Jate Sujjavanich
EEL 5666C
IMDL Final Report
Thursday, August 7, 2003

Oscar the Grouchy Garbageman

ABSTRACT **3**
EXECUTIVE SUMMARY **3**
INTRODUCTION **3**
INTEGRATED SYSTEM **4**
ATmega128 ..... *4*
MOBILE PLATFORM **5**
ACTUATION **6**
Trash Arm *7*
Platform Movement *7*
SENSORS **7**
Obstacle Avoidance *7*
Trash Can Location *8*
Trash Detection *8*
BEHAVIORS **8**
User Location *8*
Trash Can Approach *8*
IR Approach Mode *9*
Operating System *9*
CONCLUSION **10**
DOCUMENTATION **10**
ACKNOWLEDGEMENTS **10**
APPENDICES **10**

Abstract

The goal of Oscar the Grouchy Garbageman was to perform the mundane task of trash disposal. The robot's tasks were divided into functional units. These were be the microcontroller (brain), locomotion, object avoidance sensors, target sensors, trash detector, and trash holder/disposer. The platform had to hold the circuit boards, sensors, and actuators. The two actuation tasks for this robot were to manipulate the trash object and to move the platform close to the user and trash can. The sensors used were IR sensors, bump sensors, and the CMUcam. The first phase of behavior chosen was user location. The second behavior was chosen to be trash can location. IR approach mode began with the assumption that Oscar would have the target acquired by IR-distance sensors as well. The behaviors were programmed using a real-time state machine/task scheduler.

Executive Summary

Oscar the Grouchy Garbage man was based on the idea that robots should be designed to make human's lives easier. This robot was designed to perform the simple task of garbage disposal. Motors and a servo were chosen to perform the movement of the robot. IR sensors and bump sensors were chosen as they are standard obstacle avoidance tools.

Introduction

One of the hopes for robotics is that some day, mundane tasks can be used so that humans can use their time for more important things. The goal of Oscar the Grouchy Garbageman was to perform the mundane task of trash disposal. The finished robot would locate a human, accept a piece of trash, and take it to a garbage can. The sensors needed to find the user and target garbage can as well know when trash had been accepted.

Integrated System

The robot's tasks were divided into functional units. These were be the microcontroller (brain), locomotion, object avoidance sensors, target sensors, trash detector, and trash holder/disposer. All items needed interface with the microcontroller both electrically and mechanically.

ATmega128

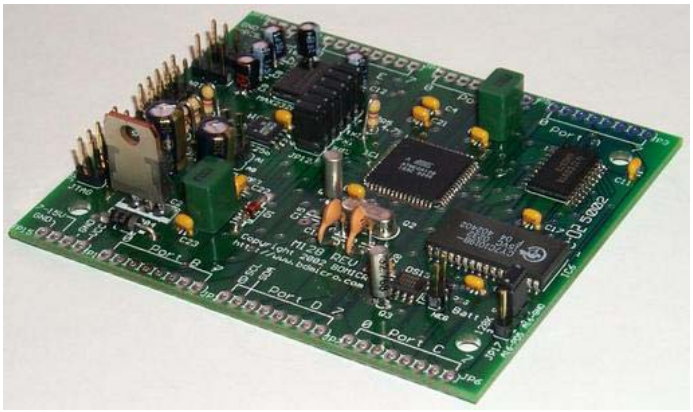

Figure 1 - BDMicro's M128 REV B Board

The M128 REV B board housing the ATmega128 microcontroller from BDmicro was chosen. It appeared to be a cheaper alternative to the LetATwork 2 board chosen by others in the class. It turned out to be the same price because unnecessary parts were purchased to go on the board. In actuality, the minimum parts could have been purchased, and more money could have been saved (see Table 1).

Note that use of this board requires good surface mount device soldering skills.

Part	Purchase Cost	Parts purchased post-class experience
M128 REV B board	$24	$24
ATmega128 chip	$16	Free (samples available)
Power system components	$8	$8
Headers	Free (from IMDL lab)	Free
Reset Switch	$1	$1

Table 1 - ATmega128 Board Costs

Mobile Platform

The platform was designed with the idea of simplicity. It had to hold the circuit boards, sensors, and actuators. The batteries were placed in the rear with the support of a caster. Directly underneath were the motors and gearbox. On top, the perforated board along with the sensors and trash arm were mounted.

Initially, a ready-made platform was chosen from Pololu.com. It was later determined that it would be too small, so a revision two was created out of wood. Some errors were discovered after fabrication (motor box not centered), so the third and final revision was created. The final revision in Figure 1 was designed based on a circle and was improved.

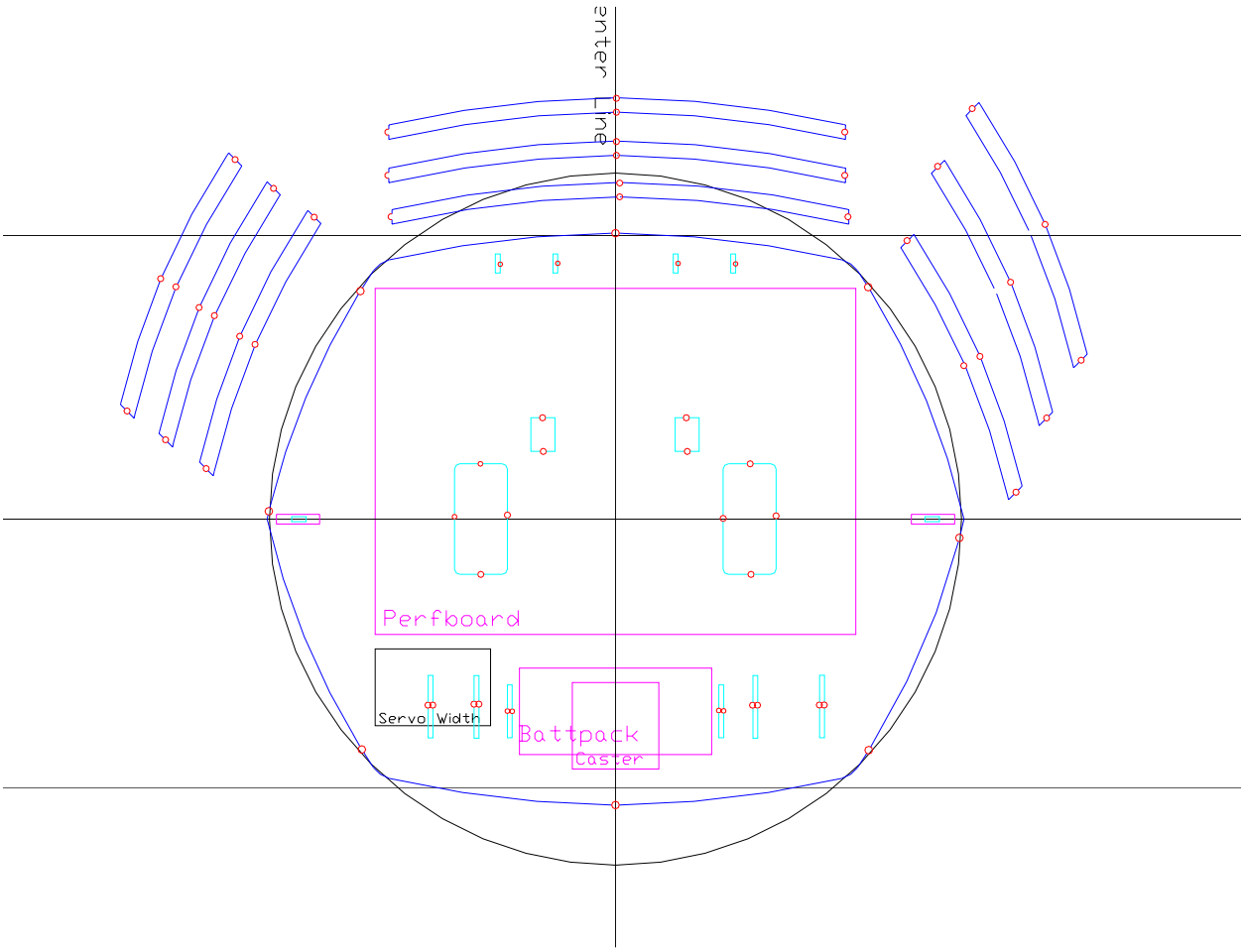

Figure 2 - Autocad Design for Final Platform

Actuation

There were two actuation tasks for this robot. The first was to move the platform close to the user and trash can. The second was to manipulate the trash object both holding it and disposing of it. The motors and gearbox provided adequate movement. The trash container was designed and attached to an un-hacked servo.

Trash Arm

Originally, the pulse width modulation signal was sent out continuously. Unfortunately, the servo had a tendency to pick up noise, and thus would turn at random intervals. The solution to this was found in software. The signal finally was only sent out when motion was required.

Platform Movement

The method of actuation for platform movement chosen was the Tamiya 70097 twin-motor gearbox. It arrived in kit form and had to be built from scratch. It was designed to be connected with a Pololu robot chassis, but it was easy to connect it to the final platform after some use of the drill press.

The controller for the motor pair chosen was the Pololu micro dual serial motor controller. This is an H-bridge chip with a serial interface. Several problems were encountered with this that range from random resets to failure to recognize the command from the chip. Eventually, placing the power for the motors on a separate solved the problem.

Sensors

Obstacle Avoidance

Obstacle avoid was achieved with the pairing of bump and IR sensors. The bump sensors were simple single-pole-double-throw switches with pull-up resistor. The IR sensors were purchased from the Mark III Robot Project. These sensors have an IR LED coupled with an IR sensor, and output an analog voltage from 0 to 2.5 V.

Trash Can Location

The core of the trash can location was the CMUcam obtained from www.seattlerobotics.com. It was successful in detected blobs of color using the color tracking mode. Its serial interface allowed for easy communication.

Trash Detection

Trash detection was implemented by having the user press a switch when the trash was deposited.

Behaviors

The behaviors were decided on using experimentation, and is divided into discrete steps. They follow.

User Location

The first behavior chosen was user location. This was implemented using a random walk with obstacle avoidance. First, Oscar was programmed to move forward while avoiding obstacles for a period of time (several seconds). After this phase, Oscar was made to turn in a direction and goes back to the forward motion phase. When the user was able to, he or she would be expected to place the trash into the holder and press the trash-location button. At this point, Oscar would go into the trash can approach mode.

Trash Can Approach

The second behavior was chosen to be trash can location. This first phase of this behavior is similar to the user location random walk. Oscar would wander around the room until the can was located by sight. When in sight, the second phase would begin.

The second phase to the trash can approach behavior was to turn until the red mass was centered in the camera's field of vision. This required the motors to be stepped in the appropriate direction. After the mass of red was centered, Oscar was designed to go into an approach (third) phase.

Whenever the trash can moved to the left or right of the field of vision, the motors' speed would be adjusted so that Oscar would turn toward the trash can. If at any time the red moved to the far left or right of the view, Oscar would try to recenter the mass again in phase two. If the red moved out of viewing range, Oscar would return to the random-walk phase. When the confidence of the image reached 255, Oscar would switch into IR approach mode.

IR Approach Mode

IR approach mode began with the assumption that Oscar would have the target acquired by IR-distance sensors as well. Oscar would then move forward turning left and right to approach the trash can at a perpendicular angle. When at the proper distance, Oscar would dump the trash. At this point, Oscar would return to user location mode.

Operating System

The behaviors were programmed using a real-time state machine/task scheduler. An event handler scheduled each event based on the timer system. In case of conflict, the system would decide based on a priority system.

The state machine examines appropriate sensor values and changes states depending on the required behavior. State changes were scheduled using the event scheduling system. This allowed for events to happen at a prescribed time.

This type of system required that each ‘process’ be well behaved. That is, they could not be permitted to run for an extended period of time. Otherwise, a misbehaved process would never allow another to run. Since this is not a real operating system and I control all the processes, this was not a problem.

Conclusion

I believe that this robot’s biggest challenge will be to determine the location of the trash can for disposal. Following a sonar beacon might be relatively simple, but determining the range to the trash can so that it may be disposed of may be a problem.

Documentation

Brian Ruck’s CMUcam report

(http://mil.ufl.edu/imdl/papers/IMDL_Report_Spring_03/ruck_brian/cmu.pdf)

Pololu Electronics (http://www.pololu.com)

Seattlerobotics.com (http://www.seattlerobotics.com

Acknowledgements

Thanks to Drs. Schwartz and Arroyo. Also the help of the TA’s (Vinh, Uriel, & Louis) in creating the platform is appreciated. Finally, thanks for the lessons in winning with the human-priest strategy.

Appendices

```
#include <avr\io.h>
#include <avr/pgmspace.h>
#include <stdbool.h>
#include <inttypes.h>
#include <string.h>
#include <stdlib.h>
```

```
#define E_TABLE_SIZE 10
#define clockHz 1000000
#define tcnt3_div 256
#define ticks_per_ms 3.9 // clockHz / tcnt3_div / 1000

/* Motor Max Speed Values */
#define m0_spd_max 60         // left
#define m1_spd_max 65         // right

/* IR Threshold Values */
#define IR_obstacle 0x40

extern void spin (char tens_ms);
extern void __INIT_UART_9600_8N1(void);
extern void __wait_tx(void);
extern void __wait_rx(void);
extern void __setup_MD(void);
extern void __reset_MD(void);

void bump_init(void);

void motor_smoother(void);
void set_motor(char motor_num, char speed, char dir);
void motor_calc(uint8_t*, uint8_t*, uint8_t*, uint8_t*);

void uart_echo(void);
void outchar(char out);
void check_switches(void);
void motor_arbiter(void);
void init_event_timer(void);
void init_event_table(void);
void sched_event(int, unsigned int);
void sample_bump(void);
void init_lcd(void);
void write_lcd(uint8_t, uint8_t);
void lcd_process(void);
void debug_init_globals(void);
void set_lcd(int);
void init_ADC(void);
void adc_process(void);
char * uint8_hex_to_string(uint8_t);

void uart1_to_lcd(void);
void update_lcd_adc(void);
void update_lcd_cam(void);
extern void __delay_10us(int);

void init_periphs(void);

void uart1_tx(char);
char uart1_rx(void);
void uart1_string_in(void);
void uart1_string_out(void);

void set_servo(int);
void servo_dump(void);
void servo_init(void);

/* Camera Routines */
void init_cam(void);
void check_cam_ack(void);
void cam_cmd(int);
void convert_track_data(void);

/* BEG Global Variables */
bool switch_one, switch_two, switch_three;   // for initial testing of bump switches

bool bump_switch[3] = {false, false, false};
bool last_bump[3] = {false, false, false};

// Motor Variables
```

```
char m0_spd, m1_spd, m0_dir, m1_dir;
uint8_t m0_spd_target, m1_spd_target, m0_dir_target, m1_dir_target;

unsigned int e_time[E_TABLE_SIZE];
char e_on[E_TABLE_SIZE];

int LCD_state = 0, LCD_char = 0, LCD_msg = -1;
uint8_t LCD_nibble;
char LCD_init_command[10] = {0x03, 0x03, 0x03, 0x02, 0x02, 0x08, 0x00, 0x0C};
unsigned int LCD_init_wait[] = {16, 8, 16, 8, 8, 8, 8, 8};

char LCD_line1[21] = "Line 1 - 01234567890";
char LCD_line2[21] = "line 2 - 01234567890";

uint8_t IR_0, IR_1;

uint8_t cam_track_data[8];
int camera_cmd;

char uart1_buf_in[40] = "                                          ";
char uart1_buf_out[40] = "                    ";

                                                                    //  "12345678901234567890"
max LCD string length
const char LCD_data1[][21] PROGMEM = {"   Jate's OSCAR     ", // 0
                                                                    "Forward State
",
                                                                    "Front Obstacle
",
                                                                    "Right Obstacle
",
                                                                    "Left Obstacle
",
                                                                    "Camera Exception
", // 5
                                                                    "   OSCAR READY
",
                                                                    "Red to the Left
",
                                                                    "Red Tracking
",
                                                                    "Red to the Right
",
                                                                    "Red not found
", // 10
                                                                    " Approaching Red
",
                                                                    "     IR Approach
",
                                                                    "  Dump Trash Now
"};
const char LCD_data2[][21] PROGMEM = {"    Calibrating...  ", // 0
                                                                    " __ __ ___ ___
",
                                                                    " __ __
",
                                                                    " __ __
",
                                                                    " __ __
",
                                                                    "ACK not returned
", // 5
                                                                    "
",
                                                                    "
",
                                                                    "
",
                                                                    "
",
```

```
", // 10

",

",

"};

12345678901234567890123456789O
const char uart1_data[][40] PROGMEM = {"\r\0",

33\r\0", // Auto on

32\r\0", // Auto off

Mode On

50\r\0", // Track Red

58 111 5 34", // dummy data

50\r\0" // Track Blue


/* END Global Variables */

int main (void)
{
        init_periphs();

        __INIT_UART_9600_8N1();

        __setup_MD();
        __reset_MD();

        bump_init();
        init_ADC();
        init_lcd();

        set_lcd(0);
        lcd_process();

        servo_init();
        init_cam();

        set_lcd(6);

        init_event_timer();

        char state = 0, next_state = 0;

        init_event_table();

        /* Line comes before so LCD will run */
//      e_on[7] = true; // camera comm

        // schedule recurring sensors
        e_on[4] = true; // motors on
//      e_on[5] = true; // bump sensor
        e_on[8] = true; // analog to digital converter (IR sensors)
        e_on[7] = true; // camera process
        e_on[9] = true; // LCD output

// Main Scheduler
while(1)
        {
                /* State Machine */
                switch (state)
                {
                        case 0 : // initial
```

```
"

"

"

"

//


 "CR 18 44 17 5 19

 "CR 18 40 17 5 19

 "PM 1\r\0", // Poll

 "TC 100 240 16 50 16

 "ACK M 53 103 49 82

 "TC 10 240 100 240 16

 };
```

```
                        // Schedule next event
                        if (!e_on[0])
                        {
                                sched_event(0, TCNT3 + 15600);
                                next_state = 5;
                        }
                        break;
                case 1 : // forward
                        set_lcd(1);
                        /* set motors */
                        m0_dir_target = 0; m0_spd_target = 50;
                        m1_dir_target = 0; m1_spd_target = 50;

                        /* next state is same or back up */
                        if (bump_switch[0] && last_bump[0])
                        {
                                next_state = 2;
                                sched_event(0, TCNT3);
                        }
                        else /* IR only matters if bumps are off */
                        {
                                if ((IR_0 > IR_obstacle) && (IR_1 > IR_obstacle))
                                {
                                        next_state = 2;
                                        sched_event(0, TCNT3);
                                }

                                if ((IR_0 > IR_obstacle) && (IR_1 <= IR_obstacle))
                                {       /* Turn Left */
                                        next_state = 4;
                                        sched_event(0, TCNT3);
                                }

                                if ((IR_0 <= IR_obstacle) && (IR_1 > IR_obstacle))
                                {       /* Turn Right */
                                        next_state = 3;
                                        sched_event(0, TCNT3);
                                }
                        }

                        break;
                case 2 : // back up
                        /* check if back-up time has passed */
                        set_lcd(2);
                        m0_dir_target = 1;  m1_dir_target = 1;
                        m0_spd_target = 70; m1_spd_target = 70;

                        if (next_state == 2)
                        {
                                sched_event(0, TCNT3 + 1950); // back up for 1/2 sec
                                next_state = 3;       // next will always be turn right to
avoid
                        }
                        break;
                case 3 : // right obstacle
                        set_lcd(3);
                        if (next_state == 3)
                        {
                                sched_event(0, TCNT3 + 975);
                                next_state = 1; // go forward again
                        }
                        m0_dir_target = 1; m0_spd_target = 60; // left motor
                        m1_dir_target = 0; m1_spd_target = 60; // right motor
                        break;
                case 4 : // left obstacle
                        set_lcd(4);
                        if (next_state == 4)
                        {
                                sched_event(0, TCNT3 + 975);
                                next_state = 1; // go forward again
                        }
```

```
        m0_dir_target = 0; m0_spd_target = 60; // left motor
        m1_dir_target = 1; m1_spd_target = 60; // right motor
        break;
case 5 : // search for red
        camera_cmd = 4; // set up for red detection
        set_lcd(8);
        m0_dir_target = 0; m0_spd_target = 0; // left motor
        m1_dir_target = 0; m1_spd_target = 0; // right motor

        /* detect red in scene */
        if ((cam_track_data[0] == 0) && (cam_track_data[1] == 0))
        { // no red detected
                next_state = 10;
                sched_event(0, TCNT3);
        }
        else
        {
                if (cam_track_data[0] < 38)
                {
                        next_state = 6;
                        sched_event(0, TCNT3);
                }
                if (cam_track_data[0] > 42)
                {
                        next_state = 8;
                        sched_event(0, TCNT3);
                }
                if ((cam_track_data[0] >= 38) && (cam_track_data[0] <= 42))
                {
                        next_state = 11;
                        sched_event(0, TCNT3);
                }
        }
        break;
case 6 : // turn left to center red - motor on period
        set_lcd(7);
        m0_dir_target = 1; m0_spd_target = 80; // left motor
        m1_dir_target = 0; m1_spd_target = 80; // right motor

        if (next_state != 7)
        {
                next_state = 7;
                sched_event(0, TCNT3 + 300);
        }

        if (cam_track_data[0] > 38)
        { // red is recentered
                next_state = 5;
                sched_event(0, TCNT3);
        }
        break;
case 7 : // turn left to center red - motor off while waiting for cam
        m0_dir_target = 0; m0_spd_target = 0;
        m1_dir_target = 0; m1_spd_target = 0;

        if (next_state != 5)
        {
                next_state = 5;
                sched_event(0, TCNT3 + 90);
        }
        break;
case 8 : // turn right to center red - motor on period
        set_lcd(9);
        m0_dir_target = 0; m0_spd_target = 80; // left motor
        m1_dir_target = 1; m1_spd_target = 80; // right motor

        if (next_state != 9)
        {
                next_state = 9;
                sched_event(0, TCNT3 + 300);
        }
```

```
        if (cam_track_data[0] < 42)
        { // red is recentered
                next_state = 5;
                sched_event(0, TCNT3);
        }
        break;
case 9 : // turn left to center red - motor off while waiting for cam
        m0_dir_target = 0; m0_spd_target = 0;
        m1_dir_target = 0; m1_spd_target = 0;

        if (next_state != 5)
        {
                next_state = 5;
                sched_event(0, TCNT3 + 90);
        }
        break;
case 10 : // no red detected
        set_lcd(10);
        // stop
        m0_dir_target = 0; m0_spd_target = 50; // left motor
        m1_dir_target = 0; m1_spd_target = 50; // right motor

        if (next_state == 10)
        {
                next_state = 14;
                sched_event(0, TCNT3 + 3900);
        }

        if ((IR_0 > IR_obstacle) && (IR_1 > IR_obstacle))
        {
                next_state = 16;
                sched_event(0, TCNT3);
        }

        if ((IR_0 > IR_obstacle) && (IR_1 <= IR_obstacle))
        {       /* Turn Left */
                next_state = 14;
                sched_event(0, TCNT3);
        }

        if ((IR_0 <= IR_obstacle) && (IR_1 > IR_obstacle))
        {       /* Turn Right */
                next_state = 15;
                sched_event(0, TCNT3);
        }

        if ((cam_track_data[0] != 0) && (cam_track_data[1] != 0))
        {
                next_state = 5;
                sched_event(0, TCNT3);
        }
        break;
case 14 : // right obstacle
        set_lcd(3);
        if (next_state == 14)
        {
                sched_event(0, TCNT3 + 975);
                next_state = 10; // go forward again
        }
        m0_dir_target = 1; m0_spd_target = 60; // left motor
        m1_dir_target = 0; m1_spd_target = 60; // right motor
        break;
case 15 : // left obstacle
        set_lcd(4);
        if (next_state == 15)
        {
                sched_event(0, TCNT3 + 975);
                next_state = 10; // go forward again
        }
        m0_dir_target = 0; m0_spd_target = 60; // left motor
```

```
                m1_dir_target = 1; m1_spd_target = 60; // right motor
                break;
        case 16 : // back up
                /* check if back-up time has passed */
                set_lcd(2);
                m0_dir_target = 1;  m1_dir_target = 1;
                m0_spd_target = 70; m1_spd_target = 70;

                if (next_state == 16)
                {
                        sched_event(0, TCNT3 + 2700); // back up for 3/4 sec
                        next_state = 15;       // next will always be turn right to
avoid
                }
                break;
        case 11 : // red centered, approach color
                set_lcd(11);

                if ((IR_0 > 0x40) || (IR_1 > 0x40))
                { // in range of red, change state to IR approach
                        next_state = 12;
                        sched_event(0, TCNT3);
                }
                else // not yet in range of red
                {
                        if (cam_track_data[0] < 38)
                        {
                                // go slowly
                                m0_dir_target = 0; m0_spd_target = 42;
                                m1_dir_target = 0; m1_spd_target = 50;
                        }
                        if (cam_track_data[0] > 42)
                        {
                                // go slowly
                                m0_dir_target = 0; m0_spd_target = 50; // left
                                m1_dir_target = 0; m1_spd_target = 42; // right
                        }
                        if ((cam_track_data[0] >= 38) && (cam_track_data[0] <= 42))
                        {
                                // go slowly
                                m0_dir_target = 0; m0_spd_target = 50;
                                m1_dir_target = 0; m1_spd_target = 50;
                        }
                        if ((cam_track_data[0] < 20) || (cam_track_data[0] > 60))
                        {
                                next_state = 5;
                                sched_event(0, TCNT3);
                        }
                }

                break;
        case 12 : // locked on by color, now move in w/ infrared
                set_lcd(12);

                if ((IR_0 > 0x50) && (IR_1 > 0x50))
                {
                        next_state = 13; // dump trash
                        sched_event(0, TCNT3);
                }
                else
                {
                        if (IR_0 > IR_1) // IR_0 is right
                        {
                                m0_dir_target = 0; m0_spd_target = 50;
                                m1_dir_target = 0; m1_spd_target = 30;
                        }
                        if (IR_0 < IR_1)
                        {
                                m0_dir_target = 0; m0_spd_target = 30;
                                m1_dir_target = 0; m1_spd_target = 50;
                        }
```

```
                        if (IR_0 == IR_1)
                        {
                                m0_dir_target = 0; m0_spd_target = 40;
                                m1_dir_target = 0; m1_spd_target = 40;
                        }
                }
                break;
        case 13 : // Dump Trash
                set_lcd(13);

                m0_dir_target = 0; m0_spd_target = 0;
                m1_dir_target = 0; m1_spd_target = 0;
                motor_smoother();
                set_motor(0, m0_spd, m0_dir);
                set_motor(1, m1_spd, m1_dir);
                set_motor(0, m0_spd, m0_dir);
                set_motor(1, m1_spd, m1_dir);
                set_motor(0, m0_spd, m0_dir);
                set_motor(1, m1_spd, m1_dir);

                motor_smoother();
                servo_dump();
                lcd_process();

                // 1 sec later
                TCNT3 = TCNT3 - 3900; // bring timer back to state before dump

                while (1); // freeze

                break;
} // end switch(i);

/* Event Scheduler */
int n, next_event = -1;

unsigned int t = TCNT3; // stablize so lower priority won't go if count occurs

for (n = 0; n < E_TABLE_SIZE; n++)
{
        if ((t >= e_time[n]) && (e_on[n]) // past valid event time?
                && ((t - e_time[n]) < 7800)) // no overflow true?
        {
                next_event = n; // assuming n is retained
                break;          // assuming n is retained
        }
} // end for


switch (next_event)
{
        case 0 :
/* Change state on or after time next_state_time */
                state = next_state;           // advance state
                e_on[0] = false; // turn off state change event
                break;
        case 1 : // no event occurs
                break;
        case 2 : // set both motors
/*              set_motor(0, m0_spd, m0_dir);
                set_motor(0, m0_spd, m0_dir);
                set_motor(0, m0_spd, m0_dir);
                set_motor(1, m1_spd, m1_dir);
                set_motor(1, m1_spd, m1_dir);
                set_motor(1, m1_spd, m1_dir);
                e_on[2] = false;*/
                break;
        case 4 : // Periodic Motor Set
                motor_smoother();
                set_motor(0, m0_spd, m0_dir);
                set_motor(1, m1_spd, m1_dir);
                e_time[4] = TCNT3 + 39;
```

```
                        break;
                case 5 : // Periodic Bump sensor sample
                        sample_bump();
                        e_time[5] = TCNT3 + 39; // next in 39/3.9 or 10 ms
                        break;
                case 7 : // Camera communications
                        // load buffer with first command
                        cam_cmd(camera_cmd); // track color specified
                        check_cam_ack(); // error detection
                        convert_track_data(); // put into array

//                      uart1_to_lcd(); // show on LCD for now

                        sched_event(7, TCNT3 + 390); // next in 100 ms
                        break;
                case 8 :
                        adc_process();
                        e_time[8] = TCNT3 + 82;
                        break;
                case 9 : // LCD commands
                        update_lcd_adc();
                        update_lcd_cam();
                        lcd_process();
                        e_time[9] = TCNT3 + 164;
                        break;
                case -1 :
                        break; // event not scheduled
        } // end switch(n)
    }
    return 0;
}

void uart_echo() // echoes one character
{
    char letter;
    __wait_rx();
    letter = UDR0; // get typed character

    __wait_tx();
    UDR0 = letter; // output character
}

void outchar(char out)
{
    __wait_tx();
    UDR0 = out;
}

void motor_smoother()
{
    /* Set motor 0 */
/*  motor_calc(&m0_spd, &m0_spd_target, &m0_dir, &m0_dir_target);
    motor_calc(&m1_spd, &m1_spd_target, &m1_dir, &m1_dir_target);*/
    m0_dir = m0_dir_target;
    m0_spd = m0_spd_target;

    m1_dir = m1_dir_target;
    m1_spd = m1_spd_target; // just don't smooth for now

}

void motor_calc(uint8_t* spd, uint8_t* s_target, uint8_t* dir, uint8_t* d_target)
{
    /* change direction if speed near zero */
    if (*dir != *d_target)
    {
        if (*spd < 20)
        {
            *dir = *d_target;      // set contents of pointer dir equal to *d_target
        }
        else
```

```
                {
                *spd -= 20;
                }
        }
        else
        {
                if (*spd < *s_target)
                {
                        *spd += ((*s_target - *spd) / 3);
                }
                else
                {
                        *spd -= ((*spd - *s_target) / 3);
                }
        }
}

void set_motor(char motor_num, char speed, char dir)
{
        __wait_tx();
        UDR0 = 0x80;

        __wait_tx();
        UDR0 = 0x00; // device type

        __wait_tx();
        UDR0 = (motor_num << 1) + dir; // 0:motor number:direction(0-1)

        __wait_tx();

        if (motor_num == 0)
                UDR0 = (uint8_t)((speed * m0_spd_max) / 100);

        if (motor_num == 1)
                UDR0 = (uint8_t)((speed * m1_spd_max) / 100);
}

void bump_init()
{
        char temp = ~((1 << 3) | (1 << 4) | (1 << 5)); // prepare bits 3, 4, 5 for zeroing
        DDRE = DDRE & temp; // set up 3, 4, 5 for input
}

void check_switches()
{
        if (PINE & (1 << 3)) // check E3(L)
                switch_one = false; // E3 high
        else
                switch_one = true;

        if (PINE & (1 << 4)) // check E4
                switch_two = false;
        else
                switch_two = true;

if (PINE & (1 << 5)) // check E5
                switch_three = false;
        else
                switch_three = true;

}

void init_event_timer()
{
        TCCR3A = 0; // all zeros, no OC, normal timing mode
        TCCR3B = (1 << CS32);
        // TCCR3C // left alone
        // TCNT3 can now be read at will
} // end init-event-timer

void init_event_table()
```

```
{
	int e;
	for (e = 0; e < E_TABLE_SIZE; e++)
	{
		e_on[e] = false;
	}
}

void sched_event(int event, unsigned int time)
{
	e_time[event] = time;
	e_on[event] = true;
}

void sample_bump()
{
	/* check bump switch state */
	last_bump[0] = bump_switch[0];
	last_bump[1] = bump_switch[1];
	last_bump[2] = bump_switch[2];

	bump_switch[0] = !(PINE & 0x08); // not converts to H logic
	bump_switch[1] = !(PINE & 0x10);
	bump_switch[2] = !(PINE & 0x20);
}

void init_lcd()
{
	// Uses PORTA at this time
	// bit3-0 = data DB7-4
	// bit 4 = RS
	// bit 5 = enable
	DDRA = DDRA | 0x3F; // 0b00111111

	/* What follows is commands to set up the LCD */
	/* 4 bit mode, Two lines, no cursor */
	__delay_10us(1500);
	write_lcd(0, 0x03);
	__delay_10us(410);
	write_lcd(0, 0x03);
	__delay_10us(10);
	write_lcd(0, 0x03);
	__delay_10us(410);
	write_lcd(0, 0x02);
	__delay_10us(4);
	write_lcd(0, 0x02);
	write_lcd(0, 0x08);
	__delay_10us(4);
	write_lcd(0, 0x00);
	write_lcd(0, 0x0C);
	__delay_10us(4);
}

void write_lcd (uint8_t RS, uint8_t nibble)
{
	PORTA = PORTA & 0xC0; // 11000000
	PORTA = PORTA | (RS << 4);
	PORTA = PORTA | nibble; // only allow nibble through
	PORTA = PORTA | (1 << 5); // stobe enable high
	PORTA = PORTA & ~(1 << 5); // strobe enable low
}

void lcd_process()
{
	int i;
	/* Clear home */
	write_lcd(0, 0x00);
	write_lcd(0, 0x01);
	__delay_10us(164);

	/* Output First Line */
```

```
        for (i = 0; i < 20; i++)
        {
                write_lcd(1, (LCD_line1[i] >> 4));
                write_lcd(1, (LCD_line1[i] & 0x0F));
                __delay_10us(4);
        }

        /* Cursor to begin of second line */
        write_lcd(0, 0x0C);
        write_lcd(0, 0x00);
        __delay_10us(4);

        /* Output Second Line */
        for (i = 0; i < 20; i++)
        {
                write_lcd(1, (LCD_line2[i] >> 4));
                write_lcd(1, (LCD_line2[i] & 0x0F));
                __delay_10us(4);
        }
}

void set_lcd(int msgnum)
{
        int j;

        if (msgnum != LCD_msg)
        {
                PGM_P a = LCD_data1[msgnum];
                for (j = 0; j < 21; j++)
                {
                        LCD_line1[j] = PRG_RDB(a + j);
                }

                a = LCD_data2[msgnum];
                for (j = 0; j < 21; j++)
                {
                        LCD_line2[j] = PRG_RDB(a + j);
                }
        }
        LCD_msg = msgnum;

}

void init_ADC()
{
        ADCSR = (1 << ADPS2);
        ADMUX = 0x60; //0b01100000
}

void adc_process()
{
        // select ADC0
        ADMUX = ADMUX & (0xE0); // use 0b11100000 - only the sel bits cleared
        // start conversion, wait for flag
        // enable ADSC
        ADCSR = ADCSR | (1 << ADEN);
        ADCSR = ADCSR | (1 << ADSC);
        while (!(ADCSR & (1 << ADIF)));
        // store value in variable
        IR_0 = ADCH;

        // select ADC1
        ADMUX = ADMUX | (0x01);
        // start conversion, wait for flag
        ADCSR = ADCSR | (1 << ADSC);
        while(!(ADCSR & (1 << ADIF)));
        // store value in variable
        IR_1 = ADCH;
        // disable ADSC to save power
        ADCSR = ADCSR & (~(1 << ADEN));
}
```

```
void update_lcd_adc()
{
	char temp[2];
	utoa(IR_0, temp, 16);
	LCD_line2[1] = temp[0];
	LCD_line2[2] = temp[1];

	utoa(IR_1, temp, 16);
	LCD_line2[4] = temp[0];
	LCD_line2[5] = temp[1];
}

void init_periphs()
{
	/* UART1 for CMUcam */
	UBRR1H = 0x00;
	UBRR1L = 0x0C; // 9600 baud
	UCSR1A = 0x02;
	UCSR1B = 0x18;
	UCSR1C = 0x06;

	/* PWM on OC1A */
	TCCR1A = 0x82;		//; Select OCA clear-comp reset-top, PWM11-PWM10=10
					//0b10000010
	TCCR1B = 0x18;		//; WGM13:2=11 for fastPWM TOP=ICR1, CS12:0=000
					//0b00011000
	ICR1H = (uint8_t)(20000 >> 8);
	ICR1L = (uint8_t)(20000);		// set pulse freq to be 20000 or 20ms

	OCR1AH = (uint8_t)(1400 >> 8);
	OCR1AL = (uint8_t)(1400);

	// OC1A/B5 is set to output
	DDRB |= (1 << DDB5);
//	TCCR1B = (TCCR1B & ~(0x06)); // set TCCR1B 2:0 to be 001
//	TCCR1B = (TCCR1B | 0x01);	// this set TCCR1B activates PWM

}

void uart1_tx(char byte)
{
	while (!(UCSR1A & (1 << UDRE1))); // loop till UDR1 is empty
	UDR1 = byte;
}

char uart1_rx()
{
	char byte;
	while (!(UCSR1A & (1 << RXC1)));
	byte = UDR1;
	return byte;
}

void uart1_string_in()
{
	/* This function reads in a string from uart1 terminated by \r */
	/* String is input into a global string array */
	int i = 0;
	char byte = 0;
	while ((byte != ':') && (i < 40))
	{
		byte = uart1_rx();
		uart1_buf_in[i++] = byte;
	}

	/* Set last byte to be null termination */
	uart1_buf_in[i] = 0;
}

void uart1_string_out()
```

```
{
	int i = 0;
	while ((i < 40) && (uart1_buf_out[i] != 0))
	{
		uart1_tx(uart1_buf_out[i++]);
	}
}

void set_servo(int pw)
{
	OCR1AH = (uint8_t)(pw >> 8);
	OCR1AL = (uint8_t)(pw);
}

void servo_dump()
{
	set_servo(1100);
	TCCR1B = (TCCR1B & ~(0x06)); // set TCCR1B 2:0 to be 001
	TCCR1B = (TCCR1B | 0x01);     // this set TCCR1B activates PWM
	spin(200);

	set_servo(2000);
	spin(50);

	TCCR1B = (TCCR1B & ~(0x07)); // deactivate PWM
}

void servo_init()
{
	set_servo(2000);
	TCCR1B = (TCCR1B & ~(0x06)); // set TCCR1B 2:0 to be 001
	TCCR1B = (TCCR1B | 0x01);     // this set TCCR1B activates PWM
	spin(50); // wait 50 ms for servo to reach position
	TCCR1B = (TCCR1B & ~(0x07)); // deactivate PWM
}

void init_cam()
{
	// First command must be run '\r'
	// so replies are aligned
	cam_cmd(0);

	cam_cmd(0);
//	uart1_to_lcd();
	check_cam_ack();

	cam_cmd(1);
//	uart1_to_lcd();
	check_cam_ack();

	spin(250);
	spin(250);

	cam_cmd(2);
//	uart1_to_lcd();
	check_cam_ack();

	cam_cmd(3);
//	uart1_to_lcd();
	check_cam_ack();

	/* temporarily test blue */
/*	while (1)
	{
		cam_cmd(6);
		uart1_to_lcd();
		spin(50);
	}*/
}

void cam_cmd(int num)
```

```
{
        strcpy_P(uart1_buf_out, uart1_data[num]);
        uart1_string_out();
        uart1_string_in();
}

void check_cam_ack(void)
{
        int i;
        bool ack = false;
        /* Checks uart1_buf_in for ACK */
        for (i = 0; i < 5; i++)
        {
                if ((uart1_buf_in[i] == 'A') &&
                        (uart1_buf_in[i + 1] == 'C') &&
                        (uart1_buf_in[i + 2] == 'K'))
                {
                        ack = true;
                }
        }

        if (ack)
        {
                return;
        }
        else
        {
                set_lcd(5); // load ACK not received msg
                lcd_process(); // display it
                while (1); // loop forever
        }
}

void uart1_to_lcd()
{
        int i;
        /* dump uart1 receive buffer */
        for (i = 0; i < 20; i++)
        {
                LCD_line1[i] = uart1_buf_in[i];
                LCD_line2[i] = uart1_buf_in[i + 20];
        }
        lcd_process();
}

void convert_track_data()
{
        int i = 4, j = 0;
        cam_track_data[0] = atoi(uart1_buf_in + 6);

        while (j <= 7)
        {
                while (uart1_buf_in[i++] != ' '); // move to next space
                cam_track_data[j++] = atoi(uart1_buf_in + i);
        }
}

void update_lcd_cam()
{
        char temp[3];
        utoa(cam_track_data[0], temp, 10);
        LCD_line2[7] = temp[0];
        LCD_line2[8] = temp[1];
        LCD_line2[9] = temp[2]; // x coor

        utoa(cam_track_data[6], temp, 10);
        LCD_line2[11] = temp[0];
        LCD_line2[12] = temp[1];
        LCD_line2[13] = temp[1]; // num pixels
}
```

```
//                                                                  " __ __ ___ ___
",

// New endline after this
```