is looking at the object (this is for audience entertainment; it is not necessary to get a reading.) When his head is done wagging, it goes back to center so this is when the color reading is taken. If the object is determined to be blue or orange he wags his tail while lighting his orange and blue LED's, then backs up randomly left or right and continues searching. If it is determined to be red or yellow he lights his red LED and goes forward for a set amount of time then backs up, turns, and continues searching. If none of the above are found to be true he simply backs away from the object and continues searching.

Please see the code titled "led.c" for the functions that light the panel and "main.c" for the main code that controls A.J.

See the figure titled flowchart for a flowchart of his behaviors and reactions.

Conclusion

A.J. is an autonomous robot capable of moving around a room without colliding with objects. When calibrated, he is capable of detecting colors of objects and determining what preprogrammed action to take. He uses IR sensors, bump switches and a color sensor to receive information from his surroundings. He reacts not only with actions but also by using lights. A.J. moves around using hacked servos attached to wheels and moves his head and tail using servos.

A.J.'s color sensor could be considerably improved. If given the chance, I would like to investigate other methods for detecting color that perhaps are so sensitive to light changes. I would also like to improve some of his movements. His backing up to head toward an object is not accurate and he sometimes over shoots his target. Sometimes the object he is looking at is not dead center and by turning his head, he could get a better "look" at the object. I would like to include code that would take IR readings as he turns his head and take note of any positions that give closer readings. When he is done wagging his head, it would return to the closer position not the center position. Then it would really appear he is truly "seeing" objects. I would also like to implement the top-jaw that would open and close using a servo as I had originally intended. Using objects that are small enough to fall into his mouth so that he could take them to a "home" base is one of my more lofty goals.

Sources

Mekatronix. 316 NW 17th Street, Suite A, Gainesville, FL 32603; www.mekatronix.com

Edmunds Scientific. 1-800-728-6999; www.scientificsonline.com

Appendix

B. main.c

C. calbr.c

D. ir.c

E. led.c

F. motor.c

G. servo.c

G. output.txt

Appendix A: main.c

```
//*************************************************
//*** THIS PROGRAM EXECUTES COLLISION AVOIDANCE   ***
//*** WRITTEN BY: LESLEY BOYLSTON                 ***
//*** NOVEMBER 13, 2001                          ***
//*************************************************


//****************************************INCLUDES***************
#include <mil.h>
#include <hc11.h>
#include <analog.h>
#include "motor.c"
#include "ir.c"
#include "led.c"
#include "servo.c"
#include "calbr.c"

//*****************************DEFINES*****************************
extern void _start(void); //for reset vectors

#define DUMMY_ENTRY (void (*) (void)) 0xFFFF //for interrupts

#define BUMPER analog(4)
//*************************PROTOTYPES*********************
void forward(int);

//*********************************MAIN*****************************
void main (void)
{
 int i,j,color;
 unsigned rand;

 rand = TCNT;

 //INITIALIZATIONS
 init_pwm();
 init_pwm2();
 init_analog();
 init_leds();

 //PRESS FRONT RIGHT BUMPER TO START
 while (BUMPER <120 );

 calibrate();

 while (BUMPER < 120);

 while(1)
 {head_center();
  tail_center();
  //WHILE NO OBJECTS GO FORWARD
  while(!(IRMcheck() || IRRcheck() || IRLcheck())){
   forward(10);
```

```
   green();}

  //DETECT WHERE OBJECT IS AND READ COLOR OR MOVE TOWARDS IT
  if(IRMcheck())
   {
    red();
    stop();
    head_wag();
    color = detect();
    if ((color == Blue) || (color == Orange))
     {blue_orange();
      tail_wag();}
    if ((color == Red) || (color == Yellow))
      {red();
       for(i=0;i<25000;i++)
        forward(10);}
    if(rand & 0x0001)   //randomly turn
     for(i=0;i<25000;i++)
      for(j=0;j<1;j++)
       back_2_left();
    else
     for(i=0;i<25000;i++)
      for(j=0;j<1;j++)
       back_2_right();
   }//end if

   if(IRRcheck())
    {
     red_green();
     for(i=0;i<10000;i++)
       back_2_left();
    }

   if(IRLcheck())
    {
     red_green();
     for(i=0;i<10000;i++)
       back_2_right();
    }
 }//end while
}//end main


//*************************************************************
void forward(int i){          //moves motors forward
 duty2 = 20;
 duty3 = i;
 }//end forward

//*************************************VECTORS******************
#pragma abs_address:0xffd6
  /* change the above address if your vector starts elsewhere */

void (*interrupt_vectors[])(void) =
        {

        DUMMY_ENTRY,   /* SCI, RS232 protocol */
```

```
        DUMMY_ENTRY,   /* SPI, high speed synchronous serial*/
        DUMMY_ENTRY,   /* Pulse accumulator input edge */
        DUMMY_ENTRY,   /* Pulse accumulator overflow */
        DUMMY_ENTRY,   /* Timer overflow */
        TOC5_isr,      /* TOC5 */
        TOC4_isr,      /* TOC4 */
        TOC3_isr,      /* TOC3 */
        TOC2_isr,      /* TOC2 */
        DUMMY_ENTRY,   /* TOC1 */
        DUMMY_ENTRY,   /* TIC3 */
        DUMMY_ENTRY,   /* TIC2 */
        DUMMY_ENTRY,   /* TIC1 */
        DUMMY_ENTRY,   /* RTI */
        DUMMY_ENTRY,   /* IRQ */
        DUMMY_ENTRY,   /* XIRQ */
        DUMMY_ENTRY,   /* SWI */
        DUMMY_ENTRY,   /* ILLOP */
        DUMMY_ENTRY,   /* COP */
        DUMMY_ENTRY,   /* CLMON */
           _start      /* RESET */
        };

#pragma end_abs_address

//*********************************************************
```

Appendix B: calbr.c

```
//*********************************************
//This routine calibrates the color sensor for***
//the room the robot is in.                   ***
//Programmer: Lesley Boylston                 ***
//Date: November 2001                         ***
//*********************************************


//Using CdS sensor on analog 5,6,7 pins 48,49,50

#define sensor1 analog(5)
#define sensor2 analog(6)
#define sensor3 analog(7)
#define BUMPER analog(4)

#define orangeindex 0
#define yellowindex 1
#define redindex 2
#define blueindex 3
#define blackindex 4

#define orangeindex2 1
#define yellowindex2 0
#define redindex2 3
#define blueindex2 2
#define blackindex2 4


#define Blue 1
#define Orange 2
#define Red 3
#define Yellow 4
#define Black 5
#define None 6

int cal1[5];
int cal2[5];
int cal3[5];

void calibrate(void);
void blink_orange(void);
void blink_red(void);
void blink_blue(void);
void blink_blue_orange(void);
int  within_range(int,int);
void wait(int secs);
int detect(void);

void calibrate(void){
//this functions calibrates the color sensor
int i,j,k,temp;

for(i=1;i<6;i++)
 {
```

```
clear_leds();      //blink red light 3 times
blink_red();

switch(i){
 case 1:{
  while(BUMPER < 20) red_blue();
  while(!(IRMcheck())) red_blue();
  blue();
  wait(2);
  blue_green();
  wait(2);
  cal1[blueindex] = sensor1;
  cal2[blueindex] = sensor2;
  cal3[blueindex] = sensor3;
  break;}
 case 2:{
  while(BUMPER < 20) red_orange();
  orange();
  wait(2);
  orange_green();
  wait(2);
  cal1[orangeindex] = sensor1;
  cal2[orangeindex] = sensor2;
  cal3[orangeindex] = sensor3;
  break;}
 case 3:{
  while(BUMPER < 20) red_orange();
  orange();
  wait(2);
  orange_green();
  wait(2);
  cal1[redindex] = sensor1;
  cal2[redindex] = sensor2;
  cal3[redindex] = sensor3;
  break;}
 case 4:{
  while(BUMPER< 20) red_blue();
  blue();
  wait(2);
  blue_green();
  wait(2);
  cal1[yellowindex] = sensor1;
  cal2[yellowindex] = sensor2;
  cal3[yellowindex] = sensor3;
  break;}
 case 5:{
  while(BUMPER < 20) red_blue_orange();
  blue_orange();
  wait(2);
  orange_blue_green();
  wait(2);
  cal1[blackindex] = sensor1;
  cal2[blackindex] = sensor2;
  cal3[blackindex] = sensor3;
  break;}
 default:{break;}
  }//end switch
```

```
 }//end reading loop
 //turn on all lights to signal done.
  all();
  wait(2);
  clear_leds();

}//end calibrate



void blink_red(void){
 int j;

 for(j=0;j<3;j++){
   red();
   wait(1);
   clear_leds();
   wait(1);}
}


void blink_orange(void){
 int j;

 for(j=0;j<3;j++){
   orange();
   wait(1);
   clear_leds();
   wait(1);}
}



int within_below(int sensor, int var){
 int t1;

 t1 = (sensor >= (var - 5)) && (sensor <= var);
 return t1;
}

int within_above(int sensor, int var){
 int t1;

 t1 = (sensor >= var) && (sensor <= var + 5);
 return t1;
}

int within_range(int sensor, int var)
{
 int t1;

 t1 = (sensor >= (var-5)) && (sensor <= (var+5));
 return t1;
 }

void wait(int secs){
```

```
//This function wait a specified number of seconds

 int q,j;
 for(q=0;q<(3*secs);q++)
 {
  j=32000;
  while(j>0)
   j--;
 }//end loop
}//end function




int detect(){
 int i,color,color1,color2,color3,R1,R2,R3;

 R1 = sensor1;
 R2 = sensor2;
 R3 = sensor3;
 color = None;

 for(i=0;i<5;i++)
  {if(R1 >= cal1[i])
    continue;
   else
    break;}

   if(i == 0)
    if (within_below(R1,cal1[i])) color1 = Orange;
    else color1 = None;
   else
    if(i == 5)
     if (within_above(R1,cal1[i-1])) color1 = Black;
     else color1 = None;
   color1 = get_color(i,R1,1);

  for(i=0;i<5;i++)
  {if(R2 >= cal2[i])
    continue;
   else
    break;}

   if(i == 0)
    if (within_below(R2,cal2[i])) color2 = Yellow;
    else color2 = None;
   else
    if(i == 5)
     if (within_above(R2,cal2[i-1])) color2 = Black;
     else color2 = None;
   color2 = get_color(i,R2,2);

 for(i=0;i<5;i++)
  {if(R3 >= cal3[i])
    continue;
   else
```

```
    break;}

   if(i == 0)
    if (within_below(R3,cal3[i])) color3 = Orange;
    else color3 = None;
   else
    if(i == 5)
     if (within_above(R3,cal3[i-1])) color3 = Black;
     else color3 = None;
   color3 = get_color(i,R3,3);

  if(((((color1 + color2 +color3)/3)*10)%10) > 5)
   color = ((color1 + color2 +color3)/3) + 1;
  else
   color = ((color1 + color2 +color3)/3);
  return color;
 }

int get_color(int i, int RR1, int sensor){
 int guess1,guess2,temp;

 if(sensor == 1)
  {guess1 = cal1[i] - RR1;
  guess2 = RR1 - cal1[i-1];}
 if(sensor == 2)
  {guess1 = cal2[i] - RR1;
  guess2 = RR1 - cal2[i-1]; }
 if(sensor == 3)
{  guess1 = cal3[i] - RR1;
  guess2 = RR1 - cal3[i-1];}

  if(sensor == 2){
   if (guess2 >= guess1)
    temp = i-1;
   else
    temp = i;
   if((temp == 0) || (temp == 2))
    temp++;
   if((temp == 1) || (temp == 3))
    temp--;}
  else
   if(guess2 >= guess1)
    temp = i;
   else
    temp = i-1;
 return temp;}

 void blink_blue(void){
 int j;
 for(j=0;j<3;j++){
   blue();
   wait(1);
   clear_leds();
   wait(1);}}
```

Appendix C: ir.c

```
//************************************************************
//Written by: Lesley Boylston.                                ****
//Date: November 2001                                         ****
//This file contains functions to  check the IR sensors on****
//on A.J. and return a 1 or 0 based on the reading.       ****
//************************************************************




//***************************DEFINES**************************
#define IRM analog(2)
#define IRL analog(1)
#define IRR analog(3)

//***************************GLOBALS**********************


//***************************PROTOTYPES******************
int IRMcheck (void);
int IRLcheck (void);
int IRRcheck (void);

//*********************************FUNCTIONS*********************
********



int IRMcheck(void){
//This function returns 1 if middle IR is too close.
//Returns 0 if not too close.

 int t1,temp;
 temp = IRM;
 if (temp < 150)
  t1 = (temp > 130);
 else
  t1 = 0;
  return  t1;
}


int IRRcheck(void){
//This function returns 1 if right IR is too close.
//Returns 0 if not too close.

 int t1,temp;
 temp = IRR;
 if (temp < 150)
  t1 = (temp > 105);
 else
  t1 = 0;
 return  t1;
```

```
}

int IRLcheck(void){
//This function returns 1 if left IR is too close.
//Returns 0 if not too close.

 int t1,temp;

 temp = IRL;
 if (temp < 150)
  t1 = (temp > 105);
 else
  t1 = 0;
 return  t1;
}
```

Appendix D: led.c

```
//*********************************
//WRITTEN BY LESLEY BOYLSTON     ***
//NOVEMBER 2001                  ***
//THIS FILE INCLUDES FUNCTIONS TO***
//LIGHT THE LED PANEL            ***
//*********************************



void init_leds(void);
void clear_leds(void);
void red(void);
void green(void);
void blue(void);
void orange(void);
void red_green(void);
void orange_green(void);
void blue_green(void);
void blue_orange(void);
void orange_blue_green(void);
void all(void);


void init_leds (void){
 //Set DDRD for output on pins 2-5
 SET_BIT(DDRD, 0x3C);
 //CLEAR LEDS
 CLEAR_BIT(PORTD, 0x3C);
 }

void clear_leds(void){
 CLEAR_BIT(PORTD, 0x3C);}

void red (void){
 clear_leds();
 SET_BIT(PORTD, 0x10);
 }

void green (void){
 clear_leds();
 SET_BIT(PORTD, 0x20);
 }

void blue (void){
 clear_leds();
 SET_BIT(PORTD, 0x08);
 }

void orange (void){
 clear_leds();
 SET_BIT(PORTD, 0x04);
 }
```

```
void red_green(void){
 clear_leds();
 SET_BIT(PORTD, 0x30);}

 void orange_green(void){
  clear_leds();
  SET_BIT(PORTD, 0x24);}

void blue_green(void){
  clear_leds();
  SET_BIT(PORTD, 0x28);
  }

void blue_orange(void)
 {clear_leds();
  SET_BIT(PORTD, 0x0C);
  }

void orange_blue_green(void){
  clear_leds();
  SET_BIT(PORTD, 0x2C);}

void all(void){
  clear_leds();
  SET_BIT(PORTD, 0x3C);}

void red_blue(void){
  clear_leds();
  SET_BIT(PORTD, 0x18);}

void red_orange(void){
  clear_leds();
  SET_BIT(PORTD, 0x14);}

void red_blue_orange(void){
  clear_leds();
  SET_BIT(PORTD, 0x1C);}
```

Appendix E: motor.c

```
//*****************************************
//***ADAPTED BY: LESLEY BOYLSTON FROM CODE***
//***WRITTEN BY SCOTT NORTMAN             ***
//***OCTOBER 28, 2001                     ***
//***FILE THAT DEFINES ISR's AND FUNCTIONS***
//***TO RUN MOTORS                        ***
//***LEFT MOTOR: PIN 29, OC3, PA5.        ***
//***RIGHT MOTOR: PIN 28, OC2, PA6        ***
//*****************************************




//*******************DEFINES*****************
#define PERIOD 30000

#pragma interrupt_handler TOC2_isr, TOC3_isr; //isr's are here

//*******************GLOBALS*********************
int duty2;
int duty3;

//**********************Function Prototypes******************
void init_pwm(void);
void TOC2_isr(void);
void TOC3_isr(void);

void stop(void);

void back_2_left(void);
void back_2_right(void);






/*******************FUNCTIONS************************/




void stop(void){        //stops motors
      duty2 = 0;
      duty3 = 0;

}//end stop



void back_2_left(void){ //moves right motor back and stops left
```

```
      duty2 = 10;
      duty3 = 0;
}//end back_2_left


void back_2_right(void){ //stops right motor and moves left motor back
      duty2 = 0;
      duty3 = 5;
}//end back_2_right



void init_pwm(void){    //initialization function

 INTR_OFF();      //interrupts off

//set interrupt for OC3
 SET_BIT(TMSK1, 0x20);
 SET_BIT(TCTL1, 0x20);
 CLEAR_BIT(TCTL1, 0x10);

 //set interrupt for OC2
 SET_BIT(TMSK1, 0x40);
 SET_BIT(TCTL1, 0x80);
 CLEAR_BIT(TCTL1, 0x40);

 INTR_ON();

}//end init_pwm

//**********************ISR's*******************************
void TOC2_isr(void)
{
 int temp = 0;

 //Clear the flag
 CLEAR_FLAG(TFLG1, 0x40);

 //Set the number of eclock for the duty cycle
 temp = (duty2 / 100.0) * PERIOD;

 if(temp < 500){
  CLEAR_BIT(TCTL1, 0x40);
  SET_BIT(CFORC, 0x40);}
 else
  if(temp > (PERIOD - 500)){
   SET_BIT(TCTL1, 0x40);
   SET_BIT(CFORC, 0x40);}
  else
   if(TCTL1 & 0x40){
    CLEAR_BIT(TCTL1, 0x40);
    TOC2 += temp;}
   else {
    SET_BIT(TCTL1, 0x40);
    TOC2 += (PERIOD - temp);}

}// end isr
```

```
//*****************************************
void TOC3_isr(void)
{
 int temp = 0;
 CLEAR_FLAG(TFLG1, 0x20);     //clear flag

 temp = (duty3/100.0)*PERIOD;

 if(temp<500){
  CLEAR_BIT(TCTL1, 0x10);
  SET_BIT(CFORC, 0x20);
  }
 else
  if(temp > (PERIOD - 500)){
   SET_BIT(TCTL1, 0x10);
   SET_BIT(CFORC, 0x20);}
  else
   if(TCTL1 & 0x10){
    CLEAR_BIT(TCTL1, 0x10);
    TOC3 += temp;}
   else {
    SET_BIT(TCTL1, 0x10);
    TOC3 += (PERIOD - temp);}

}//END ISR
```

Appendix F: servo.c

```
//*********************************************
//This file contains functions to move the   ***
//servos and wait routines.                  ***
//Written by Lesley Boylston                     ***
//October 2001                                        ***
//*********************************************


//*********************DEFINES*******************
#define PERIOD 30000
#pragma interrupt_handler TOC4_isr, TOC5_isr; //isr's are here

//***********************Global Variables********************
int front;
int back;

//***********************Function Prototypes******************
void init_pwm2(void);
void TOC4_isr(void);
void TOC5_isr(void);
void head_wag(void);

void servowait(int);
void tail_wag(void);
void head_center(void);
void tail_center(void);
void head_tail_wag(void);
//************************Functions*********************
void servowait(int secs){
//This functions waits an experimentally found time
//in order for the servos to move smoothly.
 int q,j;
 for(q=0;q<(secs);q++)
 {
  j=2500;
  while(j>0)
   j--;
 }//end loop
}//end function

void head_center (void)
 {
  front = 3254;}

void tail_center(void)
 {
  back = 3150;}

void head_wag(void)
{
   int j,i;
   while(front > 2900)
    {front -= 10;
```

```
      servowait(1);}
    servowait(1);

    for(j=0;j<2;j++)
    {for(i=0;i<73;i++)
     {front += 10;
      servowait(1);
     }
     for(i=0;i<79;i++)
     {front -= 10;
      servowait(1);
     }

    }//end for
    while(front < 3254)
     {front += 10;
      servowait(1);}
  }//end head_wag

void tail_wag(void)
{
    int j,i;
    while(back > 2610 )
     {back -= 10;
      servowait(1);}
    servowait(1);

    for(j=0;j<2;j++)
    {for(i=0;i<72;i++)
     {back += 15;
      servowait(1);
     }
     for(i=0;i<72;i++)
     {back -= 15;
      servowait(1);
     }

    }//end for

    while(back < 3150)
     {back += 10;
      servowait(1);}
  }//end tail_wag

void head_tail_wag(void){
 int i,j;
 while(front > 2900)
  {front -= 10;
   servowait(1);}
 while(back > 2610 )
  {back -= 10;
   servowait(1);}
 servowait(1);
 for(j=0;j<4;j++){
  for(i=0;i<73;i++)
   {front += 10;
    back += 15;
```

```
    servowait(1);}
  for(i=0;i<73;i++)
   {front -= 10;
    back -= 15;
    servowait(1);}
 }
 while(front<3254)
 {front += 10;
  servowait(1);}
 while(back < 3150)
 {back += 10;
  servowait(1);}
}//end head_tail_wag


void init_pwm2(void){   //initialization function

 INTR_OFF();      //interrupts off

//set interrupt for OC4
 SET_BIT(TMSK1, 0x10);
 SET_BIT(TCTL1, 0x08);
 CLEAR_BIT(TCTL1, 0x04);

 //set interrupt for OC5
 SET_BIT(TMSK1, 0x08);
 SET_BIT(TCTL1, 0x02);
 CLEAR_BIT(TCTL1, 0x01);
 CLEAR_BIT(PACTL, 0x04);

 INTR_ON();

}//end init_pwm


//**********************ISR's*******************************
void TOC4_isr(void)
{
 int temp = 0;

 //Clear the flag
 CLEAR_FLAG(TFLG1, 0x10);

 temp = front;

 if(temp < 500){
  CLEAR_BIT(TCTL1, 0x04);
  SET_BIT(CFORC, 0x10);}
 else
  if(temp > (PERIOD - 500)){
   SET_BIT(TCTL1, 0x04);
   SET_BIT(CFORC, 0x10);}
  else
   if(TCTL1 & 0x04){
    CLEAR_BIT(TCTL1, 0x04);
    TOC4 += temp;}
   else {
```

```
    SET_BIT(TCTL1, 0x04);
    TOC4 += (PERIOD - temp);}

}// end isr


//*****************************************
void TOC5_isr(void)
{
 int temp = 0;
 CLEAR_FLAG(TFLG1, 0x08);     //clear flag

 temp = back;

 if(temp<500){
  CLEAR_BIT(TCTL1, 0x01);
  SET_BIT(CFORC, 0x08);
  }
 else
  if(temp > (PERIOD - 500)){
   SET_BIT(TCTL1, 0x01);
   SET_BIT(CFORC, 0x08);}
  else
   if(TCTL1 & 0x01){
    CLEAR_BIT(TCTL1, 0x01);
    TOC5 += temp;}
   else {
    SET_BIT(TCTL1, 0x01);
    TOC5 += (PERIOD - temp);}
}//END ISR
```