Coding:

Full code can be seen in the Appendix. Below is a description of each section.

Initialization/Setup

This code only runs once. In this section pins are assigned to input/output, libraries to run the stepper motor and servo motor are loaded, and both the stepper and servo objects are created and initialized.

A boolean variable SetUp is set to = false.

Also in this section seven arrays are created, six of which will serve to store the reference values for the six m&m colors, and one of which will be reused for each m&m test. Each array holds three photoresistor readings and a tally for how many m&ms that array has identified.

Functions

Brief overview of each function and when it is used.

MovetoLimit()

This function moves the stepper motor clockwise until the limit switch is hit. It also uses a basic for loop to time how long the stepper has been moving. If too much time has passed, that means the top disk has gotten jammed, in which case the function will briefly reverse the stepper direction and then continue clockwise. This does an excellent job of catching mechanical jams. Function is called by *Sort()* and *SetupSort()*.

setLED()

Function sets the RGB LED to a set of three PWM values. Function is called by *CollectColor()*.

CollectColor(int samples, int ColorArray[])

Function takes a color array and int *samples* as arguments. When called, function flashes the white LED and takes a photoresistor value to determine if the chamber is loaded. If so, the function shines red, blue, and green light from the RGB LED, takes corresponding photoresistor values, and stores them in the array. With *samples* > 1, each photoresistor value is an average of (*samples*) readings. This feature may not have a significant effect on accuracy. If no m&m is detected, Array is set to zeros. Function is called by *CollectReferences()*.

Sort(int ID)

Function controls servo motor and stepper motor to dispense m&m into one of compartments 1 through 6, depending on ID. If fed zero, stepper continues clockwise to load another m&m.

SetupSort(int ID)
Has the exact same function as *Sort()* but with slightly different rotation values calibrated for the expectation that there is only one m&m in the loading site. Only called during *CollectReferences()*

CollectReferences(int Color1[] … int Color6[])
Function takes all reference arrays and, using *SetupSort()* calls *CollectColor()* to collect initial references for the six colors, based off of the first six m&ms it sees.

FindMatch(int TestColor[], int Color1[] … int Color6[])
Function takes the test array and compares it to the reference values in the six reference arrays using a least squares approach. For the closest array, the function returns an int with the corresponding ID, if no m&m was detected (ie. TestArray = [0,0,0,..]), function returns zero. *FindMatch* also calls *UpdateReference()* with the ID of the identified color.

UpdateReference(TestArray[], ColorArray[])
Function takes the test array and the color reference of the identified color. Then, using the tally for that color (the fourth value of the reference array) it updates the reference values for that color to be an average of all identified m&ms of that color, including the one just identified (TestArray[]). This feature was added to correct for some occasional misidentifications and has increased robustness dramatically. Function is called by *FindMatch()*.

EndScript(int match, int VoidTally, int TestArray[], int Color1[] … int Color6[])
This function keeps track using the variable VoidTally of how many consecutive times the machine has failed to load an m&m into the chamber. If that number reaches 10, it is assumed that the m&m reservoir is empty and the function prints the statistics for the run and ends the program using exit(0);

Main Loop
The main loop starts by checking if boolean SetUp == false. If so, it calls *CollectReferences()* to collect initial reference reference values and sets SetUp = true.

With an m&m now loaded, the loop calls *CollectColor()* to sense the loaded m&m.
Then, *FindMatch()* identifies the m&m and updates the associated reference values.
Using *Sort()* the program sorts the m&m and simultaneously loads another.

The identification ID is fed to *EndScript()* so that if the chamber has been empty ten times in a row, the program will terminate.