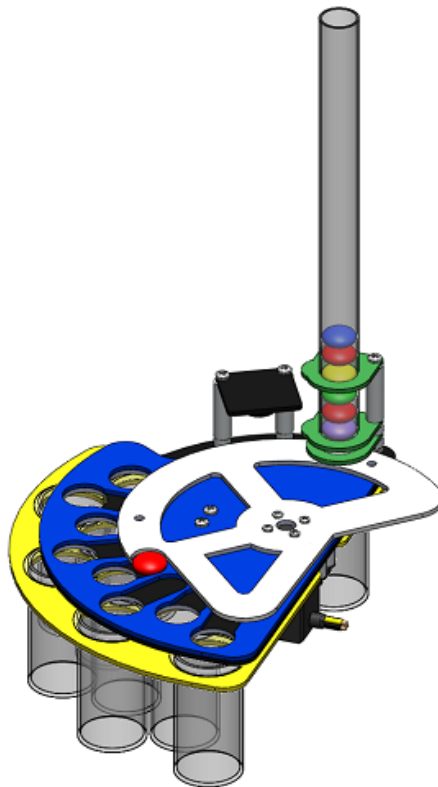


M&M's® Color Sorter

Final Project Report

LAURENT CHRIS GOUDET

December 11, 2009



Abstract:

Color sorting is a recurrent problem in the food industry. Such machines are necessary for sorting grains, seeds or coffee. This project, entitled "M&M's® Color Sorter", develops a proof of concept for color sorting by using candies. Given a random assortment of M&M's® candies, consisting of up to 8 different colors, the device will be able to sort them by color in a reasonable time (maximum of 10 seconds per candy) and with accuracy higher than 80% (4 candies out of 5 sorted correctly).

Introduction

Color detection is an important problematic in a wide range of industrial areas. For instance, powders, granules or liquids as well as metals, glass, types of paper, plastics and all kinds of textiles are checked and detected according to significant color features.

A typical color sensor is based on a photodiode which measure the intensity reflected by the object for a red, green and blue light source. More advanced systems may use more colors or more specifics colors, such orange light sources of different wavelength in order to detect the hue of an orange (the fruit). Some sensors, like the one used in this project, used the same principle but with 3 or more photodiodes with green, red and blue filters in front of them. In this case the light source may be white.

The prototype

Different mechanical designs are possible for a candy sorter but these designs are similar: a first actuator is moving the candies one by one in the “sorting zone” and one or many actuators are sorting them.

Regarding this project, the mechanical design have been found on YouTube¹ and recreated under Solidworks® (all the mechanical drawings can be found in Appendix B). It was chosen because it is one of the most mechanically straightforward M&M's® sorter designs, since it includes only one mobile part (the rotor). A 3D representation of the prototype can be found below (Figure 1):

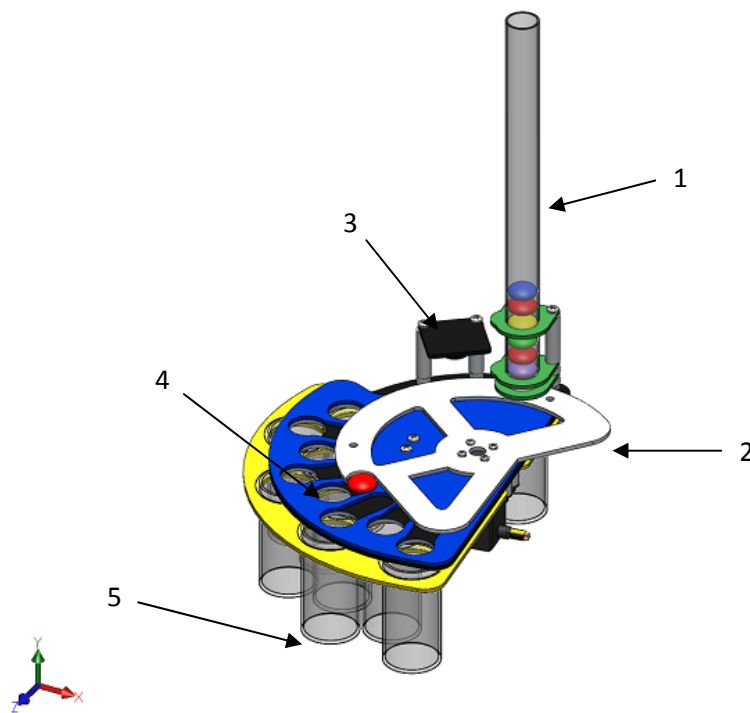


Figure 1: 3D representation of the prototype

¹ <http://www.youtube.com/watch?v=r7TRyHnvJsM>

The non-sorted M&M's® are introduced through a feeder tube (1). The rotor (2) then moves one candy under the color sensor board, mounted at (3). The color sensor determines the color of the candy, and then the rotor moves the candy into the channel (4) corresponding to the determined color. By simply reversing the rotor's direction, the candy is pushed down the channel and into the container (5) below.

The parts have been made by a 3D printer, thanks to Mike from the ECF. They have then been painted in white, black and blue. These colors are not completely arbitrary: the white color of the rotor is used for the white calibration – explained later – and the black color of the base is used to detect when the feeder tube is empty. Some photos of the final prototype can be found below (Figures 2, 3 & 4).

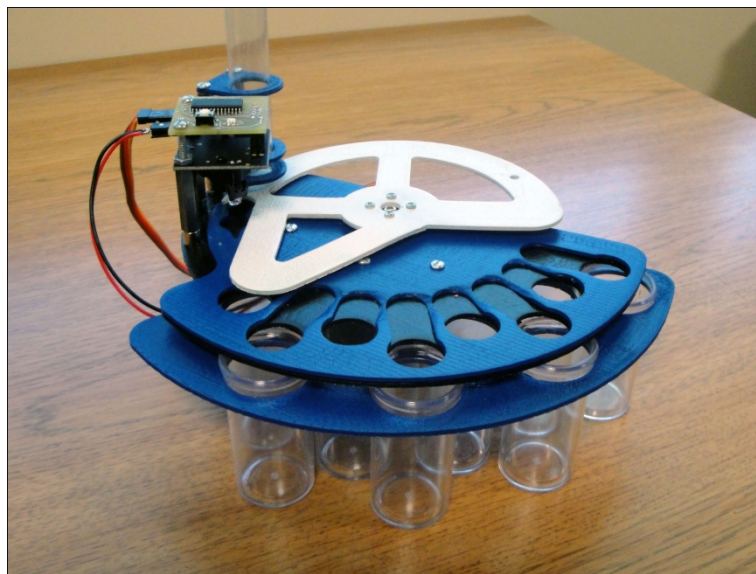


Figure 2: The prototype

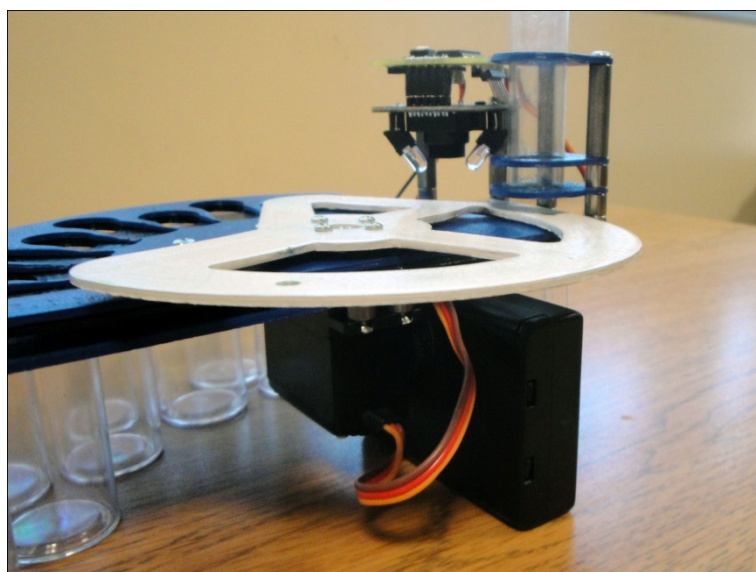


Figure 3: The servo and the battery pack

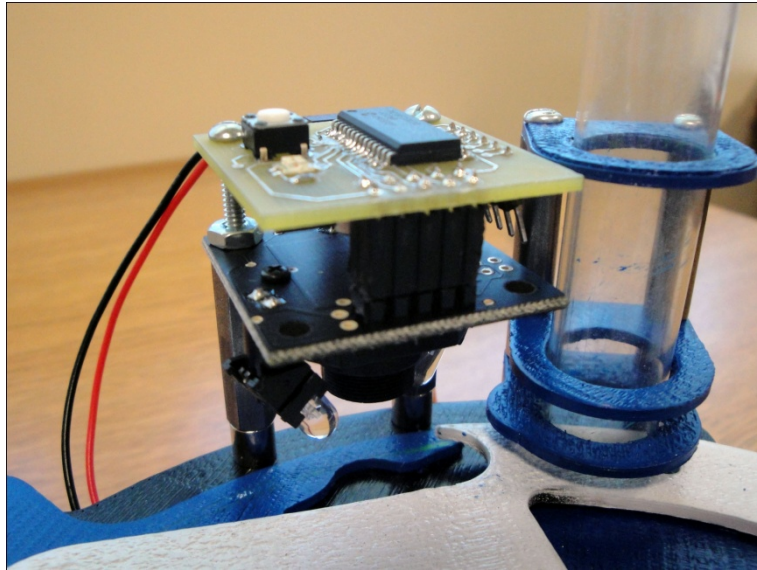


Figure 4: The color sensor and the PCB board

The Hardware

Electrical overview

The overall electrical of the system is depicted in Figure 5 (below):

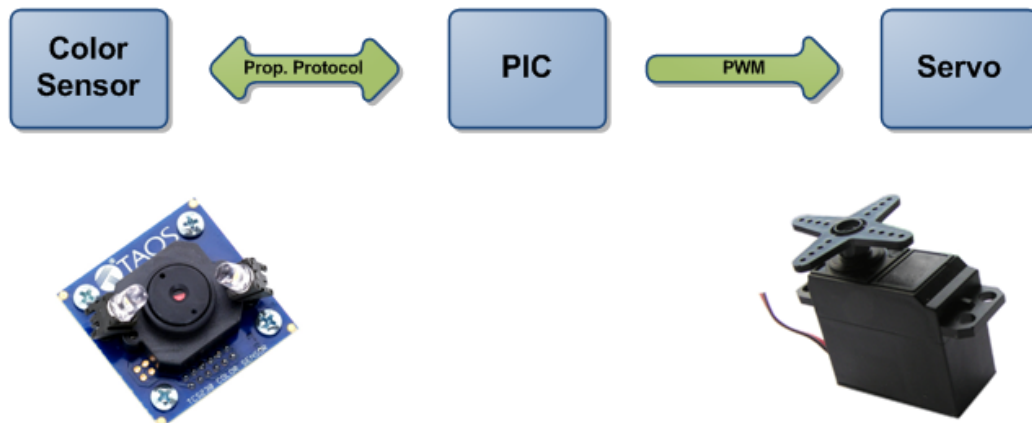


Figure 5: Electrical overview of the system

The color sensor and the servo, which is connected to the rotor, are both connected to the PIC, which is driving both of them.

Color Sensor

The color sensor which is used for detecting the color of the candies is the TCS230 sensor module from TAOS®. It combines configurable silicon photodiodes and a current-to-frequency converter on single monolithic CMOS integrated circuit, as shown on the Figure 6 (below).

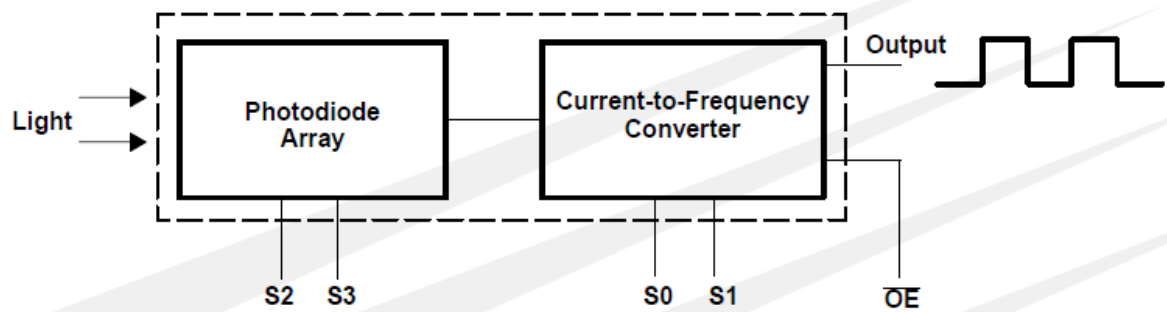


Figure 6: Functional block diagram of the TCS230

The light-to-frequency converter reads an 8 x 8 array of photodiodes. Sixteen photodiodes have blue filters, 16 photodiodes have green filters, 16 photodiodes have red filters, and 16 photodiodes are clear with no filters.

The inputs to the device can be chosen so that the sensor detects one of these colors, and in response the device outputs a square wave (50% duty cycle) with frequency directly proportional to light intensity (irradiance) of the chosen color.

In our application, this sensor needs to be used in association with a lens, to focus the object to analyze, and a white light source (white LEDs). For that reason, it has been bought with the daughterboard AXE045 from PIXACE (Figure 7). The development of a daughterboard for this sensor would have been more expensive because the CMOS lenses are hard to find in small quantities.



Figure 7: The AXE045 daughterboard for TCS color sensor from PIXACE

The PCB board

The whole system has first being designed and tested on a Harris board, a development board using a PIC 18F452 microcontroller. Because it was interesting to have a final device that can be easily used and moved, a PCB board has then been developed to hold the sorting program.

This PCB board is composed of a microcontroller, a 3.3V voltage regulator, a push button, a bi-color LED, some connectors and few passives components. The schematic of the board is available in appendix C. In order to reduce the number of components, there is not any crystal on the board: the software used the internal clock of the microcontroller.

The microcontroller has been chosen regarding the following specifications:

- Compatibility: be compatible with the current development board (a Harris board). This board is using a PIC 18F452 so the microcontroller should had to be picked up into the PIC 18F family in order to keep the same hardware architecture (same Timers) and the same C Compiler (C18), to keep the same libraries.
- Package: be small enough to fit on a 1.2-in x 1.2-in PCB board but can easily be soldered (thus not a QFN or a TQFP package)
- Functionalities: have a Low Voltage Detection module. The system is designed to be used on batteries so a low voltage detection feature is interesting in order to inform the user that he has to replace the batteries.
- Price: be, of course, as cheap as possible

Based on these constrains, the microcontroller which has been chosen is the PIC 18K23K20. A 6 pins connector has been designed on the PCB board to program it (using the ICSP technology).

The PCB board itself has been developed using the software suite Proteus ISIS & ARES by Labcenter Electronics. It has then been manufactured by 4PCB.com, using the bare bone service. In consequence, the board is a 2-layer FR-4 0.062" thick without any legends (Figure 8).

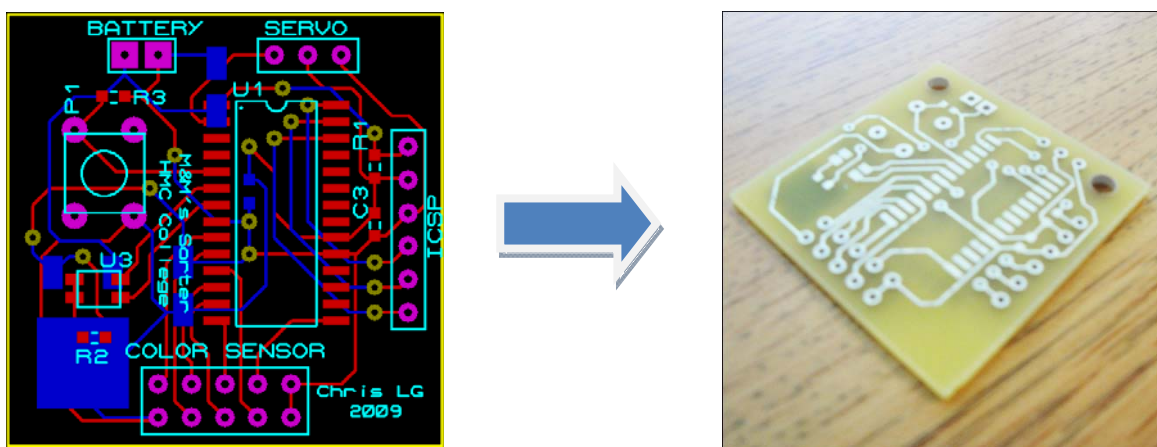


Figure 8: The PCB board, from the CAD software to the board

The Software

General operation

The general operation of the embedded software is depicted by the flowchart below (Figure 9):

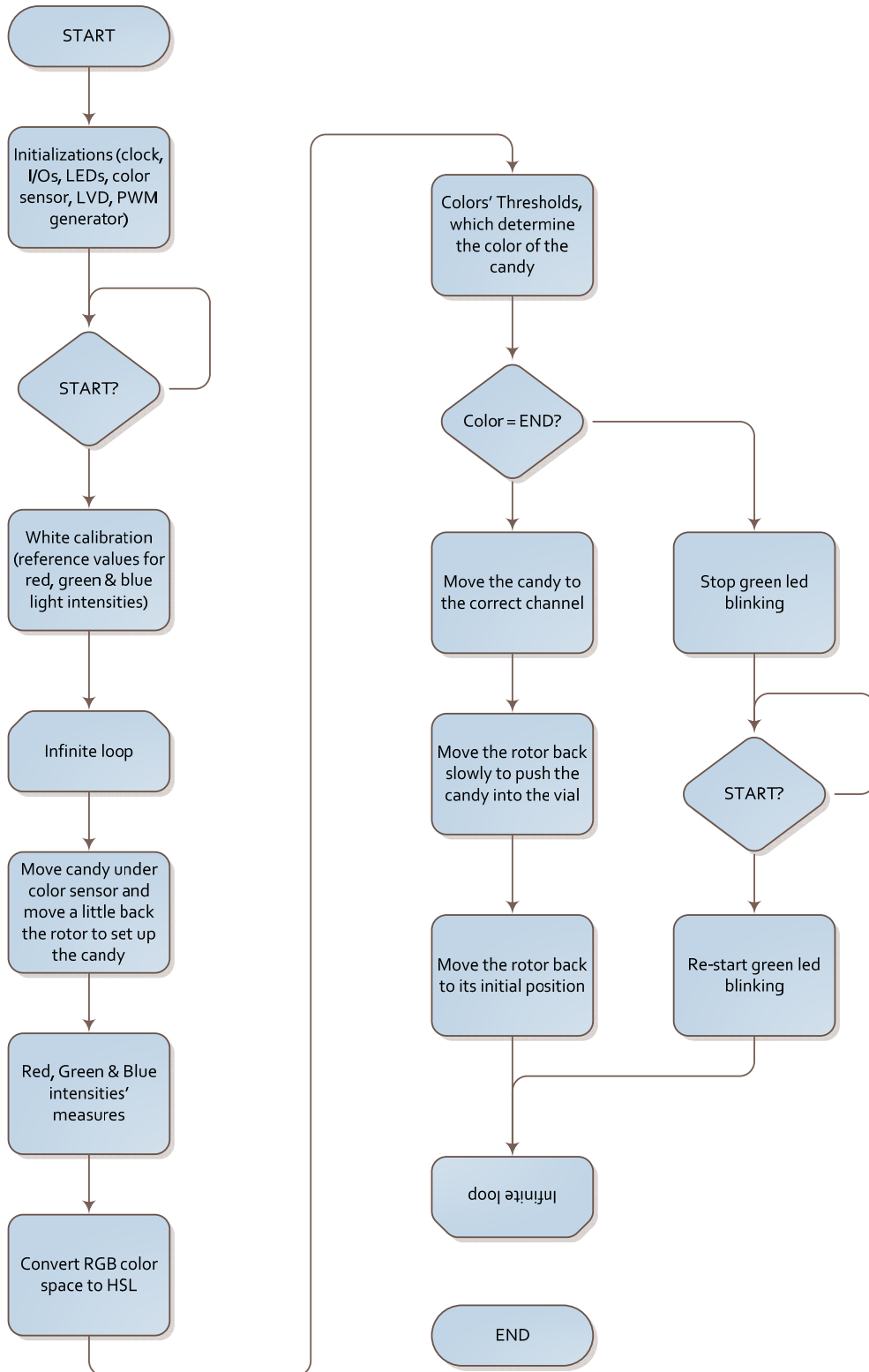


Figure 9: General operation of the embedded software

The frequency meter

The software implements a frequency meter for measuring the output frequency of the color sensor. In fact, as explained before, the output of the color sensor is a square wave (50% duty cycle) with frequency directly proportional to light intensity (irradiance).

For that purpose, the timer 1 of the microcontroller is used. This latter is set to count the low-to-high transitions on its external source input, RC0 (Port C, bit 0). The timer is started and, after a certain amount of time, stopped. The result is a number, in the timer 1 register, representing the output frequency of the color sensor.

The “amount of time” to wait until stop the timer have been determined by experimentation. In fact, the maximum output frequency of the color sensor is, according to the datasheet, 600 kHz but in practice this maximal frequency is closer to 2 kHz. As a consequence, the capture time could have been set according to this maximum but, in practice, the output of the color sensor never reaches this value.

The white calibration

The first step in order to use the color sensor is to proceed to a white calibration. In fact, as shown on the diagram below (Figure 10), the relative responsivity of the red, green and blue photodiodes are not the same. As a consequence, a white calibration is necessary in order to normalize the values given by the color sensor.

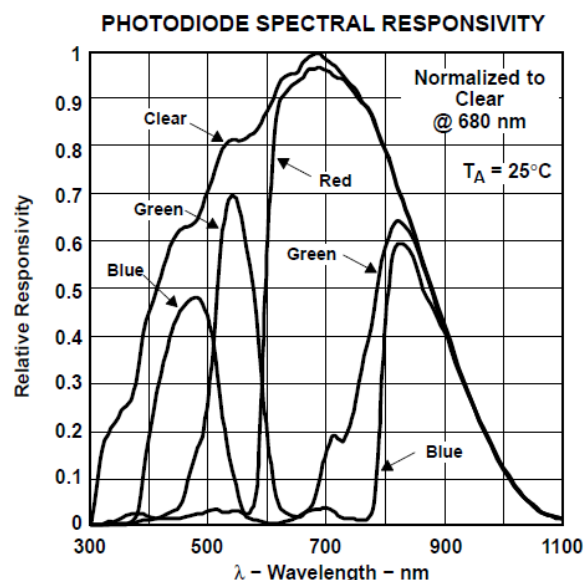


Figure 10: Photodiode Spectral Responsivity of the Color Sensor

Another benefit of this white calibration is to get extremum values for the red, green and blue intensity. In fact, the output frequency of the color sensor depends of the light intensity. By measuring this frequency for the red, green and blue light reflected by a white surface, one can know the maximum values of light intensity.

Thus, to achieve the white calibration, the light intensity reflected by the rotor (which is white for this purpose) is measured by the red, green and blue photodiodes. These values are then stored in order to be used as a reference the futures color detections.

The color space conversion

The color information given by the color sensor uses the RGB space, i.e. there is one value for the red intensity, one for the green intensity and one for the blue intensity. This color space can be represented as a cube, whose axes are defined by those 3 primary colors (Figure 11).

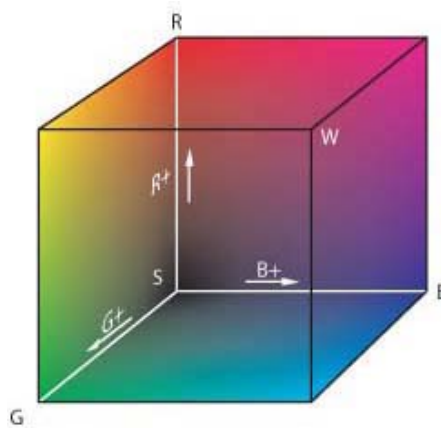


Figure 11: Graphical representation of the RGB color space

Such color space is not compliant with our application because it will imply to use 3 thresholds values for each color. Furthermore, in this color space the result of the detected color will depend of luminosity conditions, i.e. if there is more lights in the room where the device is used the values from the color sensor will be different.

For resolving this problem, the embedded software proceeds to a color space conversion to the HSL color space. The HSL (Hue, Saturation & Luminance) space is a related representation of points in the RGB color space, which attempt to describe colors more naturally than RGB. It can be arranged as a double-cone as shown below (Figure 3).

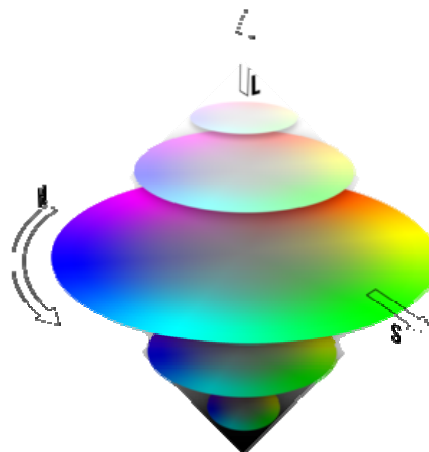


Figure 12: Double-cone representation of the HSL color space

In our application, all the benefit of this color space the hue value. The hue, which is equal to the ratio of each primary (RGB) color to the others, specifies the base color and can be represented as a circle (Figure 4).

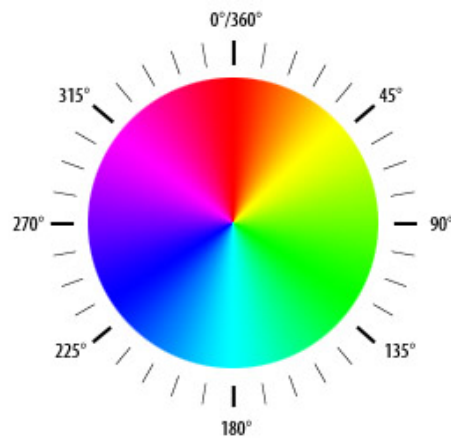


Figure 13: Circle representation of the Hue

Thresholds values for candies' colors can be thus easily define from this circle and only one number is necessary. Furthermore, the hue value will not depend of the luminosity conditions so the reliability of the color detection process will be better.

The colors' thresholds

The colors' thresholds have been set using the hue value of the HSL color space. These values have been found by experimentation. The following table (Table 1) shows some experimental results of the hue value. Some values in this table are negatives because 360° is equivalent to 0°, so 350° is equivalent to -10° (useful for the calculations).

	Red	Orange	Yellow	Green	Blue	Purple	Pink
1	-0.55	12.03	44.04	127.46	216.53	298.05	346.62
2	3.33	10.42	42.84	121.95	223.60	272.68	349.90
3	3.24	10.64	45.00	123.70	220.14	289.62	352.02
4	5.70	10.22	49.10	122.51	216.31	285.65	353.57
5	-5.26	11.13	42.85	122.53	223.32	272.25	357.50
6	8.54	11.62	44.23	121.71	220.04	274.26	352.28
7	9.44	11.47	44.64	128.78	221.17	293.74	353.45
8	4.28	12.29	45.63	124.76	216.32	277.35	344.74
9	5.06	12.32	43.45	122.56	218.46	298.33	346.34
10	3.08	12.06	44.25	126.36	218.38	289.23	346.62
Average	3.69	11.42	44.60	124.23	219.43	285.12	350.30
<i>Standard deviation</i>	<i>4.24</i>	<i>0.78</i>	<i>1.81</i>	<i>2.50</i>	<i>2.71</i>	<i>10.29</i>	<i>4.13</i>

Table 1: Experimental values of hues (after color space conversion)

For each average value of hue, the medians with the two closer values have been calculated and set as thresholds for the corresponding color.

Some remarks about this table: first, these tests have been done without real Pink and Purple M&M's because these colors have not been found. The Pink and Purple colors are thus detected from printed drawings.

Then, one can see that there is not any overlap between the colors by using the hue circle conversion. That implies that, except in case of an error from the mechanism, the candies will be expected to be sort with a 100% of accuracy.

The brown color is not represented in this table. In fact, the brown color is not even represented on the hue circle. But, this color can be detected by using the luminance value of the HSL color space. Few others tests have been necessary to find a good threshold value of luminance for the brown color.

At last, one more "color" have been added: the color corresponding to the case where there is not any M&M's left in the feeder tube. For that purpose, the base of the mechanism is in black and a threshold has been set to detect this color, namely "END" color. As the one for the brown color, this threshold is based on the luminance value.

The PWM Generator

The PWM Generator is used to generate a PWM signal to drive the servo. The position of the servo is defined by the width of a pulse (in range 0.5ms to 2.5ms). For instance, sending a 1.5ms pulse to the servo, tells the servo that the desired position is 90 degrees. In order for the servo to hold this position, the command must be sent at about 50Hz, or every 20ms (Figure 14).

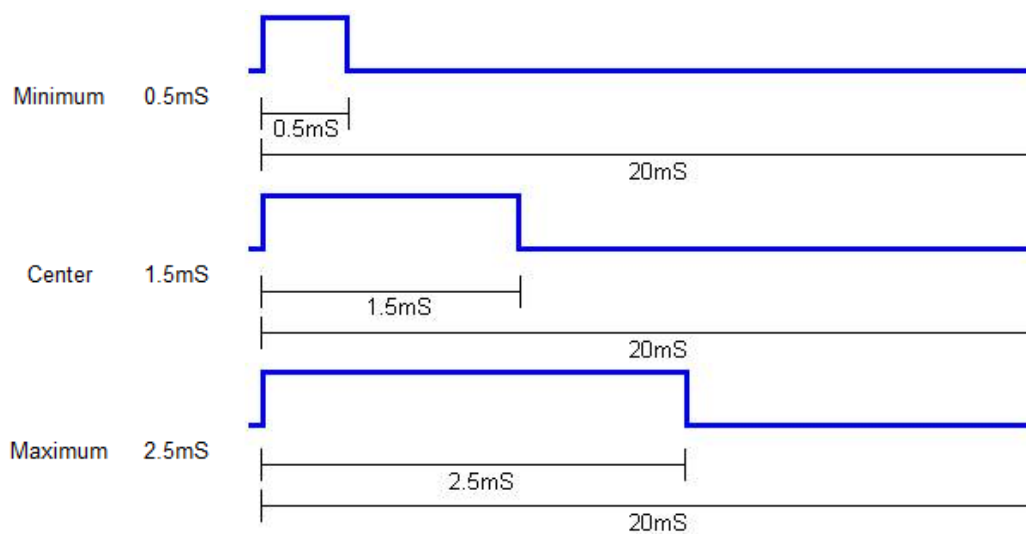


Figure 14: Position of the servo vs. width of the pulse

For this purpose two timers are used: the timer 0 is set to generate an interrupt every 20ms and the timer 3, which also used interrupts, is set to a value which represent the width of the high period of the PWM signal (Figure 15). The code for this PWM generator have been taken over and adapted from the code of the lab 7.

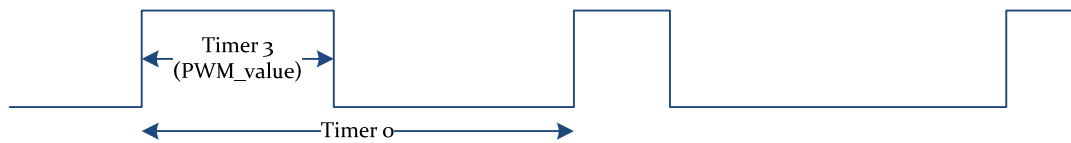


Figure 15: Implementation of the PWM generator

The other functionalities

A blinking feature has been implemented to allow the green and the red LEDs to blink when the system is running. The green LED stop blinking and stay on when there is no more M&M's in the feeder tube. The red LED, it, is blinking when a low voltage have been detected (functionality explained below).

In order to optimize the code of the program and the resources of the PIC, this blinking feature is using the same timer's interrupts than the PWM generator (Timer 0). Given that the period of the timer 0 is set to 20ms, a counter is waiting until 25 interrupts before flip the LED output. The LED is thus blinking at a frequency of 1 Hz.

The program also implements the Low-Voltage Detect module embedded into the PIC. This is a programmable circuit that allows the user to specify both a device voltage trip point and the direction of change from that point. If the device experiences an excursion past the trip point in that direction, an interrupt flag is set.

Once a low voltage has been detected, the green LED is turned off and the red LED is blinking to indicate that the batteries need to be replaced. To reduce the number of false detections a counter has been set so the device will enter in the "low batteries mode" only after 10 low voltage detections. In fact, the servo is sometime pulling a lot of current and, as a consequence, the voltage on the board is dropping. It is also why the voltage detection limit has been set, by experimentation, to only 2.3V.

Results & Conclusions

In order to confirm the results of the project, experiments have been conducted a relatively large quantity of M&M's. Each color of M&M's has been tested 50 times. The results of these experiments are shown in the chart below (Figure 14):

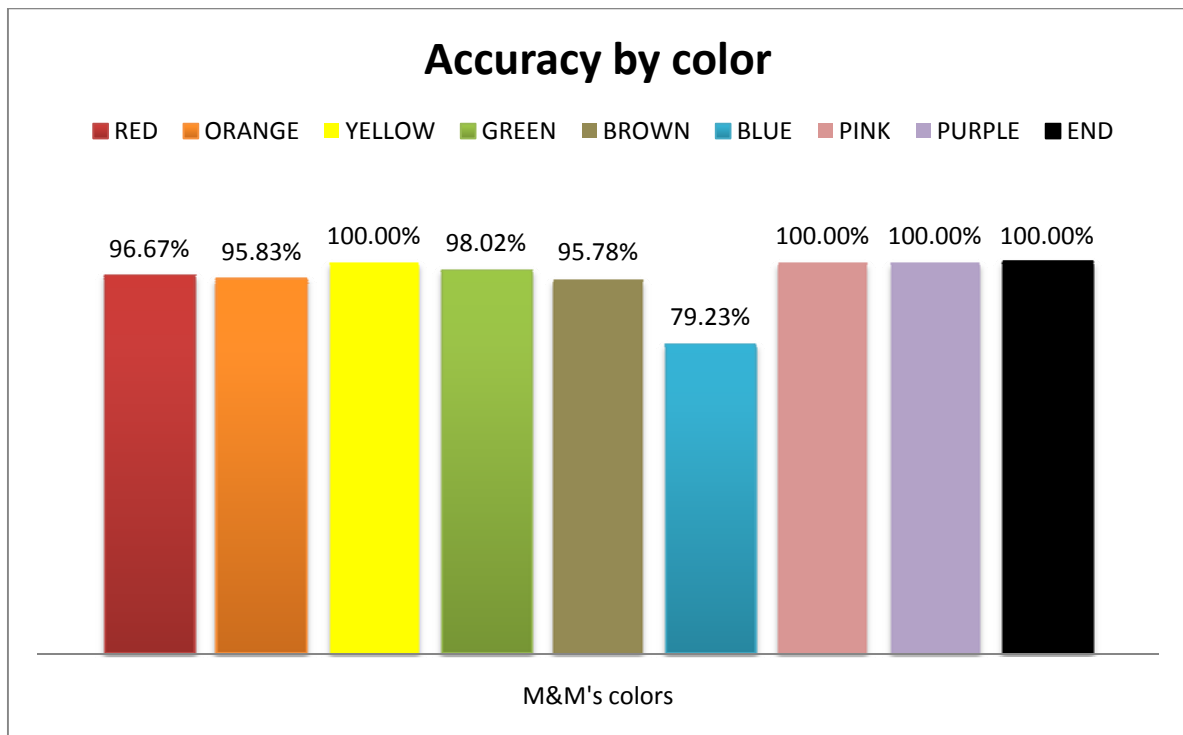


Figure 16: Accuracy by color of the sorter

Some conclusions can be drawn from these experiments. First, the little inaccuracy between the red and the orange M&M's is due to an error of detection. Sometimes the M&M's are not well positioned under the color sensor and, since these colors are quite similar, the color is not well detected (red M&M's become orange and orange M&M's become red).

This issue could maybe be solve by using another parameter, the luminance, to distinguish the difference between the red and the orange color, or by improving the positioning of the M&M's under the color sensor.

The others inaccuracy are explained by the fact that, sometimes, the M&M's are "jumping" to the next channel. This phenomenon is due to the rotor, which is a little twisted. This problem could be solved by using another material for the rotor, more rigid.

Beside these little inaccuracies, the project is a success. The system meets its specifications (at least 4 M&M's out of 5 well sorted) and the final aspect of the prototype is a success. Since the device is working on batteries, one possible improvement will be to optimize the power consumption by, for instance, entered the microcontroller in sleep mode when the feeder tube is empty.

References





- ⇒ Color sensors applications:
<http://www.baumerelectric.com/be27.html?L=1>
- ⇒ RGB color space:
http://en.wikipedia.org/wiki/RGB_color_space
- ⇒ HSL color space:
http://www.chaospro.de/documentation/html/paletteeditor/colorspace_hsl.htm
- ⇒ PIC 18F23K20 datasheet:
<http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en530013>
- ⇒ TCS230 color sensor datasheet
<http://www.taosinc.com/productdetail.aspx?product=3>

Appendixes

- ⇒ Part list
- ⇒ Mechanical drawings
- ⇒ Schematic of the PCB board
- ⇒ Code of the embedded software (PIC)

M&M Sorter – Parts List

Prototype:

<p><i>Item name:</i> Servo</p>	<p><i>Quantity:</i> 1</p>	
<p><i>Specification:</i> Standard hobby servos with cross control horn. Dim. 41x20x40mm</p>		
<p><i>Retail price:</i> \$4.90 + shipping at http://www.hobbypartz.com/kahaoubromo14.html</p>		
<p><i>Item name:</i> Plastic Vials</p>	<p><i>Quantity:</i> 9</p>	
<p><i>Specification:</i> Crystal clear polystyrene vials with snap caps. 7 Dram: 2-1/16" in height with an inside diameter of 1"</p>		
<p><i>Retail price:</i> \$3.95 + shipping at http://www.acornnaturalists.com/store/SNAP-CAP-VIALS-clear-plastic-7-dram-12-per-package-P421C0.aspx</p>		
<p><i>Item name:</i> Plastic Tube</p>	<p><i>Quantity:</i> 1</p>	
<p><i>Specification:</i> Clear Rigid Acrylic Plastic Tube 5/8" outside diameter with 1/16" wall 1' length</p>		
<p><i>Retail price:</i> \$ 2.58 + shipping for 3' at http://www.arcatapet.com/item.cfm?cat=470</p>		
<p><i>Item name:</i> Velcro Fasteners</p>	<p><i>Quantity:</i> 8</p>	
<p><i>Specification:</i> Velcro Fasteners dots 5/8"</p>		
<p><i>Retail price:</i> \$2.49 for 15 at Office Depot http://www.officedepot.com/a/products/570109/Velcro-Sticky-Back-Fasteners-Coins-5/</p>		
<p><i>Item name:</i> Color Sensor</p>	<p><i>Quantity:</i> 2</p>	
<p><i>Specification:</i> PICAXE Color Sensor</p>		
<p><i>Retail price:</i> \$44.95 + shipping at http://www.sparkfun.com/commerce/product_info.php?products_id=8924</p>		
<p><i>Item name:</i> Flat Ribbon</p>	<p><i>Quantity:</i> 1</p>	
<p><i>Specification:</i> Mouser Part #: 517-1M-1021-010-12</p>		
<p><i>Retail price:</i> \$3.18 + shipping at Mouser http://www.mouser.com/ProductDetail/3M/1M-1021-010-3365-0120-00-AB-00-0/?qs=dMyV3i0KP7N%252bDtnULMV4QQ%3d%3d</p>		

Hardware: screws, standoffs, spacers:

<i>Item name:</i> 3/8" Standoffs	<i>Quantity:</i> 6	
<i>Specification:</i> #4-40 x 3/8" Threaded Standoffs		
<i>Retail price:</i> Found in the stockroom		
<i>Item name:</i> 1" Standoffs	<i>Quantity:</i> 4	
<i>Specification:</i> #4-40 x 1" Threaded Standoffs.		
<i>Retail price:</i> Found in the stockroom		
<i>Item name:</i> Spacers	<i>Quantity:</i> 6	
<i>Specification:</i> #4 x 1/4" Thru-hole Aluminum Spacers.		
<i>Retail price:</i> Found in the stockroom		
<i>Item name:</i> 1/4" Flat Machine	<i>Quantity:</i> 4	
<i>Specification:</i> #4-40 x 1/4" Flat Head Machine Screws		
<i>Retail price:</i> Found in the stockroom		
<i>Item name:</i> 1/4" Pan Machine	<i>Quantity:</i> 8	
<i>Specification:</i> #4-40 x 1/4" Pan Head Machine Screws		
<i>Retail price:</i> Found in the stockroom		
<i>Item name:</i> 1/2" Pan Machine	<i>Quantity:</i> 4	
<i>Specification:</i> #4-40 x 1/2" Pan Head Machine Screws		
<i>Retail price:</i> Found in the stockroom		
<i>Item name:</i> 1" Pan Machine	<i>Quantity:</i> 4	
<i>Specification:</i> #4-40 x 1" Pan Head Machine Screws		
<i>Retail price:</i> Found in the stockroom		
<i>Item name:</i> Hex Nuts	<i>Quantity:</i> 4	
<i>Specification:</i> #4-40 Hex Nuts		
<i>Retail price:</i> Found in the stockroom		
<i>Item name:</i> Metal screws	<i>Quantity:</i> 8	
<i>Specification:</i> #2 x 3/16" Pan Head Sheet Metal Screws		
<i>Retail price:</i> \$6.20 at www.microfasteners.com		

PCB Board:

Resistors

2	R1,R3	10k (0603)	652-CR0603-JW-103GLF (Mouser)	\$0.03
1	R2	68 (0603)	301-68-RC (Mouser)	\$0.04

Capacitors

1	C1	22uF (SMD)	647-UCD1C220MCL6GS (Mouser)	\$0.09
2	C3,C4	100nF (0603)	81-GRM39Y104Z16D (Mouser)	\$0.06
1	C5	10uF (SMD)	647-UWF1C100MCL1GB (Mouser)	\$0.06

Integrated Circuits

1	U1	PIC18F23K20	579-PIC18F23K20-I/SO (Mouser)	\$2.31
1	U2	L1117-3	Found in MicroP's lab	-
1	U3	LED-BICO	859-LTST-C155GEKT (Mouser)	\$0.20

Connectors

1	BATTERY	CONN-SIL2	Found in MicroP's lab	-
1	COLOR SENSOR	CONN-DIL10	517-975-01-10 (Mouser)	\$1.70
1	ICSP	CONN-SIL6	Found in MicroP's lab	-
1	P1	PUSH BUTTON	Found in MicroP's lab	-
1	SERVO	CONN-SIL3	Found in MicroP's lab	-

Subtotal components:

\$5.94

Miscellaneous

1	BATTERY HOLDER		12BH331/CS-GR (Mouser)	\$1.45
1	PCB BOARD + SHIPPING		4PCB.com Bare Bone Service	\$67.67

Total PCB:

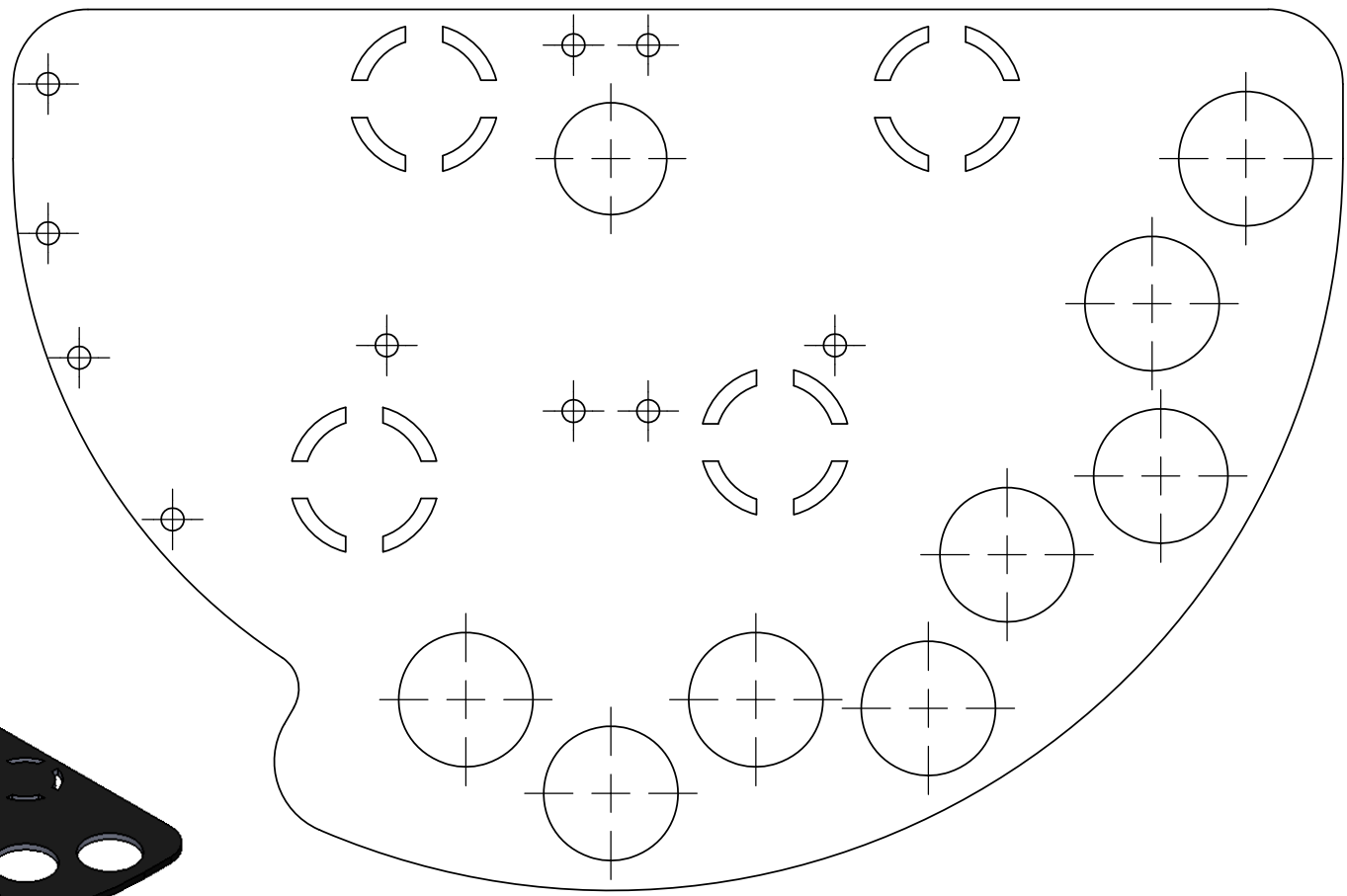
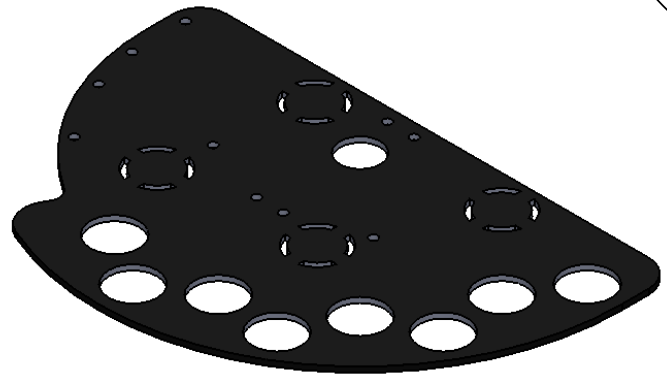
\$75.06

Global budget:

- Prototype: \$64.30
- PCB board: \$75.06

Laurent Goudet
lgoudet@hmc.edu
909-480-5341

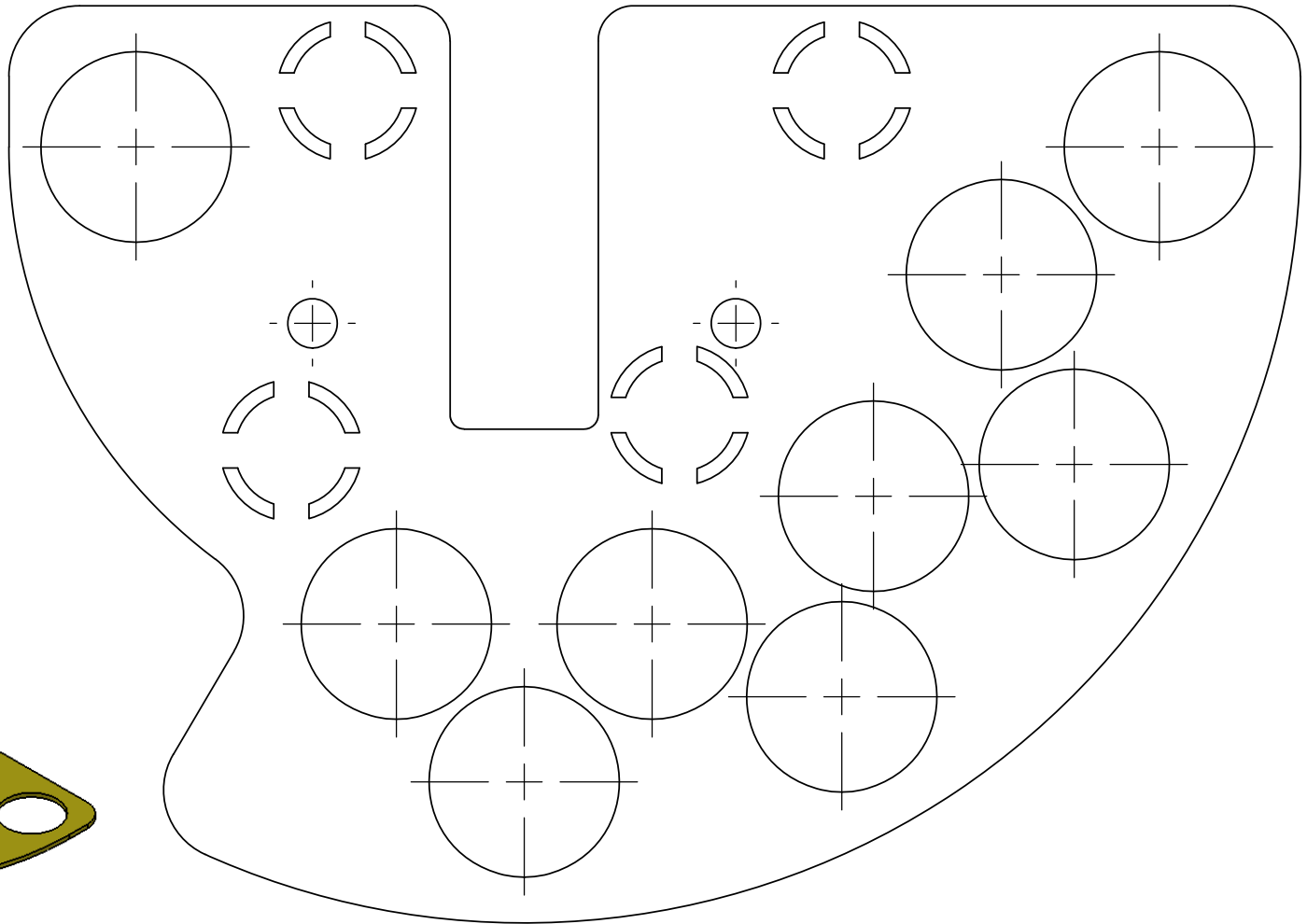
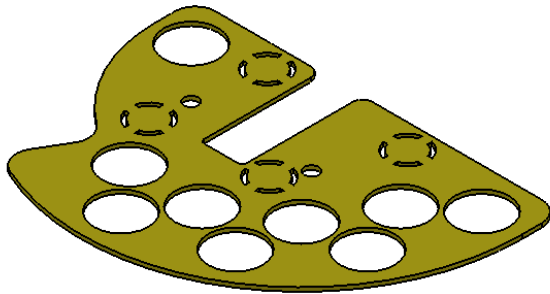
Thickness: 1/16"
Quantity: 2
Color: black if available



PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE>, IS

		UNLESS OTHERWISE SPECIFIED:		NAME	DATE			
		DIMENSIONS ARE IN INCHES		DRAWN		TITLE:		
		TOLERANCES:		CHECKED				
		FRACTIONAL ±		ENG APPR.				
		ANGULAR: MACH ± BEND ±		MFG APPR.				
		TWO PLACE DECIMAL ±		Q.A.		SIZE DWG. NO. REV		
		THREE PLACE DECIMAL ±		COMMENTS:				
		INTERPRET GEOMETRIC TOLERANCING PER:				SCALE: 1:1	WEIGHT:	SHEET 1 OF 1
		MATERIAL						
		FINISH						
NEXT ASSY	USED ON							
APPLICATION		DO NOT SCALE DRAWING						

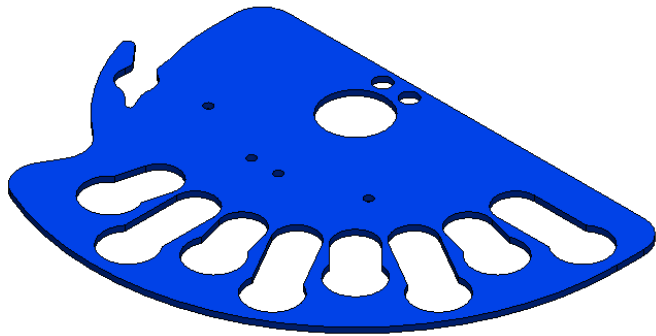
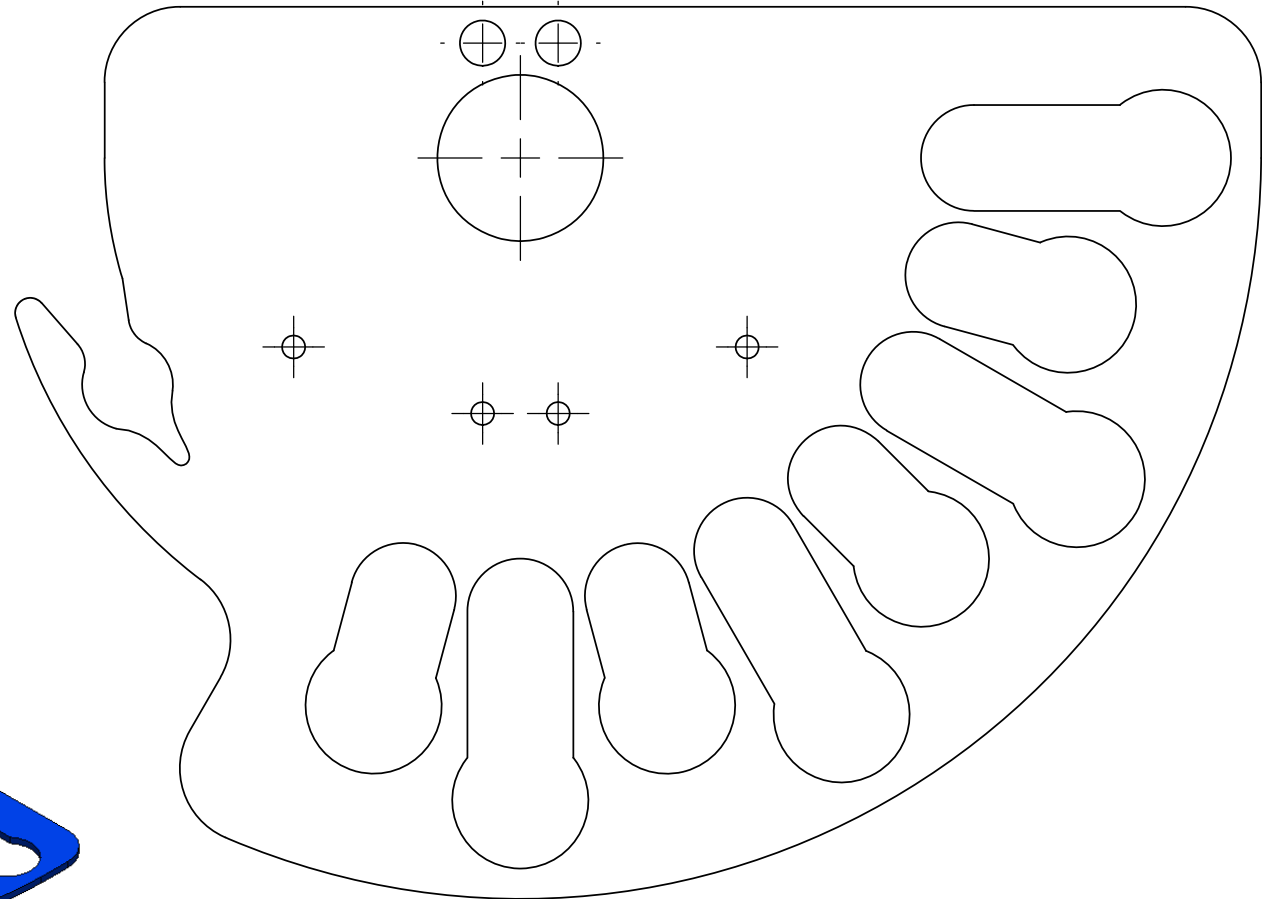
Thickness: 1/16"
 Quantity: 2
 Color: yellow if available



PROPRIETARY AND CONFIDENTIAL
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE>, IS

		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN INCHES	DRAWN			TITLE:
		TOLERANCES:	CHECKED			
		FRACTIONAL ±	ENG APPR.			
		ANGULAR: MACH ± BEND ±	MFG APPR.			
		TWO PLACE DECIMAL ±	Q.A.			SIZE DWG. NO. REV A Vial Holder
		THREE PLACE DECIMAL ±	COMMENTS:			
		INTERPRET GEOMETRIC TOLERANCING PER:				SCALE: 1:1 WEIGHT: SHEET 1 OF 1
		MATERIAL				
		FINISH				
NEXT ASSY	USED ON					
APPLICATION		DO NOT SCALE DRAWING				

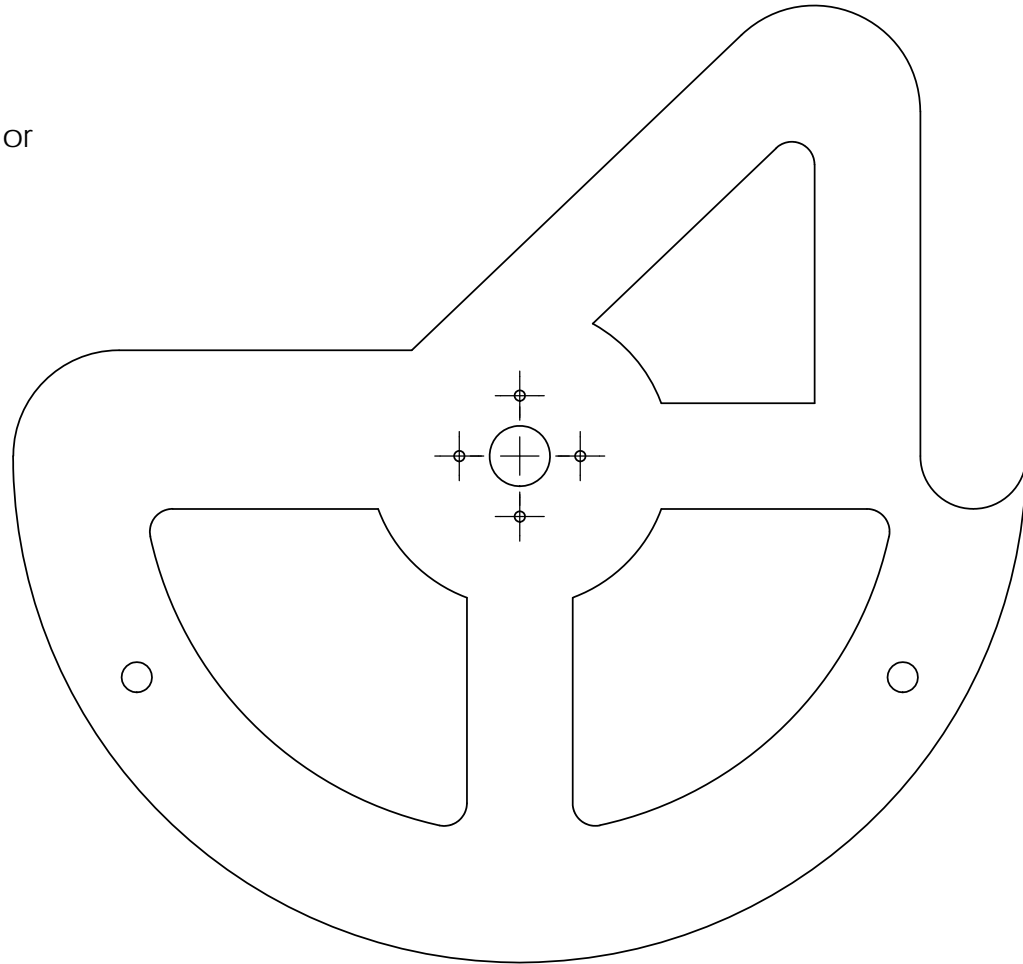
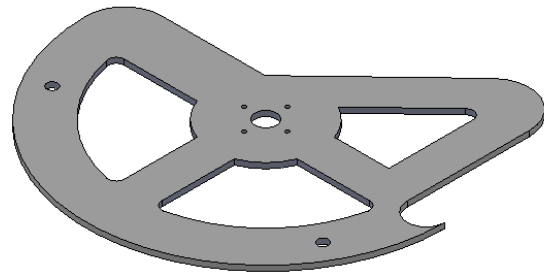
Thickness: 1/16"
 Quantity: 2
 Color: blue if available



PROPRIETARY AND CONFIDENTIAL
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE>, IS

		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN INCHES	DRAWN			TITLE:
		TOLERANCES:	CHECKED			
		FRACTIONAL ±	ENG APPR.			
		ANGULAR: MACH ± BEND ±	MFG APPR.			
		TWO PLACE DECIMAL ±	Q.A.			SIZE DWG. NO. REV A Separator
		THREE PLACE DECIMAL ±	COMMENTS:			
		INTERPRET GEOMETRIC TOLERANCING PER:				
		MATERIAL				SCALE: 1:2 WEIGHT: SHEET 1 OF 1
		FINISH				
NEXT ASSY	USED ON					
APPLICATION		DO NOT SCALE DRAWING				

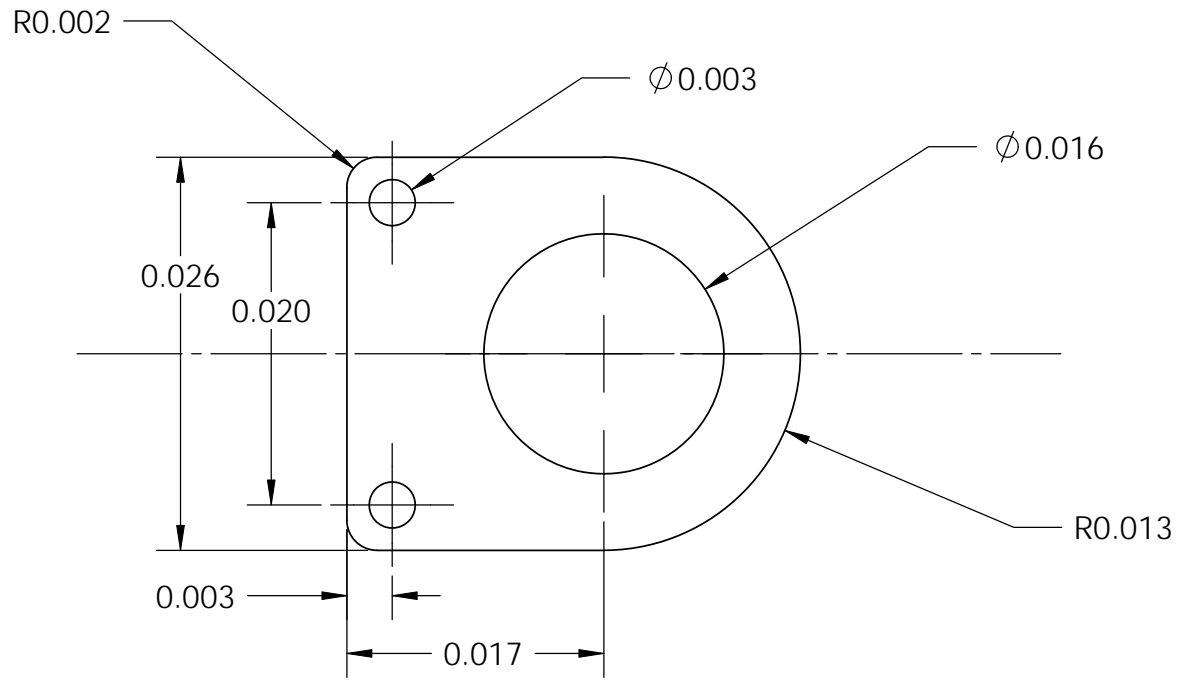
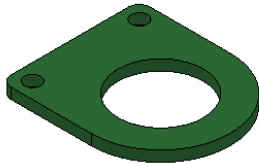
Thickness: 1/16"
 Quantity: 2
 Color: white if available or another light color



PROPRIETARY AND CONFIDENTIAL
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE>, IS

		UNLESS OTHERWISE SPECIFIED:		NAME	DATE			
		DIMENSIONS ARE IN INCHES	DRAWN			TITLE:		
		TOLERANCES:	CHECKED					
		FRACTIONAL ±	ENG APPR.					
		ANGULAR: MACH ± BEND ±	MFG APPR.					
		TWO PLACE DECIMAL ±	Q.A.			SIZE A DWG. NO. Rotor REV SCALE: 1:1 WEIGHT: SHEET 1 OF 1		
		THREE PLACE DECIMAL ±	COMMENTS:					
		INTERPRET GEOMETRIC TOLERANCING PER:						
		MATERIAL						
		FINISH						
NEXT ASSY	USED ON							
APPLICATION		DO NOT SCALE DRAWING						

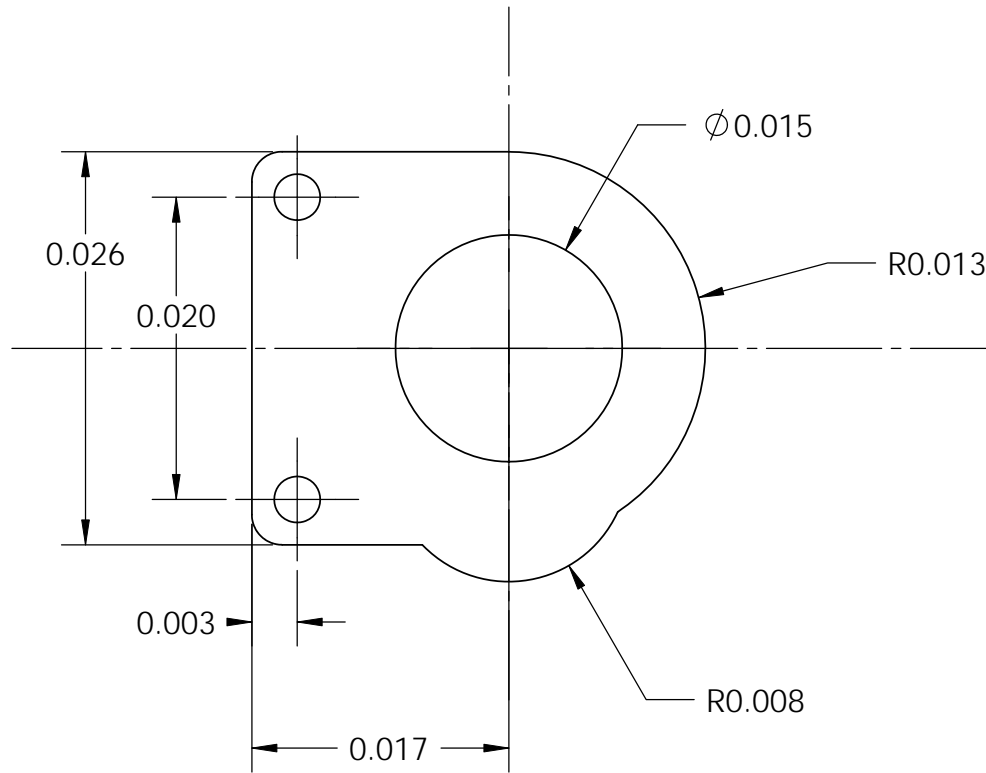
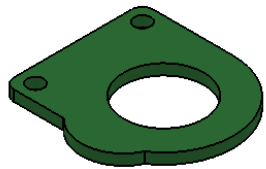
Thickness: 1/16"
 Quantity: 4
 Color: green if available



PROPRIETARY AND CONFIDENTIAL
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

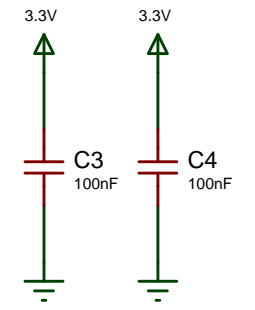
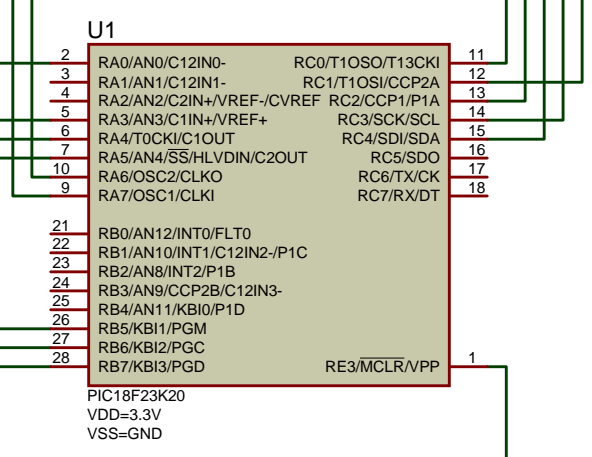
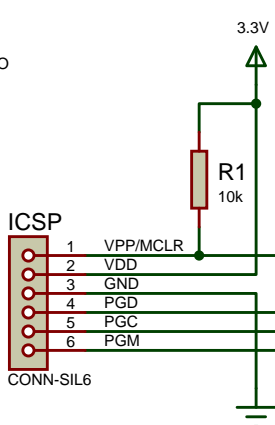
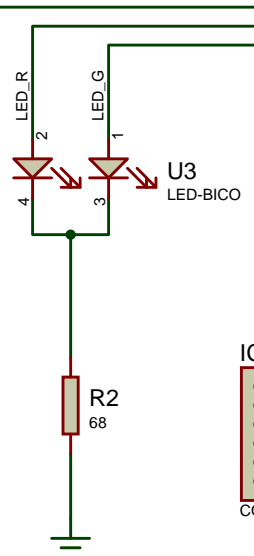
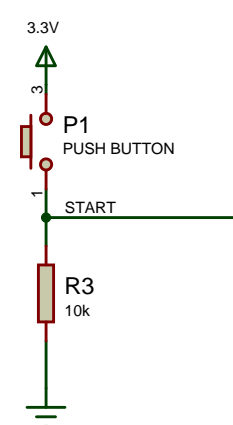
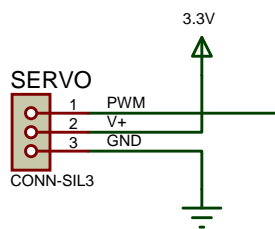
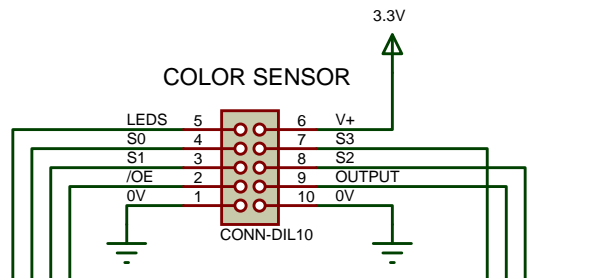
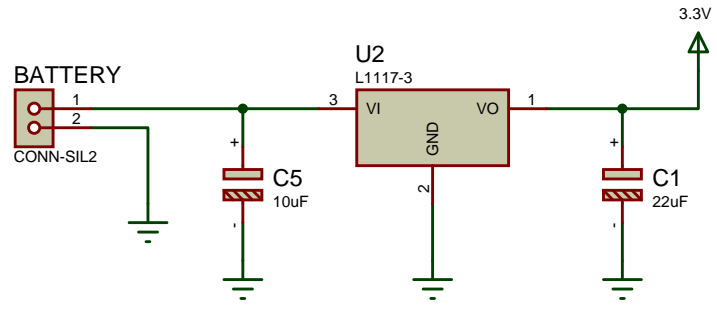
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN INCHES		DRAWN		TITLE:
		TOLERANCES:		CHECKED		
		FRACTIONAL \pm		ENG APPR.		
		ANGULAR: MACH \pm BEND \pm		MFG APPR.		
		TWO PLACE DECIMAL \pm		Q.A.		SIZE DWG. NO. REV A ube Holder H
		THREE PLACE DECIMAL \pm		COMMENTS:		
		INTERPRET GEOMETRIC TOLERANCING PER:				
		MATERIAL				SCALE: 2:1 WEIGHT: SHEET 1 OF 1
		FINISH				
NEXT ASSY	USED ON					
APPLICATION		DO NOT SCALE DRAWING				

Thickness: 1/16"
 Quantity: 2
 Color: green if available



PROPRIETARY AND CONFIDENTIAL
 THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN INCHES		DRAWN		TITLE:
		TOLERANCES:		CHECKED		
		FRACTIONAL ±		ENG APPR.		
		ANGULAR: MACH ± BEND ±		MFG APPR.		
		TWO PLACE DECIMAL ±		Q.A.		SIZE DWG. NO. REV Tube Holder L
		THREE PLACE DECIMAL ±		COMMENTS:		
		INTERPRET GEOMETRIC TOLERANCING PER:				
		MATERIAL				SCALE: 2:1 WEIGHT: SHEET 1 OF 1
		FINISH				
NEXT ASSY	USED ON					
APPLICATION		DO NOT SCALE DRAWING				



FILE NAME:	MM Sorter.DSN	DATE:	12/8/2009
DESIGN TITLE:	M&M Sorter	PAGE:	1 of 1
PATH:	C:\MM Sorter.DSN	BY:	Chris LG
REV:	1	TIME:	2:16:21 AM


```

/*****
*   E155 MicroP's Final Project - M&M's Sorter
*****/
*   FILENAME: M&M's Project.c
*   Version: 2, Updated on 12/06/2009
*
*   DESCRIPTION: This program is able to sort up to 8 different colors of M&M's.
*   For that, it drives a TCS230 color sensor to detect the color of the candy and
*   a servo to sort it once its color have been detected.
*-----
*   Laurent Chris Goudet - Harvey Mudd College
*****/

#include <p18f23k20.h>
#include <timers.h>
#include <delays.h>
#include <capture.h>

/*****
*   I/O Definitions
*****/

// Color Sensor
#define LED          LATAbits.LATA7           // White LEDs
#define S0           LATCbits.LATC1           // Output frequency select
#define S1           LATCbits.LATC3           // Output frequency select
#define S2           LATCbits.LATC2           // Photodiode type select
#define S3           LATAbits.LATA6           // Photodiode type select
#define OE           LATCbits.LATC4           // Enable output (active low)

// Servo
#define PWM          LATAbits.LATA0           // PWM signal output

// Misc.
#define LED_R        LATAbits.LATA4           // Green status LED
#define LED_G        LATAbits.LATA5           // Red status LED
#define START        PORTAbits.RA3           // Start push button

/*****
*   Macros Definitions
*****/

// Friendly names for colors
#define RED          1
#define ORANGE       2
#define YELLOW       3
#define GREEN        4
#define BROWN        5
#define BLUE         6
#define PINK         7
#define PURPLE       8
#define END          9

// Macros used for the color space conversion
#define MAX(a, b)    ((a) > (b)) ? (a) : (b)
#define MIN(a, b)    ((a) < (b)) ? (a) : (b)

/*****
*   Global Variables Definitions
*****/

// Reference values from the white calibration
unsigned int red_ref;
unsigned int green_ref;
unsigned int blue_ref;

// Absolute values of light intensities
unsigned int red_abs;
unsigned int green_abs;
unsigned int blue_abs;

// Normalized values (in range 0 to 1) of light intensities
float red;
float green;
float blue;

// Hue and luminance values from the color space conversion

```

```

float hue;
float luminance;

// Width of the high period of the PWM signal
unsigned volatile int PWM_value;

// Variables used for the green led blinking
unsigned volatile char green_led_on;
unsigned volatile char led_count;
unsigned volatile char stop_blinking;

// Variable used for the Low Voltage Detector
unsigned volatile char lvd_count;

/*****
*   Prototype Definitions
*****/

void initialization(void);           // System initialization
void white_calibration(void);       // White calibration of the color sensor
void selec_color(int color);       // Photodiodes filters selection
unsigned int get_data(void);        // Get data from the color sensor
void convert_RGB_to_HSL(void);     // Color space conversion
char which_color(void);            // Colors' thresholds
void high_isr(void);               // Interrupts catch up function
// Move the servo to a specific angle with a certain speed
void move_to_angle(unsigned char angle, unsigned char speed);

/*****
*
*   interrupt_at_high_vector(void) - Function placed at the interrupt vector and
*   transferring control to the ISR proper.
*
*****/

#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
    _asm GOTO high_isr _endasm
}
#pragma code /* return to the default code section */

/*****
*
*   main(void) - Main routine
*
*****/

void main(void)
{
    unsigned char color;           // The color of the candy
    initialization();              // Initiaization color sensor & PWM Generator
    while(!START);                // Wait for a user input
    white_calibration();           // White calibration of the color sensor
    green_led_on = 1;              // Start main loop => green LED blinking
    while(1){

        move_to_angle(0,0);        // Move servo to initial position
        move_to_angle(30,1);       // Move candy under color sensor
        move_to_angle(20,1);       // Move rotor back to set up the candy

        // RED intensity measure
        selec_color(RED);          // Set photodiodes filters to Red
        red_abs = get_data();       // Measure the ouput frequency of the color sensor
        // The measured value must not be greater than the calibration value
        if(red_abs>red_ref){red_abs=red_ref;}
        // Normalization of the value (in range 0 to 1)
        red = ((float) red_abs) / ((float) red_ref);

        // GREEN intensity measure
        selec_color(GREEN);        // Set photodiodes filters to Green
        green_abs = get_data();     // Measure the ouput frequency of the color sensor
        // The measured value must not be greater than the calibration value
        if(green_abs>green_ref){green_abs=green_ref;}
        // Normalization of the value (in range 0 to 1)
    }
}

```

```

green = ((float) green_abs) / ((float) green_ref);

// BLUE intensity measure
selec_color(BLUE); // Set photodiodes filters to Blue
blue_abs = get_data(); // Measure the output frequency of the color sensor
// The measured value must not be greater than the calibration value
if(blue_abs>blue_ref){blue_abs=blue_ref;}
// Normalization of the value (in range 0 to 1)
blue = ((float) blue_abs) / ((float) blue_ref);

convert_RGB_to_HSL(); // Color space conversion
color = which_color(); // Detect candy's color from thresholds

if(color==END){ // If no more candies => wait for refill
    stop_blinking = 1; // Stop green led blinking
    while(!START); // Wait for start button
    stop_blinking = 0; // Re-start led blinking
}
else{
    // Move the candy to the corresponding channel
    move_to_angle(60+15*color,1);
    // Move back the rotor to push the candy into the vial
    move_to_angle(15*color,1);
    // Move the rotor back to its initial position
    move_to_angle(0,0);
}

} // while(1)
} // main()

/*****
*
* void initialization(void) - Initialization of the clock, the I/Os, the color
* sensor, the green led blinking, the Low Voltage Detector and the PWM generator
*
*****/
void initialization(void)
{
    // Clock initialization
    OSCCONbits.IRCF2 = 1; // Set internal clock to 16 Mhz

    // PORTA Configuration
    PORTA = 0;
    LATA = 0;
    // => Configure all inputs as digital I/Os
    ANSEL = 0xE0;
    // => PORTA directions configuration
    // ==> RA0: PWM (output)
    // ==> RA3: Start button (input)
    // ==> RA4: Red LED (output)
    // ==> RA5: Green LED (output)
    // ==> RA6: S3 (output)
    // ==> RA7: White LEDs (output)
    TRISA = 0x0E;

    // PORTC Configuration
    PORTC = 0x00;
    // => PORTC directions configuration
    // ==> RC0: Output color sensor (input)
    // ==> RC1: S0 (output)
    // ==> RC2: S2 (output)
    // ==> RC3: S1 (output)
    // ==> RC4: /OE (output)
    TRISC = 0xE1;

    // Color sensor initialization
    S0 = 1; S1 = 1; // Output scaling set at full speed
    OE = 0; // Enable color sensor output

    // Green led blinking initialization
    green_led_on = 0; // Green led off
    led_count = 0; // Reset green led blinking counter
    stop_blinking = 0; // Green led blinking off

    // High / Low Voltage Dectector (HLVD) intialization
    // => Bit 7: VDIRMAG = 0 => Event occurs when voltage falls below trip point
    // => Bit 4: HLVDEN = 1 => HLVD enabled

```

```

// => Bit 3-0: HLVDL<3:0> = 1000 => Voltage Detection Limit set to 2.6V
HLVDCON = 0x15;           // Configure HLVD module
lvd_count = 0;           // Reset detections counter
PIR2bits.HLVDIF = 0;     // Reset interrupt flag

// PWM Generator initialization
PWM = 0;                 // Set PWM output in low state

// => Timer 0 is fixed and generate an interruption every 20ms
OpenTimer0( TIMER_INT_ON &
            T0_16BIT &
            T0_SOURCE_INT &
            T0_PS_1_2 );

// => Timer 1 stand for the width of the modulation
OpenTimer3( TIMER_INT_ON &
            T3_16BIT_RW &
            T3_SOURCE_INT &
            T3_PS_1_1 &
            T3_SYNC_EXT_OFF );

// => Timer 0 value: 20ms
// Fosc / 4 = 4 Mhz
// With 1:2 prescaler => 2 Mhz => T = 0.5µs
// 20ms / 0.5µs = 40,000 = 0x9C40
WriteTimer0(0xFFFF - 0x9C40);

// => PWM_value (width of the high period)
// PWM_value = 2000 => High period = 0.5ms => Servo angle = 0°
// PWM_value = 10000 => High period = 2.5ms => Servo angle = 180°
PWM_value = 6000;

// => Clear interrupt flags
INTCONbits.TMR0IF = 0;   // Timer 0 interrupt flag
PIR2bits.TMR3IF = 0;    // Timer 3 interrupt flag

// => Enable interrupts
INTCONbits.PEIE = 1;    // Enable peripheral interrupts
INTCONbits.GIE = 1;    // General interrupt enable switch
}

/*****
 *
 * void white_calibration(void) - Proceed to the white calibration of the color
 * sensor => measure the light intensity reflected by a white surface for each
 * color filters (red, green & blue)
 *
 *****/
void white_calibration(void)
{
    // RED
    selec_color(RED);    // Set photodiodes filters to RED
    red_ref = get_data(); // Get a reference value (maximum red light intensity)
    // GREEN
    selec_color(GREEN);  // Set photodiodes filters to GREEN
    green_ref = get_data(); // Get a reference value (maximum green light intensity)
    // BLUE
    selec_color(BLUE);   // Set photodiodes filters to BLUE
    blue_ref = get_data(); // Get a reference value (maximum blue light intensity)
}

/*****
 *
 * void selec_color(int color) - Select the color of the filters in front of the
 * photodiodes
 *
 *****/
void selec_color(int color)
{
    if(color == RED){ S2 = 0; S3 = 0; } // Red filters
    else if(color == GREEN){ S2 = 1; S3 = 1; } // Green filters
    else if(color == BLUE){ S2 = 0; S3 = 1; } // Blue filters
    Delay10TCYx(2); // 1µs necessary delay for updating filters color
}

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

```

```

* unsigned int get_data(void) - Frequency meter which measure the output
* frequency of the color sensor (input = RC1)
*
*****/
unsigned int get_data(void)
{
    unsigned int value;           // The measured value
    LED = 1;                      // Power up the white LEDs
    Delay10KTCYx(10);            // 5ms power-up delay for the LEDs
    OpenTimer1( TIMER_INT_OFF &
                T1_16BIT_RW &
                T1_SOURCE_EXT &
                T1_PS_1_1 &
                T1_OSC1EN_OFF &
                T1_SYNC_EXT_OFF); // Start timer 1 for counting pulses on RC1
    Delay10KTCYx(30);            // Wait 15ms to get a compliant number
    value = ReadTimer1();         // Read the value from the timer 1
    CloseTimer1();               // Stop the timer 1
    LED = 0;                      // Turn off the white LEDs
    return value;                 // Return the measured value
}

/*****
*
* void convert_RGB_to_HSL(void) - Convert RGB (in range 0 to 1) to HSL
* (in range 0 to 1)
*
*****/
void convert_RGB_to_HSL(void)
{
    // This algorithm is based on the numerous algorithms available on Internet
    // regarding RGB to HSL space conversion and have been adapted to the
    // requirements of this program
    float fmax, fmin;
    fmax = MAX(MAX(red, green), blue);
    fmin = MIN(MIN(red, green), blue);
    luminance = fmax;
    if (fmax == red){
        hue = (green - blue) / (fmax - fmin);
    }
    else if (fmax == green){
        hue = 2 + (blue - red) / (fmax - fmin);
    }
    else{
        hue = 4 + (red - green) / (fmax - fmin);
    }
    hue = hue * 60; // The ouput is the hue circle (in range 0 to 360)
    if (hue < 0) {hue += 360;}
}

/*****
*
* char which_color(void) - Determine what is the color of the M&M's regarding
* the hue and the luminance
*
*****/
char which_color(void)
{
    // These thresholds values have been determined from the hue circle and
    // adjusted by some experimentations
    int color;
    if(luminance < 0.023) color = END; // No candy
    else if(luminance < 0.25) color = BROWN;
    else if(hue >= 8 && hue < 28) color = ORANGE;
    else if(hue >= 28 && hue < 84) color = YELLOW;
    else if(hue >= 84 && hue < 172) color = GREEN;
    else if(hue >= 172 && hue < 252) color = BLUE;
    else if(hue >= 252 && hue < 318) color = PURPLE;
    else if(hue > 320 && hue < 350) color = PINK;
    else color = RED;
    return color;
}

/*****
*
* void move_to_angle(unsigned char angle, unsigned char speed) - This function
* allow the servo to be moved to a specific angle with an adjustable speed

```

```

*
*****/
void move_to_angle(unsigned char angle, unsigned char speed)
{
    unsigned int new_angle;
    // PWM_value = 2000 => High period = 0.5ms => Angle = 0°
    // PWM_value = 10000 => High period = 2.5ms => Angle = 180°
    // The new angle is being calculated based on these extremum
    new_angle = 2000 + ((unsigned int)angle * 45);
    // The current PWM_value is incremented until it reaches the new angle value
    while(PWM_value != new_angle){
        if(PWM_value < new_angle){
            PWM_value ++;          // +1 step
        }
        else{
            PWM_value --;          // -1 step
        }
        if(speed!=0){              // If speed = 0 => full speed
            Delay1KTCYx(speed);    // Else wait for 'speed' x 250µs
        }
    }
    Delay10KTCYx(254); // Then wait for that the servo reaches its final position
}

/*****
*
*   high_isr(void) - Interrupts sub-routine. Process Timer 0, Timer 1 and ADC
*   interrupts
*
*****/

#pragma interrupt high_isr
void high_isr(void){
    // Timer 0 interrupt routine
    if (INTCONbits.TMR0IF){
        // Green LED blinking
        // The timer 3 interrupt occurs every 20ms so the state of the green LED
        // is changing (when blinking is on) every 25 * 20 ms = 0.5s
        led_count++;
        if(led_count >= 25){
            if(PIR2bits.HLVDIF){          // If a Low Voltage have been detected
                LED_G = 0;                // => The green LED is turn off
                LED_R = ~ LED_R;          // And the red LED is blinking
                if(lvd_count < 10){       // If 10 detections => stop detections
                    lvd_count ++;        // Increment detection counter
                    PIR2bits.HLVDIF = 0; // The LVD interupt flag is reset
                }
            }
            else if(green_led_on){        // If the green led is on
                LED_R = 0;                // The red LED is turn off
                if(stop_blinking){
                    LED_G = 1;            // If blinking off => always on
                }
                else{
                    LED_G = ~ LED_G;      // If not => the green LED is blinking
                }
            }
            else{
                LED_G = 0;                // If green led off => output to low
                LED_R = 1;                // The red led is turn on by default
            }
            led_count = 0;                // Then the counter is reset
        }
        // PWM Generator
        // Timer 0 interrupt => begining of a new period
        PWM = 1;                          // PWM output in high state
        WriteTimer3(0xFFFF - PWM_value); // Set high state period
        WriteTimer0(0x63BF);              // Reload Timer 0 value (20ms)
        T3CONbits.TMR3ON = 1;            // Turn on Timer 3 (high period timer)
        INTCONbits.TMR0IF = 0;           // Reset Timer 0 interrupt flag
    }
    // Timer 3 interrupt routine
    else{
        PWM = 0;                          // PWM output in low state
        T3CONbits.TMR3ON = 0;            // Stop Timer 3 (high period timer)
        PIR2bits.TMR3IF = 0;            // Reset Timer 3 interrupt flag
    }
}

```

C:\Users\Laurent\Desktop\MicroP's final project\No More Blue M&Ms!\Software\M&M's Project.c

}