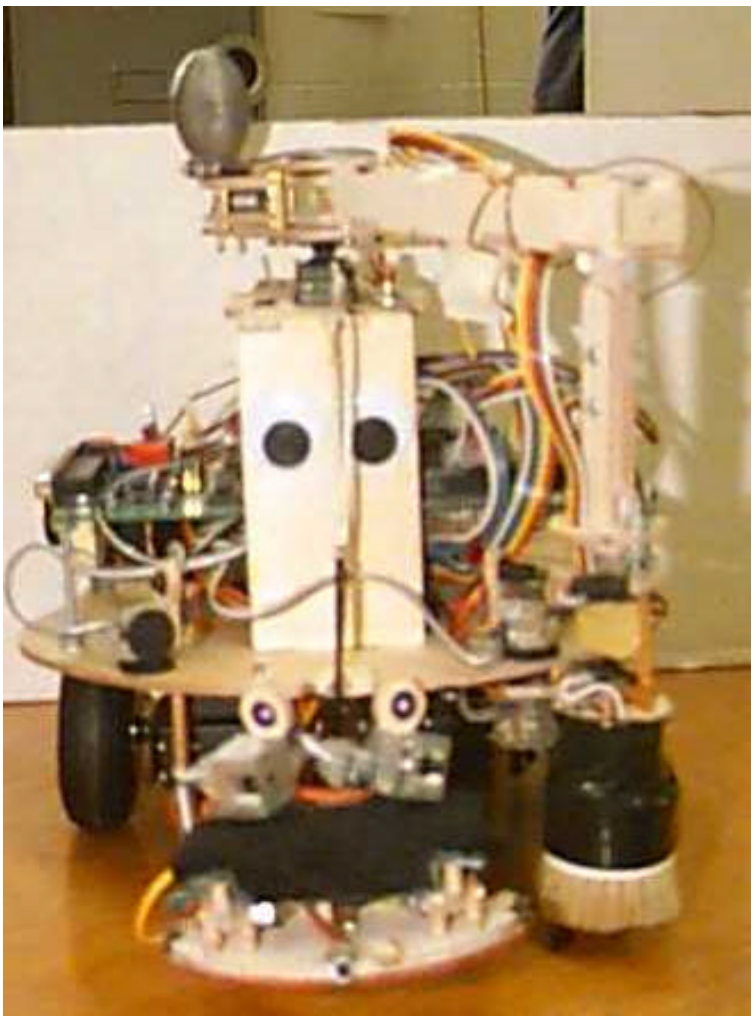
Lothar: A Tile Identifying and Placing Robot



Benjamin R. Paul
IMDL Summer 1999
August 3, 1999

Table of Contents

Abstract

Lothar is a tile placing robot that follows colored trails to find 1 1/4 inch tiles. It then captures, identifies, and places the tiles. Lothar's primary function is to find tiles and then based on the tiles color, place them in an appropriately colored area. He was designed to roam about looking for lines on the ground which then lead to colored tiles. Once a tile is found and properly positioned in the grabber, it's color is identified. From here Lothar looks for the appropriate colored area to place this tile.

Executive Summary

Lothar was designed to manipulate his environment by identifying and moving tiles around on the floor. While a tile placement robot has been built before, I wanted to add color and give it the ability to seek and select tiles. To accomplish this task I built a color sensor and a tile grabber as well a wrote behaviors that demonstrate there usefulness.

The color senor arm provides a means of looking and detecting colors in different locations around the robot. The arm uses a spring and two bump switches to protect the servo that drives it. This design could also be used to detect obstacles.

The tile grabber provides a means of detecting tiles and a way of holding them in place. Because the tiles are too light to be detected by any other means, the tile grabber presses down onto the tile thus sensing it's presences. The tile grabber also has a sensor that detects when a tile is in position and is ready to be moved and placed.

Introduction

In this paper I will describe my tile robot, Lothar. Lothar finds tiles, identifies their color, and then slides them into a predetermined colored area on the ground. A mobile platform, a collection of senors, actuators, and behaviors enable Lothar to complete this task.

I wanted to build a robot that manipulated its environment in a constructive manner. After evaluating many ideas including a sand painting robot and one that would make mosaics out of M&M's, I decided tiles would be the easiest medium to implement and clean up. This idea had been done before by Alan Senior with his robot, TileBot. Senior's robot dispensed tiles which meant it had to be reloaded. Also, it placed thin black plastic 'tiles' which limited TileBot to creating only black and white images. I wanted to build a robot that could find and identify colored tiles in it's environment and then arrange them to create images. Due to time constraints and unforseen difficulties, I was only able to complete the first part.

Lothar consists of a color sensor on a rotating arm and a tile grabber supported by a 1/8 inch plywood structure propelled by two hacked servos. Lothar's processing power is provided by a Motorola HC11 EVBU board expanded with 32k of RAM, a clock divider to provide a 40 kHz signal, two memory mapped output ports, two memory mapped input ports, and a multiplexed analog port.

This paper is a description of Lothar's mobile platform, actuators, and sensors which are each discussed in detail in the following sections.

Integrated System

Lothar is systems are divided into four main systems:

1. Microprocessor and Port Expansion Board

2. Color Sensor and The Color Sensor Arm

3. Tile Grabber Assembly

4. Drive System and Infer-red Sensors

The microprocessor and expansion board provides the logic and power requirements of Lothar. The color sensor and tile grabber works together in to capture tiles and identify their colors. The drive system and infer-red sensors allow Lothar to move safely and wall follow. Due to the sequential nature of ICC11 I found it best to use each system as needed, instead of treating them as objects that with accessor functions. Only one behavior is running at a time so all the systems resources are available to the currently active behavior.

Mobile Platform

The mobile platform needed to be able to provide structure to support all of Lothar's components and be able to move accurately when sliding tiles.

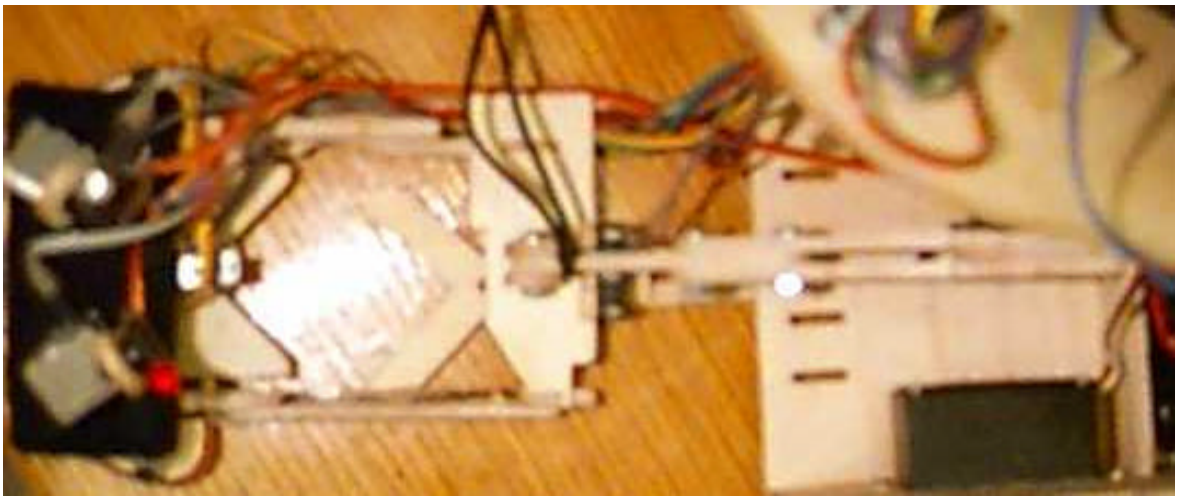
The physical structure of the mobile platform consists of an open-ended box which houses the primary battery pack, the two drive servos, and the grabber servo. The two circuit boards and the color turret tower are attached to the main chasse or top of the box. A V-shaped tile jaw is attached on the bottom front for pushing and positioning tiles. I had to remove a barrier that enclosed the batteries when I started using six C batteries instead of the originally designed for six AA batteries. I added a secondary six pack of AA batteries that had to be taped onto the main chasse due to the lack of space in the battery compartment. I also had great difficulty accessing the two circuit boards because the color turret made it necessary to stack them.

Movement is provided by two partially hacked 45 inch/ounce servos with three inch wheels attached. These servos are positioned close to the front for greater control of the jaw. There is also a closed-ended nut glued to the rear to provide a pivot for dragging the rear end. This drive system was not very accurate or responsive primarily because the servos I used were of very low quality and the refresh rate needed by the servo was too slow. In retrospect, I should have used geared stepper motors.

Actuation

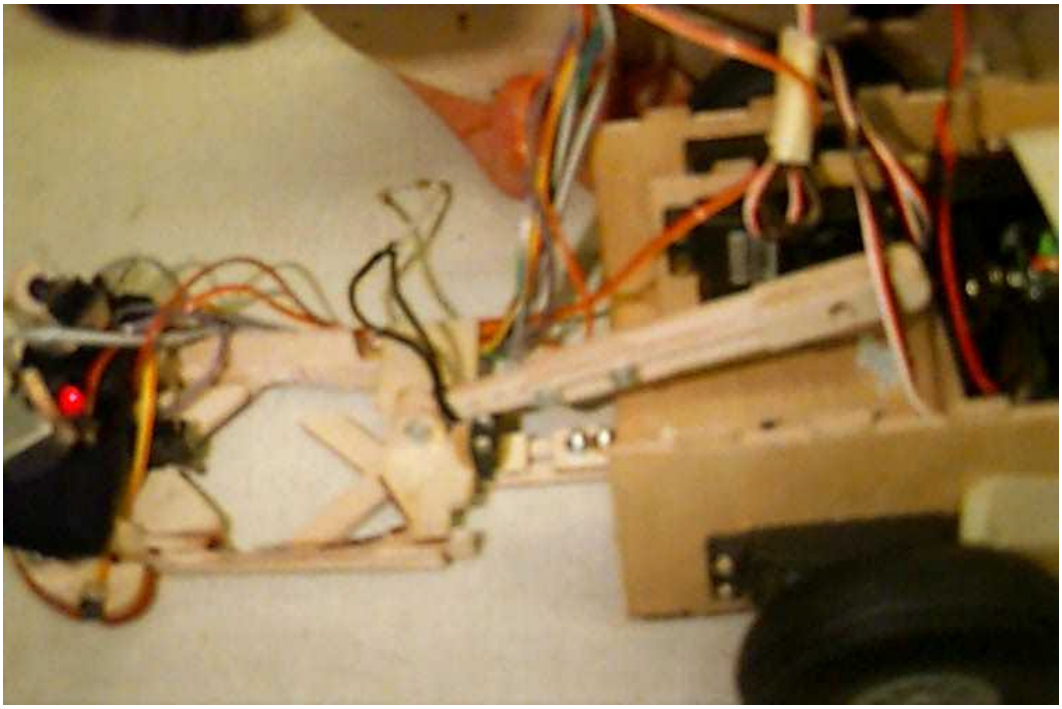
To find and arrange tiles, Lothar needed to hold tiles, move the color sensor to follow lines, and determine the color of captured tiles. Two assemblies where designed for this purpose: the tile-gripping assembly, and the color sensor turret.

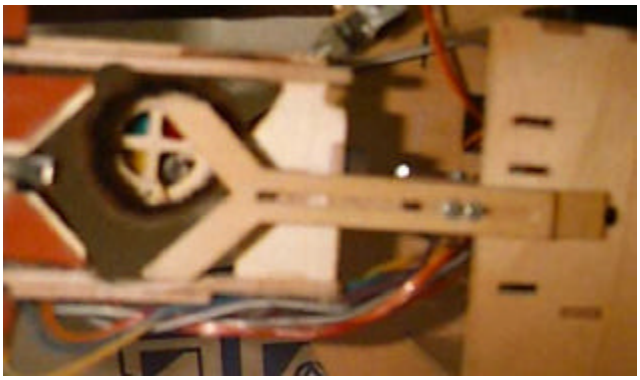
Tile-gripping Assembly:



The tile-gripping assembly enables Lothar to sense and grip tiles. The gripper is actuated by a 45 inch/ounce servo mounted above the left drive servo. The gripper is pulled up by a rubber band so that its at rest in the open position when the servo is not activated. The gripper consists of four parts: the major armature, the ankle, two minor armatures, and the foot. The servo is attached to the major armature which is then connected to the ankle. The ankle has a limit switch attached that comes in contact with the ram when the grabber is closed and there are no tiles under the foot. Due to their light weight, impact with tiles can not be detected and this is the only

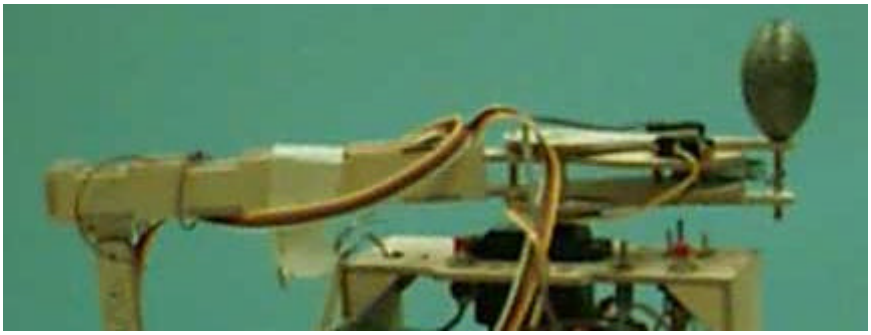
way to detect them. The ankle is connected to the foot by the two minor armatures. The foot contains a limit switch that makes contact if a tile is properly in position against the tile jaw. The foot also has three bump switches and two IR sensors for obstacle avoidance. The under side of the foot is covered in rubber for grabbing tiles that are not yet fully in position.





Color Sensor Turret Arm:

The color sensor turret arm moves and supports the color sensor, allowing Lothar to determine the color of tiles in the V-shaped jaw as well as finding and following colored trails. It consists of a 45 inch/ounce servo, a pivot housing, and a 12 inch arm. The purpose of the pivot housing is to protect the servo from stripping gears by providing a spring to absorb impact. The pivot housing is a pair of identical 1/8 inch plywood supports that are spaced ½ inch apart by 1/8 inch brass rod. A spring and a counter balance are attached to the brass spacing rod farthest from where the servo is mounted. The brass spacer closest to where the servo is mounted serves as a pivot for the arm which is also attached to the other end of the spring. Two limit switches are mounted to the arm and make contact when force is applied to the arm. This also aids in protecting the servo.



Sensors

Color Senor:

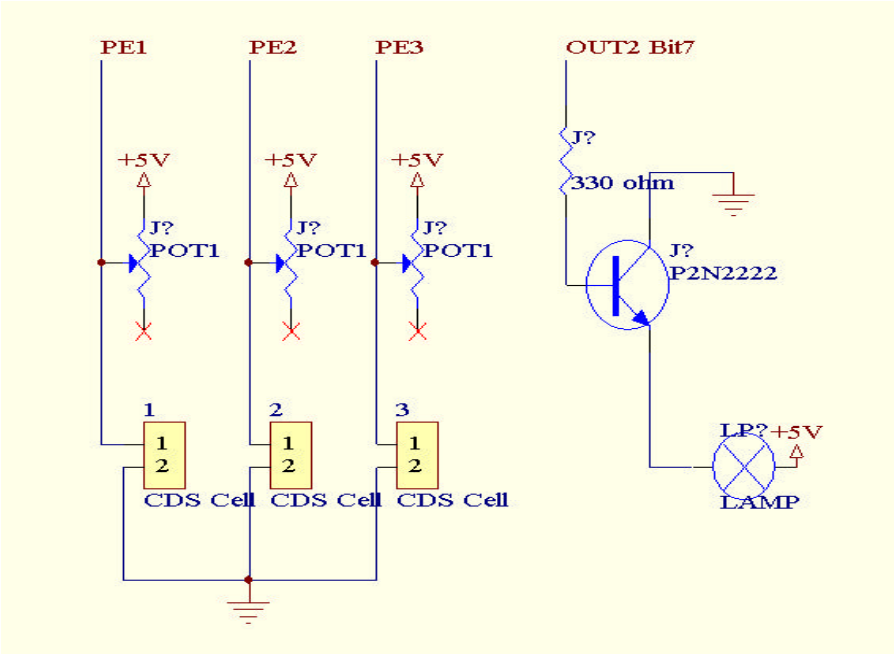
Lothar needed the ability to sense color to identify tiles and find trails. The sensor I designed to accomplish this task consists of an incandescent lamp, three CDS cells, three color filters, a curtain to keep out ambient light, and a screw reinforced wood support structure. To assure that this sensor would read dependably I tested it throughly.

The incandescent lamp was used instead of LEDs because the light it produces covers more of the visible spectrum than can be expected from LEDs. After trying a small 6 volt 25 milliamp light, I decided that a larger lamp was needed and implemented a 6 volt 100 milliamp light. Due to the greater power requirement, a simple driver circuit consisting of a P2N2222 transistor and a 330 ohm resister which can be seen on the right in Figure 1 was also added. This lamp provides enough light to allow the sensors to read the color of the tiles but being incandescent, it requires more than .3 seconds to reach full luminescence. This requires that the lamp stay on or have a warm-up period more than .3 seconds each time the light is turned on.

Three CDS are used to sense the amount of light reflected off of a tile and filtered through three

different color filters. A CDS cell is a solid state device which increases conductivity when light is present. Using this fact, I constructed three voltage divider circuits using 10 k potentiometers as seen on the left of Figure 1. The signal is then converted from analog to digital using PE1, PE2, and PE3 on the HC11. My concerns about inconsistent readings from these sensors where quieted by how consistent the experimental data was.

The selection of the color filters or jells as they are called in the lighting industry was very important and challenging. Instead of choosing three from the 159 filters that would best enable Lothar to distinguish the tiles, I chose eight. I designed my sensor so that the jells could be mounted to wheels and these wheels could be changed rather easily. Three wheels where constructed and tested on each tile color. The table below shows which jells where used on which wheel.

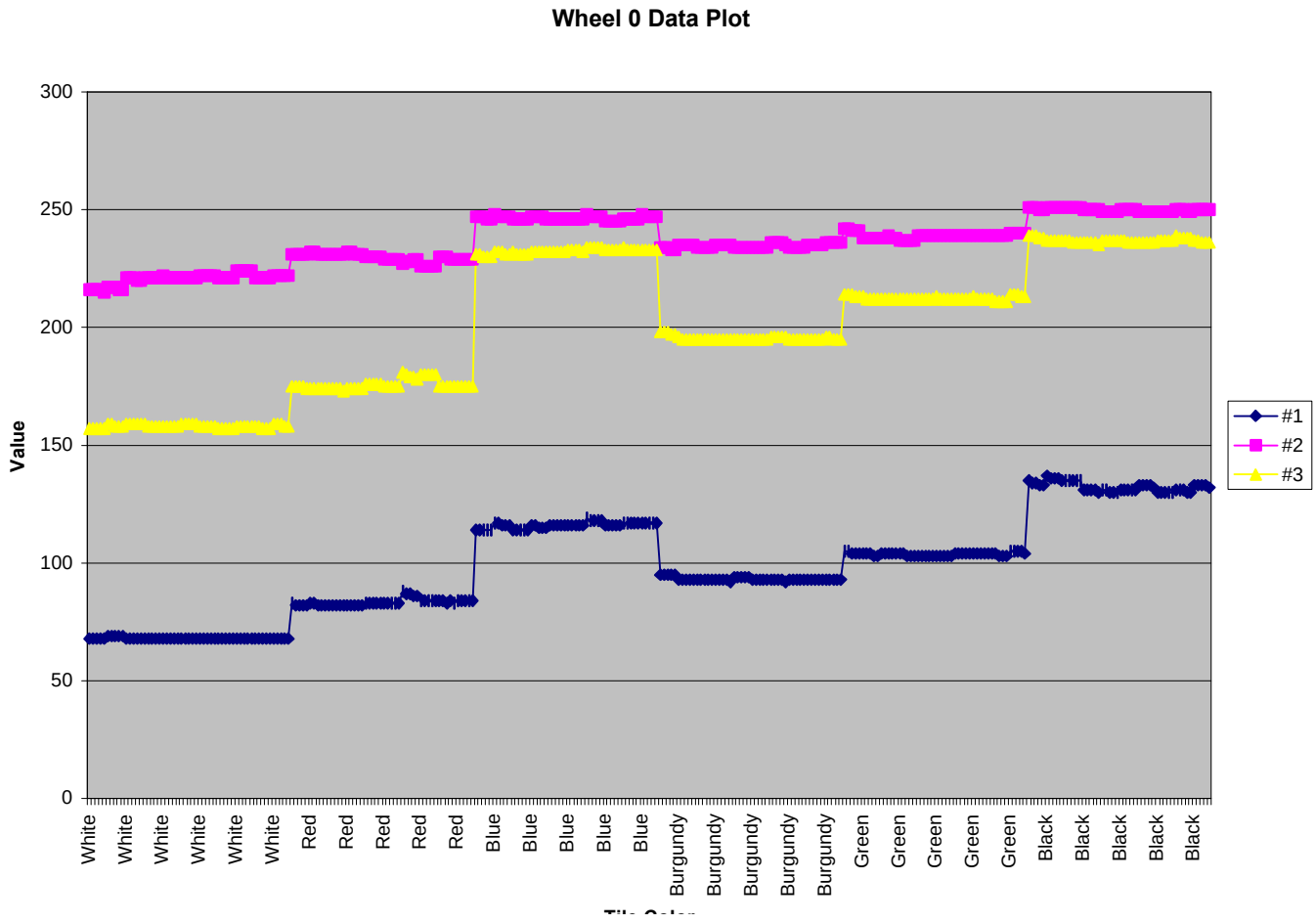


Wheel	Position	Color/Name	Y(%)	Part Number
0	1	Yellow	80.00	#139
	2	True Red	6.87	#109
	3	Just Blue	5.93	#79
1	1	Primary Green	14.97	#139
	2	Pail Blue	54.39	#63
	3	Flame Red	17.97	#164
2	1	Yellow	80.00	#101
	2	Magenta	10.92	#113
	3	Marine Blue	41.32	#131

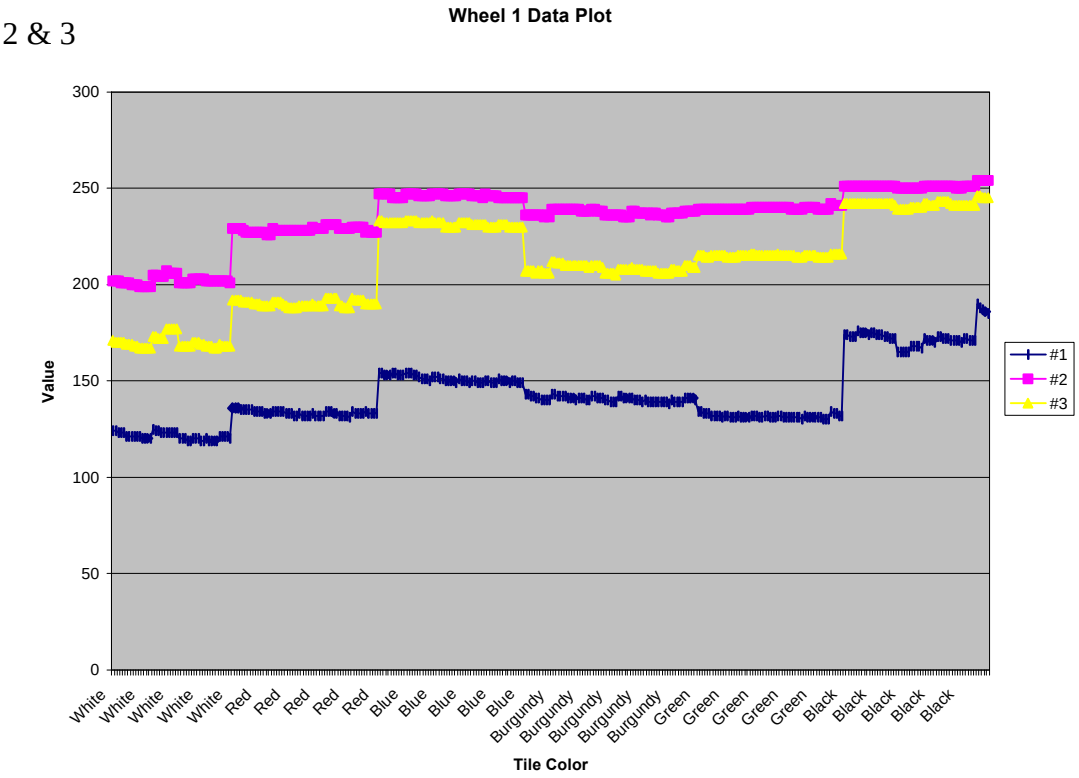
Table 1.

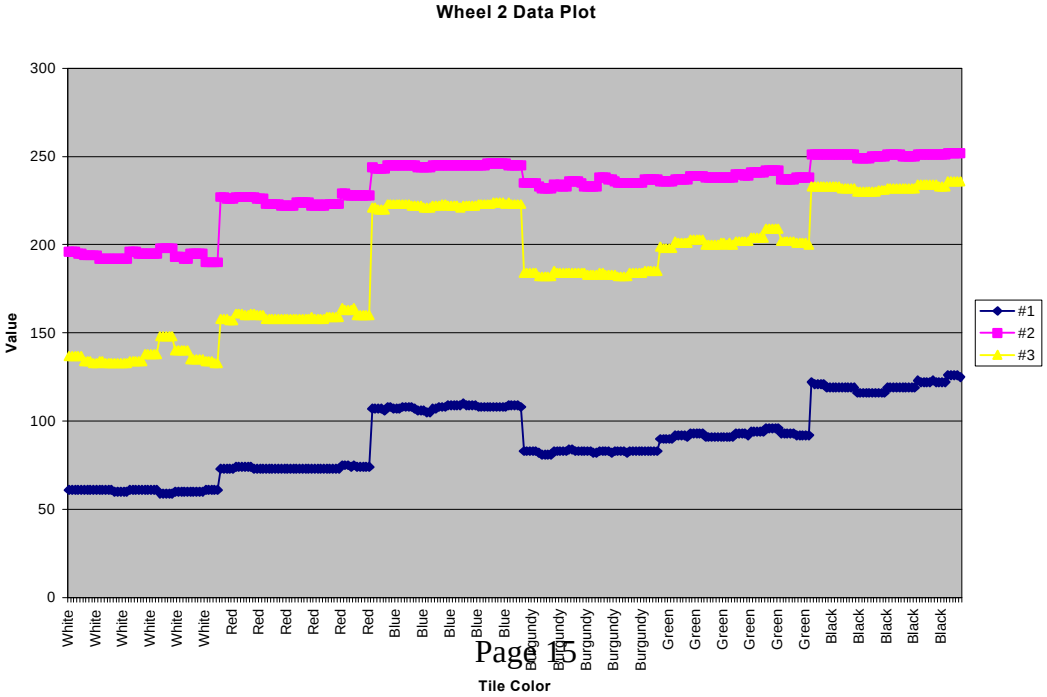
The mechanical structure consists of five disks held together by screws and the circuit board. Two of the disks combine to form the wheels previously mentioned. Two other disks are used to hold the circuit board in place and the last supports the lamp. Paint brush bristles where wrapped around this frame blocking out light from outside.

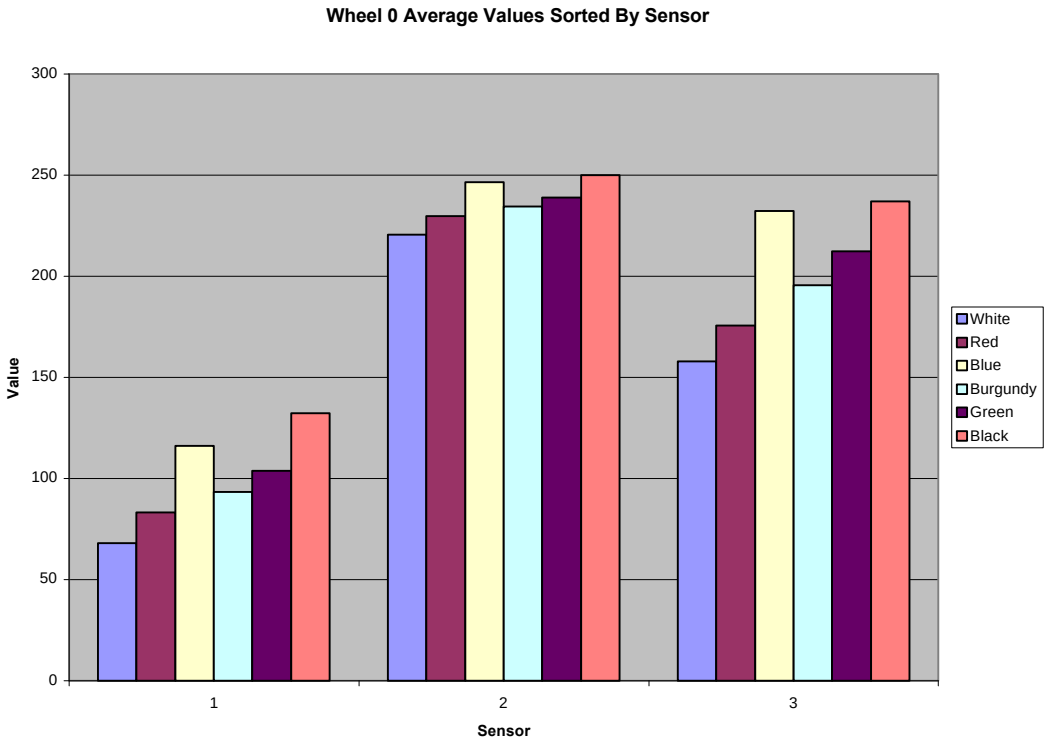
The test consisted of using each of the three wheels described in Table 1 on all of the tiles. A minium of 45 readings where taken for every wheel and tile combination. The test software warmed up the sensor and took five readings (one from each CDS cells) and waited for input on the keyboard. The software would then loop, waiting for input and then receive five readings. To provide variance, I picked up and set the sensor down again between every five readings. Below are the charts describing the test results.



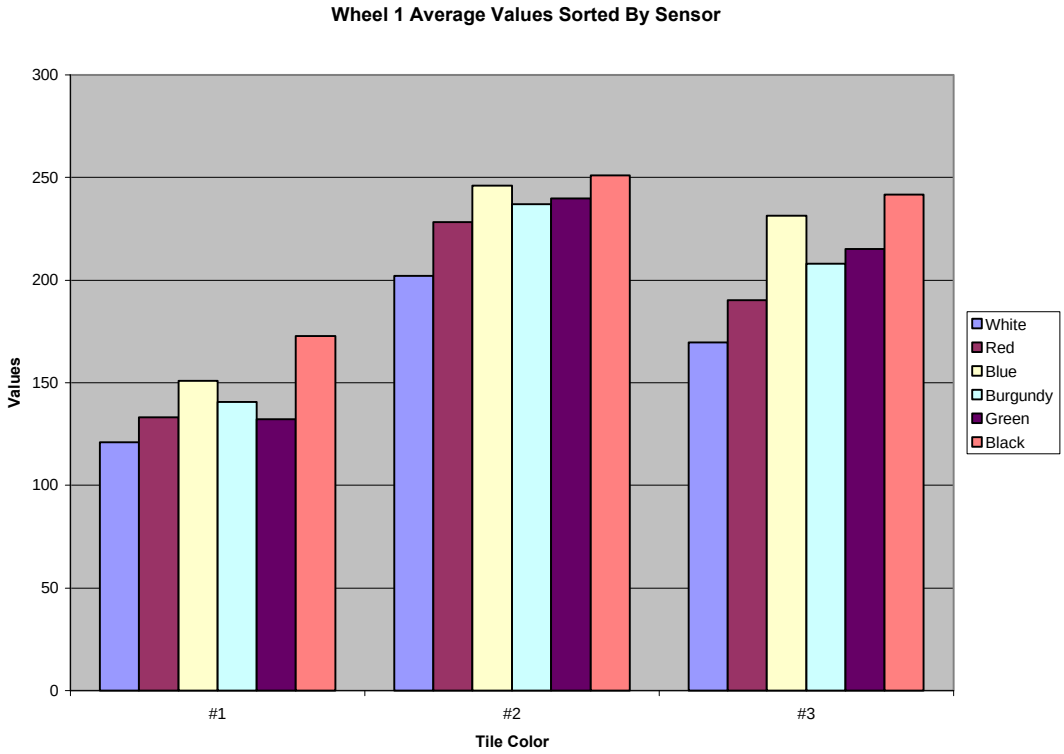
Graph 2 & 3

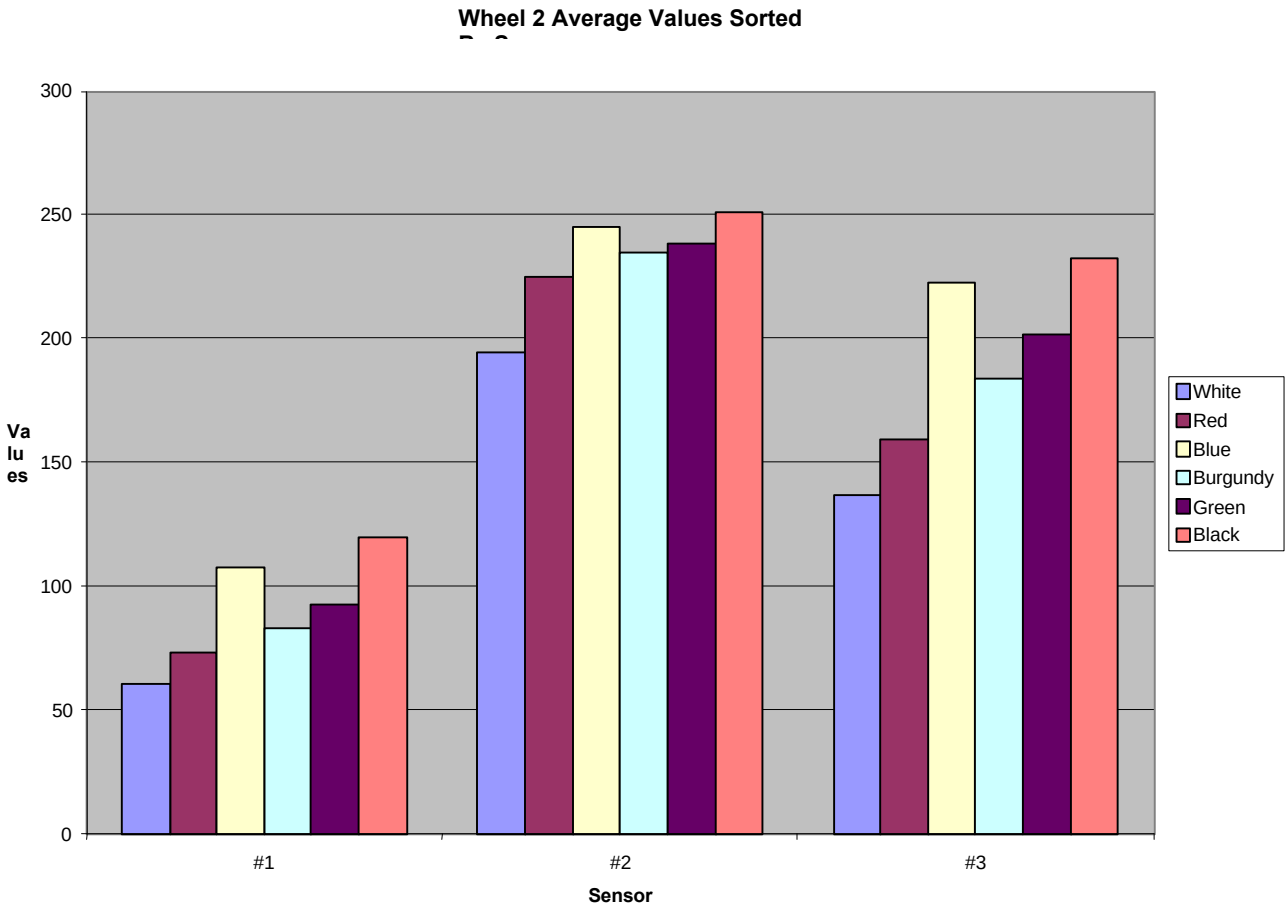

Wheel 1 Data Plot
Value
300
250
200
150
100
50
0
White
Red
Blue
Burgundy
Green
Black
Tile Color
#1
#2
#3


Wheel 2 Data Plot
Value
300
250
200
150
100
50
0
White
Red
Blue
Burgundy
Green
Black
Tile Color
#1
#2
#3


Wheel 0 Average Values Sorted By Sensor
300
250
200
150
100
50
0
Value
1
2
3
Sensor
White
Red
Blue
Burgundy
Green
Black

Graph 4 & 5


Wheel 1 Average Values Sorted By Sensor
300
250
200
150
100
50
0
Values
#1
#2
#3
Tile Color
White
Red
Blue
Burgundy
Green
Black



Graph 6

After inspecting the data, I chose to use wheel two because it provided the greatest distribution for each of the sensors. This sensor works well but there are some important changes I would have liked to make. The sensors are too spread out and this means that they need at least one square inch of color to identify it. The light shines mainly on the area seen by sensors one and three, leaving sensor two in the dark. The senor needs to be about 3/8 inches from the ground to look at tiles and on the ground to best identify colors on the ground.

Infer-red Sensors:

The IR sensors are used to detect obstacles and walls by emitted infer-red light and then quantifying how much of this light is returned. I used six of these sensors which are constructed using a 40 kHz modulated signal, a infer-red LED, and a hacked Sharp infer-red sensor. These sensors where then connected to a multiplexed A-D port. These sensors returned values ranging from 88 to 129 with larger values indicating something is close to the sensor.

Limit-switches:

Four limit-switches were used. Two to detect when the senor armature comes in contact with something, one to detected when a tile is in the jaw, and one to detect when a tile is under the foot.

Behaviors

To find and identify tiles, I developed behaviors for the color sensor, the tile grabber, and the drive system. I had a few difficulties developing these behaviors. First, I was completely terminal dependant to initiate them. Second, being used to Java and C++, I had difficulties writing large code using non-object-oriented ICC11. While I found it limiting, ICC11 allowed for far quicker code development than writing in assembly.

Color Calibration:

To identify colors, I had to first enter the colors to be identified and calibrate the sensors. During development of the sensor, I manually put the sensor on colors and pressed a key at a terminal. This was time consuming and not autonomous. I found that stored values also worked, but not as well as a fresh calibration. I believe this is due to the battery levels effect on the color sensors light. When Lothar's color arm and physical structure where completed, I developed a self calibration which involved setting him on a piece of posterboard with the colors arranged so that the color sensor turret could move the color sensor over them and a reading could be taken. Once the color values are stored in memory, a bubble-sort puts the color values of each sensor in order from lowest to highest. A table is then made by finding the midpoint between adjacent color values.

Color Identification:

The table created by the color calibration behaviors can then be used to identify colors by

determining which midpoints the new value is between. The color identification returns the color that is presently under the color sensor and sets a global variable to the percentage of sensors that agree.

Find Tile:

The tile finding behavior has Lothar follow a black line and closes the grabber and checks the state of its limit switches every few moves. Because of the possibility of false readings from the color sensor, the only dependable means of tile detection is checking the state of the limit switches on the grabber assembly. It checks the grabber by closing it and checking its two limit switches. If the limit switch on the gripper foot is activated, the color sensor is brought over to identify the color of the tile. If the tile is the correct color, all the LEDs blink. If it is not the right color the tile is released and the find color behavior is called to get Lothar back on the line. The limit switch between the ram and the grabber detects wether there is a tile under the grabber foot. If a tile is under the foot, Lothar is moved forward .5 inches and the grabber is checked again. This cycle is repeated till the tile is no longer under the gripper foot. Hopefully the tile will end up in the tile jaw and the gripper can fully close and the tile can be identified and dealt with. Otherwise, the tile is lost and the find color behavior is called to get Lothar back on the line.

Obstacle Avoidance:

The obstacle avoidance behavior checks the infer-red sensors for obstacles and moves accordingly. This behavior tries to move forward, but if an obstacle is found, Lothar is turned away from the obstacle. This is done by choosing to turn to the side whose infer-red sensors have

the lowest reading. While turning, the infer-red sensors on the side which is turning are checked to assure that Lothar does not turn into an obstacle. The number of interrupted turns are counted and when this number breaches ten, Lothar backs up and tries again.

Find Color:

In this behavior, Lothar moves about avoiding obstacles and looking for a color on the ground. This behavior is essentially obstacle avoidance with the addition of periodic checks of the color sensor after every move.

Wall Following:

This behavior involves veering left till an object is encountered. If the object is a specified distance away from the left infer-red sensors, Lothar goes forward. When the object gets too close Lothar turns right till the object is the desired distance away.

Conclusion

In summary, Lothar is capable of wall following, obstacle avoidance, finding colors, following trails, grabbing tiles, and identifying them. There are many areas that I would like to improve.

I feel that using cheep servos instead of geared stepper motors was a mistake. I chose to use servos because they did not require driving circuitry, but their poor performance far out weighed there ease on implementation. I would also like to greatly reduce the size of the color sensors and use multiple color sensors instead of the arm. I believe I could get the sensor down to the size of a dime and reduce the power requirement by 150%. I learned a great deal this summer, unfortunately much of what I learned did not make it onto Lothar, there simple was not enough time.

Appendix A:

Vendor Information

IR emitters/detectors:

- LED emitters: Mekatronics
  316 NW 17th St., Suite A
  Gainesville, FL 32603
  (407) 672-6780
  http://www.mekatronics.com
  (purchased from Scott Jantz)
  $0.75 each

- Detectors: Sharp GPIU58Y via Mekatronics
  (purchased from Scott Jantz)
  $3.00 each

Servos:

- 45 ounce/inch servos: Tower Hobbies
  PO Box 9078
  Champaign, IL 61826-9078
  1-800-637-4989
  http://www.towerhobbies.com
  $9.99 each

```
//*************************************************************************
//*
//*         File Name:        L_base.h
//*         Purpose:          to provide a common header file for all of Lothar's
//*                           code.  This is where Global Constants and including
//*                           other code will be handled.
//*
//*************************************************************************


//*****************************  INCLUDES ***********************************
#include <analog.h>
#include <clocktjp.h>
#include <servome.h>
#include <isrdecl.h>
#include <vectors.h>
#include <stdio.h>

//*****************************  END OF INCLUDES ****************************

//*****************************  CONSTANTS **********************************
/* VT100 clear screen */
char c1, tile_color[], clear[]= "\x1b\x5B\x32\x4A\x04";

/* VT100 position cursor at (y,x) = (3,12) command is "\x1b[3;12H"*/
char place[]= "\x1b[1;1H";    /*Home*/
char line8[]= "\x1b[8;1H";
char line9[]= "\x1b[9;1H";
char line15[]= "\x1b[15;1H";
char line16[]= "\x1b[16;1H";
//******************  General Constants *****************
#define TRUE      1
#define FALSE     0

#define LEFT      1
#define RIGHT     0


//******************  INPUT/OUTPUT CONSTANTS  *************
#define output1 *(unsigned char*)(0x4c00)
#define output2 *(unsigned char*)(0x4d00)
#define input1 *(unsigned char*)(0x4c00)
#define input2 *(unsigned char*)(0x4d00)

//******************  Left & Right Servo Constants ********
#define STOP      0

//******************  Grabber Servo Constants *************

//******************  Servo Constants *********************
#define GRABBER               1
#define COLOR_SERVO           2
#define LEFT_SERVO            3
#define RIGHT_SERVO           4


//******************  Sensor Constants ********************
#define TILE_IN_JAW           0x02
```

```
#define UNDER_GRABBER   0x01
#define LEFT_BUMPER           0x10
#define CENTER_BUMPER   0x08
#define RIGHT_BUMPER    0x04
#define LEFT_ARM        0x40
#define RIGHT_ARM       0x20
#define MY_BUTTON             0x80

#define WRONG_COLOR                     3
#define     TILE_UNDER_GRABBER          4

#define IR_LEFT               0x00
#define IR_GROUND       0x01
#define IR_RIGHT        0x02
#define IR_BACK               0x03
#define IR_GLEFT        0x05
#define IR_GRIGHT       0x04

#define BATT                  0x07

//******************  LED Constants *********************
#define LED_LEFT        0x01
#define LED_GROUND            0x02
#define LED_RIGHT       0x04
#define LED_BACK        0x08
#define LED_GLEFT       0x10
#define LED_GRIGHT            0x20

#define LED_ALL         0xff
//******************  Color Senor Constants ***************

#define CS_LIGHT  0x08    // output port 2 bit3
#define     CS_TILE           800
#define CS_HOME         1400

#define CS_SERVO  1           // argument for init_color()
#define CS_DEFAULT      0           // argument for init_color()

#define MARKER          0
#define TILE            1

//Color Constants
#define MARKER_BLACK    0
#define MARKER_RED            1
#define MARKER_BLUE           2
#define MARKER_GREEN    3
#define MARKER_WHITE    4

//Tile Constants
#define TILE_BLACK            0
#define TILE_RED        1
#define TILE_BLUE             2
#define TILE_GOLD             3
#define TILE_WHITE            4
```

```
#define output1 *(unsigned char*)(0x4c00)
#define output2 *(unsigned char*)(0x4d00)
#define input1 *(unsigned char*)(0x4c00)
#define input2 *(unsigned char*)(0x4d00)

//***************************  END OF CONSTANTS ****************************

//***************************  GLOBAL VERIABLES ****************************

//******************  Output Ports *********************
int _output1;                       //current value of output port 1
int _output2;                       //current value of output port 2

//******************  Color Sensor *********************
int colors[4][11];                  //colors[sensor][value] - color values from
                                              //servo_get_colors() or terminal_get_colors()
int colors_length;                  //number of entries in colors[]

int CSorder[4][11][2];        //CSorder[sensor][order][0 - color, 1 - value] -
                                              //sorted table created by color_sort()
int CSorder_length;                 //number of entries in CSorder[][][]

int color_table[2][4][10];          //color_table[MARKER/TILE][sensor][divition] - color table
created by
                                              //buile table, used by colorID();
int table_length[2];                //number of entries in color_table[4][10]

int percent_match;                        //percent match from colorID() values 0,33,66,100

int colorID_out[2][4];              //color detected by each sensor in colorID()
int color_cal_pos[5];          //positions of colors for servo_get_colors(void);

int servo_pos[5];             //current position of each servo;

//******************  Slow Servo veriables *********************
int ss_step = 50 ;            //size of each movement
int ss_wait = 100 ;           //wait between each movement

//******************  Grabber servo veriables *********************
int grabber_up = 2400;
int grabber_mid = 2550;
int grabber_down = 2650;


int left_servo_stop = 2500;
int right_servo_stop = 2500;


//****************** Obstical Avoidance Veribles *************************
int ir_range = 125;

int speed = 1000;             // rate for forward and backward movements
int forward_factor = 12;      // number of msec to run for an inch
int reverse_factor = 12;      // number of msec to run for an inch
int left_factor = 20;         // number of msec to turn a degree
int right_factor = 20;        // number of msec to turn a degree
```

```
//*********************************************************************
//*
//*         File Name:        L_cal
//*         Purpose:          to calabrate Lothar's servo actators
//*         Version:          6.0
//*
//*         Version Pre-History:
//*               This file is L_servo with sensor calabration added
//*
//*         Version History:
//*               Ver 0 - add sensor (bump and IR) calabration to L_servo
//*               Ver 1 - incorporate color sensing
//*               Ver 2 - built a cool user interface
//*               Ver 3 - added line following and get tile behavoirs
//*               Ver 4 - added the ability to create use to different color tables
//*                           added a tile color calabration
//*               Ver 5 - adding wall following and obstical avoidance
//*                           added a MY_BUTTON
//*               Ver 6 - adding terminal interface
//*
//*********************************************************************

//*****************************  INCLUDES ***********************************
#include <L_base.h>


/************************** Prototypes ********************************/
void init(void);

//*****************************  Servo Prototypes  ******************
void slow_servo_cal(void);
void drive_servo_cal(void);
void grabber_servo_cal(void);
void slow_servo(int servo, int new_pos);
void grabber_servo_test(void);

//*****************************  IR/Bumper Prototypes  ******************
void sensor_cal(void);
void ir_on(char ir);
void ir_off(char ir);
int read_MxAD(char x);

//************  Color Sensor Prototypes  *******************
void init_color(void);
void get_colors_saved(int type);
void get_colors_terminal(void);
void get_colors_servo(void);
void button_servo_get_colors(void);
void build_table(int type);
void color_sort(void);
void calc_table(int type);
int   colorID(int type);

void display_color_values(void);
void display_CSorder(void);
void display_table(int type);

void draw_lothar(void);
```

```
//***********  line_follow Prototypes  ********************
void line_follow(int line_color);
int get_tile(int line_color, int tile_color);
int look_for_line(int line_color);
int check_grabber(int tile_color);
void stop(void);
void lf_forward(int speed);
void lf_reverse(void);
void lf_turn_left(void);
void lf_turn_right(void);
void open_grabber(void);
void close_grabber(void);


//***********  IR Output  ********************
void blink_LED(char theLED);


//*********** Obstical avoidance *************
int obsticle_IR(void);
int forward(int dist);
int reverse(int dist);
int left(int degree);
int right(int degree);
void avoid_obsticles(void);
int find_color(int color);

//***********  wall_follow Prototypes  ********************
void veer_left(void);
void veer_right(void);
void follow_wall(void);

// Menu Stuff
char menu1[]= "\t\t*********************************************\n";
char menu2[]= "\t\t*                                           *\n";
/***************************** Main *********************************/

void main(void)
{
      int pulse_width;
      char menu_input;
      int i;
      init();

      get_colors_saved(MARKER);
      build_table(MARKER);
      get_colors_saved(TILE);
      build_table(TILE);

      //*********  terminal free interface  ***********

      if ((input1 & 0x03) == 0x00)        //mode 0
      {
            while ((input2 & (CENTER_BUMPER | MY_BUTTON)) != (CENTER_BUMPER | MY_BUTTON))
            {
                  printf("%sMode 0, ", clear);
                  printf("Press CENTER bumper to manualy callabrate color sensor\n");
```

```
            printf("Press button to go get to work\n");
            while((input2 & (CENTER_BUMPER | MY_BUTTON )) == 0)
                  blink_LED(LED_ALL);

            if ((input2 & CENTER_BUMPER ) != 0)
            {
                  for (seconds = 0; seconds < 15; i++)
                        blink_LED(LED_GLEFT | LED_GRIGHT);
                  button_servo_get_colors();
                  display_color_values();
                  printf("press LEFT bumper to save colors as TILES\n");
                  printf("press RIGHT bumper to save colors as RIGHT\n");
                  while((input2 & (LEFT_BUMPER | RIGHT_BUMPER)) == 0)
                        blink_LED(LED_GLEFT | LED_GRIGHT);
                  if ((input2 & LEFT_BUMPER) != 0)
                  {
                        ir_off(LED_ALL);
                        ir_on(LED_GLEFT);
                        build_table(TILE);
                  }
                  else
                  {
                        ir_off(LED_ALL);
                        ir_on(LED_GRIGHT);
                        build_table(MARKER);
                  }
            }
            else
            {
                  while (1)
                  {
                        find_color(MARKER_BLACK);
                        if (get_tile(MARKER_BLACK, TILE_GOLD) == TRUE)
                              find_color(MARKER_BLUE);
                        open_grabber();
                  }
            }
      }
}

else if ((input1 & 0x03) == 0x01)         //mode 1
{
      printf("%sMode 1, ", clear);
      while((input2 &  MY_BUTTON ) == 0)
            blink_LED(LED_BACK);
      get_colors_servo();
      display_color_values();
      build_table(MARKER);
      while((input2 &  MY_BUTTON ) == 0)
            blink_LED(LED_GRIGHT | LED_GLEFT);
      get_colors_servo();
      display_color_values();
      build_table(TILE);

      while((input2 &  MY_BUTTON ) == 0)
            blink_LED(LED_RIGHT | LED_LEFT);
      while (1)
      {
```

```
            find_color(MARKER_BLACK);
            get_tile(MARKER_BLACK, TILE_GOLD);
            find_color(MARKER_BLUE);
            open_grabber();
      }

}

else if ((input1 & 0x03) == 0x02)         //mode 2
{
      printf("%sMode 2, ", clear);
      while((input2 &  MY_BUTTON ) == 0)
            blink_LED(LED_RIGHT | LED_LEFT);
      while (1)
      {
            find_color(MARKER_BLACK);
            get_tile(MARKER_BLACK, TILE_GOLD);
            find_color(MARKER_BLUE);
            open_grabber();
      }

}
else if ((input1 & 0x03) == 0x03)         //mode 3
{


}

menu_input = '0';
while(menu_input != 'x')
{
      printf("%s", clear);
      printf("%s", place);
      printf("%s", menu1);
      printf("%s", menu2);
      printf("\t\t*       Lothar's Servo Menu of Power        *\n");
      printf("%s", menu2);
      printf("%s", menu1);

      printf("\t1) Drive Servo Calabrator\n");
      printf("\t2) Slow Servo Calabrator\n");
      printf("\t3) Grabber Servo Calabrator\n");
      printf("\t4) Sensor Calabration\n");
      printf("\t5) get_tile(MARKER_BLACK)\n");
      printf("\t6) line_follow(MARKER_BLACK)\n");
      printf("\t7) Full Grabber Test\n");
      printf("\tx) Exit\n");

      menu_input = getchar();

      if (menu_input == '1')  drive_servo_cal();
      else if (menu_input == '2')   slow_servo_cal();
      else if (menu_input == '3')   grabber_servo_cal();
      else if (menu_input == '4')   sensor_cal();
      else if (menu_input == '5')   get_tile(MARKER_BLACK, TILE_GOLD);
      else if (menu_input == '6')
      {
            init();
```

```
                get_colors_saved(MARKER);
                build_table(MARKER);
                line_follow(MARKER_BLACK);
            }

            else if (menu_input == '7')   ;
            else if (menu_input == 'x')   printf("later");
            else printf("Throw me a bone, %u is not a menu option.", menu_input);
      }
}


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

//*         NAME:       init
//*         PURPOSE:    start-up stuff
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void init(void)
{
      init_servome();
      init_serial();
      init_clocktjp();
      init_analog();
      init_color();


      servo_pos[COLOR_SERVO] = 1400;
      servo(COLOR_SERVO, servo_pos[COLOR_SERVO]);
      wait(300);
      servo(COLOR_SERVO, 0);
      servo_pos[GRABBER] = grabber_up;
}




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

//*         NAME:       sensor_cal
//*         PURPOSE:
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void sensor_cal(void)
{
      int i;
      char c1;
      ir_on(LED_ALL);
```

```
	get_colors_saved(MARKER);
	build_table(MARKER);
	draw_lothar();
	c1 = '0';
	while(c1 != 'x')
	{
		printf("\x1b[4;25H[%u] ",read_MxAD(IR_BACK));		//back IR
		printf("\x1b[11;16H[%u] ",read_MxAD(IR_LEFT));		//left IR
		printf("\x1b[11;35H[%u] ",read_MxAD(IR_RIGHT));		//right IR
		printf("\x1b[12;25H[%u] ",read_MxAD(IR_GROUND));	//center IR
		printf("\x1b[17;23H[%u] ",read_MxAD(IR_GLEFT));		//left grabber IR
		printf("\x1b[17;29H[%u] ",read_MxAD(IR_GRIGHT));	//right  grabber IR

		printf("\x1b[6;25H[%u] ",read_MxAD(BATT));	//BATT level

		if ((input2 & TILE_IN_JAW) != 0)   //TILE_IN_JAW
			printf("\x1b[15;27H!");
		else
			printf("\x1b[15;27H ");

		if ((input2 & UNDER_GRABBER) != 0)     // UNDER_GRABBER
			printf("\x1b[13;27H!");
		else
			printf("\x1b[13;27H ");


		if ((input2 & LEFT_BUMPER) != 0)   // LEFT_BUMPER
			printf("\x1b[17;19H!");
		else
			printf("\x1b[17;19H ");


		if ((input2 & CENTER_BUMPER) != 0)   // CENTER_BUMPER
			printf("\x1b[18;27H!");
		else
			printf("\x1b[18;27H ");

		if ((input2 & RIGHT_BUMPER) != 0)   // RIGHT_BUMPER
			printf("\x1b[17;36H!");
		else
			printf("\x1b[17;36H ");


		if ((input2 & LEFT_ARM) != 0)    //LEFT_ARM
			printf("\x1b[8;28H!");
		else
			printf("\x1b[8;28H ");

		if ((input2 & RIGHT_ARM) != 0)   //RIGHT_ARM
			printf("\x1b[10;28H!");
		else
			printf("\x1b[10;28H ");



///
		printf("\x1b[20;15Hinput1 =");
		for(i = 128; i >= 1; i=i/2)
```

```
{
      if (i == 8)
            printf("-");
      if ((input1 & i) != 0)
            printf("1");
      else
            printf("0");
}

printf("\t input2 =");
for(i = 128; i >= 1; i=i/2)
{
      if (i == 8)
            printf("-");
      if ((input2 & i) != 0)
            printf("1");
      else
            printf("0");
}


printf("\x1b[12;50HMarker:");
i = colorID(MARKER);
if (i == MARKER_BLACK)
      printf("BLACK, %u   ", percent_match);
else if (i == MARKER_RED)
      printf("RED,   %u   ", percent_match);
else if (i == MARKER_BLUE)
      printf("BLUE,  %u   ", percent_match);
else if (i == MARKER_GREEN)
      printf("GREEN, %u   ", percent_match);
else if (i == MARKER_WHITE)
      printf("WHITE, %u   ", percent_match);

printf("\x1b[13;50HTile:  ");
i = colorID(TILE);
if (i == TILE_BLACK)
      printf("BLACK, %u   ", percent_match);
else if (i == TILE_RED)
      printf("RED,   %u   ", percent_match);
else if (i == TILE_BLUE)
      printf("BLUE,  %u   ", percent_match);
else if (i == TILE_GOLD)
      printf("GOLD,  %u   ", percent_match);
else if (i == TILE_WHITE)
      printf("WHITE, %u   ", percent_match);

if ((SCSR & 0x20) != 0)
{
printf("\x1b[21;15H");
      c1 = SCDR;
      if (c1 =='r')
      {
            slow_servo(GRABBER, grabber_up);
            servo(GRABBER, 0);
            printf("grabber up                    ");
      }
      else if (c1 =='g')
```

```
                {
                      slow_servo(GRABBER, grabber_down);
                      printf("grabber down                  ");
                }
                else if (c1 =='c')
                {
                      printf("Calabrate marker or tile (m or t)??     ");
                      c1 = getchar();
                      init_color();
                      get_colors_servo();
                      if (c1 == 'm')
                            build_table(MARKER);
                      else if (c1 == 't')
                            build_table(TILE);
                      draw_lothar();
                }
                else if (c1 =='+')
                {
                      slow_servo(COLOR_SERVO, servo_pos[COLOR_SERVO] - 50);
                      printf("color arm moving left, %u      ", servo_pos[COLOR_SERVO]);
                      servo(COLOR_SERVO, 0);
                }

                else if (c1 =='-')
                {
                      slow_servo(COLOR_SERVO, servo_pos[COLOR_SERVO] + 50);
                      printf("color arm moving right, %u      ", servo_pos[COLOR_SERVO]);
                      servo(COLOR_SERVO, 0);
                }
          }
     }
 }


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

//*         NAME:       draw_lothar
//*         PURPOSE:
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void draw_lothar(void)
{

      printf("%s", clear);
      printf("\t\tSensor Calabration  \n\n");
      printf("          @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n");
      printf("           @                                  @\n");
      printf("            @                                 @\n");
      printf("             @     BATT:                      @\n");
      printf("             @                                @\n");
      printf("             @           _( )_____________@_________          _ \n");
      printf("             @          (_______________________________|-----(_)\n");
      printf("             @            ( )              @\n");
```

```
      printf("             @                          @\n");
      printf("Press:        @                        @\n");
      printf(" g) grab       @@@@@@@@@@( )@@@@@@@@@@\n");
      printf(" r) release      *******<   >******** \n");
      printf(" c) color_cal   *        ( )         * \n");
      printf(" -) arm left    *                    * \n");
      printf(" +) arm right    ( )              ( ) \n");
      printf("                  *******( )******** \n");
}

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

//*         NAME:       drive_servo_cal
//*         PURPOSE:
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void drive_servo_cal(void)
{
      char c1 = '0';
      printf("%s %s %s %s", clear, place, menu1, menu2);
      printf("\t\t*        Drive Servo Calabration  \n");
      printf("%s %s\n\n", menu2, menu1);

      printf("Press: \n");
      printf("\t'+' to raise\n");
      printf("\t'-' to lower\n");
      printf("\t's' to save this value\n");
      servo(LEFT_SERVO, left_servo_stop);
      while(c1 != 's')
      {
            printf("%sleft_servo_stop = %u\n", line15, left_servo_stop);
            c1 = getchar();
            if (c1 == '+')   left_servo_stop++;
            else if (c1 == '-') left_servo_stop--;
            else if (c1 != 's') printf("%sbunk input\n", line15);
            servo(LEFT_SERVO, left_servo_stop);
      }
      c1 = '0';
      servo(RIGHT_SERVO, right_servo_stop);
      while(c1 != 's')
      {
            printf("%s right_servo_stop = %u\n",line16, right_servo_stop);
            c1 = getchar();
            if (c1 == '+')   right_servo_stop++;
            else if (c1 == '-') right_servo_stop--;
            else if (c1 != 's') printf("%s bunk input\n", line16);
            servo(RIGHT_SERVO, right_servo_stop);
      }
      servo(LEFT_SERVO, 0);
      servo(RIGHT_SERVO, 0);
}

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

```
//*

//*         NAME:       slow_servo_cal
//*         PURPOSE:
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void slow_servo_cal(void)
{
      char c1;
      printf("%s %s %s %s", clear, place, menu1, menu2);
      printf("\t\t*        Lothar's slow_servo_cal  \n");
      printf("%s %s\n\n", menu2, menu1);
      while(c1 != 's')
      {
            printf("Enter ss_wait:\n");
            ss_wait = read_int();
            printf("Enter ss_step:\n");
            ss_step = read_int();

            slow_servo(COLOR_SERVO, 1400);
            slow_servo(GRABBER, grabber_up);
            slow_servo(COLOR_SERVO, 3000);
            slow_servo(GRABBER, grabber_down);
            printf("Press 's' to save\n");
            c1 = getchar();
      }
}
//*************************************************************
//*

//*         NAME:       grabber_servo_cal
//*         PURPOSE:    start-up stuff
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void grabber_servo_cal(void)
{
      int pulse_width;
      char c1;

      printf("%s", clear);
      printf("%s", place);
      printf("%s", menu1);
      printf("%s", menu2);
      printf("\t\t*        Lothar's Grabber Calabrator        *\n");
      printf("%s", menu2);
      printf("%s", menu1);

      c1 = 'n';
      slow_servo(GRABBER, grabber_up);
      pulse_width = grabber_up;
```

```
  printf("Up Position\n");
  printf("grabber_up = %u \n", pulse_width);
  printf("Press: \n");
  printf("\t'-' to raise\n");
  printf("\t'+' to lower\n");
printf("\t'b' to go back to original value\n");
  printf("\t's' to save this value\n");
printf("\t'x' to exit\n");

  while(c1 != 'x' && c1 != 's')
  {
        printf("grabber_up %u \n", pulse_width);
        c1 = getchar();
               if (c1 == '+')
              pulse_width = pulse_width + 50;
        else if (c1 == '-')
              pulse_width = pulse_width - 50;
        else if (c1 == 'b')
              pulse_width = grabber_up;
        else if (c1 == 's')
              grabber_up = pulse_width;
        else if (c1 == 'x')
              printf("later");
        else
              printf("\n\n Get with the program butthead\n");


        slow_servo(GRABBER, pulse_width);


  }
  c1 = 'n';
  slow_servo(GRABBER, grabber_down);
  pulse_width = grabber_down;

  printf("Down Position\n");
  printf("grabber_down = %u \n", pulse_width);
  printf("Press: \n");
  printf("\t'-' to raise\n");
  printf("\t'+' to lower\n");
printf("\t'b' to go back to original value\n");
  printf("\t's' to save this value\n");
printf("\t'x' to exit\n");

  while(c1 != 'x' && c1 != 's')
  {
        if ((input2 & TILE_IN_JAW) != 0)
              printf("TILE_IN_JAW = TRUE\n");
        else
              printf("TILE_IN_JAW = FALSE\n");
        printf("grabber_down = %u \n", pulse_width);
        c1 = getchar();
        slow_servo(GRABBER, grabber_up);
        wait(600);
               if (c1 == '+')
              pulse_width = pulse_width + 50;
        else if (c1 == '-')
```

```
                pulse_width = pulse_width - 50;
          else if (c1 == 'b')
                pulse_width = grabber_down;
          else if (c1 == 's')
                grabber_down = pulse_width;
          else if (c1 == 'x')
                printf("later");
          else
                printf("\n\n Get with the program butthead\n");


          slow_servo(GRABBER, pulse_width);

    }
    servo(GRABBER, 0);
}



void grabber_servo_test(void)
{
    char c1;

    c1 = '0';
    printf("%s", clear);
    printf("%s", place);
    printf("%s", menu1);
    printf("%s", menu2);
    printf("\t\t*        Lothar's Grabber Servo Test        *\n");
    printf("%s", menu1);
    printf("%s", menu2);

    printf("\tPress: \n\t 'u' for up, \n\t 'd' for down \n\t 'x' for exit\n");
    while(c1 != 'x')
    {
          c1 = getchar();
          if (c1 == 'u') servo(GRABBER, grabber_up);
          if (c1 == 'd') servo(GRABBER, grabber_down);
          if (c1 != 'u' && c1 != 'd' && c1 != 'x')
          {
                printf("Throw me a bone, '%c' is not a option.\n\n", c1);
          }
    }
}



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

//*         NAME:       slow_servo
//*         PURPOSE:
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
```

```
//*************************************************************
void slow_servo(int the_servo, int new_pos)
{
      if (servo_pos[the_servo] < new_pos)
      {
            while(servo_pos[the_servo] < new_pos)
            {
                  servo_pos[the_servo] = servo_pos[the_servo] + ss_step;
                  servo(the_servo, servo_pos[the_servo]);
                  wait(ss_wait);
            }
      }
      else
      {
            while(servo_pos[the_servo] > new_pos)
            {
                  servo_pos[the_servo] = servo_pos[the_servo] - ss_step;
                  servo(the_servo, servo_pos[the_servo]);
                  wait(ss_wait);
            }
      }
      servo(the_servo, servo_pos[the_servo]);
}

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

//*         NAME:       read_MxAD
//*         PURPOSE:    returns the voltage applyed to pin X of the
//*                           muxed A-D port
//*         INPUT:            X - A to D to be read
//*         OUTPUT:           int value aplied to pin num
//*         EFFECTS:
//*   CALLS:            printf(), analog()
//*         CONSTANTS:  IR_LEFT, IR_GROUND, IR_RIGHT, IR_BACK
//*
//*************************************************************

int read_MxAD(char x)
{
      if (x < 0x00 || x > 0x07)
      {
            printf("bad argument: read_MxAD(%x)\n");
            return 0;
      }

      CLEAR_BIT(_output2,0x07);
      SET_BIT(_output2,x);
      output2 = _output2;
      return analog(0);
}

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

//*         NAME:       ir_on
//*         PURPOSE:    turn on the 40 kHz modulated IR LED
//*         INPUT:            ir - the LED to turn on
```

```
//*         OUTPUT:
//*         EFFECTS:    turns on LED
//*   CALLS:            SET_BIT()
//*         CONSTANTS:  LED_LEFT, LED_RIGHT, LED_GROUND
//*                           LED_BACK, LED_ALL
//*
//*****************************************************************
void ir_on(char ir)
{
      SET_BIT(_output1, ir);
      output1 = _output1;
}

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

//*         NAME:       ir_off
//*         PURPOSE:    turn off the 40 kHz modulated IR LED
//*         INPUT:            ir - the LED to turn off
//*         OUTPUT:
//*         EFFECTS:    turns on LED
//*   CALLS:            CLEAR_BIT()
//*         CONSTANTS:  LED_LEFT, LED_RIGHT, LED_GROUND
//*                           LED_BACK, LED_ALL
//*
//*****************************************************************
void ir_off(char ir)
{
      CLEAR_BIT(_output1, ir);
      output1 = _output1;
}

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

//*         NAME:       init_color
//*         PURPOSE:    initialize and calabrate the color sensor
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*****************************************************************************

void init_color()
{
      printf("init_color()\n");
      color_cal_pos[0]  = 1400;
      color_cal_pos[1]  = 1900;
      color_cal_pos[2]  = 2400;
      color_cal_pos[3]  = 2900;
      color_cal_pos[4]  = 3400;

      if((_output2 & CS_LIGHT) == 0)                  //if the color sensor light is off
      {
            //printf("turning light on\n");           //turn it on

            SET_BIT(_output2, CS_LIGHT);              //
```

```
            output2 = _output2;
            wait(300);                                          //and wait till it warms up
      }
      get_colors_saved(MARKER);
      build_table(MARKER);
      get_colors_saved(TILE);
      build_table(TILE);

}
//*************************************************************
//*

//*         NAME:       get_colors_saved
//*         PURPOSE:    get and store color input for evaluation later
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void get_colors_saved(int type)
{
      if (type == MARKER)
      {
                colors[1][0] = 136;
                colors[1][1] = 100;
                colors[1][2] = 110;
                colors[1][3] = 126;
                colors[1][4] = 89;
                colors[1][5] = 0;
                colors[1][6] = 0;
                colors[1][7] = 0;
                colors[1][8] = 0;
                colors[1][9] = 0;
                colors[1][10] = 0;
                colors[2][0] = 247;
                colors[2][1] = 228;
                colors[2][2] = 226;
                colors[2][3] = 237;
                colors[2][4] = 196;
                colors[2][5] = 0;
                colors[2][6] = 0;
                colors[2][7] = 0;
                colors[2][8] = 0;
                colors[2][9] = 0;
                colors[2][10] = 0;
                colors[3][0] = 242;
                colors[3][1] = 211;
                colors[3][2] = 225;
                colors[3][3] = 239;
                colors[3][4] = 207;

            colors_length = 5;
      }
      else if(type == TILE)
      {
                colors[1][0] = 136;
                colors[1][1] = 98;
```

```
                colors[1][2] = 126;
                colors[1][3] = 89;
                colors[1][4] = 86;
                colors[1][5] = 0;
                colors[1][6] = 0;
                colors[1][7] = 0;
                colors[1][8] = 0;
                colors[1][9] = 0;
                colors[1][10] = 0;
                colors[2][0] = 245;
                colors[2][1] = 227;
                colors[2][2] = 240;
                colors[2][3] = 202;
                colors[2][4] = 198;
                colors[2][5] = 0;
                colors[2][6] = 0;
                colors[2][7] = 0;
                colors[2][8] = 0;
                colors[2][9] = 0;
                colors[2][10] = 0;
                colors[3][0] = 238;
                colors[3][1] = 206;
                colors[3][2] = 236;
                colors[3][3] = 195;
                colors[3][4] = 196;
                colors[3][5] = 0;
                colors[3][6] = 0;
                colors[3][7] = 0;
                colors[3][8] = 0;
                colors[3][9] = 0;
                colors[3][10] = 0;
            colors_length = 5;
      }
      else
            printf("get_colors_saved recieved a bad argument: %u\n",type);
}

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

//*         NAME:       terminal_get_colors

//*         PURPOSE:    get and store color input for evaluation later
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void get_colors_terminal(void)
{
      int i,j;
      char c1;
      printf("Color Cal\n\n");
      printf("Place color  under the sensor and press any key\n\n", i);
      printf("Then press 'r' to reread, 'x' to quit, and any key to continue\n");
      for(i = 0; i < 11 && c1 != 'x'; i++)
      {
```

```
            c1 = 'r';
            while (c1 == 'r')
            {
                  printf("color %u:", i);
                  getchar();
                  colors[1][i] = analog(1);
                  colors[2][i] = analog(2);
                  colors[3][i] = analog(3);
                  printf("%u, %u, %u \n", colors[1][i], colors[2][i], colors[3][i]);
                  c1 = getchar();
            }
      }
      colors_length = i;
}


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

//*         NAME:       servo_get_colors
//*         PURPOSE:    get and store color input for evaluation later
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void get_colors_servo(void)
{
      int i,j;
      char c1;
      printf("get_colors_servo()\n");
      for(i = 0; i < 5; i++)
      {
            slow_servo(COLOR_SERVO, color_cal_pos[i]);
            wait(700);
            colors[1][i] = analog(1);
            colors[2][i] = analog(2);
            colors[3][i] = analog(3);
            printf("color %u: %u, %u, %u \n", i, colors[1][i], colors[2][i], colors[3][i]);
            //getchar();
      }
      colors_length = i;
      slow_servo(COLOR_SERVO, CS_HOME);
      servo(COLOR_SERVO, 0);
}

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

//*         NAME:       button_servo_get_colors

//*         PURPOSE:    get and store color input for evaluation later
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
```

```
//*************************************************************
void button_servo_get_colors(void)
{
      int i,j;
      char c1;
      printf("button_servo_get_colors()\n");
      slow_servo(COLOR_SERVO, color_cal_pos[2]);
      wait(700);
      servo(COLOR_SERVO, 0);
      for(i = 0; i < 5; i++)
      {
            wait(700);
            colors[1][i] = analog(1);
            colors[2][i] = analog(2);
            colors[3][i] = analog(3);
            printf("color %u: %u, %u, %u \n", i, colors[1][i], colors[2][i], colors[3][i]);
            //getchar();
            while((input2 & MY_BUTTON) == 0)
            {
                  if (i == 0)
                        blink_LED(LED_LEFT);
                  else if (i == 1)
                        blink_LED(LED_LEFT | LED_GLEFT);
                  else if (i == 2)
                        blink_LED(LED_LEFT | LED_GLEFT | LED_GRIGHT);
                  else if (i == 3)
                        blink_LED(LED_LEFT | LED_GLEFT | LED_GRIGHT | LED_RIGHT);
                  else if (i == 4)
                        blink_LED(LED_ALL);
            }
            ir_on(LED_ALL);
      }
      colors_length = i;
      slow_servo(COLOR_SERVO, CS_HOME);
      servo(COLOR_SERVO, 0);
}


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

//*         NAME:       build_table
//*         PURPOSE:    initialize and calabrate the color sensor
//*         INPUT:            color_method -    CS_SERVO to use get_colors_servo()
//*                                             CS_DEFAULT to use stored values

//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************************

void build_table(int type)
{
      //printf("build_table(%u)\n", type);
      //display_color_values();
      color_sort();
      //display_CSorder();
```

```
	calc_table(type);
	//display_table(type);
}



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

//*		NAME:		color_sort
//*		PURPOSE:	sort by values the colors for each sensor
//*		INPUT:
//*		RETURNS:
//*		EFFECTS:
//*	CALLS:
//*
//*****************************************************************
void color_sort()
{
	int j, k, i;
	int temp;
	CSorder_length = colors_length;
//fill CSorder with 1,2,3..11 for each sensor
	for (i = 0; i < CSorder_length; i++)
	{
		CSorder[1][i][0] = i;
		CSorder[1][i][1] = colors[1][i];
		CSorder[2][i][0] = i;
		CSorder[2][i][1] = colors[2][i];
		CSorder[3][i][0] = i;
		CSorder[3][i][1] = colors[3][i];
	}

	//display_CSorder();
	//printf("sorting\n");

	for (i = 1; i<= 3; i++)
	{
		for (j = CSorder_length -1; j >= 0; j--)
		{
			for (k = 0; k < j; k++)
			{
				if (CSorder[i][k][1] > CSorder[i][k+1][1])
				{
					temp = CSorder[i][k][1];
					CSorder[i][k][1] = CSorder[i][k+1][1];
					CSorder[i][k+1][1] = temp;

					temp = CSorder[i][k][0];
					CSorder[i][k][0] = CSorder[i][k+1][0];
					CSorder[i][k+1][0] = temp;
				}
				//printf("%u, %u\n", CSorder[i][k][0], CSorder[i][k][1]);
			}
			//printf("\n");
		}
	}
}
```

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

//*         NAME:       calc_table
//*         PURPOSE:    sort by values the colors for each sensor
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void calc_table(int type)
{
      int i, j;

      table_length[type] = CSorder_length - 1;
      for (i = 1; i <= 3; i++)
      {
            for (j = 0; j < table_length[type] ; j++)
            {
                  color_table[type][i][j] = ((CSorder[i][j+1][1] - CSorder[i][j][1] + 1)/2 ) +
CSorder[i][j][1];
            }
      }
}

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

//*         NAME:       colorID
//*         PURPOSE:    to determine what color is under the sensor
//*         INPUT:
//*         RETURNS:    int - color matching the current values
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
int colorID(int type)
{
      int i,j,sensor[4];
      wait(100);
      for (i = 1; i <= 3; i++)                                    //get sensor values
            sensor[i] = analog(i);                                      //

      for (i = 1; i <= 3; i++)                                    //determine which color
matches
      {                                                                     //the values
for each sensor
            j = 0;
            for(j = 0; (sensor[i] >= color_table[type][i][j]) && (j < table_length[type]); j++);
            colorID_out[type][i] = CSorder[i][j][0];
            //printf("colorID(%u):  i= %u j= %u\n",type, i, j);
            //printf("colorID_out[type][i] =  %u   CSorder[i][j][0]= %u\n",colorID_out[i],
CSorder[i][j][0]);
      }
```

```
      if (colorID_out[type][1] == colorID_out[type][2] && colorID_out[type][1] ==
colorID_out[type][3])
      {
            percent_match = 100;
            return colorID_out[type][1];
      }

      else if (colorID_out[type][1] == colorID_out[type][2])
      {
            percent_match = 66;
            return colorID_out[type][1];
      }

      else if (colorID_out[type][1] == colorID_out[type][3])
      {
            percent_match = 66;
            return colorID_out[type][1];
      }

      else if (colorID_out[type][2] == colorID_out[type][3])
      {
            percent_match = 66;
            return colorID_out[type][2];
      }
      else if (colorID_out[type][1] != colorID_out[type][2] && colorID_out[type][1] !=
colorID_out[type][3])
      {
            percent_match = 33;
            return (colorID_out[type][1] + colorID_out[type][2] + colorID_out[type][3])/3;
      }
      else
      {
            printf("/n/nERROR: colorID(MARKER) is funky/n");
            return 99;
      }
}



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

//*         NAME:       display_CSorder
//*         PURPOSE:    displays the contents of CSorder
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void display_CSorder()
{
      int i;
      printf("CSorder:\n");
      printf("   1\t   2\t  3\n\n");
      for (i = 0; i < 11; i++)
      {
            printf("%u  %u\t", CSorder[1][i][0], CSorder[1][i][1]);
```

```
            printf("%u  %u\t", CSorder[2][i][0], CSorder[2][i][1]);
            printf("%u  %u\n", CSorder[3][i][0], CSorder[3][i][1]);
      }
      printf("\n");
}

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

//*         NAME:       display_table
//*         PURPOSE:    displays the contents of CSorder
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void display_table(int type)
{
      int i;
      printf("color_table[%u]:\n", type);
      for (i = 0; i < 10 && color_table[type][1][i] != 0; i++)
      {
            printf("\tcolor_table[%u][1][%u] = %u\n", type, i, color_table[type][1][i]);
            printf("\tcolor_table[%u][2][%u] = %u\n", type, i, color_table[type][2][i]);
            printf("\tcolor_table[%u][3][%u] = %u\n", type, i, color_table[type][3][i]);
      }
      printf("\n");
}
//*************************************************************
//*

//*         NAME:       display_color_values

//*         PURPOSE:    displays the contents of CSorder
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void display_color_values()
{
      int j,i;
      for(j = 1; j<=3; j++)
      {
            for(i = 0; i <= 10; i++)
            {
                  printf("\t\tcolors[%u][%u] = %u;\n", j,i, colors[j][i]);
            }
      }
}
```

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

//*         NAME:       get_tile
//*         PURPOSE:    follow a colored line while opening and closing
//*                           the gripper till a tile is in the proper position
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
int get_tile(int line_color, int tile_color)
{
      int g_result;
      int steps_sence_grab = 5;
      int on_line = TRUE;

      slow_servo(COLOR_SERVO, CS_HOME);
      servo(COLOR_SERVO, 0);
      ir_on(LED_ALL);
      while(on_line == TRUE)
      {
            printf("colorID(MARKER) = %u", colorID(MARKER));
            while(colorID(MARKER) == line_color && percent_match >= 66)
            {
                  lf_forward(100);
                  printf("on line and going lf_forward\n");
                  wait(20);
                  stop();
                  if (steps_sence_grab >= 5)
                  {
                        steps_sence_grab = 0;
                        g_result = check_grabber(tile_color);
                        if(g_result == TILE_UNDER_GRABBER)
                        {
                              while(g_result == TILE_UNDER_GRABBER)
                              {
                                    printf("TILE_UNDER_GRABBER loop g_result == %u\n",
g_result);
                                    lf_forward(100);
                                    wait(70);
                                    stop();
                                    g_result = check_grabber(tile_color);
                              }
                        }

                        if (g_result == TRUE)
                        {
                              servo(GRABBER, grabber_mid);
                              return TRUE;
                        }
                        else if(g_result == WRONG_COLOR)
                        {
                              lf_reverse();
                              wait(600);
                              stop();
                              servo(GRABBER, grabber_mid);
```

```
                              lf_forward(400);
                              wait(600);
                              stop();
                              open_grabber();
                              if (find_color(MARKER_BLACK) == FALSE)
                                    return FALSE;
                        }

                  }
                  else steps_sence_grab++;

            }
            open_grabber();
            on_line = look_for_line(line_color);
      }
      return FALSE;
}



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

//*         NAME:       check_grabber
//*         PURPOSE:    start-up stuff
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
int check_grabber(int tile_color)
{
      close_grabber();
      wait(300);
      if ((input2 & TILE_IN_JAW) != 0)
      {
            printf("YIPPY, tile in jaw\n");
            slow_servo(COLOR_SERVO, CS_TILE);
            servo(COLOR_SERVO, 0);
            if (colorID(TILE) == tile_color)
            {
                  printf("I found the tile I was looking for!!\n");
                  seconds = 0;
                  while(seconds < 20)
                        blink_LED(LED_ALL);
                  slow_servo(COLOR_SERVO, CS_HOME);
                  return TRUE;
            }
            else
            {
                  printf("nope, that's not it\n");
                  slow_servo(COLOR_SERVO, CS_HOME);
                  servo(COLOR_SERVO, 0);
                  open_grabber();
                  return WRONG_COLOR;
            }
      }
```

```
      else if ((input2 & UNDER_GRABBER) == 0)
      {
            printf("there's a tile under the foot\n");
            open_grabber();
            return TILE_UNDER_GRABBER;
      }
      else
      {
            printf("the tile holder is empty\n");
            open_grabber();
            return FALSE;
      }
}


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

//*         NAME:       line_follow
//*         PURPOSE:
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
void line_follow(int line_color)
{
      int i;
      int on_line = TRUE;


      printf("line_follow(%u)", line_color);

      while(on_line == TRUE)
      {
            printf("colorID(MARKER) = %u", colorID(MARKER));
            while(colorID(MARKER) == line_color && percent_match >= 66)
            {
                  lf_forward(100);
                  printf("on line and going lf_forward\n");
                  wait(10);
                  stop();
            }
            on_line = look_for_line(line_color);
      }
}

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

//*         NAME:       look_for_line
//*         PURPOSE:
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
```

```
//*************************************************************
int look_for_line(int line_color)
{
      int i, last_PM;
      int turns = 12;
      int turn_time = 20;
      int lost_line = 0;
      static int last_turn;
      while (lost_line < 2)
      {
            if (last_turn == LEFT)
            {
                  printf("lost the line and turning left\n");
                  last_PM == percent_match;
                  for(i = 0; (colorID(MARKER) != line_color && percent_match >= last_PM ) && (i
< turns); i++)
                  {
                        lf_turn_left();
                        wait(turn_time);
                        last_PM == percent_match;
                        stop();
                        wait(100);
                  }
                  if (i == turns)
                  {
                        printf("turned left and didn't find the line\n");
                        lf_turn_right();
                        wait((turns*turn_time)+50);
                        stop();
                        last_turn = RIGHT;
                        lost_line++;
                  }
                  else
                  {
                        printf("turned left and found the line\n");
                        last_turn = LEFT;
                        lost_line = 0;
                        return TRUE;
                  }
            }

            else if (last_turn == RIGHT)
            {
                  printf("lost the line turning right\n");
                  last_PM == percent_match;
                  for(i = 0; (colorID(MARKER) != line_color && percent_match >= last_PM ) && (i
< turns); i++)
                  {
                        lf_turn_right();
                        wait(turn_time);
                        last_PM == percent_match;
                        stop();
                        wait(100);
                  }
                  if (i == turns)
                  {
                        printf("turned right and didn't find the line\n");
                        lf_turn_left();
```

```
                        wait((turns*turn_time)+50);
                        stop();
                        last_turn = LEFT;
                        lost_line++;
                  }
                  else
                  {
                        printf("turned right and found the line\n");
                        last_turn = RIGHT;
                        lost_line = 0;
                        return TRUE;
                  }
            }
            if (lost_line >= 2)
            {
                  stop();
                  printf("I lost the line\n");
                  return FALSE;
            }
      }
      return FALSE;
}

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

//*         NAME:       lf_forward
//*         PURPOSE:    move lothar forward
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:    turns on left and right servos
//*   CALLS:            servo()
//*
//*************************************************************
void lf_forward(int speed)
{
      servo(LEFT_SERVO, 2500 - speed);                    //go forward
      servo(RIGHT_SERVO, 2500 + speed);                         //
                  //

}

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

//*         NAME:       lf_reverse
//*         PURPOSE:    move lothar in reverse
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:    turns on left and right servos
//*   CALLS:            servo()
//*
//*************************************************************
void lf_reverse()
{
      servo(LEFT_SERVO, 3000);                        //go forward
      servo(RIGHT_SERVO, 2000);                             //
}
```

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

//*         NAME:       stop
//*         PURPOSE:    stops lothar from moving
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:    turns off left and right servos
//*   CALLS:            servo()
//*
//*************************************************************
void stop()
{
      servo(LEFT_SERVO, 0);                           //go forward
      servo(RIGHT_SERVO, 0);                                //
}


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

//*         NAME:       lf_turn_left
//*         PURPOSE:    turns Lothar left
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:    turns on left and right servos
//*   CALLS:            servo()
//*
//*************************************************************
void lf_turn_left()
{
      servo(LEFT_SERVO, 2850);                        //go forward
      servo(RIGHT_SERVO, 2850);                             //
}

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

//*         NAME:       lf_turn_right
//*         PURPOSE:    turns Lothar right
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:    turns on left and right servos
//*   CALLS:            servo()
//*
//*************************************************************
void lf_turn_right()
{
      servo(LEFT_SERVO, 2250);                        //go forward
      servo(RIGHT_SERVO, 2250);                             //
}

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

//*         NAME:       open_grabber
//*         PURPOSE:
```

```
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*****************************************************************
void open_grabber(void)
{
      servo(COLOR_SERVO, 0);
      slow_servo(GRABBER, grabber_up);
      servo(GRABBER, 0);
}

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

//*         NAME:       close_grabber
//*         PURPOSE:
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*****************************************************************
void close_grabber(void)
{
      servo(COLOR_SERVO, 0);
      slow_servo(GRABBER, grabber_down);
}


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

//*         NAME:             blink_LED
//*         PURPOSE:
//*         INPUT:
//*         RETURNS:
//*         EFFECTS:
//*   CALLS:
//*
//*****************************************************************
void blink_LED(char the_LED)
{
      ir_off(LED_ALL);
      wait(100);
      ir_on(the_LED);
      wait(100);
}


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

//*         NAME:       -forward
//*         PURPOSE:    move lothar forward <dist> inches
//*         INPUT:            distence  - number of inches to move forward
```

```
//*         OUTPUT:           int - TRUE if compleated, FALSE if not
//*         EFFECTS:    turns on left and right servos and IR's
//*   CALLS:            servo(), obsticle_IR(), ir_on(), ir_off()
//*
//*************************************************************
int forward(int dist)
{
      printf("forward(%u)\n", dist);
      if (_output1 != LED_ALL)
      {
            printf("LEDs wern't on, turning them on\n");
            ir_on(LED_ALL);
            wait(200);
      }

      servo(LEFT_SERVO, 2500 - speed);                              //go forward
      servo(RIGHT_SERVO, 2500 + speed);                                   //
      msec = 0;
      seconds = 0;
      while (seconds <= (dist*forward_factor)/1000)         //while no obsticle and
      {                                                                   //distence
has not been reached
            if (obsticle_IR() == TRUE)
            {
                  stop();
                  printf("Shit's in the way, had to stop\n");
                  return FALSE;
            }
      }


      while (msec <= (dist*forward_factor)%1000);                 //while no obsticle and
      {                                                                   //distence
has not been reached
            if (obsticle_IR() == TRUE)
            {
                  stop();
                  printf("Shit's in the way, had to stop\n");
                  return FALSE;
            }
      }
      printf("It's all good G\n");
      return TRUE;
}

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

//*         NAME:       reverse
//*         PURPOSE:    move lothar forward <dist> inches
//*         INPUT:            distence  - number of inches to move forward

//*         OUTPUT:           int - TRUE if compleated, FALSE if not
//*         EFFECTS:    turns on left and right servos and IR's
//*   CALLS:            servo(), obsticle_IR(), ir_on(), ir_off()
//*
//*************************************************************
int reverse(int dist)
```

```
{
      printf("reverse(%u)\n", dist);
      ir_off(LED_ALL);
      ir_on(LED_BACK);

      servo(LEFT_SERVO, 2500 + speed);                              //go forward
      servo(RIGHT_SERVO, 2500 - speed);                                   //
      msec = 0;
      seconds = 0;
      while (seconds <= (dist*reverse_factor)/1000)         //while no obsticle and
      {                                                                                 //distence
has not been reached
            if (read_MxAD(IR_BACK)   > ir_range)
            {
                  stop();
                  printf("Shit's in the way, had to stop\n");
                  return FALSE;
            }
      }


      while (msec <= (dist*reverse_factor)%1000);                 //while no obsticle and
      {                                                                                 //distence
has not been reached
            if (read_MxAD(IR_BACK)   > ir_range)
            {
                  stop();
                  printf("Shit's in the way, had to stop\n");
                  return FALSE;
            }
      }
      printf("It's all good G\n");
      stop();
      return TRUE;
}



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

//*         NAME:       left
//*         PURPOSE:    move lothar forward <dist> inches
//*         INPUT:            distence  - number of inches to move forward

//*         OUTPUT:           int - TRUE if compleated, FALSE if not
//*         EFFECTS:    turns on left and right servos and IR's
//*   CALLS:            servo(), obsticle_IR(), ir_on(), ir_off()
//*
//*************************************************************
int left(int degree)
{

      printf("left(%u)\n", degree);

      servo(LEFT_SERVO, 2525);                                          //turn left
      servo(RIGHT_SERVO, 2525);                                         //
```

```
      msec = 0;
      seconds = 0;
      while (seconds <= (degree*left_factor)/1000)          //while no obsticle and
      {                                                                       //distence
has not been reached
            if (read_MxAD(IR_LEFT) > ir_range ||
                  read_MxAD(IR_GLEFT) > ir_range)
            {
                  stop();
                  printf("Shit's in the way, had to stop\n");
                  return FALSE;
            }
      }


      while (msec <= (degree*left_factor)%1000);                  //while no obsticle and
      {                                                                       //distence
has not been reached
            if (read_MxAD(IR_LEFT) > ir_range ||
                  read_MxAD(IR_GLEFT) > ir_range)
            {
                  stop();
                  printf("Shit's in the way, had to stop\n");
                  return FALSE;
            }
      }
      printf("It's all good G\n");
      stop();
      return TRUE;
}


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

//*         NAME:       right
//*         PURPOSE:
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
int right(int degree)
{
      printf("right(%u)\n", degree);

      servo(LEFT_SERVO, 2475);                                    //turn right
      servo(RIGHT_SERVO, 2475);                                   //

      msec = 0;
      seconds = 0;
      while (seconds <= (degree*right_factor)/1000)         //while no obsticle and
      {                                                                       //distence
has not been reached
            if (read_MxAD(IR_RIGHT) > ir_range||
                  read_MxAD(IR_GRIGHT) > ir_range)
            {
```

```
                  stop();
                  printf("Shit's in the way, had to stop\n");
                  return FALSE;
            }
      }


      while (msec <= (degree*right_factor)%1000);                  //while no obsticle and
      {                                                                        //distence
has not been reached
            if (read_MxAD(IR_RIGHT) > ir_range||
                  read_MxAD(IR_GRIGHT) > ir_range)
            {
                  stop();
                  printf("Shit's in the way, had to stop\n");
                  return FALSE;
            }
      }
      printf("It's all good G\n");
      stop();
      return TRUE;
}



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

//*          NAME:       obsticle_IR
//*          PURPOSE:    return TRUE if an obsticle is found FALSE not
//*          INPUT:
//*          OUTPUT:           int - TRUE if an obsticle is found FALSE not
//*          EFFECTS:    reads IR sensors
//*    CALLS:            read_MxAD()
//*          CONSTANTS:  IR_LEFT, IR_GROUND, IR_RIGHT, IR_BACK
//*
//*************************************************************
int obsticle_IR()
{
      if (  read_MxAD(IR_RIGHT)  < ir_range &&
                  read_MxAD(IR_GLEFT)  < ir_range &&
                  read_MxAD(IR_GRIGHT)  < ir_range &&
                  read_MxAD(IR_BACK)   < ir_range)
      {
            printf("obsticle_IR() = false\n");
            return FALSE;
      }
      else
      {
            printf("obsticle_IR() = true %u\t %u\t %u\t %u\t %u\t \n",read_MxAD(IR_GROUND)
,read_MxAD(IR_RIGHT), read_MxAD(IR_GLEFT),      read_MxAD(IR_GRIGHT) ,read_MxAD(IR_BACK));
            return TRUE;
      }
}


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

```
//*

//*         NAME:       avoid_obsticles
//*         PURPOSE:
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:
//*   CALLS:
//*         CONSTANTS:
//*
//*************************************************************
void avoid_obsticles()
{
      int board = 0;
      ir_on(LED_ALL);
      minutes = 0;
      seconds = 0;
      slow_servo(COLOR_SERVO, 2000);
      servo(COLOR_SERVO,0);

      ir_on(LED_ALL);
      while (minutes < 1  && read_MxAD(IR_BACK) < ir_range)
      {
            while(board < 10 && read_MxAD(IR_BACK) < ir_range)
            {
                  if (obsticle_IR() == FALSE)
                  {
                        printf("Nothing in the way, going forward\n");
                        forward(5);
                        board = 0;
                  }
                  else
                  {
                        printf("obstical ahead\n");
                        board++;
                        if( read_MxAD(IR_GLEFT) < read_MxAD(IR_GRIGHT))
                  {
                        printf("turning left\n");
                        left(20);
                  }
                  else
                  {
                        printf("turning right\n");
                              right(20);
                        }
                  }
            }
            printf("I'm jammed backing up\n");
            reverse(1);
            board = 0;
      }
      if (read_MxAD(IR_BACK) < ir_range)
      {
            printf("bumper stop\n");
      }
}

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

```
//*

//*         NAME:       find_color
//*         PURPOSE:
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:
//*   CALLS:
//*         CONSTANTS:
//*
//*************************************************************
int find_color(int color)
{
      int board = 0;
      minutes = 0;
      seconds = 0;

      ir_on(LED_ALL);
      printf("find_color(%u)\n", color);
      slow_servo(COLOR_SERVO, CS_HOME);
      servo(COLOR_SERVO,0);
      while (minutes < 1 && (colorID(MARKER) != color || percent_match < 66))
      {
            while(board < 10 && (colorID(MARKER) != color || percent_match < 66))
            {

                  if (obsticle_IR() == FALSE )
                  {
                        printf("Nothing in the way, going forward\n");
                        lf_forward(100);
                        wait(50);
                        stop();
                        board = 0;
                  }
                  else
                  {
                        printf("obstical ahead\n");
                        board++;
                        if( read_MxAD(IR_GLEFT) < read_MxAD(IR_GRIGHT))
                  {
                        printf("turning left\n");
                        left(20);
                  }
                  else
                  {
                        printf("turning right\n");
                              right(20);
                        }
                  }
            }
            if (board == 10)
            {
                  printf("I'm jammed backing up\n");
                  reverse(1);
                  board = 0;
            }
      }
      if (colorID(MARKER) == color && percent_match >= 66)
```

```
      {
            printf("found the color\n");
            slow_servo(GRABBER, grabber_up);
            return TRUE;
      }
      else
      {
            printf("found squat\n");
            return FALSE;
      }
}

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

//*         NAME:       veer_left
//*         PURPOSE:    moves Lothar forward while turning to the left
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:    turns on left and right servos
//*   CALLS:            servo()
//*
//*************************************************************
void veer_left()
{
      servo(LEFT_SERVO, 2400);
      servo(RIGHT_SERVO, 2950);
}

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

//*         NAME:       veer_right
//*         PURPOSE:    moves Lothar forward while turning to the left
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:    turns on left and right servos
//*   CALLS:            servo()
//*
//*************************************************************
void veer_right()
{
      servo(LEFT_SERVO, 2500);
      servo(RIGHT_SERVO, 3000);
}


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

//*         NAME:       follow_wall
//*         PURPOSE:
//*         INPUT:
//*         OUTPUT:
//*         EFFECTS:
//*   CALLS:
//*
//*************************************************************
```

```
void follow_wall()
{
      seconds = 0;
      minutes = 0;
      ir_on(LED_ALL);
      while(minutes < 2)
      {

            while (obsticle_IR() == FALSE & (read_MxAD(IR_LEFT) <= 120 && read_MxAD(IR_LEFT)
>=100))
            {
                  printf("I'm following a wall   \n");
                  lf_forward(50);
            }

            while (read_MxAD(IR_LEFT) > 120)
            {
                  lf_turn_right();
            }

            while (read_MxAD(IR_LEFT) < 100)
            {
                  veer_left();
            }


            if (obsticle_IR() == TRUE)
            {
                  printf("%sSMEG ahead turning right     ");
                  lf_turn_right();
            }

      }
      stop();
}
27


27
```