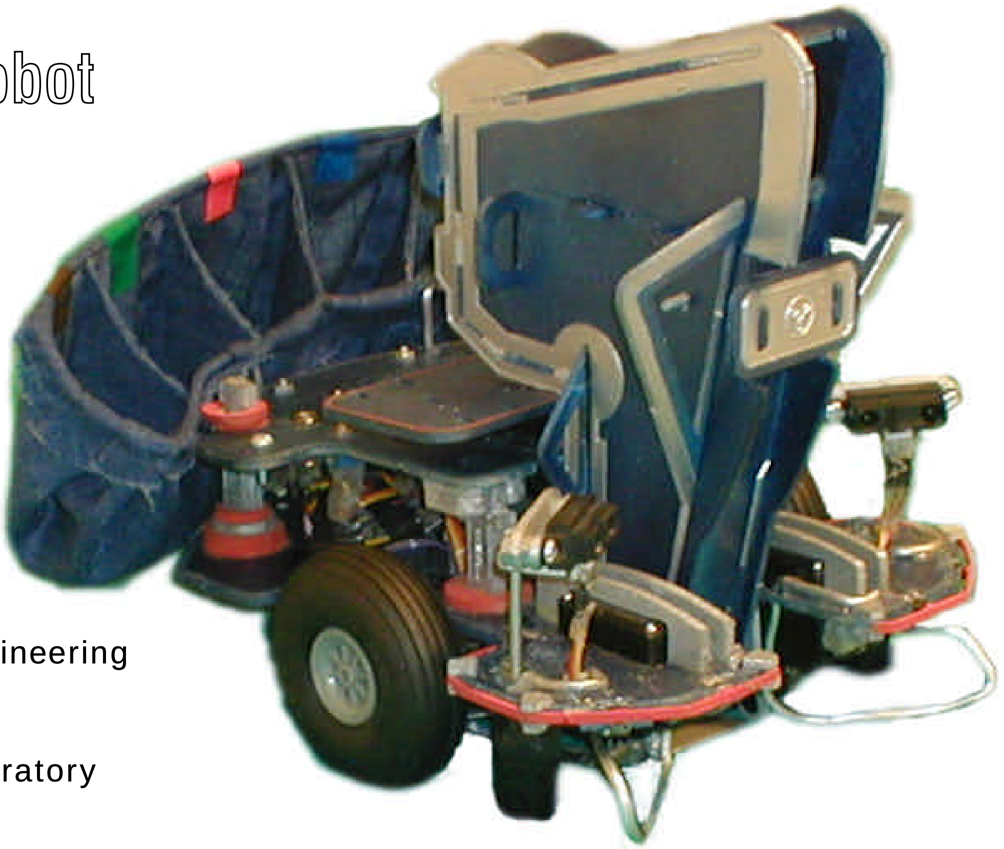
EM

The M&M™ Sorting Robot



University of Florida
Dept. of Computer & Electrical Engineering

EEL 5666C
Intelligent Machines Design Laboratory
Spring 2002

Final Paper
April 23, 2002
Vinh Trinh

1.0 ABSTRACT

Color detection has been a sparsly travelled path in introductory robotics courses due to the lack of cheap reliable color detection schemes, uncontrollable environments, and success rates of at most 50%. To address this issue I have developed a process that is cheap, easy to reproduce, and has an accuracy of close to 100%. My robot, although it does not do anything particulary useful, is a wonderful "proof of concept" example that color detection can be done painlessly, and accurately. My hope is to provide future students with the confidence in knowing that a cheap reliable color detection scheme is well within their grasps, opening a doorway to more complex robots which implement color detection.

2.0 ACKNOWLEDGEMENTS

I would like to thank Aamir Qaiyumi, Uriel Rodriguez, and Prof. Arroyo for their guidance, patience, and motivation throughout this coarse, Dr. Schwartz for equipming me with the skills required to code and debug hardware/software issues (A skill only acquired after the completion of EEL 4744), And my girlfirend Heather Bryant, for building my M&M bins.

3.0 EXECUTIVE SUMMARY

My robot's name is EM. EM moves around randomly on a tabletop avoiding obstacles and the table edge. In the meantime, EM will funnel in any M&M™ candies in its path towards a break beam that initiates a sorting sequence. EM will then determine the color of the M&M™ and place it into the corresponding color bin also located on the robot.
EM utilizes "Edge of the world Dectectors" to detect table top edges, IR Emitter/Detectors to detect obstacles that are close, and a bump switch network which is used as a backup for the IR Emitter/Detectors.

4.0 THE BASICS

4.1 Development Board

HC11 Based TJPRO II By MEKATRONIX
(http://www.mekatronix.com)

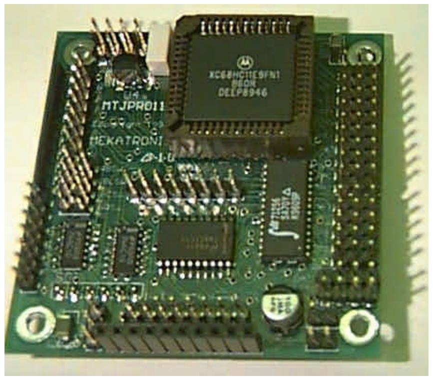


Figure 1

Cost: $90

This little board is unbelievable. At only 2.5" by 2.5" in size it can be easily mounted on even the smallest robots and can be easily hidden out of site. Many students made the mistake of purchasing the MRC11 + EXPANSION enticed by the 64k of RAM and a few more features. However I have noticed that the limiting features have consistently been the availabilty of Analog to Digital Converters, and the Amount of Servo Control, not RAM (I have yet to meet a person that has written more than 20k of code) or the ability to use DC Motors. If you plan on using DC Motor control then I suggest the MRC11+EXPANSION since it has built in a built in H-Bridge Network. But for most begining robotics students I highly recommend going with the TJPRO II.

4.2 Bump Detection

4 - Bump Switches

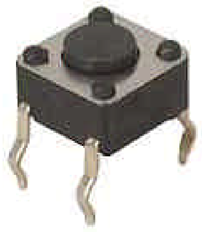
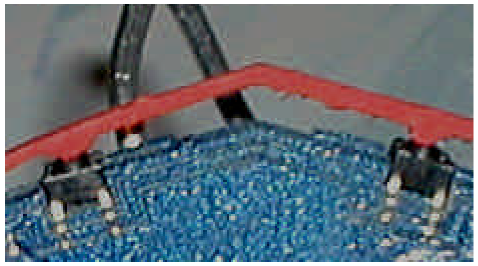



Figure 2

Cost: Free in the lab

Press the button, it completes the circuit, enough said.
The TJ PROII has a voltage divided bump switch network built in that is connected to an A to D port. My robot rarely bumps into things but I put these on anyway since they are a great way to initiate operating modes for your robot. For instance, if your robot needs calibration, you can write your calibration routines to follow if a back bump switch is pressed. And then run the main program when the front bump switch is pressed and so on. It is a common practice to put a dab of superglue on the surface of the button so that a robot's bumper stays in contact with the switch.

4.3 Object Detection

2 - SHARP GP2D12: IR Emitter/Detector Pair

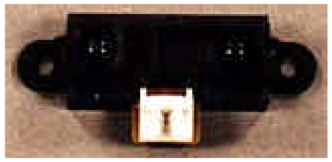


Figure 3

Strengths:

- Built in 40khz Oscillator
- Cheap
- Reliable
- No Hassle, Plug and Play

Weaknesses:

- Power Hungry
- Cannot be powered by the digital outs of the microcontroller

Cost: Approx $15 each (available at MEKATRONIX)

These things work beautifully. I mounted 2 of them in the front of my robot, angling them towards the center. This allows me to detect objects which are coming head on using only 2 emitter/detector pairs. They come built in with a 40khz modulation/demodulation circuitry and are virtually plug and play devices. Be sure however to give these units their own 5V regulated supply because if you try to power them off the digital outs of a uC they will most likely reset the board since they pull a lot of current. Save yourself some time and buy these things so that you can work on the more important aspects of your robot.

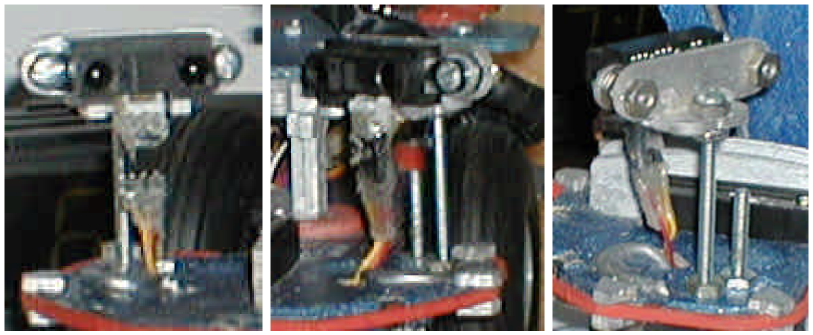


Figure 4

4.4 Edge of the World Detection

2 - CDS cells
2 - Ultra Bright LEDs
2 - 47 kOhm Resistors

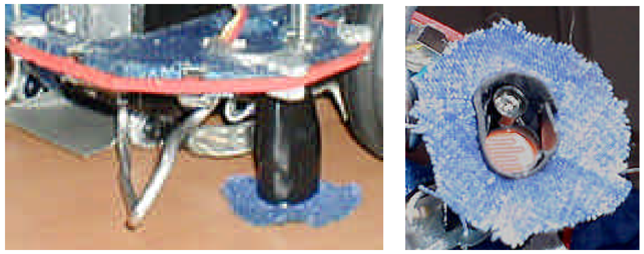


FIgure 5

Cost:

- $5 For Ulta Bright LED's
- CDS Cells and resistors are Free in lab

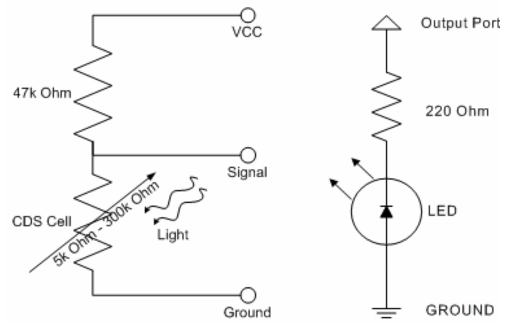


Figure 6

Edge of the world detection is done by detecting variances in light. To detect light, CDS Cells are commonly used. As you can see from the pictures, I have columnated the entire unit along with the LED to prevent any direct light from hitting the CDS Cell. I have also added a cloth skirt to further prevent any reflective ambient light to make its way to the CDS cell. When operating in a Bright Room (much like the MIL lab) I turn the LEDS off and look for bright ambient light to signify table edges. In dark rooms, the LEDS are turned on and the CDS Edge detection network is programmed to look for DARK SPOTS to signify table edges. This method of edge of the world detection is very reliable and works well just as long as you set the correct mode of operation, i.e. light room or dark room.

4.5 Wheel Actuation

2 - Hacked T-53 Tower Hobby Servos
(http://www.towerhobbies.com)

Figure 7

Cost:
$10 Each at tower hobbies

"Hacking" a servo means to modify a servo so that it continously rotates. This allows a servo to act as a Pulse Width Controlled motor for small robots. For details on how to hack a servo please refer to this website.

http://www.rdrop.com/~marvin/explore/servhack.htm

The only drawback of using a hacked servo is that they have to be calibrated in order to function properly. Mis-Calibration will prohibit your robot from moving straight or being able to turn in place. Also, If speed is a major issue I would go with more powerful DC motors, however an H-Bridge will be necessary to implement DC motor control.

5.0 MOBILE PLATFORM

I completely designed EM from the ground up using AutoCAD 2002. During the AutoCAD learning process I've went through 3 platform revisions, the third is the one you see now. EM is built with 5-ply 1/8" thick aircraft grade plywood and was cut out on the T-TECH machine in the IMDL lab.

5.1 Design Constraints

- Fit comfortably Inside my bag
  - -My bag measures 8" by 15"
  - -To be able to transport my robot around easily and inconspicuously
- Centered Wheels
  - - allows the robot to turn in place
  - - ensures that the robot's turning radius is the same size as the robot's length.
- Ability to Hide wires, battery, and uC Board
  - - Adds to the overall aesthetic of the robot
- Easy Access to uC Board
- Will provide 2 Degrees of Freedom for a Sorting Arm.
- Does not interfere with the movement of a Sorting Arm.
- Has some sort of ARM that can pick up m&ms

After evaluating and re-evaluating the constraints I designed the following 4 main parts of EM.

1.) The Chassis
2.) The uC Board Mount
3.) The Rotating Platform
4.) The Sorting Unit

5.1.2 Bill of Materials

- 5 Servo Motors ($10 each at tower hobby)
- 1 Thick tin modeling wire ($3 at Michaels Craft Store)
- 4 Bump Switches (Free in lab)
- 1 Yard of Cloth ($1 at walmart)
- 48" x 12" of 5-ply 1/8" aircraft grade plywood
- 2 IR Emitter Detector Pairs
- 1 TJPRO II development board
- 3 Cans of spraypaint (Black,Silver,Blue)
- 1 tube of orange enamel paint
- 3 tubes of GOOP

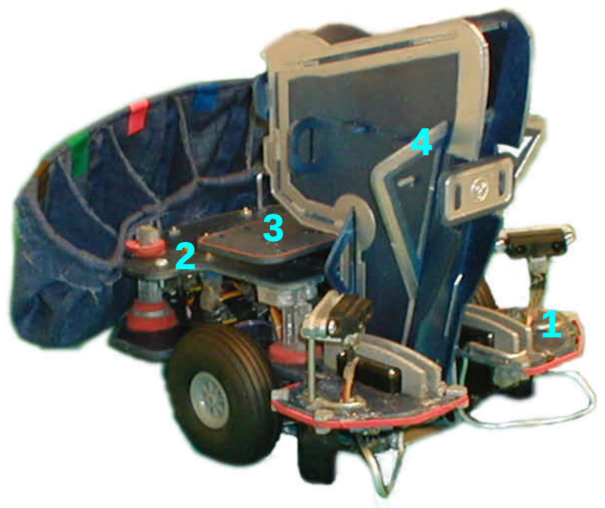


Figure 8

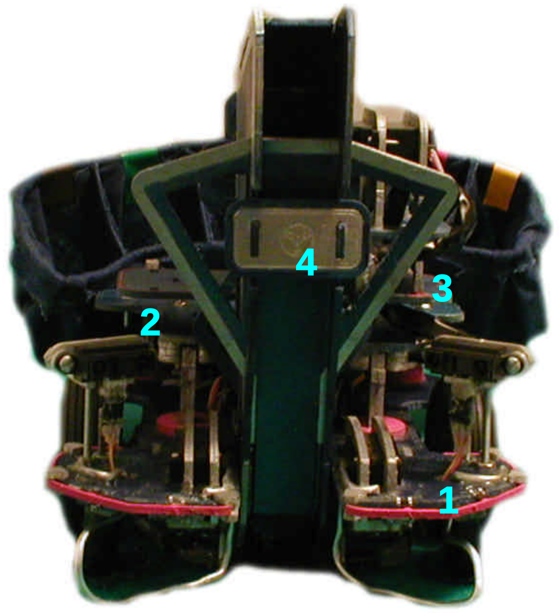


Figure 9

5.2 The Chassis

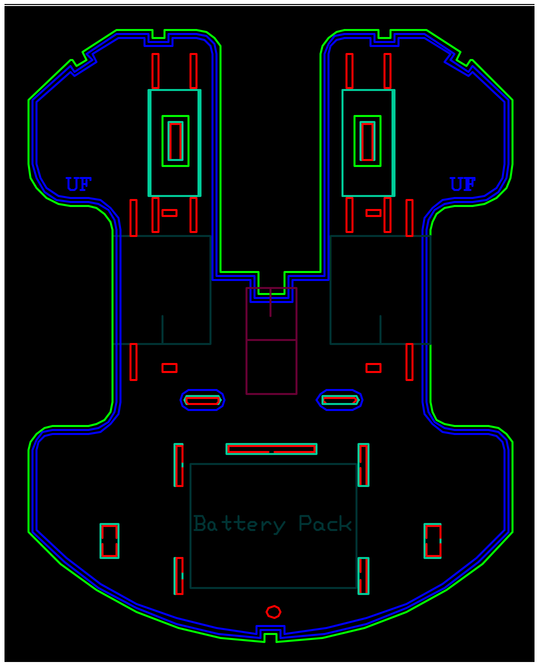


Figure 10

The maximum length of the robot was not to exceed 10", and so the chassis design began with a 9.5" diameter circle. In order to fit inside my bag, the robot could be no greater than 7.5" in width. The trim tool was used to trim off the sides of the circle until it met this specification. I then cut out centered indention on each side for the wheels and small indentions on the front and back for the bump switches. Then one long indention was cut out from the front to allow for sorting arm movement. Finally, the fillet tool was used to smoothen out sharp corners, and the correct holes were added for connecting other components.

5.3 Rotating Platform

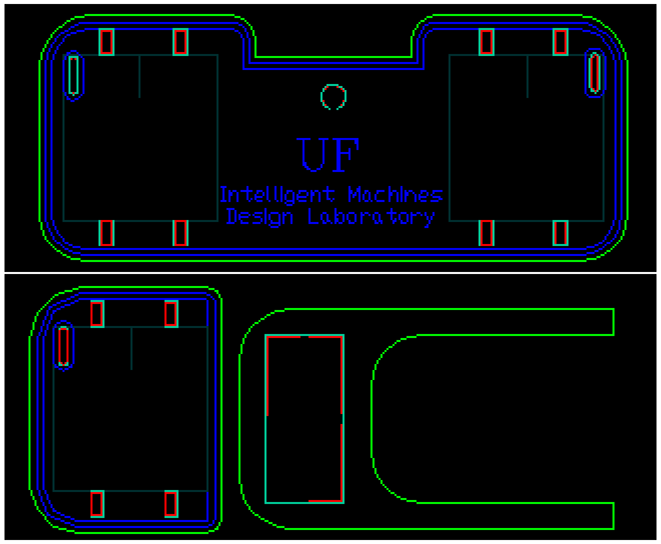


Figure 11

The platform is supported by a servo mounted on the center of the chassis. The axis of rotation is located about the "UF".

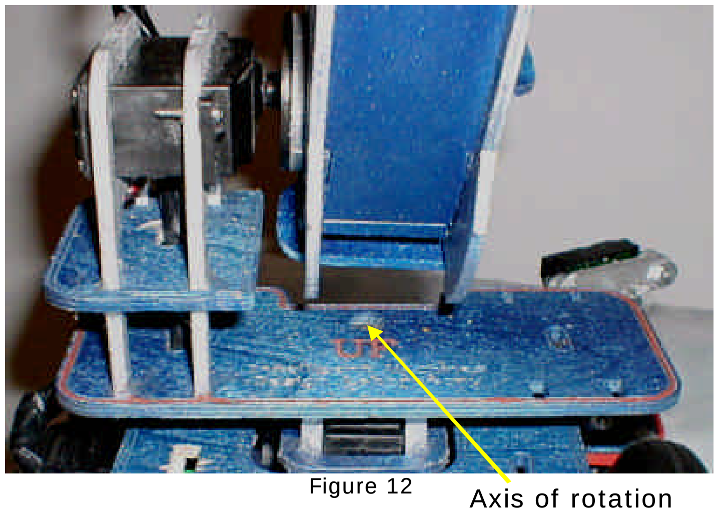


Figure 12

5.4 The uC Board Mount

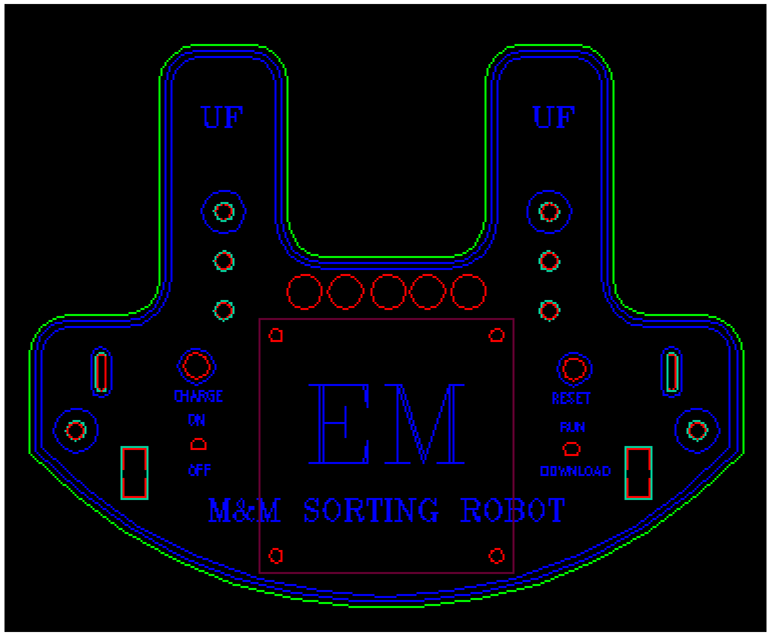


Figure 13

This unit was made by simply offsetting the bottom half of the chassis and cutting out an indention to make room for the platform servo. The holes, switches, and charging unit were "copy and pasted" from the TALRIK AutoCAD drawing. This unit sits approximately 2" off from the chassis allowing space for the uC board and the cables. As you can see in the next picture the uC board is housed underneath. Where as the reset button, siwtches and charging circuit are accessed from the front.

Underneath

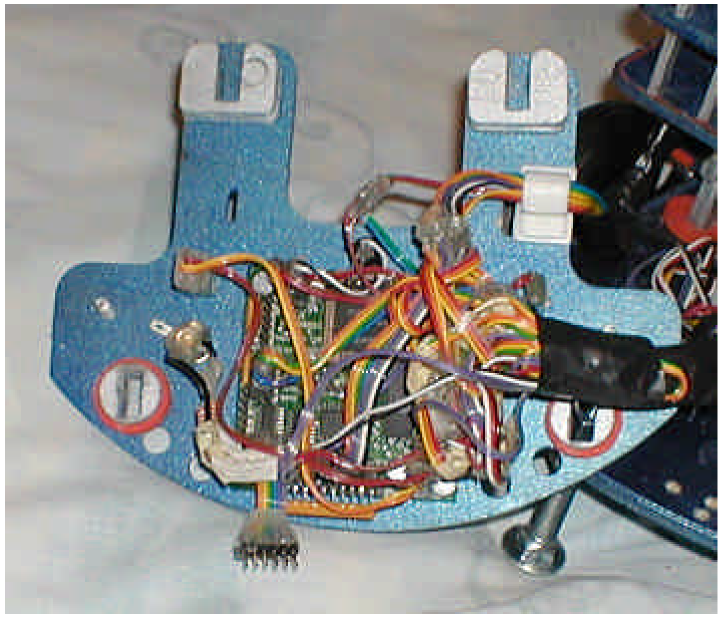


Figure 14

The Front

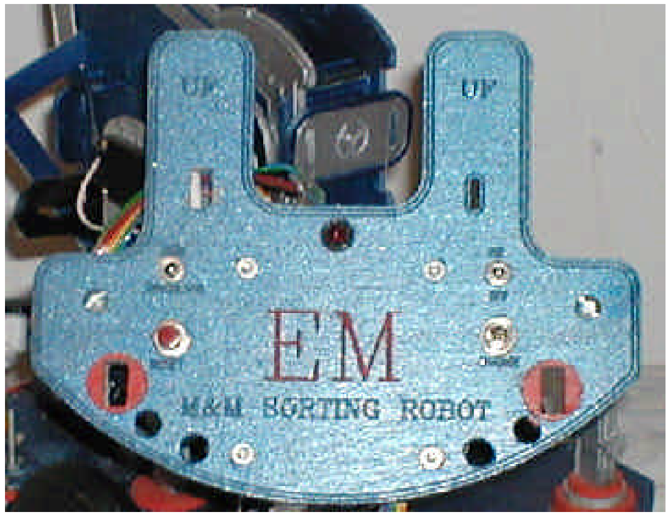


Figure 15

5.5 Sorting Unit

Outside

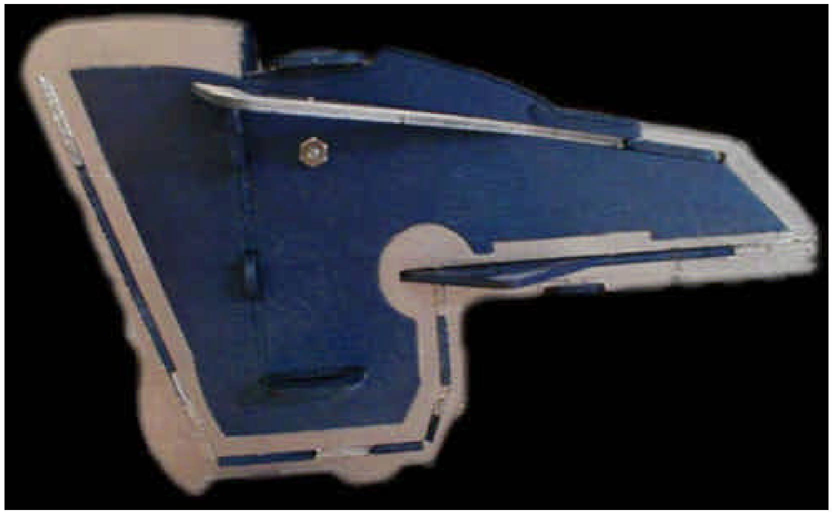


Figure 16

Inside

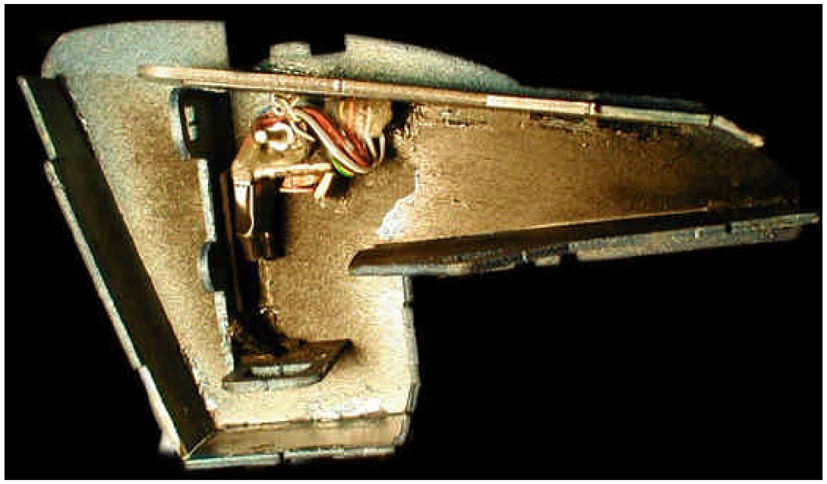


Figure 17

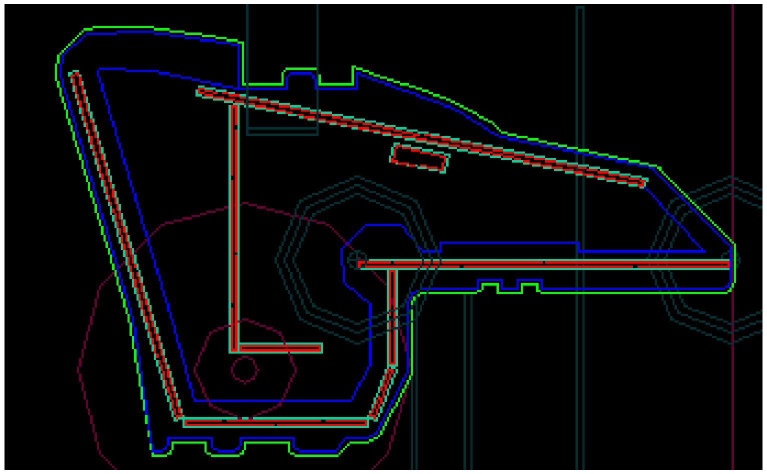


Figure 18

The sorting arm is the heart and the most dynamic part of robot. My intention was to have 3 seperate units. One to pick up the M&M™'s, one to detect the M&M™ color, and one to place the M&M™ into the correct bin. Two months and six revisions later, I have come up with this simple, yet effective design that does the work of all three.

The main design consraints were:

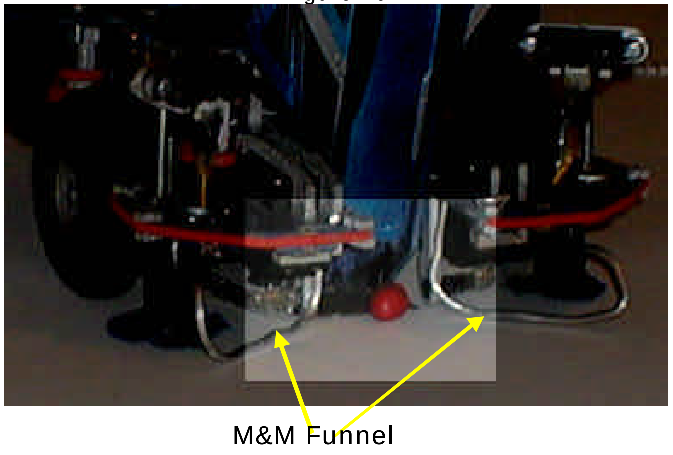
- Height - The unit could not cause the robot to be more than 8" high since my bag is only 9" high.
- Free Uninhibited Rotation - As the sorting arm rotates from one position to another it can not bump into any other parts of my robot. The most prodominant being the platform on which it sat.
- M&M's must not ever get stuck inside - By designing the sorting arm based on 1" diamter circles I have ensured that no M&M™'s will ever be stuck while they traverse the sorting unit.

6.0 UNIQUE BEHAVIORS

6.1 Picking up the M&M's

6.1.2 The Funnel

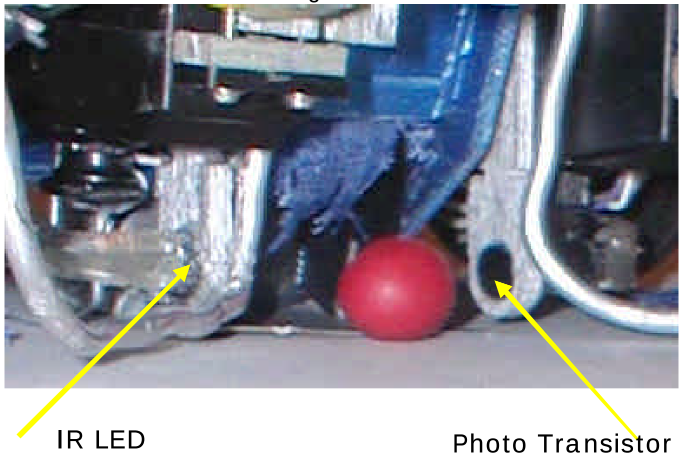
Figure 19



As EM moves forwards, M&M™ candies are funneled in towards the break beam network.

6.1.3 The Break Beam

Figure 20



Break Beam Schematic

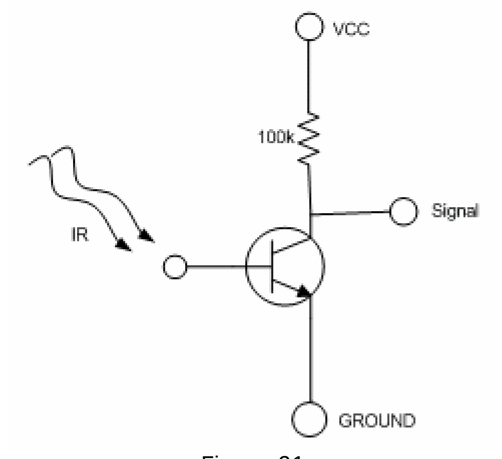


Figure 21

While IR is present on the base of the transistor, the collector has a potential of approximately 3.5 V. When no IR is present the voltage drops to approximately 0.2V . This characteristic is perfect for tying the signal right to an input bit rather than wasting an A/D port. Once the break beam is broken the arm lifts up quickly sending the M&M™ down the shaft of the sorting unit.

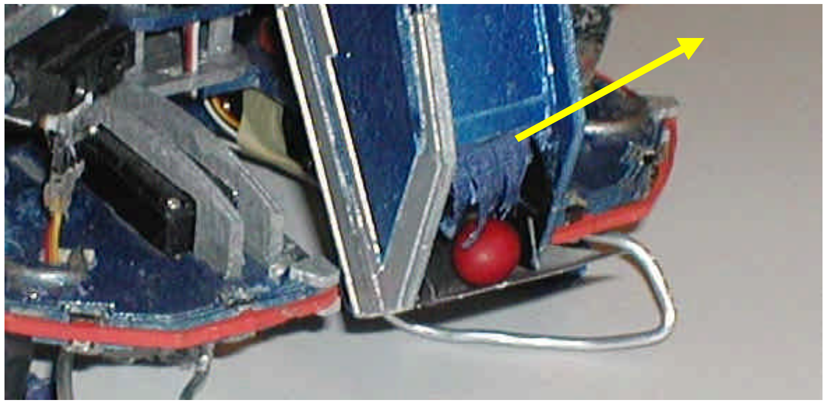


Figure 22

6.2 COLOR DETECTION

- 1 - Ultra Bright Blue LED
- 1 - Ultra Bright Green LED
- 1 - Ultra Bright Yellow LED
- 1 - Ultra Bright White LED
- 1 - CDS Cell
- 1 - 47k Ohm Resistor
- Electrical Tape, or Heat Shrink Tubing

Cost:
$9 for Ultra Bright LEDs
Resistors and CDS cell are free in lab

The idea behind color detection is simple. Different colors reflect different amounts of light. A blue M&M™ for example, when exposed to green light shines very brightly, however in the presence of red light appears to be black. Theoretically if one could just measure the amount of light reflecting off the surface of an object being exposed to different colors of light, one should be able to determine its color. That's exactly what I did.

There are two main pitfalls which hinder the accuracy of any color detection scheme, they are:

1.) The Presence of Ambient light
2.) Inconsistent Positioning of the objects.

These two problems are the by far the biggest obstacles one must overcome when attempting color detection. The actual color detection process is very straightforward and easy to reproduce once the previous 2 conditions are met. In fact, I personally think it is a good practice to sit back and spend some time working out these type of problems before jumping in and getting your hands dirty. It is good practice to ask yourself: *"How can somebody screw up my robot's behavior?"*. Solving these type of problems early in the development phase will save you a lot of time and headache later on.

6.2.1 Blocking out Ambient Light

Why do I need to block out the ambient light, and how can I do it?

If you plan on demonstrating your robot in more than one room (and I'm sure you are) then you must account for the various lighting conditions. Moreover, the brighter the ambient light, the less accurate your color readings will be. Think of a glass of fruit juice as your brightly colored M&M™. Start pouring water into it and the vibrant red of the juice begins to dilute losing its color.

The way I overcame this obstacle was by placing the M&M™ inside a controlled pitch black environment. The sorting unit I built is enclosed on all sides (Refer to Figures 16 & 17) and the inside is spray painted black to stop all reflections of light from the entrance and exit.

This works very well for small objects but what if you wanted to do color detection on large ones. My suggestion is to carry out color detection beneath your robot, using the chassis of the robot as a shield to ambient light. And then on top of that, design the color sensor so that it will be positioned completely flush with any object that you wish to carry out color detection.

6.2.2 Positioning

It is important that the object is consistently positioned in front of the color sensor. Spherical objects, such as M&M™'s, are easiest since they are relatively the same in all positions. Because of this I did not have to worry too much about the orientation of the m&m's as they fall in front of the color sensor. However, If you plan on applying color detection on something with an awkward shape you must devise a method to not only place the object in front of the sensor in a consistent manner but orientate the object so the same side is facing the sensor as well.

6.2.3 Color Sensing

The Color Sensor

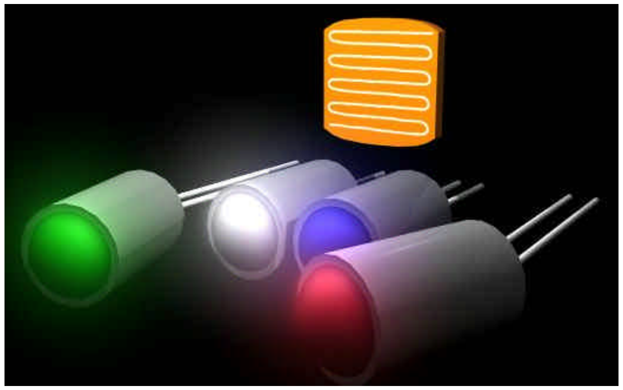
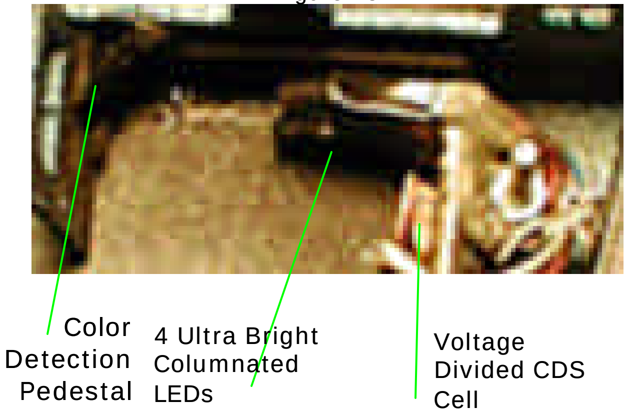


Figure 23

Figure 24

Figure 25



Once an M&M™ is securely positioned in front of the color sensor, the method I use for color detection is as follows:

1.) Turn on the green LED turn OFF all others - RECORD
2.) Turn on the white LED turn OFF all others - RECORD
3.) Turn on the blue LED turn OFF all others - RECORD
4.) Turn on the red LED turn OFF all others - RECORD
5.) Pass the 4 recorded values into a color detection function. The function runs my color detection algorithm and returns the correct M&M™ color.

The following is a schematic of my color sensor. (See figures 23-25)

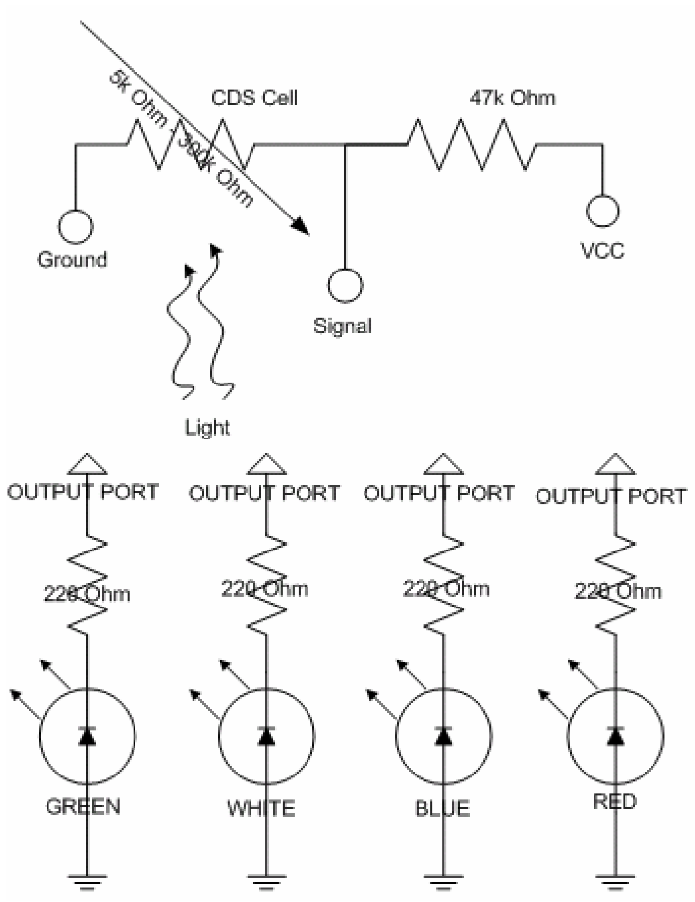


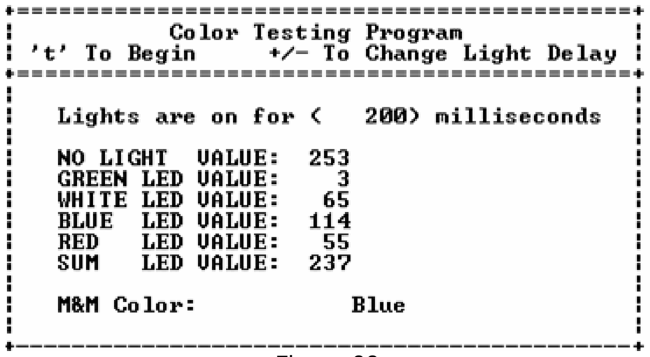
Figure 26

6.2.4 Data Collection

Figure 27

	Green	White	Blue	Red	G+W+B+R	G+W+B-R	Red	G+W+B+R	G+B-R	W+R-B-G	G+W+B	G+W+B+R
Yellow	55	175	117	193	540		193		-21	196	347	
Yellow	72	177	124	192	565		192		4	173	373	
Yellow	83	182	123	191	579		191		15	167	388	
Yellow	70	178	120	189	557		189		1	177	368	
YELLOW	70	178	121	191	560.25		191.25		-0.25	178.25	369	
	Green	White	Blue	Red	G+W+B+R	G+W+B-R	Red	G+W+B+R	G+B-R	W+R-B-G	G+W+B	G+W+B+R
Red	2	85	89	164	340		164		-73		176	340
Red	2	72	75	160	309		160		-83		149	309
Red	4	83	85	166	338		166		-77		172	338
Red	3	80	84	160	327		160		-73		167	327
RED	2.75	80	83.3	163	328.5		162.5		-76.5		166	328.5
	Green	White	Blue	Red	G+W+B+R	G+W+B-R	0	G+W+B+R	G+B-R	W+R-B-G	G+W+B	G+W+B+R
Blue	8	88	125	68	289	153	68	289				
Blue	16	95	120	56	287	175	56	287				
Blue	15	88	127	54	284	176	54	284				
Blue	8	80	115	46	249	157	46	249				
BLUE	11.75	87.75	122	56	277.25	165.25	56	277.25				
	Green	White	Blue	Red	G+W+B+R	G+W+B-R	Red	G+W+B+R	G+B-R		G+W+B	G+W+B+R
Green	81	157	139	104	481		104	481	116	41		
Green	62	136	123	84	405		84	405	101	35		
Green	84	157	134	101	476		101	476	117	40		
Green	85	153	127	106	471		106	471	106	47		
GREEN	78	150.8	131	99	458.25		98.75	458.25	110	40.75		
	Green	White	Blue	Red	G+W+B+R	G+W+B-R	Red	G+W+B+R	G+B-R	W+R-B-G	G+W+B	G+W+B+R
Brown	1	42	64	71	178	36						
Brown	6	72	96	76	250	98						
Brown	6	60	75	80	221	61						
Brown	6	60	79	72	217	73						
BROWN	4.75	58.5	78.5	75	216.5	67						
	Green	White	Blue	Red	G+W+B+R	G+W+B-R	Red	G+W+B+R	G+B-R	W+R-B-G	G+W+B	G+W+B+R
Orange	5	124	88	187	404		187		-94		217	404
Orange	12	149	108	191	460		191		-71		269	460
Orange	5	132	104	187	428		187		-78		241	428
Orange	8	133	106	179	426		179		-65		247	426
ORANGE	7.5	134.5	102	186	429.5		186		-77		243.5	430

The first 4 columns of the table represent the data recorded from the CDS Cell through the A/D when the corresponding LED is turned on. For example, starting from the top left, the CDS cell measures 55 units for a yellow M&M™ in a green light, 175 in white light, 117 in blue, and 193 in red. The program I wrote called "CLRTST.C" was used to quickly obtain test data and can be found in APPENDIX B . This is a sample screenshot of that program.



```
+==========================================+
|             Color Testing Program        |
| 't' To Begin     +/- To Change Light Delay |
+==========================================+
|                                          |
|   Lights are on for (   200) milliseconds |
|                                          |
|   NO LIGHT  VALUE:  253                  |
|   GREEN LED VALUE:    3                  |
|   WHITE LED VALUE:   65                  |
|   BLUE  LED VALUE:  114                  |
|   RED   LED VALUE:   55                  |
|   SUM   LED VALUE:  237                  |
|                                          |
|   M&M Color:           Blue              |
|                                          |
+------------------------------------------+
```

Figure 28

I tabulated 4 trial runs for each color M&M™ and then calculated the average which I then underlined to be able to see quickly. It is crucial at this point to have a database program such as EXCEL that will allow one to quickly do calculations on a large amount of data and display that data in a organized manner. It is also a big plus to be able to change the font colors so that you can hide values that you don't care for any longer.

6.2.5 The Color Detection Algorithm

To begin, please refer frequently to figure x in order to follow the color detection process. If you look at all the colors you will notice that brown has consistently smaller values for ALL lights. Using this first trend we can add all the light values (G + B + W + R) which is calculated and displayed in the next column. Notice that brown is 216, significantly less than the other colors. The next closest would be blue which is at 277. I take the average of these 2 values (approx 240) and create the first conditional of my algorithm. (See Figure 29). Now even if this conditional is met there is still a SLIGHT chance that the M&M™ could still be blue. So what I need is one more nested conditional that does nothing but distinguish between blue or brown. Looking back at the table we notice that the GREEN, WHITE, and BLUE readings for the brown M&M are less than the blue M&M, however the RED reading for brown is greater. Using this data we create one more column (G + W + B - R) and we notice that there is a very nice gap between the blue and brown. Using this data we now have a way to determine if the M&M™ is brown.

At this point we can assume that if any readings get pass our first conditional than the color can not be brown. And so in excel we can ignore all values of brown. With one color already gone it becomes even easier to find minimum or maximum light trends for the various M&M™'s. Let us do one more example. Now that brown is out of the equation we look back at our data and see that BLUE now has the smallest RED value by far (56)! So to keep things consistent I make a column for just red (R). We notice that the next closest M&M™ is green with a value of 98. So we take the average of these 2 values (56 + 98 / 2 = 76) and create our next conditional. Now all we need is a way to distinguish between blue and green and we can then take blue completely out of the picture. So looking back at the table we notice blue is smaller than green for all color readings. With this information we create one more (G + W + B + R) column, take the average for blue and green and create yet another nested conditional that will distinguish blue from all the rest of the M&M™'s.

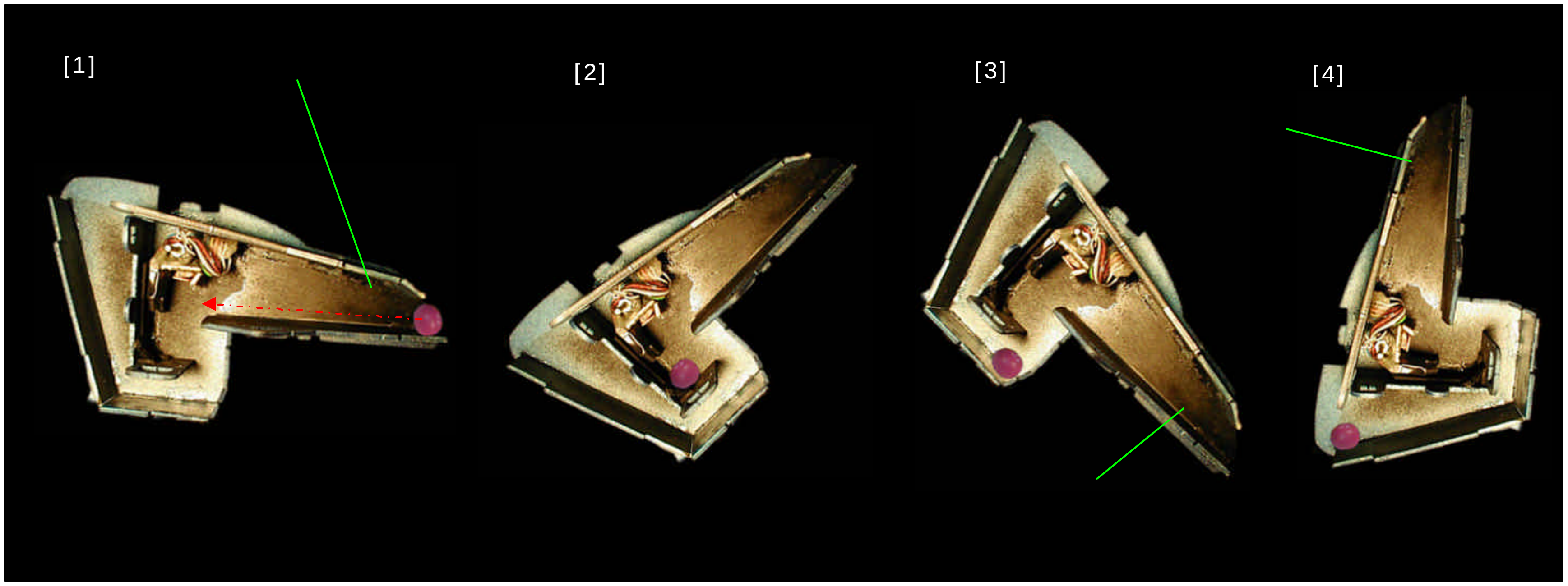
So now after 6 lines of code we have already distinguished 2 of the 6 M&M™'s and it becomes easier and easier with each M&M™ we can take out of the equation. I continue my analysis for the rest of the M&M™'s the same way and produce the color detection algorithm which you see here.

```
int detectColor(int white, int red, int blue, int green) {
  if ((white + red + blue + green) <= 240) {
  /* Brown, Perhaps Blue */
    if ((green + white + blue - red) <= 116) return 1; // brown
    else return 2; // blue
  }
  else if ((red <= 78)) {
  /* Blue, Perhaps Green */
    if ((green + white + blue + red) <= 367) return 2; // blue
    else return 3; // green
  }
  else if ((green + blue - red) >= 55) {
  /* GREEN, Perhaps Yellow */
    if ((white + red - green - blue) <= 110) return 3; // green
    else return 5; // yellow
  }
  else if ((green+white+blue) >= 308) return 5; // yellow
  else if ((green + white + blue + red) >= 372) return 6; // orange
  else return 4; // red   }
```

Figure 29

6.3 M&M Positioning

Figure 30



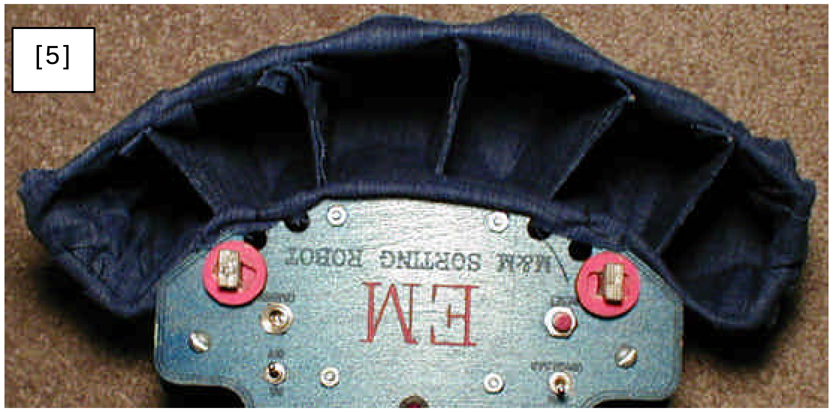


Figure 31

[1] - The M&M is lifted up and travels down the shaft of the sorting unit and onto the color detection pedestal.

[2] - This is where color detection takes place. Refer to Section 7.0 for details on the color detection sensor and how color detection is implemented.

[3] - M&M is dropped down to the bottom level of the sorting arm. At this point the arm is then positioned in front corresponding color bin.

[4] - M&M is sent down the last chute into color bins located at the back of the robot

[5] - The color bins

7.0 CONCLUSION

EM moves around on a table top avoiding the edges, avoiding obstacles, and even responding to bumps. In the meantime, EM is able to pick up M&M's and sort them by color with an accuracy of close to 100%. I can happily say that EM turned out to be a complete success. Every aspect of EM, from edge avoidance, to color detection, worked out better than I could have ever expected. Had I the opportunity to redo this project I would create a design that could be completely snap and locked together with nothing but wood, (no screws, no glue, no tape).

IMDL has turned out to be one of the greatest learning experiences of my life. This class has given me the opportunity to apply almost all of the skills taught as an electrical engineer at the University of Florida, and even acquire some new ones. After the completion of this class I have become better at C programming, better with Electronics, a MASTER at AutoCAD, unbelievably immaculate when it comes to debugging Hardware/Software issues, and to top it all off, I'm even more comfortable with public speaking. This robot is the capstone to my career as an undergraduate electrical engineering student and I am very proud at the ease at which I was able to create it.

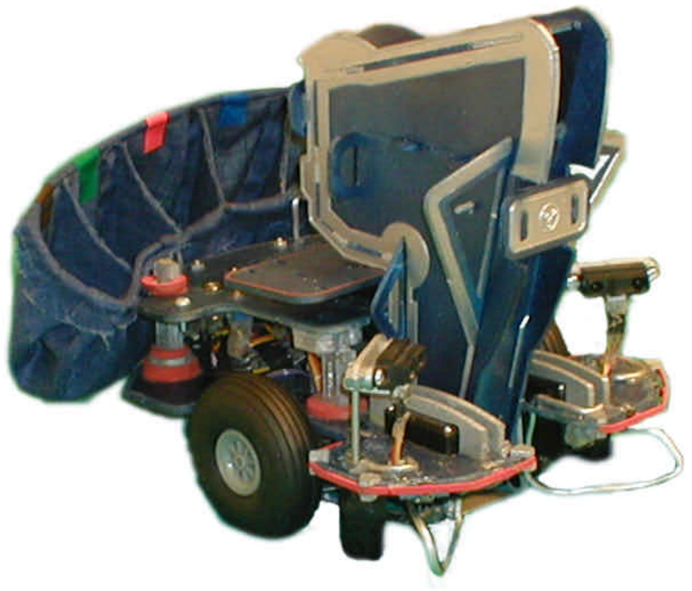


Figure 33

APPENDIX A: COMPLETE MANUAL CONTROL OF ROBOT THROUGH SCI

```
/*-----------------------------------------------------------------------
 Module:        C:\icc11\em\MANCTRL.C
 Author:        Vinh Trinh
 Project:
 State:
 Creation Date: 3/02/02
 Description:   Allows manual Control of robot movement and
                servos.
-----------------------------------------------------------------------*/
/*-----------------------INCLUDES--------------------------------------*/
#include <tjpbase.h>
#include <stdio.h>
#include <math.h>
#include <hc11.h>
#include <mil.h>
#define LEFT_MOTOR 0
#define RIGHT_MOTOR 1
#define sampleRate    4000
#define maxspeed    99
#define minspeed    1
#define maxaccel    19
#define minaccel    1
#define maxchange   99  // Maximum amount the servo can change by
#define minchange   1   // Minimum amount the servo can change by
#define servo_delta   1   // Amount to increment change
/*-----------------------END OF INCLUDES-------------------------------*/
#pragma interrupt_handler sci_hand;
void init_sci(void);
void sci_hand();

/* GLOBALS */
char scibuffer;
int  sciflag;
/* Servo Dead Areas*/



/*---------------------MAIN--------------------------------------------*/

void main(void){

char clear[]= "\x1b\x5B\x32\x4A\x04";   /*clear screen*/
char place[]= "\x1b[1;1H";        /*position at (1,1)*/

/* Motor Commands */
char  f =       'e';
char  b =       'd';
char  l =       's';
char  r =       'f';
char  s =       'x';
char  speed_u =     't';
char  speed_d =     'g';
char  accel_u =     'q';
char  accel_d =   'a';
char  lm_off =    'w';
char  rm_off =    'r';

/* Servo Commands */
char  arm_u =     'y';
char  arm_d =     'h';
char  push_u =    'i';
char  push_d =    'k';
char  plat_u =    'p';
char  plat_d =    ';';
char  arm_change_u =  'u';
char  arm_change_d =  'j';
char  push_change_u = 'o';
char  push_change_d = 'l';
char  plat_change_u = '[';
char  plat_change_d = ']';
char  change_mode = 'z';
char  inc_delay =   'v';
char  dec_delay =   'b';
char  colortest =   'c';

/* LED TOGGLE */
char LED0 = '0';
char LED1 = '1';
char LED2 = '2';
char LED3 = '3';
char LED4 = '4';
char LED5 = '5';
char LED6 = '6';
char LED7 = '7';
char *LEDOFF[3] = {"+"};
char *LEDON[3] = {"-"};

char action, oldaction;
char *printaction[20] = {"Idle"};
char *printmode[20] = {"Increment"};

/* Initial Motor Values */
int   speed = 20,
    accel = 1,
    lm_status = 0,
    rm_status = 0;


/* Initial Servo Values */
int   armpw = arm_init_pw,
    platpw = plat_init_pw,
    pushpw = push_init_pw,
    arm_change = 20,
    plat_change = 10,
    push_change = 20,
    armPosition = 1 ;

/* Initial LED Values */
int l0Status = 0,
  l1Status = 0,
  l2Status = 0,
  l3Status = 0,
  l4Status = 0,
  l5Status = 0,
  l6Status = 0,
  l7Status = 0,
  LEDMASK = 0x00;

int servomode = 0; /* 0 = Incremental 1 = Continuous */
int servodelay = 5;
int colorDelay = 0;
int noLight;

  init_analog();
  init_motortjp();
  init_clocktjp();
  init_servotjp();
  init_serial();
  init_sci();

  noLight = COLOR_CDS;
  printf("%s", clear);
  printf("%s", place);
  printf("+-------------------------------------------------------------+\n");
printf("|============================================================|\n");
printf("|                Manual Robot Control Program                |\n");
printf("|============================================================|\n");
printf("+------------------------------+-----------------------------+\n");
printf("|       MOTOR STATUS           |       MOTOR COMMANDS        |\n");
printf("+------------------------------+-----------------------------+\n");
  printf("|                              |                              |\n");
printf("|                              | Forwards..................e   |\n");
  printf("| Action:                      | Backwards.................d   |\n");
  printf("| Left Motor:                  | Left......................s   |\n");
  printf("| Right Motor:                 | Right.....................f   |\n");
  printf("| Motor Speed:                 | Stop......................x   |\n");
  printf("| Motor Acceleration:          | Speed Up..................t   |\n");
  printf("|                              | Speed Down................g   |\n");
  printf("|                              | Acceleration Up...........q   |\n");
  printf("|                              | Acceleration Down.........a   |\n");
  printf("|                              | Turn Off Left Motor.......w   |\n");
  printf("|                              | Turn Off Right Motor......r   |\n");
  printf("|                              |                              |\n");
  printf("+------------------------------+------------------------------+\n");
  printf("|      SERVO STATUS            |       SERVO COMMANDS         |\n");
  printf("+------------------------------+------------------------------+\n");
  printf("|                              |                              |\n");
  printf("| Sorting Arm PW:              | Sorting Arm Up............y   |\n");
  printf("| Delta:                       | Sorting Arm Down..........h   |\n");
  printf("|                              | +/- Arm Movement ........u/j  |\n");
  printf("| Platform PW:                 |                              |\n");
  printf("| Delta:                       | Platform Up...............p   |\n");
  printf("|                              | Platform Down.............;   |\n");
  printf("| Push Arms PW:                | +/- Platform Movement ...[/]  |\n");
  printf("| Delta:                       |                              |\n");
  printf("|                              | Push Arms Up..............i   |\n");
  printf("| MODE:                        | Push Arms Down............k   |\n");
  printf("|------------------------------| +/- Push Arm Movement....o/l  |\n");
  printf("| Current SCI BUFFER:          |                              |\n");
  printf("| Servo Delay:                 | Mode Change...............z   |\n");
  printf("|                              | +/- Servo Delay..........v/b  |\n");
  printf("+------------------------------+------------------------------+\n");
  printf("|         LED DRIVER TOGGLE  - Use # Pad to toggle LED's      |\n");
  printf("| 0:            1:             2:             3:              |\n");
  printf("| 4:            5:             6:             7:              |\n");
  printf("+-------------------------------+-----------------------------+\n");
  printf("| Color Sensor Reading:         | Color Arm Positions......c   |\n");
  printf("+-------------------------------+-----------------------------+\n");

stop();

printf("\x1b[11;21H%8d", lm_status);
printf("\x1b[12;21H%8d", rm_status);
printf("\x1b[13;21H%8d", speed);
printf("\x1b[14;21H%8d", accel);
printf("\x1b[25;21H%8d", armpw);
printf("\x1b[26;21H%8d", arm_change);
printf("\x1b[28;21H%8d", platpw);
printf("\x1b[29;21H%8d", plat_change);
printf("\x1b[31;21H%8d", pushpw);
printf("\x1b[32;21H%8d", push_change);
printf("\x1b[37;21H%8d", servodelay);
printf("\x1b[34;18H%11s", *printmode);
printf("\x1b[41;6H%3s", *LEDOFF);
printf("\x1b[41;20H%3s", *LEDOFF);
printf("\x1b[41;36H%3s", *LEDOFF);
printf("\x1b[41;52H%3s", *LEDOFF);
printf("\x1b[42;6H%3s", *LEDOFF);
printf("\x1b[42;20H%3s", *LEDOFF);
printf("\x1b[42;36H%3s", *LEDOFF);
printf("\x1b[42;52H%3s", *LEDOFF);
```

```
  /* Put Servos Into Initial Positions */

  servo(plat,platpw);
  wait(500);
  servo(arm,armpw);
  servo(push,pushpw);
  wait(1000);
  servo(arm,0);
  servo(plat,0);
  servo(push,0);

while(1){
  if (sciflag == 1) {
    sciflag = 0;
    action = scibuffer;
    if (action == f || action == b || action == l || action == r){
      servo(arm,0);
      servo(plat,0);
      servo(push,0);
    }

    if     (action == f) {
      forwards(speed);
      *printaction = "Forwards";
      lm_status = 1;
      rm_status = 1;
    }
    else if(action == b) {
      backwards(speed);
      *printaction = "Backwards";
      lm_status = 1;
      rm_status = 1;
    }
    else if(action == l) {
      left(speed);
      *printaction = "Left";
      lm_status = 1;
      rm_status = 1;
    }
    else if(action == r) {
      right(speed);
      *printaction = "Right";
      lm_status = 1;
      rm_status = 1;
    }
    else if(action == speed_u && speed <= maxspeed) {
      speed = speed + accel;
      printf("\x1b[13;21H%8d", speed);
    }
    else if(action == speed_d && speed >= minspeed) {
      speed = speed - accel;
      printf("\x1b[13;21H%8d", speed);
    }
    else if(action == accel_u && accel <= maxaccel) {
      accel++;
      printf("\x1b[14;21H%8d", accel);
    }
    else if(action == accel_d && accel >= minaccel) {
      accel--;
      printf("\x1b[14;21H%8d", accel);
    }
    else if(action == lm_off) {
      lmoff();
      lm_status = 0;
      printf("\x1b[11;21H%8d", lm_status);
    }
    else if(action == rm_off) {
      rmoff();
    rm_status = 0;
    printf("\x1b[11;21H%8d", rm_status);
  }
  else if(action == s) {
    stop();
    *printaction = "IDLE";
    lm_status = 0 ;
    rm_status = 0 ;
  }
  else if(action == arm_change_u && arm_change < maxchange) {
    arm_change = arm_change + servo_delta ;
    printf("\x1b[26;21H%8d", arm_change) ;
  }
  else if(action == arm_change_d && arm_change > minchange) {
    arm_change = arm_change - servo_delta ;
    printf("\x1b[26;21H%8d", arm_change) ;
  }
  else if(action == plat_change_u && plat_change < maxchange) {
    plat_change = plat_change + servo_delta ;
    printf("\x1b[29;21H%8d", plat_change) ;
  }
  else if(action == plat_change_d && plat_change > minchange) {
    plat_change = plat_change - servo_delta ;
    printf("\x1b[29;21H%8d", plat_change) ;
  }
  else if(action == push_change_u && push_change < maxchange) {
    push_change = push_change + servo_delta ;
    printf("\x1b[32;21H%8d", push_change) ;
  }
  else if(action == push_change_d && push_change > minchange) {
    push_change = push_change - servo_delta;
    printf("\x1b[32;21H%8d", push_change) ;
  }

  else if(action == change_mode) {
    servomode = servomode ^ 1;
    if (servomode == 1) *printmode = "Continuous";
    else       *printmode = "Increment";
    printf("\x1b[34;18H%11s", *printmode);
  }
  else if(action == inc_delay || action == dec_delay) {
    if (action == inc_delay) servodelay++;
    else           servodelay--;
    printf("\x1b[37;21H%8d", servodelay);
  }
  else if(action == colortest) {
    switch(armPosition) {
      case 1:
        for (armpw = arm_init_pw; armpw <= sortPos1; armpw = armpw + arm_change){
          wait(servodelay);
          servo(arm, armpw);
        }
        noLight = COLOR_CDS;
        armPosition = 2;
        break;
      case 2:
        for (armpw; armpw >= sortPos2; armpw = armpw - arm_change){
          wait(servodelay);
          servo(arm, armpw);
        }
        armPosition = 3;
        break;
      case 3:
        for (armpw; armpw <= sortPos3; armpw = armpw + arm_change){
          wait(servodelay);
          servo(arm, armpw);
        }
        armPosition = 4;
        break;
      case 4:
        for (armpw; armpw >= arm_init_pw; armpw = armpw - arm_change){
          servo(arm, armpw);
        }
        armPosition = 1;
        break;
      default:
        servo(arm, arm_init_pw);
        armPosition = 1;
        break;

    }
  }
  else if(action == LED0 || action == LED1 || action == LED2 || action == LED3
      || action == LED4 || action == LED5 || action == LED6 || action == LED7) {
      if (action == LED0) {
        LEDMASK = LEDMASK ^ 0x01;
        l0Status = l0Status ^ 1;
        (l0Status == 1) ? printf("\x1b[41;6H%3s", *LEDON)
          :printf("\x1b[41;6H%3s", *LEDOFF);
      }
      else if (action == LED1) {
        LEDMASK = LEDMASK ^ 0x02;
        l1Status = l1Status ^ 1;
        (l1Status == 1) ? printf("\x1b[41;20H%3s", *LEDON)
          :printf("\x1b[41;20H%3s", *LEDOFF);
      }
      else if (action == LED2) {
        LEDMASK = LEDMASK ^ 0x04;
        l2Status = l2Status ^ 1;
        (l2Status == 1) ? printf("\x1b[41;36H%3s", *LEDON)
          :printf("\x1b[41;36H%3s", *LEDOFF);
      }
      else if (action == LED3) {
        LEDMASK = LEDMASK ^ 0x08;
        l3Status = l3Status ^ 1;
        (l3Status == 1) ? printf("\x1b[41;52H%3s", *LEDON)
          :printf("\x1b[41;52H%3s", *LEDOFF);
      }
      else if (action == LED4) {
        LEDMASK = LEDMASK ^ 0x10;
        l4Status = l4Status ^ 1;
        (l4Status == 1) ? printf("\x1b[42;6H%3s", *LEDON)
          :printf("\x1b[42;6H%3s", *LEDOFF);
      }
      else if (action == LED5) {
        LEDMASK = LEDMASK ^ 0x20;
        l5Status = l5Status ^ 1;
        (l5Status == 1) ? printf("\x1b[42;20H%3s", *LEDON)
          :printf("\x1b[42;20H%3s", *LEDOFF);
      }
      else if (action == LED6) {
        LEDMASK = LEDMASK ^ 0x40;
        l6Status = l6Status ^ 1;
        (l6Status == 1) ? printf("\x1b[42;36H%3s", *LEDON)
          :printf("\x1b[42;36H%3s", *LEDOFF);
      }
      else if (action == LED7) {
        LEDMASK = LEDMASK ^ 0x80;
        l7Status = l7Status ^ 1;
        (l7Status == 1) ? printf("\x1b[42;52H%3s", *LEDON)
          :printf("\x1b[42;52H%3s", *LEDOFF);
      }

      *(unsigned char *)(0x7000) = LEDMASK;
  }
  else {
```

```
stop();
if    (action == arm_u && armpw <= MAX_ARMPW) {
  if (servomode == 1) {
    while (armpw <= MAX_ARMPW && scibuffer != s) {
      if (armpw > DEAD_MIN && armpw < DEAD_MAX)
        armpw = DEAD_MAX;
      else {
        armpw = armpw + arm_change;
        servo(arm,armpw);
        wait(servodelay);
      }
    }
  }
  else {
    if (armpw > DEAD_MIN && armpw < DEAD_MAX)
        armpw = DEAD_MAX;
    else {
      armpw = armpw + arm_change;
      servo(arm,armpw);
    }
  }
  printf("\x1b[25;21H%8d", armpw);
}
else if (action == arm_d && armpw >= MIN_ARMPW) {
  if (servomode == 1) {
    while (armpw >= MIN_ARMPW && scibuffer != s) {
      if (armpw > DEAD_MIN && armpw < DEAD_MAX)
        armpw = DEAD_MIN;
      else {
        armpw = armpw - arm_change;
        servo(arm,armpw);
        wait(servodelay);
      }
    }
  }
  else {
    if (armpw > DEAD_MIN && armpw < DEAD_MAX)
      armpw = DEAD_MIN;
    else {
      armpw = armpw - arm_change;
      servo(arm,armpw);
    }
  }
  printf("\x1b[25;21H%8d", armpw);
}
else if (action == plat_u && platpw <= MAX_PLATPW) {
  if (servomode == 1) {
    while (platpw <= MAX_PLATPW && scibuffer != s) {
      if (platpw > DEAD_MIN && platpw < DEAD_MAX)
        platpw = DEAD_MAX;
      else {
        platpw = platpw + plat_change;
        servo(plat,platpw);
        wait(servodelay);
      }
    }
  }
  else {
    if (platpw > DEAD_MIN && platpw < DEAD_MAX)
      platpw = DEAD_MAX;
    else {
      platpw = platpw + plat_change;
      servo(plat,platpw);
    }
  }
  printf("\x1b[28;21H%8d", platpw);
}
else if (action == plat_d && platpw >= MIN_PLATPW) {
```

```
      if (servomode == 1) {
        while (platpw >= MIN_PLATPW && scibuffer != s) {
          if (platpw > DEAD_MIN && platpw < DEAD_MAX)
            platpw = DEAD_MIN;
          else {
            platpw = platpw - plat_change;
            servo(plat,platpw);
            wait(servodelay);
          }
        }
      }
      else {
        if (platpw > DEAD_MIN && platpw < DEAD_MAX)
          platpw = DEAD_MIN;
        else {
          platpw = platpw - plat_change;
          servo(plat,platpw);
        }
      }
      printf("\x1b[28;21H%8d", platpw);
    }
    else if (action == push_u && pushpw <= MAX_PUSHPW) {
      if (servomode == 1) {
        while (pushpw <= MAX_PUSHPW && scibuffer != s) {
          if (pushpw > DEAD_MIN && pushpw < DEAD_MAX)
            pushpw = DEAD_MAX;
          else {
            pushpw = pushpw + push_change;
            servo(push,pushpw);
            wait(servodelay);
          }
        }
      }
      else {
        if (pushpw > DEAD_MIN && pushpw < DEAD_MAX)
          pushpw = DEAD_MAX;
        else {
          pushpw = pushpw + push_change;
          servo(push,pushpw);
        }
      }
      printf("\x1b[31;21H%8d", pushpw);
    }
    else if (action == push_d && pushpw >= MIN_PUSHPW) {
      if (servomode == 1) {
        while (pushpw >= MIN_PUSHPW && scibuffer != s) {
          if (pushpw > DEAD_MIN && pushpw < DEAD_MAX)
            pushpw = DEAD_MIN;
          else {
            pushpw = pushpw - push_change;
            servo(push,pushpw);
            wait(servodelay);
          }
        }
      }
      else {
        if (pushpw > DEAD_MIN && pushpw < DEAD_MAX)
          pushpw = DEAD_MIN;
        else {
          pushpw = pushpw - push_change;
          servo(push,pushpw);
        }
      }
      printf("\x1b[31;21H%8d", pushpw);
    }
  }
  printf("\x1b[10;18H%11s", *printaction);
  printf("\x1b[36;21H       %c", action);
```

```
    oldaction = action;

  } /* end if */
  if (colorDelay == sampleRate) {
    printf("\x1b[44;26H%3d",(noLight - COLOR_CDS));
    colorDelay = 0;
  }
  colorDelay ++;
}/* End While */


} /* end main */

void sci_hand(void){
  scibuffer = SCDR;
  sciflag = 1;
  CLEAR_FLAG(SCSR,0x20);
}

void init_sci(void) {
  INTR_OFF();
  *((void (**)())0xffd6) = sci_hand;
  CLEAR_FLAG(SCSR, 0x20);
  SET_BIT(SCCR2, 0x20);
  INTR_ON();
}
```

APPENDIX B: COLOR DATA RETRIEVAL PROGRAM

```
#include <tjpbase.h>
#include <stdio.h>
#include <hc11.h>
#include <mil.h>

#define servoffset 50 // Servo constants for some reason arne't the same! use this to
tweak them a little.
#define arm_change 20
#define servodelay 5

#define sortPos1  3600
#define sortPos1_2  2000
#define sortPos2  2210
#define sortPos3  4460
#define sortPos4  800


/* SCI STUFF */
#pragma interrupt_handler sci_hand;
void init_sci(void);
void sci_hand();
char scibuffer;
int  sciflag;

/* Globals */
int   white,
    red,
    blue,
    green;



void main(void) {


  char clear[]= "\x1b\x5B\x32\x4A\x04";   /*clear screen*/
  char place[]= "\x1b[1;1H";        /*position at (1,1)*/
  char g[]= "Green";
  char r[]= "Red";
  char b[] = "Blue";
  char y[] = "Yellow";
  char o[] = "Orange";
  char br[] = "Brown";
  char e[] = "Error";
  char w[] = "White";
  char begin = 't';
  char upDelay = '+';
  char downDelay = '-';

  int armPosition,
    armpw,
    platpw,
    pushpw,
    noLight,
    lightOn,
    color;

  armpw = arm_init_pw;
  platpw = plat_init_pw;
  pushpw = push_init_pw;

  /* Terminal Screen*/
  printf("%s",clear);
  printf("%s",place);
  printf("+==========================================+\n");
  printf("|           Color Testing Program          |\n");
printf("| 't' To Begin     +/- To Change Light Delay |\n");
printf("+==========================================+\n");
printf("|                                          |\n");
printf("|   Lights are on for (      ) milliseconds  |\n");
printf("|                                          |\n");
printf("|   NO LIGHT  VALUE:                       |\n");
printf("|   GREEN LED VALUE:                       |\n");
printf("|   WHITE LED VALUE:                       |\n");
printf("|   BLUE  LED VALUE:                       |\n");
printf("|   RED   LED VALUE:                       |\n");
printf("|   SUM   LED VALUE:                       |\n");
printf("|                                          |\n");
printf("|   M&M Color:                             |\n");
printf("|                                          |\n");
printf("+------------------------------------------+\n");


/* Turn On all Systems*/
init_analog();
init_motortjp();
init_clocktjp();
init_servotjp();
init_serial();
init_sci();

/* Initialize Variables*/
sciflag = 0;
armPosition = 1;
noLight = COLOR_CDS;
lightOn = 200;
wait(100);
printf("\x1b[6;24H%6d",lightOn);

/* Initialize Servos*/
/*servo(plat,platpw);
wait(500);
servo(arm,armpw);
servo(push,pushpw);
wait(1000);*/
servo(arm,0);
servo(plat,0);
servo(push,0);

while(1){
  if(sciflag == 1) {
    sciflag = 0;
    if(scibuffer == begin) {
      switch(armPosition) {
        case 1:
          for (armpw; armpw <= sortPos1_2; armpw = armpw + arm_change){
            wait(servodelay);
            servo(arm, armpw);
          }
          for (; armpw <= sortPos1; armpw = armpw + arm_change){
            wait(servodelay*3);
            servo(arm, armpw);
          }
          wait(1000);
          noLight = 0;
          LEDS_OFF;
          //servo(arm,0);
          while (noLight < 230) {
          noLight = COLOR_CDS;
          printf("\x1b[8;22H%4d    ");
          printf("\x1b[8;22H%4d",noLight);
          wait(200);
          }
```

```
GREEN_ON;
wait(lightOn);
green = (noLight - COLOR_CDS);
printf("\x1b[9;22H%4d    ");
printf("\x1b[9;22H%4d",green);
LEDS_OFF;

WHITE_ON;
wait(lightOn);
white = (noLight - COLOR_CDS);
printf("\x1b[10;22H    ");
printf("\x1b[10;22H%4d",white);
LEDS_OFF;

BLUE_ON;
wait(lightOn);
blue = (noLight - COLOR_CDS);
printf("\x1b[11;22H    ");
printf("\x1b[11;22H%4d",blue);
LEDS_OFF;

RED_ON;
wait(lightOn);
red = (noLight - COLOR_CDS);
printf("\x1b[12;22H    ");
printf("\x1b[12;22H%4d",red);
LEDS_OFF;
printf("\x1b[13;22H%4d",(green+white+red+blue));

color = detectColor(white,red,blue,green);

/*
  brown:  1
  blue:   2
  green:  3
  red:    4
  yellow: 5
  orange: 6
  error:  0
*/
switch(color) {
  case 1:
    printf("\x1b[15;22H%8s",br);
    break;
  case 2:
    printf("\x1b[15;22H%8s",b);
    break;
  case 3:
    printf("\x1b[15;22H%8s",g);
    break;
  case 4:
    printf("\x1b[15;22H%8s",r);
    break;
  case 5:
    printf("\x1b[15;22H%8s",y);
    break;
  case 6:
    printf("\x1b[15;22H%8s",o);
    break;
  case 7:
    printf("\x1b[15;22H%8s",w);
    break;
  default:
    printf("\x1b[15;22H%8s",e);
    break;
} /* END SWITCH */
```

```
            armPosition = 2;
            break;
          case 2:
            for (armpw; armpw >= sortPos2; armpw = armpw - arm_change){
              wait(servodelay);
              servo(arm, armpw);
            }
            armPosition = 3;
            break;

          case 3:
            for (armpw; armpw <= sortPos3; armpw = armpw + arm_change){
              wait(servodelay);
              servo(arm, armpw);
            }
            armPosition = 4;
            break;
          case 4:
            for (armpw; armpw >= (sortPos4); armpw = armpw - arm_change){
              wait(servodelay/2);
              servo(arm, armpw);
            }
            armPosition = 1;
            break;
          default:
            servo(arm, arm_init_pw);
            armPosition = 1;
            break;
        }/* End switch(armPosition) */
      }/* End if(scibuffer) */
      else if (scibuffer == upDelay) {
          lightOn = lightOn + 10;
          printf("\x1b[6;24H%6d",lightOn);
      }
      else if (scibuffer == downDelay) {
        lightOn = lightOn - 10;
        printf("\x1b[6;24H%6d",lightOn);
      }
    }/* End if (sciflag) */
  }/* End While(1) */
}/* End Main */

void sci_hand(void){
  scibuffer = SCDR;
  sciflag = 1;
  CLEAR_FLAG(SCSR,0x20);
}

void init_sci(void) {
  INTR_OFF();
  *((void (**)())0xffd6) = sci_hand;
  CLEAR_FLAG(SCSR, 0x20);
  SET_BIT(SCCR2, 0x20);
  INTR_ON();
}

/*
  white: 7
  brown:  1
  blue:   2
  green:  3
  red:    4
  yellow: 5
  orange: 6
  error:  0
*/
int detectColor(int white, int red, int blue, int green) {
  if ((white + red + blue + green) >= 600)              // ALL COLORS
  /* WHITE THANKS TO AAMIR */
    return 7;
  else if ((white + red + blue + green) <= 250)
  /* Brown */
    return 1;
  else if ((white + blue - red) <= 32)
  /* RED */
    return 4;
  else if ((green + blue + white - red) <= 109)
  /* ORANGE */
    return 6;
  else if ((red >= 137))
  /* YELLOW */
    return 5;
  else if ((white + green + red) >= 257)
  /* GREEN */
    return 3;
  else if ((green + white + blue - red) >= 160)
  /* BLUE */
    return 2;
  else
    return 0;

  if ((white + red + blue + green) > 600)
    return 7;
  if ((white + red + blue + green) <= 331){
/* Blue, Brown */
    if ((white + blue - red) > 100 )
      /* blue */
      return 2;
    else
      /* brown */
      return 1;
  }

/* RED, GREEN, YELLOW, ORANGE */
  if ((white) < 125)
    /* red */
    return 4;
  else {
    /* yellow, geen, orange */
    if (green < 30)
      /* Orange */
      return 6;
    /* yellow, green */
    else if (red < 125)
      /* Green */
      return 3;
    else
      /* yellow */
      return 5;
  }


}
```

APPENDIX C: MAIN PROGRAM

```
/*************************** Includes ********************************/

#include <analog.h>
#include <clocktjp.h>
#include <motortjp.h>
#include <servotjp.h>
#include <serialtp.h>
#include <isrdecl.h>
#include <vectors.h>
#include <inport.h>
#include <control.h>
#include <stdio.h>
#include <math.h>
/************************ End of Includes ****************************/

/**************************** Constants *********************************/
#define MAX_SPEED       100
#define ZERO_SPEED      0
#define MIN_ARMPW   920
#define MAX_ARMPW   4500
#define MIN_PLATPW    700
#define MAX_PLATPW    4600
#define MIN_PUSHPW    700
#define MAX_PUSHPW    4000
#define DEAD_MIN    1980
#define DEAD_MAX    2150
#define arm       2
#define plat      1
#define push      0
#define plat_init_pw  2650
#define arm_init_pw   920
#define push_init_pw  4200
#define push_in_pw    680

#define BUMPER      analog(0)
#define RIGHT_IR    analog(2)
#define LEFT_IR     analog(3)
#define RIGHT_CDS   analog(7)
#define LEFT_CDS    analog(5)
#define BREAK_BEAM  inport(0)
#define COLOR_CDS analog(4)

/* Enable OC4 interrupt and all servo operations */
#define SERVOS_ON     SET_BIT(TMSK1,0x10)

/*Disable OC4 interrupt: Stops all servo holding torques, useful for energy savings*/
#define SERVOS_OFF    CLEAR_BIT(TMSK1, 0x10)

#define CDS_THRESHOLD_L 60
#define CDS_THRESHOLD_D 8
#define IR_THRESHOLD    100
#define sampledelay     10
#define speed           45
#define backwardsdelay  1100
#define doublecheck     50
#define armdelta    20
#define armdelay    5
#define platdelta   10
#define platdelay   10
#define pushdelta   30
#define pushdelay   20
#define SORTPOSITION  MAX_PLATPW
#define MAXTURNDELAY  2500
#define candy_settle  300
#define fallout_delay   500

// These dont work so try the old way
#define irled     0x08
#define cdsled      0x03
#define greenled    0x10
#define blueled     0x40
#define redled      0x80
#define whiteled    0x20
#define allled      0xF0

// COLOR DETECTION LED CONTROL CONSTANTS
#define LEDS_ON   *(unsigned char *)(0x7000) = 0xFF
#define LEDS_OFF  *(unsigned char *)(0x7000) = 0x00
#define CDS_ON  *(unsigned char *)(0x7000) = 0x03
#define WHITE_ON *(unsigned char *)(0x7000) = 0x20
#define RED_ON *(unsigned char *)(0x7000) = 0x80
#define BLUE_ON *(unsigned char *)(0x7000) = 0x40
#define GREEN_ON *(unsigned char *)(0x7000) = 0x10

/* FACING ROBOT FROM LEFT TO RIGHT*/
#define nobin     3800
#define bin1      3460
#define bin2      3060
#define bin3      2680
#define bin4      2300
#define bin5      1950
#define bin6      1700

#define brownpos    bin1
#define bluepos     bin4
#define redpos      bin5
#define greenpos    bin2
#define orangepos   bin3
#define yellowpos   bin6
#define badpos      nobin



/* COLOR TESING DEFINES */
#define servoffset 50 // Servo constants for some reason arne't the same use this to
tweak them a little.
#define push_change 10
#define arm_change 20
#define servodelay 5



#define sortPos1  4000
#define sortPos2  3600
#define sortPos3  2100
#define sortPos4  4700
#define sortPos5  4300


#define LIGHTON   200


#define UPDATE_LEDS *(unsigned char *)(0x7000) = ledmask


#pragma interrupt_handler sci_hand;
/*******************************************************************************************
***/



/* FUNCTION PROTOTYPES */
void blink(void);
void init_systems(void);
void init_servos(void);
void init_constants(void);
void scan_sensors(void);
void action(int);
void positionCandy(void);
int  getColor(void);
void sort(void);

void program1(void);


/* GLOBALS */
int avoid = 0;
int sortmode = 0;
unsigned int armpw = arm_init_pw;
unsigned int platpw = plat_init_pw;
unsigned int pushpw = push_init_pw;

int room_is_dark;
int  sciflag;
char scibuffer;
int leftCDS_init;
int rightCDS_init;
int leftCDS_reading;
int rightCDS_reading;
int bump_reading;
int program_select;
int color;
int rand;
int ledmask = 0x00;
int armPosition = 1;
int color;
int noLight;
int white,blue,red,green; // led colors






/********************************************
************ MAIN PROGRAM *******************
********************************************/

void main(void) {

init_systems();

blink();
printf("System Reset\n");

while (BUMPER < 10) {
  wait(1);
  program_select = BUMPER;
 }


if (program_select < 30) {
  room_is_dark = 0;
  program1();
}
else {
  room_is_dark = 1;
  program1();
```

```
}


}  /* End Main */
```

```
/* -------------- Blink --------------------- */
void blink(void) {
int i;
  for(i; i<10; i++) {
  ledmask = ledmask ^ cdsled;
  UPDATE_LEDS;
  wait(50);
  }
}
```

```
/* -------------- INIT SYSTEMS ---------------*/
void init_systems (void){
  init_analog();
  init_motortjp();
  init_clocktjp();
  init_servotjp();
  init_serial();
  wait(500);
  //printf("Starting\n");
}
```

```
/* ---------------- PROGRAM 1 ------------------*/
void program1(void) {
  init_servos();
  init_constants();

  while (1){
    // TURN ON SENSOR LEDS
    ledmask = ledmask | irled;
    if(room_is_dark)  ledmask = ledmask | cdsled;
    else        ledmask = ledmask & ~(cdsled);
    UPDATE_LEDS;
    avoid = 0;
    armPosition = 1;
    forwards(speed);
    wait(150);
    while (avoid == 0) {
      scan_sensors();
        wait(sampledelay);
      }  // End While
    if (avoid != 0) action(avoid);
      avoid = 0;
  }   // Infinate Loop
}





/* ------------- INIT CONSTANTS --------------*/

void init_constants(void) {

  ledmask = ledmask | irled;
  if(room_is_dark)  ledmask = ledmask | cdsled;
  else        ledmask = ledmask & ~(cdsled);
  UPDATE_LEDS;

  rightCDS_init   = RIGHT_CDS;
  leftCDS_init  = LEFT_CDS;
  wait(50);
  rightCDS_init   = (rightCDS_init + RIGHT_CDS)/2;
  leftCDS_init  = (leftCDS_init + LEFT_CDS)/2;
  wait(50);
  rightCDS_init = (rightCDS_init + RIGHT_CDS)/2;
  leftCDS_init  = (leftCDS_init + LEFT_CDS)/2;
  wait(50);
  rightCDS_init   = (rightCDS_init + RIGHT_CDS)/2;
  leftCDS_init  = (leftCDS_init + LEFT_CDS)/2;
  //printf("Left CDS Init: %d    Right CDS Init: %d\n",leftCDS_init,rightCDS_init);
}






/* --------------------- INIT SERVOS -------------------------*/
void init_servos(void) {
  servo(push,push_init_pw);
  wait(500);
  servo(plat,plat_init_pw);
  wait(500);
  servo(arm,arm_init_pw);
  wait(500);
  servo(arm,0);
  servo(plat,0);
  servo(push,0);

}
```

```
/* ---------------------- SCAN SENSORS ------------------------*/
void scan_sensors(void) {


   leftCDS_reading = abs(leftCDS_init - LEFT_CDS);
   rightCDS_reading = abs(rightCDS_init - RIGHT_CDS);
   bump_reading = BUMPER;
  //         printf("rightCDS_reading:         %d\n         leftCDS_reading:         %d",
rightCDS_reading,leftCDS_reading);


  if (BREAK_BEAM == 1)
  {
      wait(25); //allows m&m to get closer to sorting arm
      if(BREAK_BEAM == 1) avoid = 3;
  }

    else if (
        (rightCDS_reading >= CDS_THRESHOLD_D && room_is_dark == 1) ||
        (rightCDS_reading >= CDS_THRESHOLD_L && room_is_dark == 0)
        )
    {
        wait(2);
        if  (
          (rightCDS_reading >= CDS_THRESHOLD_D && room_is_dark == 1) ||
          (rightCDS_reading >= CDS_THRESHOLD_L && room_is_dark == 0)
          )
        {
          avoid = 1;
          printf("RightCDS Triggered: %d\n", rightCDS_reading);
        }
    }
    else if (
        (leftCDS_reading >= CDS_THRESHOLD_D && room_is_dark == 1) ||
        (leftCDS_reading >= CDS_THRESHOLD_L && room_is_dark == 0)
        )
    {
        wait(2);
        if  (
          (leftCDS_reading >= CDS_THRESHOLD_D && room_is_dark == 1) ||
          (leftCDS_reading >= CDS_THRESHOLD_L && room_is_dark == 0)
          )
        {
          avoid = 2;
          printf("LeftCDS Triggered: %d\n", leftCDS_reading);
        }
    }
    else if (RIGHT_IR > IR_THRESHOLD && RIGHT_IR < 175)
    {
      wait(doublecheck);
        if (RIGHT_IR > IR_THRESHOLD && RIGHT_IR < 175)
        {
          avoid = 2;
            printf("Right IR Triggered: %d\n", RIGHT_IR);
        }
    }
    else if (LEFT_IR > IR_THRESHOLD && LEFT_IR < 175)
    {
        wait(doublecheck);
        if (LEFT_IR > IR_THRESHOLD && LEFT_IR < 175)
        {
          avoid = 1;
            printf("Left IR Triggered: %d\n", LEFT_IR);
        }
    }
    else if (bump_reading > 10 ) avoid = (TCNT % 2) + 1;
  } // END FUNCTION
```

```
/* ----------------------- ACTION ------------------------ */
void action(int n) {
  switch(n) {

   case 1: // SOMETHING TO THE RIGHT

    rand = (TCNT % MAXTURNDELAY);
      stop();
      backwards(speed);
    wait(backwardsdelay);
    stop();
    if (rand < MAXTURNDELAY/4)
        rand = MAXTURNDELAY/4;

    left(speed);
    wait(rand);
    break;

    case 2: // SOMETHING TO THE LEFT
      rand = (TCNT % MAXTURNDELAY);
      stop();
      backwards(speed);
    wait(backwardsdelay);
    stop();
    if (rand < MAXTURNDELAY/4)
        rand = MAXTURNDELAY/4;

      right(speed);
      wait(rand);
      break;

    case 3: // BREAK BEAM
      stop();
      sort();
      break;

      default:
      break;
  }

} // End function
```

```
/* ------------------- SORT -------------------- */
void sort(void) {
  while (armPosition < 6)
  {
        switch(armPosition)
        {
        // GRAB THE M&M, BRING THE ARM TO A POSITION SO THAT WE CAN BEGIN COLOR DETECTION
          case 1:


          // PUSH ARMS
          /*
            for (pushpw; pushpw >= push_in_pw; pushpw = pushpw - push_change) {
              wait(servodelay);
              servo(push,pushpw);
            }
            wait(300);
            for (pushpw; pushpw <= push_init_pw; pushpw = pushpw + push_change) {
              wait(servodelay);
              servo(push,pushpw);
            }
          */

          // SORT ARM
            for (armpw; armpw <= sortPos1; armpw = armpw + arm_change)
            {
              wait(servodelay);
              servo(arm, armpw);
            }
            wait(750);
            for (armpw; armpw >= sortPos2; armpw = armpw - arm_change)
            {
              wait(servodelay);
              servo(arm, armpw);
            }


            wait(candy_settle);                       // Allow m&m to settle
            noLight = 0;
            LEDS_OFF;
            while(noLight < 230) {
            noLight = COLOR_CDS;                      // Obtain a zero reading
            //printf("No Light: %d\n",noLight);
            }


            GREEN_ON;
            wait(LIGHTON);
            green = (noLight - COLOR_CDS);
            LEDS_OFF;



            WHITE_ON;
            wait(LIGHTON);
            white = (noLight - COLOR_CDS);
            LEDS_OFF;

            BLUE_ON;
            wait(LIGHTON);
            blue = (noLight - COLOR_CDS);
            LEDS_OFF;


            RED_ON;
            wait(LIGHTON);
            red = (noLight - COLOR_CDS);
            LEDS_OFF;
          color = detectColor(white,red,blue,green);
          printf("White: %d  Red: %d  Blue: %d  Green: %d   \n", white,red,blue,green);

          /**************
          **  brown:  1 **
          **  blue:   2 **
          **  green:  3 **
          **  red:    4 **
          **  yellow: 5 **
          **  orange: 6 **
          **  error:  0 **
           **************/
          switch(color) {
            case 1:
              servo(plat,brownpos);
              break;
            case 2:
              servo(plat,bluepos);
              break;
            case 3:
              servo(plat,greenpos);
              break;
            case 4:
              servo(plat,redpos);
              break;
            case 5:
              servo(plat,yellowpos);
              break;
            case 6:
              servo(plat,orangepos);
              break;
            default:
              servo(plat,badpos);
              break;

          } /* END SWITCH */
          wait(500);
          armPosition = 2;
          break;



      // COLOR DETECTION IS FINISHED AND THE ARM IS FACING THE CORRECT BIN, DROP M&M to
BOTTOM OF CHAMBER
        case 2:
          for (armpw; armpw >= sortPos3; armpw = armpw - arm_change){
            wait(servodelay);
            servo(arm, armpw);
          }
          wait(200);
          armPosition = 3;
          break;



      // ROTATE ARM UNTIL M&M FALLS OUT
        case 3:
          for (armpw; armpw <= sortPos4; armpw = armpw + arm_change){
            wait(servodelay);
            servo(arm, armpw);
          }
          wait(fallout_delay);
          armPosition = 4;
          break;
        case 4:
          for (armpw; armpw >= sortPos5; armpw = armpw - arm_change){
            wait(servodelay);
            servo(arm, armpw);
          }
          armPosition = 5;
          break;
      // PUT PLATFORM AND ARM BACK TO IN INITIAL POSITIONS
        case 5:
          armPosition = 6;
          platpw = plat_init_pw;
          armpw = arm_init_pw;
          pushpw = push_init_pw;
          init_servos();
          break;
        default:
          armPosition = 6;
          platpw = plat_init_pw;
          armpw = arm_init_pw;
          pushpw = push_init_pw;
          init_servos();
          break;
      }/* End switch(armPosition) */
    } /*END WHILE*/
  } /* END FUNCTION */
```

```
/*************************************
*** COLOR DETECTION ALGORITHM       **
*** AUTHOR: VINH TRINH              **
*** FOR USE BY: EM                  **
**************************************
** COLOR CODES:                     **
**  white: 7                        **
**  brown:  1                       **
**  blue:   2                       **
**  green:  3                       **
**  red:    4                       **
**  yellow: 5                       **
**  orange: 6                       **
**  error:  0                       **
*************************************/

int detectColor(int white, int red, int blue, int green) {
  if ((white + red + blue + green) <= 260) {
  /* Brown, Perhaps Blue */
    if ((green + white + blue - red) <= 116) return 1; // brown
    else return 2; // blue
  }
  else if ((red <= 78)) {
  /* Blue, Perhaps Green */
    if ((green + white + blue + red) <= 367) return 2; // blue
    else return 3; // green
  }
  else if ((green + blue - red) >= 55) {
  /* GREEN, Perhaps Yellow */
    if ((white + red - green - blue) <= 110) return 3; // green
    else return 5; // yellow
  }
  else if ((green+white+blue) >= 308) return 5; // yellow
  else if ((green + white + blue + red) >= 372) return 6; // orange
  /* RED, Perhaps Brown */
  else if ((red >= 115)) return 4; // red
  else return 1;
}
```