Super Speedy Serial Skittle Sorter (5S)

Submitted by
Nickolas David King

Electrical Engineering

To
The Honors College
Oakland University

In partial fulfillment of the
requirement to graduate from
The Honors College

Mentor: Brian Dean, Professor of Engineering
Department of Engineering
Oakland University

(15 May, 2014)

Abstract

The Super Speedy Serial Skittle Sorter, or 5S for short, is a fully automated sorting machine made up of simple parts that are easily replaceable. The ideas and mechanics of large corporation sorting machines from around the world are incorporated into a small and efficient tabletop machine. By putting a Skittle into the hopper, the machine will automatically start and begin sorting Skittles. One large disk will move the Skittles along to the designated hole, already pre-determined, in a circular direction to trap doors that will lead to designated bins. This document will describe the details on how the 5S will operate and design considerations that were explored.

It was found, through the design of the system, that a microprocessor would be used to read the Skittles. Using a photoresistor or using an RGB sensor was also considered, as it could be tuned to a Skittles color and instantly actuate a sorting mechanism. This option was, however, more expensive and complicated than using a factory color sensor, the Taos TCS 3200 TCS 230. By using this sensor in conjunction with the microprocessor, sorting Skittles would be quick and efficient.

The 5S was found to be a very good implementation of a sorting machine, with many areas that could be improved with further research. This document explains the initial version of the machine and the success that was found using it.

Table of Contents

Abstract……2

Introduction……5

Conclusion/Recommendations…… 6

Computer Overview……...7

Original Considerations……7

Program Flowchart……8

Color Sensors……9

Start and Stop of Machine……10

Motor Control……11

Delays……11

Electrical Overview……12

Hopper Sensor……13

Color Sensor……15

Processor……17

Counter Sensors……18

Displays……19

Sliding Door Servos……21

Motor……21

Mechanical and Machine Design Considerations……22

Hopper Designs……22

Plinko……25

Piston……26

Table of Contents (cont.)

Conveyor Belt……27

Stress Analysis……30

Results and Discussion……32

References……37

Appendix……38

Introduction

The 5S was built as part of a senior design competition. Ten teams consisting of electrical, mechanical and computer engineers were tasked to build an automatic Skittle sorting machine. The machine was to turn on when Skittles were dropped in, sort, count, and display the time until the machine sorted all Skittles, shutting off by itself when this was accomplished. Each group was instructed to build the machine keeping cost and speed at the forefront of design consideration. The competition would take time, cost, and accuracy into each team's final score. Mistakes, such as system jamming, and incorrectly sorted skittles would add to a team's score, which would then be multiplied by the final build cost, with the lowest scoring team winning.

The 5S focused highly on cost and how quickly a Skittle could be sorted. Knowing a low cost machine would greatly benefit in the end, all design constraints were made with this in mind. Mechanically, the majority of the machine was built with wood, using metal to fashion doors for the skittles to fall through and finding the most cost-efficient servos to open these metal doors. Electrically, low cost power supplies, color sensors, self-built timers, and programming board were used.

This document will describe in detail the steps taken to create the 5S. Different designs will be discussed with the steps taken to create the 5S, as the design was completely changed and modified halfway through the semester. Problems overcame, how Skittle colors were differentiated, how Skittles were electronically counted and the total time it takes for the machine to operate is also discussed in detail.

Conclusion/Recommendations

Ten different teams built an automatic Skittle sorter, all attempting to build the quickest and cheapest Skittle sorter. The 5S was the most successful implementation of a Skittle sorting machine. Cost was kept at a minimum and speed was found to be the fastest out of all machines built. With this affirmation of original design considerations, future buildings of this machine would also generate the greatest success. At time of presentation, the machine was found to be 80 percent accurate, with the ability to improve with the allotment of more time.

The 5S could have been improved through continued laboratory experiments, as the two biggest problems were differentiating between three of the seven colors as well as jamming of the system. By putting the 5S in a stable and non-changing environment, the sensors could be calibrated much more accurately. The next problem that would be solved would be the jamming issue. The height the Skittles were fed from the copper tubes needs to be exact, as differing heights could lead to Skittles becoming jammed in the system. Along with this, the height of each sensor would be kept constant so that there would be no jamming underneath the sensor as the Skittle was read. The system could also have been improved by creating the same height between each door. As a Skittle traveled between each door, it dropped down onto wood and then back up onto a metal door. This allowed the Skittle to roll as the disk spun, not keeping the Skittle “flat.” This could lead to jamming and incorrect color sorting, as the Skittle could not be read in the most desirable location.

Computer Overview

Original Considerations

At the start of the project, an algorithm was designed that would be simple, yet efficient at sorting Skittles. Research into color sensors and the Arduino programming language and interface spearheaded the project. The Arduino code was written to be simple, aiming to be without redundancy.

Two microcontrollers were compared for the task presented; the DRAGON12-Plus and Arduino Mega. A microcontroller was needed that would not sacrifice speed at a lower cost. The DRAGON12-Plus board was found to have a faster clock with more input and output pins, yet was four times the cost of the Arduino Mega. The Arduino Mega was found to have adequate speed with enough input and output pins to satisfy the proposed design. Since its release, the Arduino Mega has seen two revisions, leading to widespread support and many companies providing comparable boards at a lower cost.

The Arduino Programming platform was chosen for the simplicity and ability to handle large, complex systems. The Arduino platform is an all-inclusive environment where one can program, design, and build, using only systems designed for Arduino boards. The Integrated Development Environment (IDE) for Arduino, along with library plugins allowed for easy programming and quick troubleshooting. The Arduino environment has many plugins available, i.e. Eclipse, Visual Studio, and many other platforms, allowing developers to use the most comfortable IDE. The Arduino programming language is simple to follow, with much of the hard coding done in the background, implemented in libraries referred to at the beginning of the code. With the hard coding running in the background, the necessary code was easy to follow and explain. Arduino IDE allowed on-the-fly changes and was easy to learn, rather than using C or C++. Both these programming languages would have required more research into the design and communication of the system.

An Arduino Uno and Arduino Due were initially considered to carry out the code for the 5S. However, after testing and building of the 5S, more input and output pins were necessary to drive all the required components. The Arduino Mega has 70 pins available for use, 54 digital pins and 16 analog pins. Initially, amount of interrupt pins was also considered, as the Arduino Mega has six. However, after careful consideration, interrupts were not used and conditional statements instead implemented. Color sensing was considered at the digital pins, as 54 pins could be used and act as both input and output.

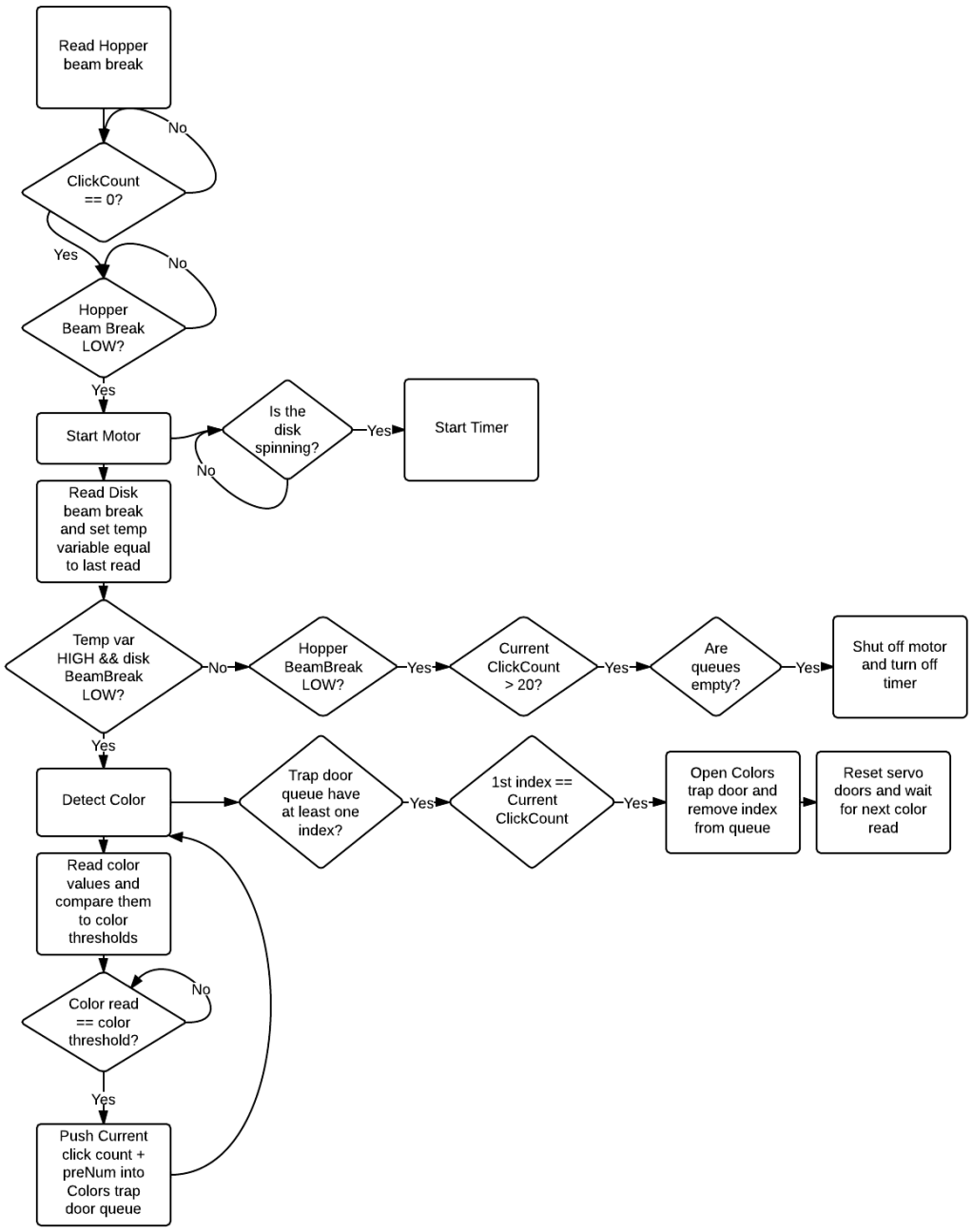


Figure 1 Program Flowchart

Figure 1 gives a general overview of how the 5S operates. If the disk has not started spinning, the Arduino code recognizes that the current click count, which represents when the disk rotates one hole, is zero. A beam break is located inside the hopper, which is set to be continuously HIGH until a Skittle falls in front of it, causing it to send a LOW signal to the Arduino, which will continue as long as a Skittle is detected in front of the sensor. When this LOW signal occurs, the motor will begin to spin and the timer will begin. The beam break on the disk consists of a photoresistor and a diode that are positioned across from each other. When the disk is in a spot for a Skittle to fall into a bin, the photoresistor and diode are lined up, sending a signal to the Arduino and the detect color method is recognized in the code. At any other time, the photoresistor and diode are blocked by the disk and do not line up. If the hopper beam break goes HIGH, there are no longer any Skittles in the hopper, which triggers the shut off conditional statements. If all of the trap door queues are empty, the motor and timer are both shut off.

Color Sensors

Many color sensors were compared when choosing how to differentiate Skittles. A color sensor using a red, green, and blue LED shined at a photoresistor was initially considered. However, after many tests, this option was deemed not fast enough to read the Skittles going through the 5S. The Taos TCS 3200 TCS 230 was next explored; a light to frequency color sensor. The Taos detected red, green, blue, and white values, giving a greater accuracy at separating colors. Documentation on the Taos was available for many versions of the sensor, allowing for ease of implementation into the 5S. Before the Taos was chosen, the HDJD-S822 was also explored, a light to voltage sensor. However, the advantages to this sensor were no different than using the Taos. Because of this, the Taos was chosen, as it was one-fourth the price.

The Taos sensor has the ability to read one Skittle at a time, preventing other colors from being detected. The Taos sensor reads the values of white, red, blue, and green for each Skittle as the Skittle goes by. The Taos sensor uses two pins, S2 and S3, for filtering between the four colors. By setting both pins LOW, the sensor filters for red; setting S2 LOW and S3 HIGH will filter for blue; setting S2 HIGH and S3 LOW filters for white; setting both pins HIGH will filter for green. The Taos color sensor also allows for output frequency scaling, achieved by another two pins, S0 and S1. Again, four ranges

are observed; by setting both pins LOW, the sensor output frequency powers down; setting S0 LOW and S1 HIGH, the sensor output frequency is set to 2%; setting S0 HIGH and S1 LOW, the sensor output frequency is set to 20%; setting both pins HIGH, the sensor output frequency is set to 100%. To achieve the best color readings, the sensor was set at an output frequency of 2%.

Queue Algorithm

The design of the 5S was to be as fast and accurate as possible. Many ideas were considered, with a continuous rotating disk chosen as the desired design. Holes were cut along the edge of the disk, intended for the Skittles, along with holes further in on the disk to supplement a counting system. A beam break was used to observe the difference between an open hole and the solidity of the disc. Each time a hole lined up, the system was able to monitor the position of the disk and send this information to the Arduino. This information needed to be relayed to the servo doors, telling the servo doors when to open after a specific criteria was met. After brainstorming, a queue system was implemented for the 5S. The Taos TCS 3200 TCS 230 would read the color of the Skittle as it passes by, storing this in a designated color queue. Each designated color queue would be assigned a number of “clicks” that would be added to the current click count. When the Skittle got to the designated servo door, the code would check to see if the number stored in the queue is equal to the current click count. If the Skittle was in the right position, the servo door would open and the Skittle would be removed from the index of the queue. To ensure the quickest sorting time, the only delays present in the code were observed for the servo doors to open.

Start and Stop of Machine

A beam break, consisting of an LED and photoresistor, located inside the hopper detects when Skittles are poured into it. A conditional statement inside the loop function of the code detects when Skittles pass by the photoresistor. If a Skittle passes by the photoresistor and the current click count is zero, the 5S will start spinning the disk.

The 5S sorts colors by reading a color threshold and storing the current click count plus a color’s distance away from a trap door, into a queue. By utilizing the same beam break inside the hopper, the microcontroller is constantly reading it to see if it ever goes LOW. If the beam break inside the hopper goes LOW, the code recognizes that the hopper is empty and starts checking queues. The 5S will shut

off when the queues no longer have any values in their indices. Using this method allows the 5S to guarantee that all of the Skittles have been sorted and counted before the disk and timer stops.

Motor Control

The motor was controlled using PWM (pulse width modulation). The PWM signal and the analog read/write functions that are built into the Arduino library were explored. AnalogWrite was used to control the speed of the motor, adjusting the speed between analog value 0 and 255, which would set the disk rotation to not moving or rotating as quickly as the motor could spin. To calculate the amount of Skittles that could be sorted in one second, the disk was marked and timed to see how long it took for one revolution. The optimal speed the 5S can sort Skittles was found to be one revolution in nine seconds, which was found by setting the motor speed to an analog value of 133.

Delays

The only delay needed to guarantee that the machine worked correctly was a servo delay. This was calculated to incorporate the speed of the disk, as the opening and closing of each door contributed to the total time the door operated. This delay was found to be 160 milliseconds, 80 milliseconds dedicated to opening the servo and the subsequent to closing.

Electrical Overview

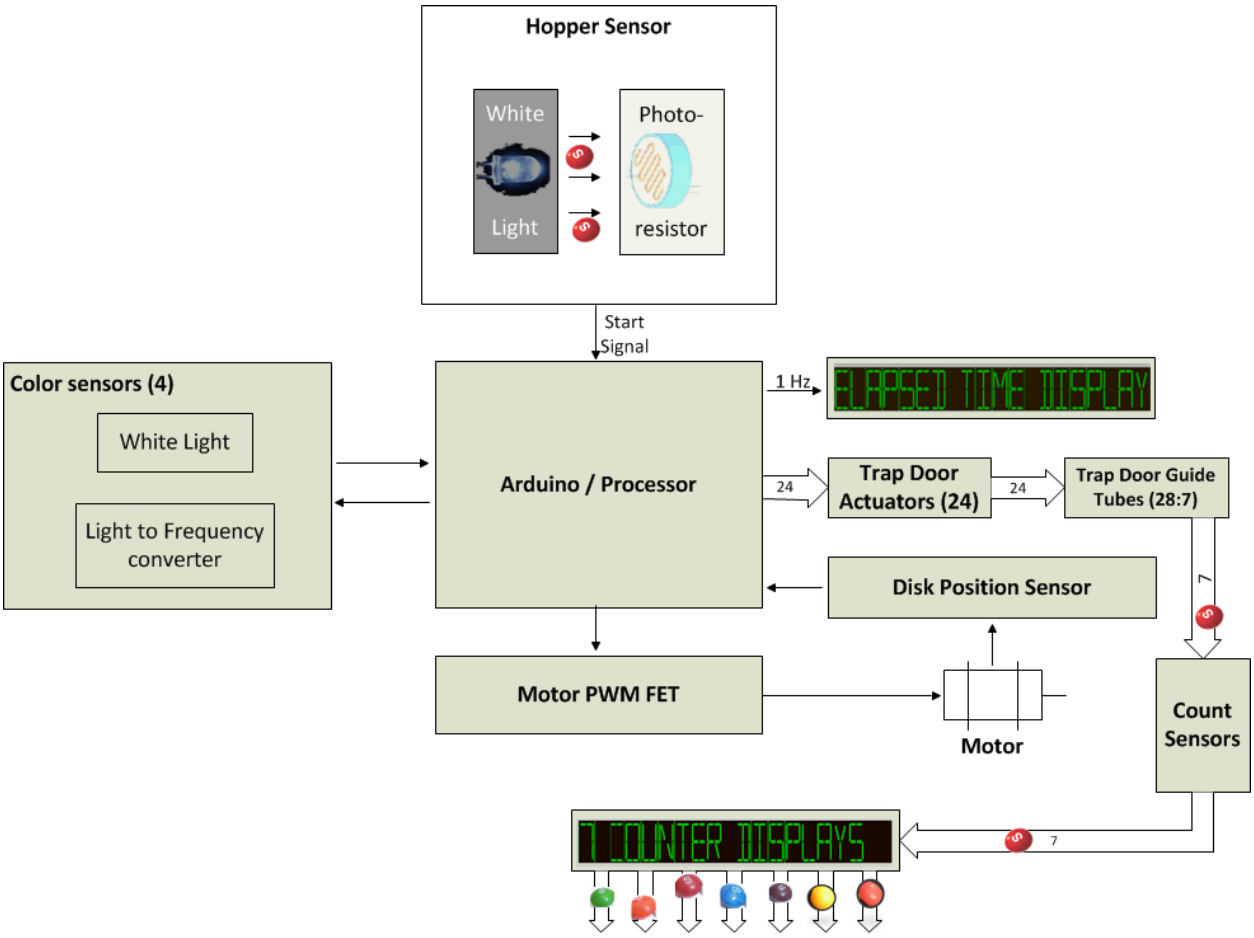


Figure 2 Electrical System Overview

A 12V start signal from the hopper is level-shifted to 5V, as the programming board only accepts 3.3V or 5V input, and fed to the processor (power supplies are omitted from the diagram for simplicity). The processor in turn sends a PWM signal to a FET to start the motor and begins sending a 1 Hz pulse to the elapsed time chronometer. The 1 Hz pulse is level-shifted to 12V in order to keep in line with all other signals being sent to the Arduino.

The spinning disk is divided into four sections with one color sensor, six sliding doors, and one hole without a door for each section. The loop is closed by a disk position sensor, or 'click counter' that uses the same type of LED light and photoresistor (PR) as the count and hopper sensors. Upon the first pulse (click) from the disk position sensor and every subsequent click thereafter, the processor will select and read all four of the color intensities available from each color sensor, parse the data and return a

color for each sensor. Thus, the four sections are processed simultaneously, i.e. in parallel. This information is tracked using a queue array system in which the sensed Skittle's sliding door is actuated as it approaches the door assigned to its color.

A system of clear tubing and PVC Tee's guides the Skittle through the light sensor for its color and into its bin. The sensor signal is fed to an independent counter and display circuit which can be easily read from 10 to 20 feet away.

Hopper Sensor

The goal for the hopper sensor is to sense anything placed in the hopper from a single candy to its full four-pound capacity. The hopper being divided into four sections, one for each feeder tube, necessitates the use of four sensors that are diode-orred to provide one input signal to the processor to start the machine.

The sensors are composed each of an LED shining directly onto a photoresistor forming a voltage divider with a pull-up resistor (Figure 3). When nothing obstructs the light, the PR has a value on the order of 100 ohms, providing a voltage of nearly zero on the non-inverting input and thus a low output. When the light is obstructed by a Skittle, the resistance of the PR increases to 100K over 47ms (see Figure 9). The non-inverting input exceeds the threshold voltage set by the potentiometer and the open collector output goes to the voltage set by its pull-up resistor, in this case the +5V start signal required by the microprocessor.

The output diodes provide isolation between the comparator outputs. The purpose of the series 100-ohm resistors is to drop any small voltage differences between the cathodes.

Power is provided by the same 12V power supply that supplies the motor. Current requirements are 10 mA for each LED, 5 mA for the PRs, and 5 mA sink current provided by the comparators for a total of 50 mA.

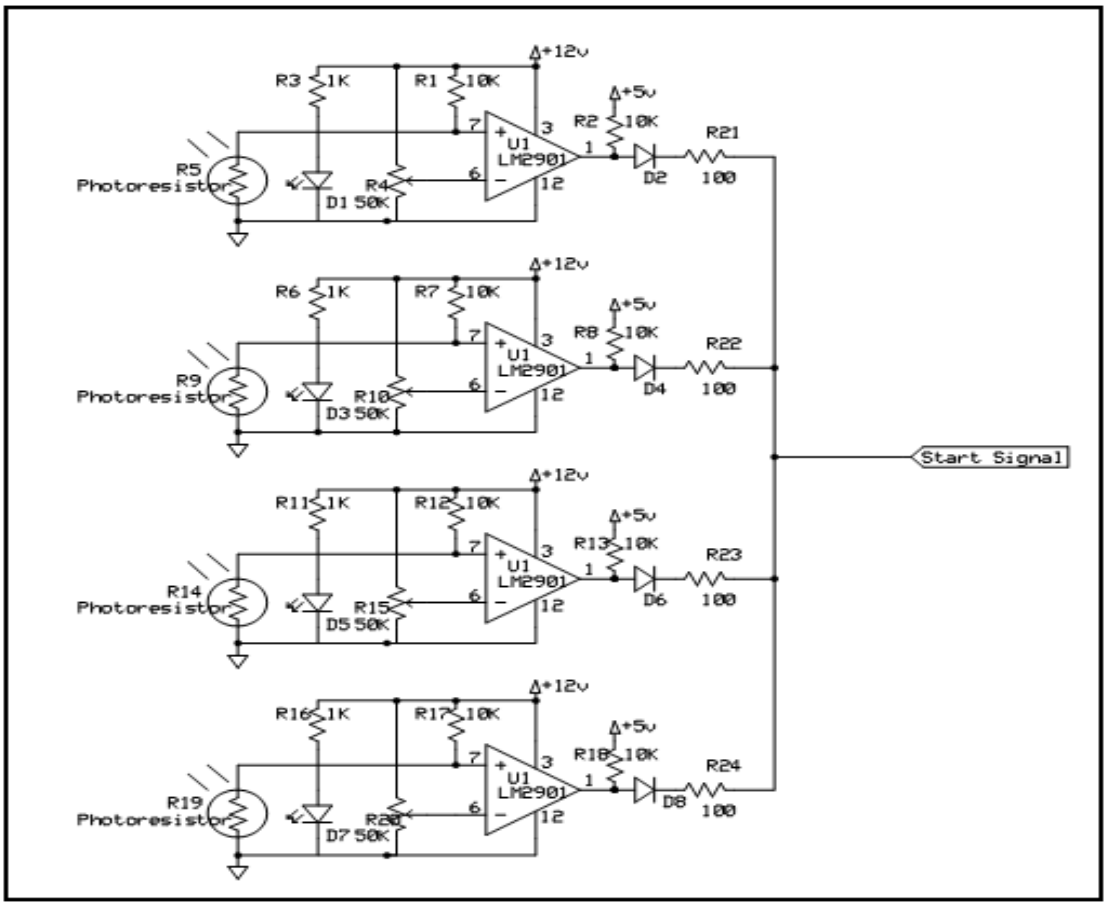


Figure 3 Hopper Sensor Schematic

Color Sensor

The original concept for color sensing was to use the same type of pulled up photoresistors used for the hopper and counters to provide a single sense voltage to either the Arduino processor or a comparator. Hopefully, the PRs could be coaxed into distinct responses for each color under controlled conditions. This offers an inexpensive alternative to light/frequency and light/voltage converters. The idea was to measure the wavelength for each Skittle color and use matching color gels to filter out the light of different colors, thus providing the lowest voltage for the Skittle that matched the gel. The idea of using comparators to immediately trigger an actuator offers the potential savings of a microprocessor-free design.

Figure 4 Testing PR's

Table 1 LED & Color Effects on PhotoSensors

LED & Color Effects on PhotoSensors									
Sensor PN	Color Temp (K)	Intensity (mA)	Empty (KKΩ)	Red (KΩ)	Orange (KΩ)	Yellow (KΩ)	Green(KΩ)	Violet (KΩ)	Std. Dev.
PDV-P8103 & Roscolux #342	5500	14.4	20	13.2	9.5	4.8	12.4	17	4.55
		0.767	45	33	28	23	21	20	5.43
		4.59	9.6	6.5	9.3	7.4	8.8	8.7	1.15
PDV-P8103 & No filter	5500	14.4	2.15	1.78	1.42	2.68	2.2	2.3	0.49
		0.767	19.7	19.8	18.3	14.7	19.8	21	2.44
		4.59	6.75	4.3	4	3.35	4.1	4.8	0.52

Experiments shown in Table 1 were conducted using the color filters over the light and again using filters over the photoresistors. Using Ocean Optics SpectraSuite software to measuring the spectrum of several white LEDs, it was found that a temperature of 4000K yielded the widest and flattest response over the visible range. The same apparatus was used previously to determine Skittle wavelength.

Controlling and varying light intensity, light-to-Skittle distance, PR-to-Skittle distance, and ambient light intrusion, the data for different colors showed that each color could have a distinct range of response voltages. Promising, yes, however, the color vs. voltage distributions overlapped with other colors to an unworkable degree.

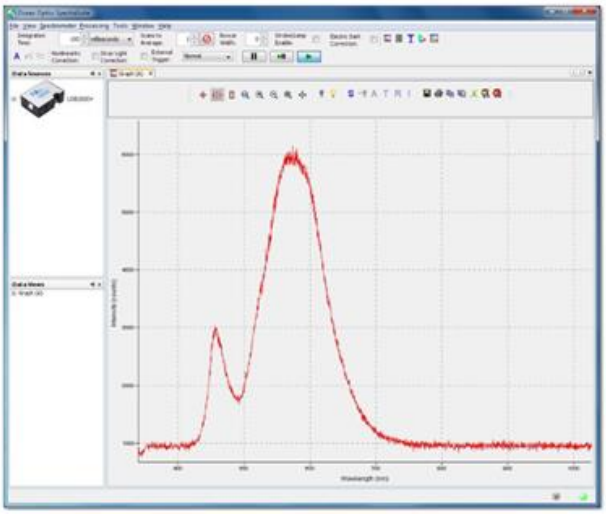


Figure 5 Wavelength Approximation Using SpectraSuite

Skittle(R) Wavelength by Color (nm)					
	Red	Orange	Yellow	Green	Violet
Center	645	605	565	550	no read
Range	625-660	585-630	540-610	530-570	

Similar experiments were carried out using the ams TCS3103 RGB-light to voltage converter. This device outputs individual red, green and blue voltages in response to light input.

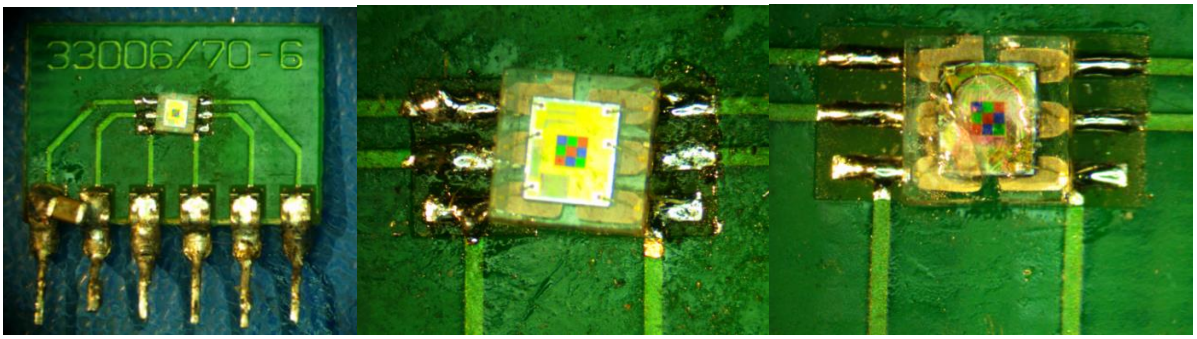


Figure 6 ams TCS3103

Figure 6 shows the ams TCS3103 sensor mounted to a surfboard, close up, and damaged during an experiment.

The datasheet suggests that it is capable of outputting a range of voltages spanning nearly the range from Vcc to ground. The most that was measured for red, green or blue outputs, however was 1.1V. The cost of the sensor, a Surfboard®, and possibly instrumentation amplifiers as well as the difficulty in soldering and otherwise working with these sensors (ambiguous datasheets, for instance) dimmed the attractiveness of this option. This option was finally decided against as time grew short and when it was found that pre-assembled light/frequency converters were reliably reading colors with off-the-shelf Arduino code.

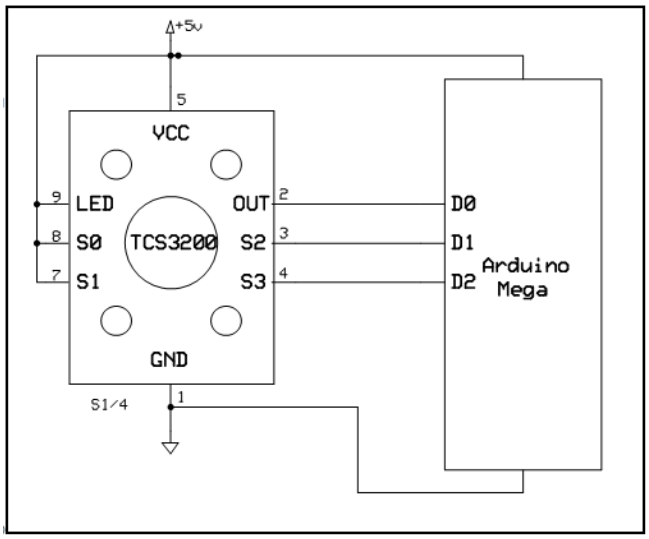


Figure 7 Taos TCS 3200 TCS 230 Connection to Arduino

The Taos TCS 3200 TCS 230 color sensor circuit board uses an ams AG color sensing chip that converts light intensity to frequency. The board provides white LED light, interface headers and power supply filtering.

The sensor uses an array of 64 photodiodes, 16 each of red, green, blue and clear. Inputs S2 and S3 are used to select which color filter is active. S0 and S1 are used for frequency scaling. The 16 MHz Arduino Mega can easily accommodate the max output frequency of 600 KHz, thus S0 is tied LOW and S1 tied HIGH to ensure the best color reads.

Processor

Servo doors were actuated on 24 of the Arduino Mega's 54 digital outputs. One provides a 1 Hz clock to drive the timer display, one is for the start signal, 9 for color sensor I/O, one for motor control, and one for the click sensor accounting for a grand total of 37 digital I/O's used.

Power is provided by the same 12V power supply that drives the motor. The onboard regulators provide 5V and 3.3V power for the microprocessor. Each I/O can source/sink 40 mA. Worst case current consumption is therefore 1.2 A. The processor will not, however, drive the servos directly, but apply < 1mA signal to the devices. Total current consumption is then less than 100 mA.

Counter Sensors

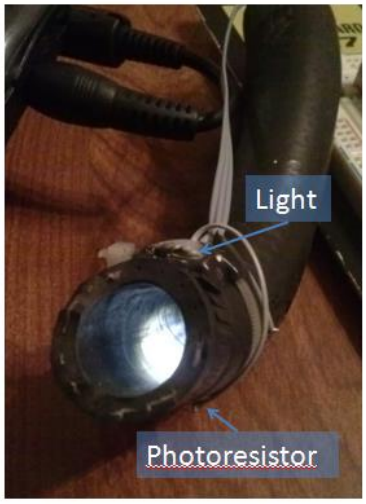


Figure 8 Counter Sensor

The counter sensors work on the same principle, using the same parts as the hopper sensor. In this case, 5mm holes are drilled through opposing sides of a rubber tube in which mount the LED light source and the photoresistor. As the Skittles fall through, they interrupt the light shining on the PR, changing its resistance and triggering a comparator circuit sending a pulse to a counter/7-segment decoder IC.

The PR response to a Skittle falling through the tube is shown in Figure 9 below (yellow trace). This particular run shows that it took 7ms for the Skittle to pass the PR, 11.5 ms for the complete PR cycle. The theoretical speed limit per sensor is therefore 1000/11.5 = 87 Skittles/second. Since the response can vary widely due to the path a particular candy may take, a low threshold was set (shown by delta x on the plot to be 1.225V). Indeed, if the threshold were to be higher than the PR peak voltage, a count would be missed. The tradeoff is a risk of false triggers caused by electrical noise (from the motor, perhaps) or ambient light events. Current requirements for each sensor are the same as those of the hopper, 10mA.

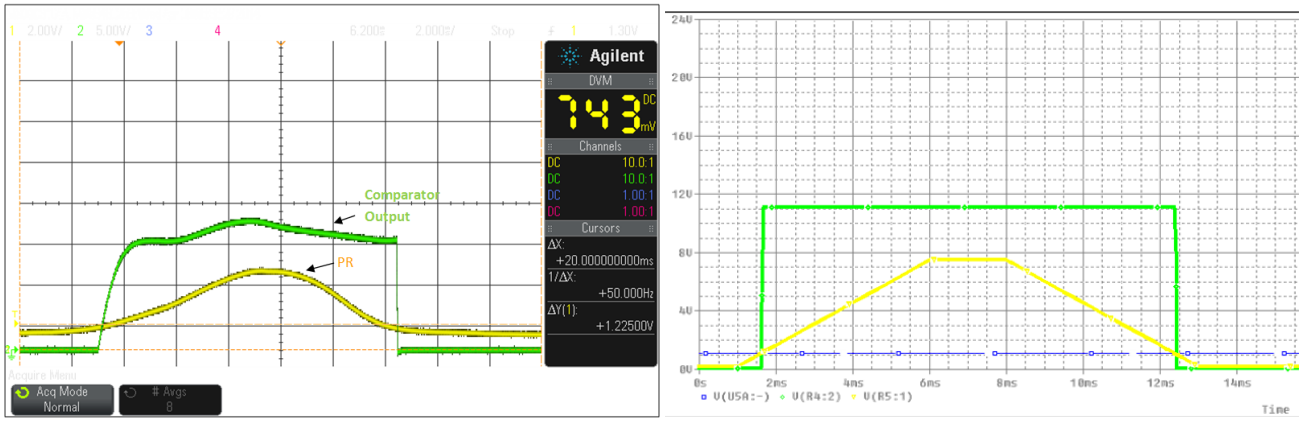


Figure 9 PR Response to Skittle Falling Through Tube

Scope trace compared with a PSpice simulation showing a 12ms fly by time for a typical drop count--not enough time for the PR to fully rise to its “dark” value, but plenty of time to reliably cross a carefully chosen comparator threshold voltage.

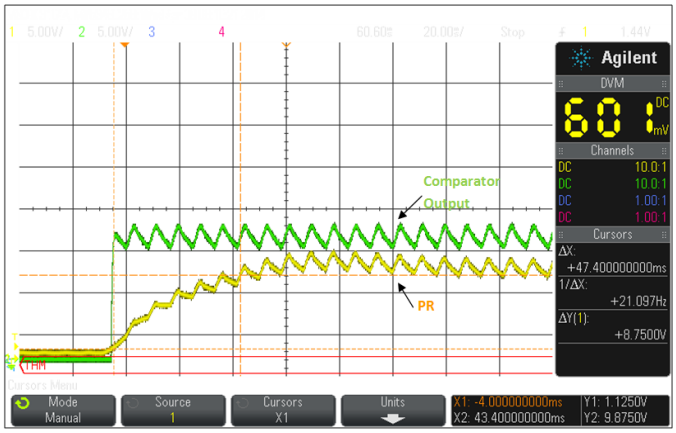


Figure 10 Response Time of PR

This trace measures the response time of the PR to be 47ms. The datasheet says 35ms. Bearing in mind how the power supply ripple obscures the rise time, the two numbers agree nicely.

Displays

Four-digit, multiplexed seven-segment LED displays that could be driven by a single I2C networkable chip were originally chosen as cheaper, bigger and brighter than LCD displays (Figure 11). The driver ICs were fine-pitch 24-pin devices that were expensive ($4), and difficult to work with. Though 20-pin mounting boards were available at reasonable price, 24-pin mounting boards were expensive and hard to find. This and the LCD display schemes would cost 7 inputs of the processor in the form of count sensor inputs and two more for the I2C communication.

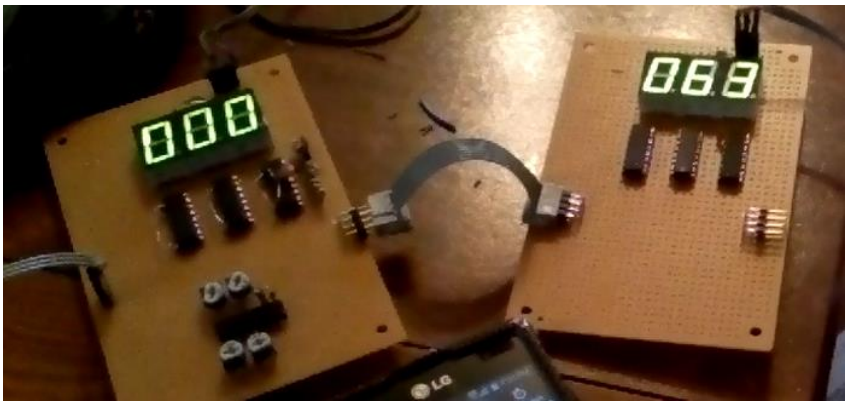


Figure 11 Displays

Instead it was decided to go with independent sensors and counter/driver ICs in order to reduce the I/O count required of the processor. This presented the opportunities to either reclaim the I/O for sorting or go with a lesser Arduino board.

The CD4026 IC provides a counter, schmitt trigger and seven segment decoding in one package. The cost is similar to the lowest cost LCD displays which may not even meet the requirements of readability. In addition, it was unclear whether eBay was a usable source, with the next best price nearly double. Half-inch yellow-green seven-segment LEDs were chosen for price and to go with a Texas Instruments counter-decoder in one. While the parts are cheap, the high number of connections required for seven three-digit displays was nearly prohibitive. In retrospect, designing and ordering etched PCBs from ExpressPCB would have saved time, but added expense. Since time is not considered as a cost in the competition, it was decided to use point-to-point soldering. Total cost per display is $3.26, including the circuit board.

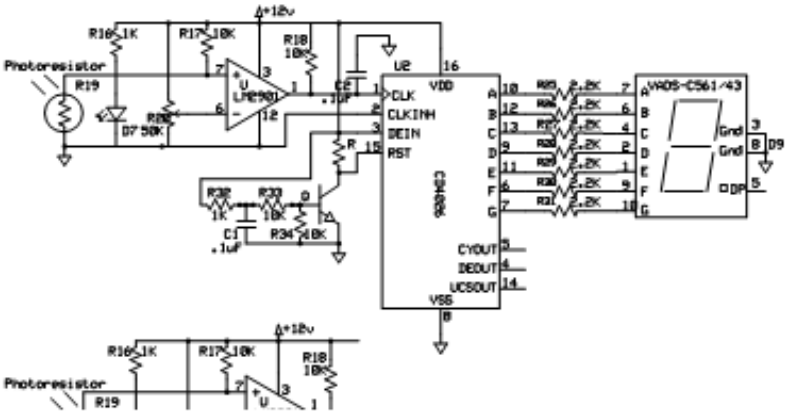


Figure 12 Schematic for Displays

While the datasheet suggests that the drivers have a limit of 3mA/segment and should be buffered, experiments showed that using no current-limiting resistors did not pose a problem for drivers or displays. Initially, 4.7K resistors were tried to keep current under the 3mA driver limit, but the displays were too dim. Instead, 2.2K resistors resulting in 4.7mA/segment yielded sufficient brightness. Each display therefore required 100mA.

The elapsed time chronometer displays are nearly identical to the counter displays, the main differences being an additional digit, the '10s' digit being tested[1] for '6' in order to clock the 'minutes' digit, and the '10s' digit being installed upside down in order to form the colon between minutes and seconds.

Sliding Door Servos

Hobby King

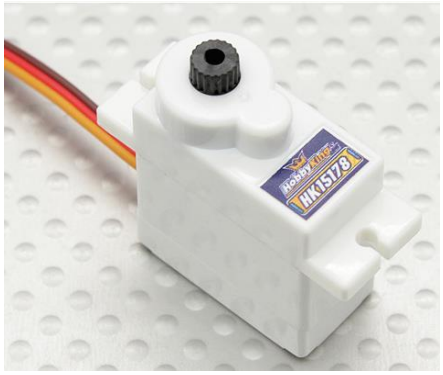


Figure 13 Hobby King Servo

The sliding door actuators operate on 5 to 6V, and require 190 mA to operate. A 5V, 8A power supply is required to power all of the servos and sensors while the Arduino, counters, timer and motor will be powered by a 12V computer power supply.

The servo in Figure 13 can travel 60 degrees in 0.09s via PWM signal. We require 40 degrees absent any type of mechanical advantage like a bell crank. The relationship between arm travel and time is not completely linear due to the inductive nature of the load, thus we must assume travel time to be more than ⅔ of 0.09s.

Motor

The motor is sufficiently oversized in order to cope with variations in friction and load encountered during the development process. A window lift motor was chosen because its relative inexpensive ($6.43), has a wide range of control voltages, (6 - 16V), torque (exact spec unknown, but enough to overcome the weight of door glass and friction of window seals), and integrated gear housing that provides for an ideal speed. Speed can be varied, by changing the Arduino's PWM signal from 0-255, which ranges the speed of the disc from 0 RPM all the way up to 18.75 RPM. In addition, the motor will accept a square shaft, on either side of the gear box, making adapting the motor for use in the project fairly simple and useable for driving both the disc and the hopper agitator at the same time.

The other motor considered was a Johnson P/N 9167AJ 13.6VDC motor used in power tools from MPJA. The ⅛" shaft driving the outside diameter of the 24" disk and 6200 RPM speed would give a speed of 6200/(24*8) = 32.3 RPM or 15 Skittles/second. While this is an ideal speed, the gear housing provided by the window motor for less than $2 more allowed for shafts that extend in both directions normal to rotation, thus we could spin both the disk and the hopper agitator with a simple single shaft.

Mechanical and Machine Design Considerations

Hopper Designs

The first part of the machine that the Skittles need to pass through is the hopper. Because of this, the hopper is the most crucial part of the system, often being overlooked. The initial design utilized a sliding plate that would pick up 6 Skittles at a time and drop them on the conveyor belt. The conveyor belt design was changed, as this hopper was slower than anticipated.

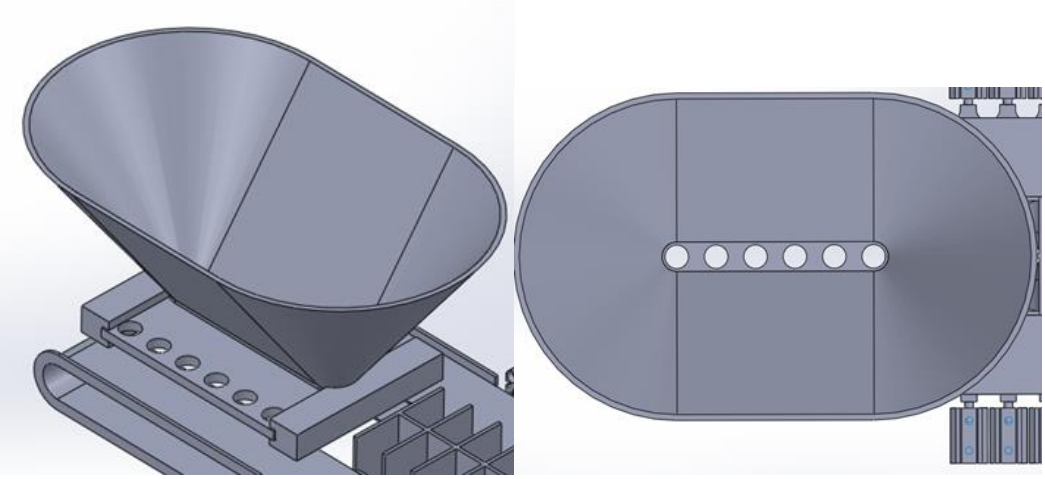


Figure 14 Initial Hopper Design

The second design incorporated more of a forced and better way to separate the Skittles one by one. As shown below, the hopper would require another motor to drive the flap wheel. This adds more complexity as well as more cost to the 5S, as it would require at least 1, maybe 2 more motors to drive the flap wheel.

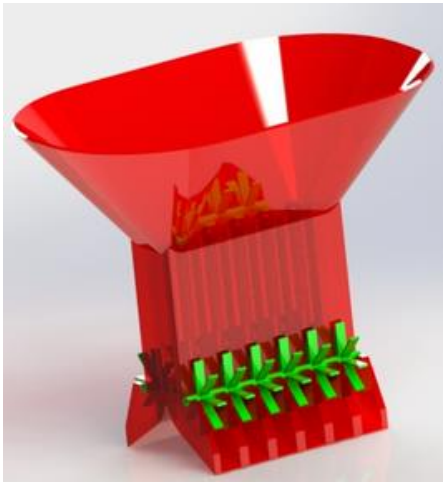


Figure 15 Second Hopper Design

The third hopper design incorporated 2 rotating discs that would grab 1 Skittle at a time. As pictured below, the Skittle would still have to be separated individually from a group of Skittles. This did not really seem like the most logical option because the Skittles are separated individually 2 different times.

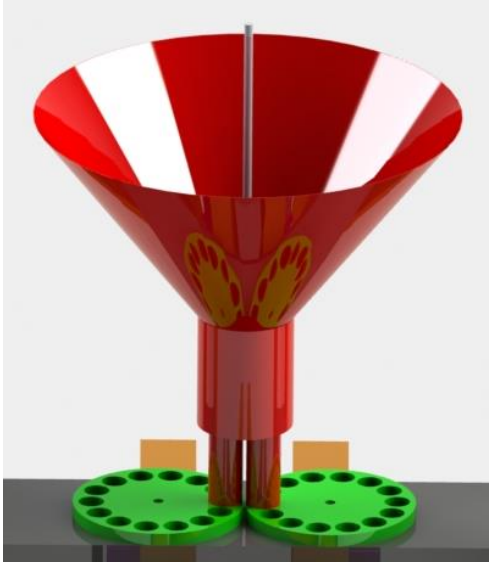


Figure 16 Third Hopper Design

The last and final hopper design ended up being a very simple agitator that spins using the same motor that spins the disc. As the agitator spins, the Skittles are pushed through each of the 4 tubes that lead down to the rotating disc.

Figure 17 Hopper Agitator

The next and second most important part of the Skittle sorting machine is the separating the Skittles individually so that the Skittle color can be read. In order for the Skittle to be read consistently and accurately, it must be held very steady and at a consistent distance away from the sensor. The importance of the Skittle position was not learned until after the first design was made and tried.

There were multiple different ideas that were considered. The first design discussed was the plinko style sorting machine. The second design was the piston or plunger design, which was determined to not sort Skittles quickly enough for the competition but did seem to be a consistent and dependable way to sort Skittles. Below is a picture of the plinko design that was found online and designed by Lego.

Plinko

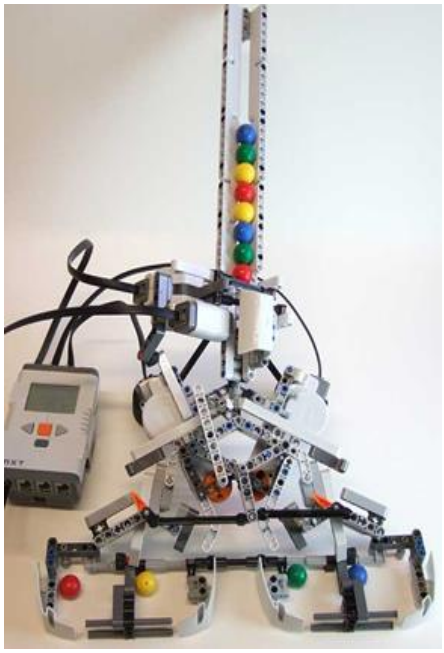


Figure 18 Sample Plinko System

The plinko system would utilize gravity to push the Skittles down and a series of paddles connected to servos would guide them to the correct bucket based on color. The cons on this set up is that the system would have to stop each Skittle one by one to read the color, and timing the paddles to move the Skittles as they fall would be difficult. Also, gravity is the only thing moving the Skittles. The pros to this system is there are not many parts involved and can be made on a low budget and easily duplicated to increase the speed at which Skittles are sorted.

Piston

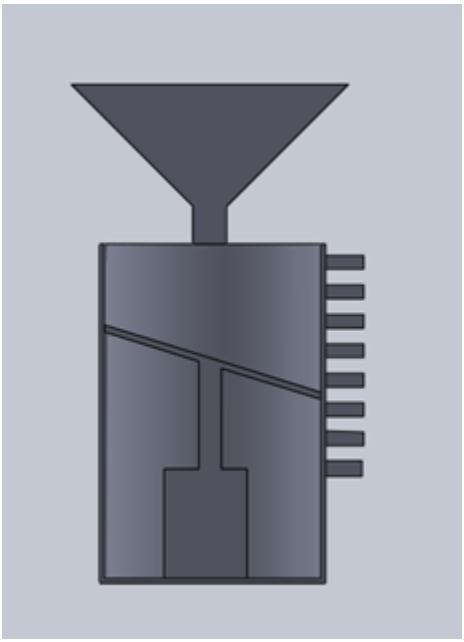


Figure 19 Sample Piston System

Another idea considered was a piston/linear actuator. The idea behind this system is the Skittles would be read and a linear actuator would push a plate up and down to the correct position where the Skittles will slide into the correct hole based on its color. The cons to this system are that you have to stop each Skittle to read the color and you are also relying on gravity to force the Skittle to the correct bucket. The pros to this idea are there are not many moving parts and cost should be low.

Conveyor Belt

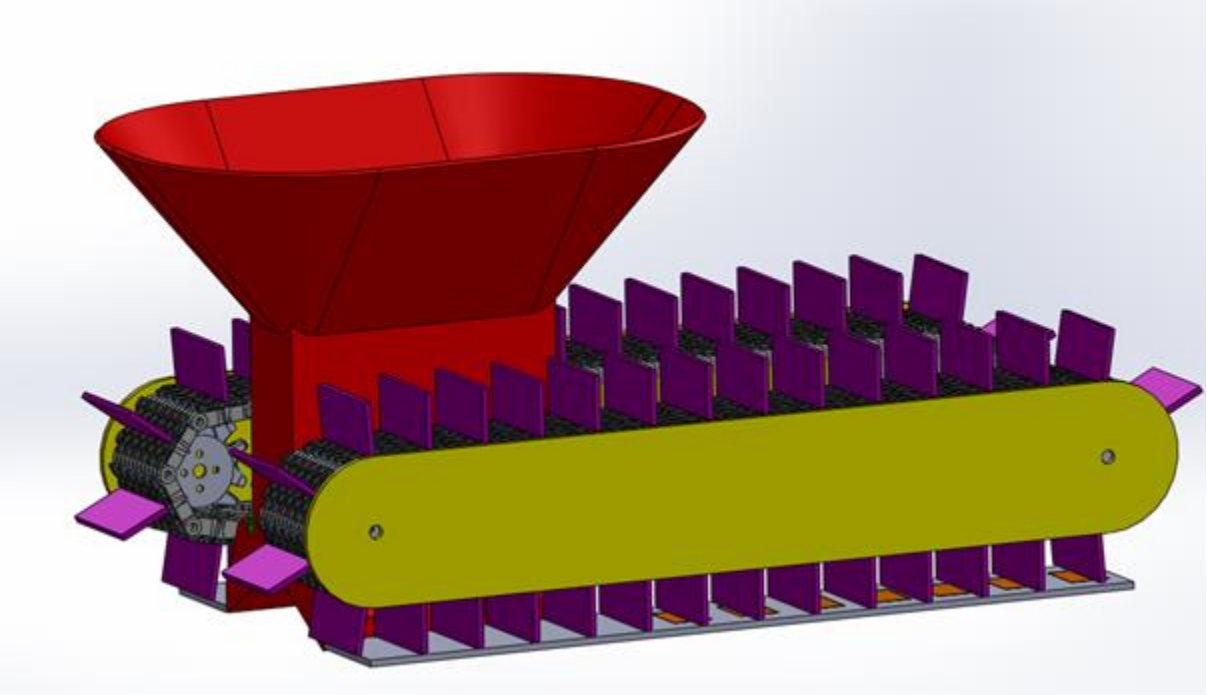


Figure 20 Sample Conveyor Belt System

The last design that was considered was a conveyor belt system. This system would feed 6 Skittles on a platform where brushes on a conveyor belt would come by and push the Skittles down the platform. There is one trap door for each color and will open according to the color of the Skittles. There will be two conveyor belt systems so the Skittles can sort just as fast. The cons to this system are there will need to be 14 servos for each trap door as well as sprockets and motors to drive the conveyor belt. The pros to this system are the Skittles are constantly moving; the Skittles do not need to stop for a sensor to read the color. Since this system does not rely on gravity, you can tune in the servos and conveyor belt to move as fast as the color sensor can read and as fast as the servos can move, to allow the Skittles through the door. This idea was built up and determined to not be a good decision because the Skittle could not be held steady enough and in a consistent enough position to read the Skittles accurately.

Then it was time to get back to the drawing board. It was decided that instead of separating the Skittles two times, it was a better idea to just separate the Skittles one time and then read the color immediately after they were separated. It was decided to try using the final hopper design as the main

part of the whole sorting machine. Instead of just using one sensor and one set of doors to sort the Skittles, it was decided to duplicate the set and have four sets of servos and sensors to speed up the sorting process. At full speed this machine is designed to sort 40 Skittles per second. However, this goal was far-reaching at time of competition, yet could be improved upon in the future.

The white in Figure 21 is the hopper; the copper tube is the part that transfers the Skittles one by one from the hopper to the rotating disc.

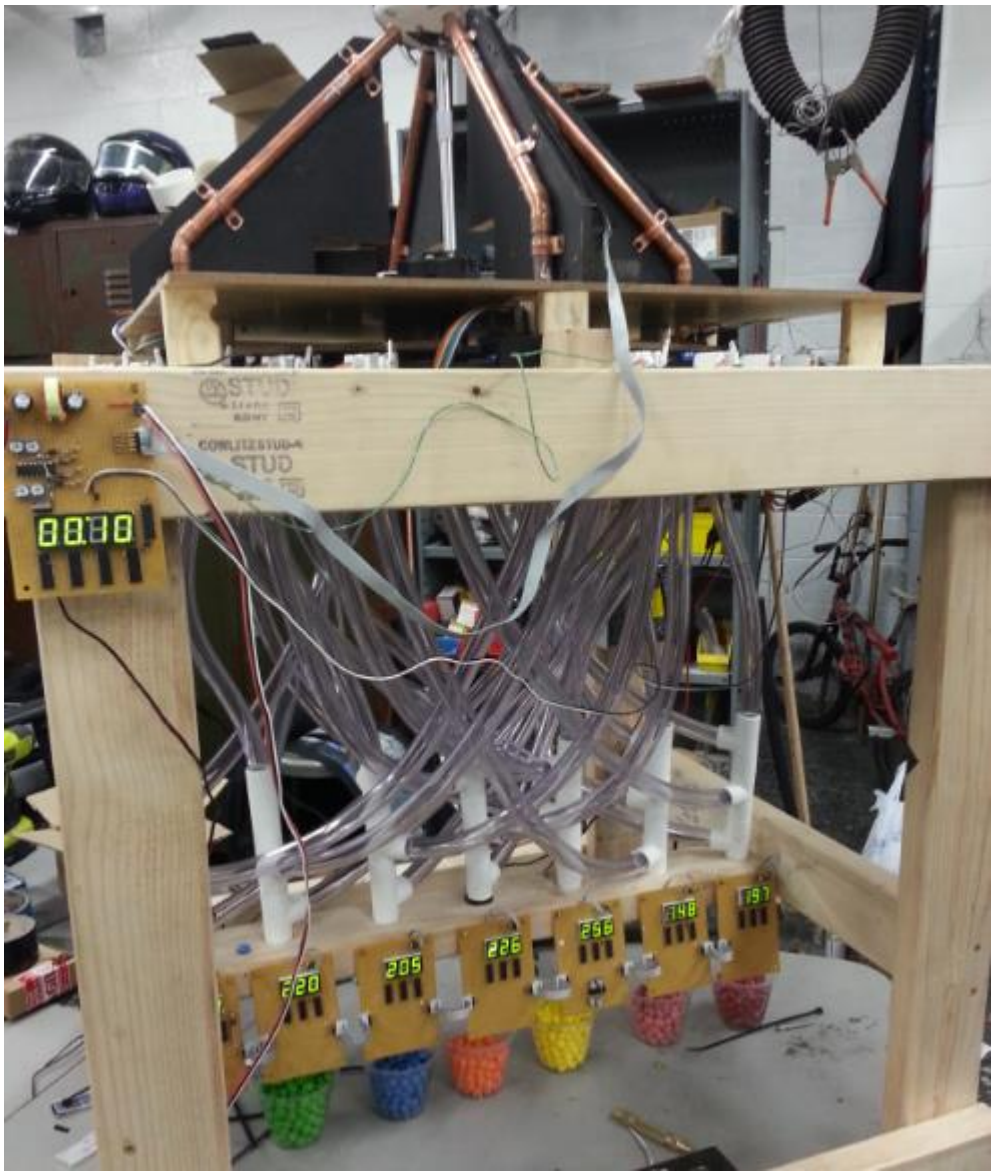


Figure 21 Entire System

The black disc rotates clockwise and once the Skittle drops out of the tube, it gets picked up by the rotating disc and then slides past the sensor. As the sensor reads the color, it is stored in the queue and the proper door opens after the proper count of clicks.

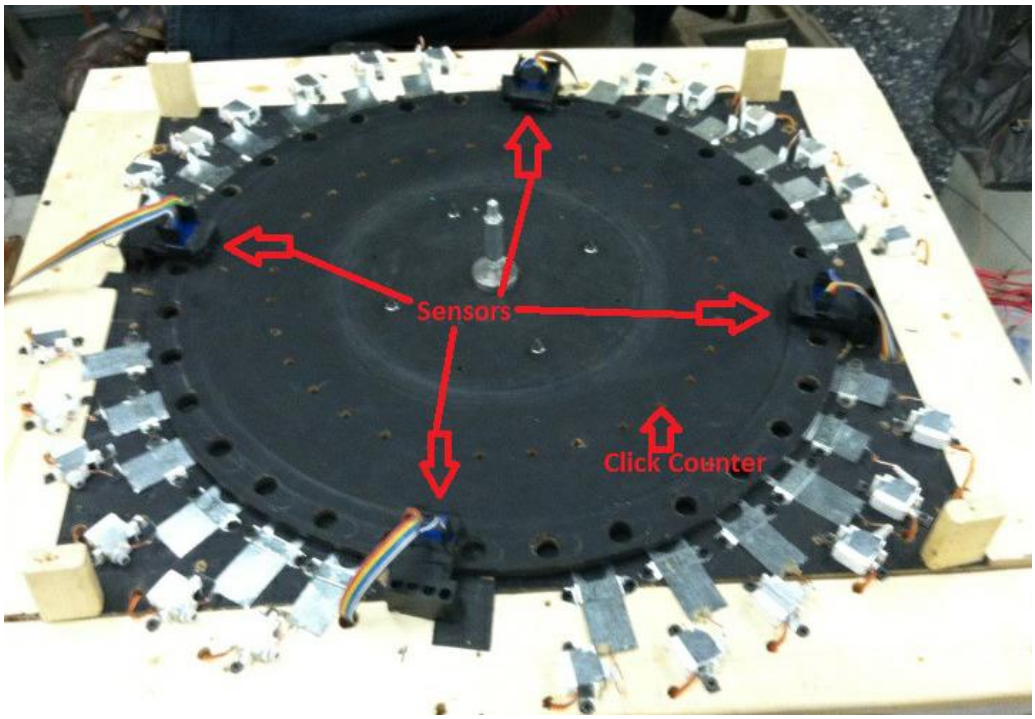


Figure 22 Rotating Disk with Servo Doors

This height that the copper tube is above the black disc is very important to keep consistent to keep just one Skittle entering the disc at a time without jamming.

Figure 23 With Top On

Once the Skittles are read and the proper trap door opens, a flexible plastic tube takes the Skittles from the machine down to the bins. Since there are four sets of doors and four sensors, there are a lot of tubes that transport the Skittles. This brings about another challenge, routing four tubes to just one so that they can be counted. With the help of PVC Tee's, all four sets of each color Skittle were able to be connected to just one output where it can be counted easily.

Stress Analysis

A finite element analysis (FEA) was done on the shaft that connects to the motor. The load is applied to an area on the top of the shaft that makes contact with the motor. The shaft is made out of 4130 Steel; and is made of 1.25" hex with a ¼" square on top that connects directly to the motor. The shaft was fixed at the bolt holes that connect to the bearing and a torque of 11 Nm was applied to the top square of the shaft. The torque was chosen because the motor has a stall torque of 11Nm. The mesh size was refined until the results reached diminishing returns. The analysis was done in the parabolic method.

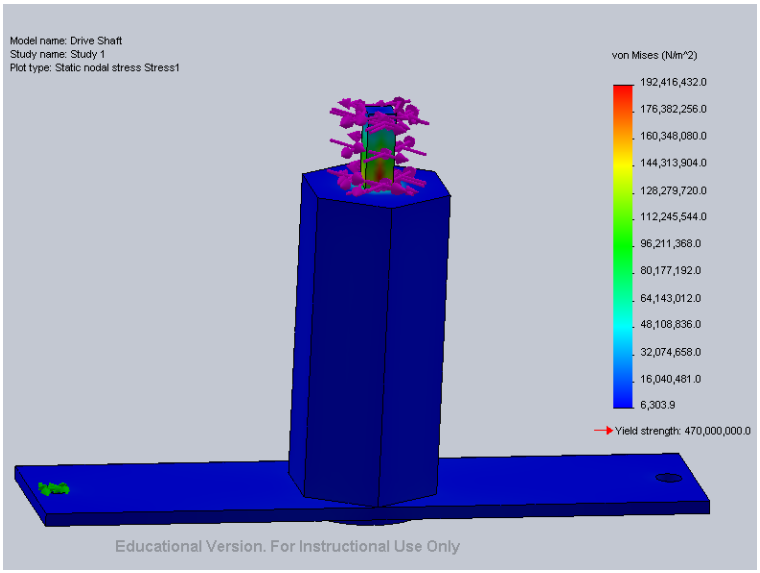


Figure 24 Von Mises Stress on Drive Shaft

The maximum stress analysis is shown above The maximum stress was recorded to be 197,748,192 N/m^2. This stress was located at the lower part of the square.. This is expected because this part of the shaft is directly connected to the motor. 4130 has a yield strength of; 470,000,000 N/m^2. The safety factor can be calculated by dividing the yield stress by the actual stress on the material. This is shown in the equation below.

$$S.F. = \frac{\sigma_y}{\sigma}$$

The factor of safety for the shaft is 2.44. With a factor of safety of 2.4, this gives assurance that the machine can run without the shaft being a concern.

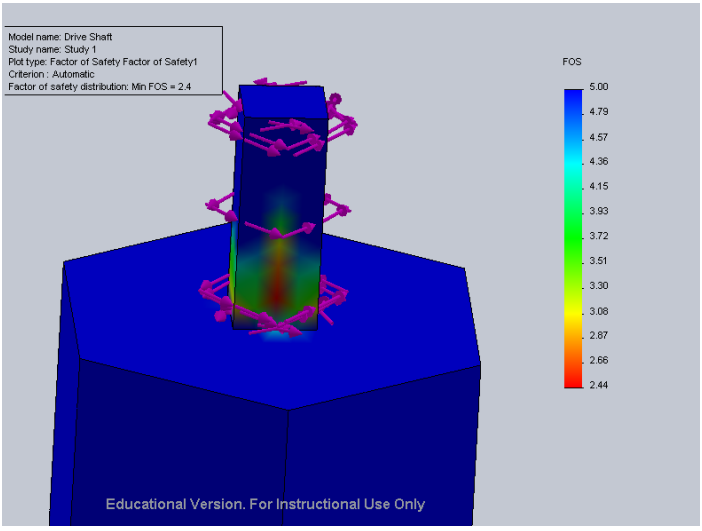


Figure 25 Factor of Safety of the Drive Shaft

Results and Discussion

It would be very inexpensive to implement the 5S. After buying all materials to build it, the cost would be approximately $180. Labor involved would be building the whole machine, wiring all sensors, servos, and displays, “zeroing” the servos, and calibrating the sensors. With tight tolerances to machining, calibration would be very quick as the sensors should output very similar results. Verifying each sensor would still need to be done to be sure a faulty reading wasn’t being output. The initial readings would take about 6-8 hours. After the initial readings were saved, verifying it for new machines would take 1-4 hours depending if there was a faulty reading. Each display takes about 10 hours to make. Setting up the servos by “zeroing” them would take only a few minutes and could be pre done if each servo arm and door was made identical. The wiring of all sensors and servos would take about 10 hours. Building the system as a whole would take about 60 hours. This includes making each servo arm, making the disk, making the servo doors, tubing for each door to the correct bin, making the base piece, making the piece to mount the position sensor and to build the finished product. In all, about 160 hours would be needed to make a whole machine. Breaking this up between 10 people, it can be completed in about 16 hours which would mean about 2 days to complete the build. Depending on demand of such Skittle sorting machines, people hired can be based off of this demand. With a high demand, more jobs would be created and help the economy.

Safety

Overall, the 5S is a safe machine. However, there are several things that users should be aware of. First, there is a blade agitator inside of the hopper, so it is not safe to reach inside of the hopper while the machine is operating. Second, there is a rotating disk, which carries Skittles around. Therefore, it is also dangerous to touch when the disk is spinning. Third, the trap doors are sharp. They close and open rapidly, so it is not wise to touch the trap doors while the machine is in use, as it might cause injury. Fourth, it is dangerous to touch the ends of wires in the sorter. They might carry high voltage electricity. Fifth, when the machine stops running due to any problems, it is important to turn off the machine first. Oftentimes, the machine stops running because some irregular shaped Skittles become stuck in the disk. However, the amount of Skittles would not be an issue as a taller tube could be used for the hopper to allow more Skittles and it would not affect the motor rotating the blades inside the hopper. As long as the

bins at the exit could hold the amount of Skittles to be sorted, the number of Skittles entering the machine would not be a factor.

Economic Factors

The cost of the Super Speedy Serial Skittle Sorter is around 180 dollars. It has two power supplies, one of them is 400W, and another one is 40W. Therefore, if the machine were operating 24/7, it would use around 0.44KW per hour, with the cost of electricity depending on location. This machine could be used for many different purposes. Some modification might be needed, like color based sorting or shape based sorting. In this case, this machine could displace some jobs like medicine pill sorting or different color recycling waste sorting. It would affect some jobs, as many sorting jobs require human efforts. It would also increase the speed of the sorting process.

Reliability

The failure rate for the machine would be dependent based off of the sensor that was used. For the current setup, a failure rate of 20% would be accepted. This is because with the TCS3200 TCS230 sensor, the values for colors overlap. This is a problem if a darker pink is sensed or a lighter red. Each might be read as the wrong color. If a new color sensor was implemented, an ideal failure rate would be 0.01%. This system would be very reliable; however, each feeding tube would be set to the correct height to eliminate the possibility of jamming. Without jamming, the machine would continue without problem.

Aesthetics

For the 5S, using a metal frame would be more aesthetically attractive, as well as making the sorter more stable and durable. The metal could be painted red in order to represent Skittles and heighten the aesthetics. Placing white borders around the counters could draw attention to them and make them look better. Additionally, using a clear hopper with the Skittles logo on it would add some more color to the technology. To stay with the Skittles theme, using white metal borders would be a good choice. Moreover, in order to see the unique mechanics on the inside without making it vulnerable, clear Plexiglas could surround it. Nonetheless, users need to get access to the inside, so constructing a door on the back of the technology would be a good idea. The clear Plexiglas would also reduce the noise produced by the technology. To make the 5S more appealingly, LEDs could be lined down the tubes. As

soon a Skittle goes down a tube, an LED would flash which would correspond to the color it was. As a red Skittle exits the disk, a red LED would flash. The same would happen for the remaining colors, and in effect a light show would occur until all the Skittles have left the system. Tubes leading to the bins could have a vacuum at the end of the tube would help alleviate Skittles from getting stuck in the tubing.

Potential Customers

There are various types of sensors available: distance, color, and sound sensors. These sensors have made daily living easier and more convenient. The color-sorting machine can be used in various fields and for many different applications. The color-sorting machine can be used in the manufacturing field, as it could be used widely for sorting bottle caps, medicine pills, candy, and so on. It sorts the different items into their designated sections based on their color. In addition, the color-sorting machine can be used in recycling factories. Plastic recycling factories need to sort the different colored plastic so it can be recycled and reused. Therefore, the color-sorting machine can be used to sort the products based on their color. Afterwards, the different colored unwrapped products will be placed back into the husking machine until they are completely unwrapped. Furthermore, the color-sorting machine can be used for organization, such as organizing library books. A better method to sort books would be to use a color-sensor machine rather than a code scanner. A different colored sticker can represent a specific bookshelf. The color-sorting machine can be attached to the book-return entry. Once people drop books into the book-return entry, the sorting machine can organize the books by the colored sticker on them.

Societal Impact

The potential benefit to society of the 5S would be increasing the speed of sorting. It also can be more efficient by allowing humans to avoid injury when it comes to sorting harmful products. However, this machine might cause people to lose jobs as the machine will do all the sorting work. Additionally, people who work with this machine might need a higher level of education, so that they are able to fix or adjust the machine by themselves. Therefore, people who have a lower of education might have a more difficult time finding a job in this type of field.

Table 2 shows results from calibrating the first sensor. This was done by taking five different Skittles and recording four different readings for each Skittle. Each reading consisted of a red, green, blue, and white value. The red, green, and blue values were put into ratios as can be seen from the figure below. The average of the b/g, g/r, and r/b ratios was recorded. In order to get a range that could be implemented into the code, +/- 3 standard deviations was used, which is shown in the column +/-. The range was the produced by taking the average of the ratios and adding or subtracting three standard deviations.

Table 2 Color Values From Color Calibration

																								ratios		
Red																						Ratios		+/-	LB	UB
Sample->	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	μ					
R	2426	2425	2418	2426	2391	2384	2391	2391	2461	2454	2460	2456	2213	2212	2207	2213	2223	2221	2215	2209	2339.8	b/g	0.744419	0.01	0.73	0.76
B	2135	2139	2140	2133	2257	2255	2258	2257	2101	2101	2101	2103	2248	2246	2245	2248	2094	2099	2090	2097	2167.35	g/r	1.247417	0.24	1.00	1.50
G	2870	2863	2863	2870	3042	3043	3046	3044	2813	2813	2813	2815	3032	3031	3032	3034	2805	2805	2799	2804	2911.85	r/b	1.080997977	0.20	0.88	1.29
W	775	768	768	774	800	800	800	806	759	760	759	759	785	779	778	779	750	751	749	743	772.1			56.76	715.33	828.87
Green																						Ratios				
Sample->	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20						
R	2124	2131	2129	2128	2254	2251	2248	2254	2069	2068	2074	2076	2074	2069	2074	2076	2233	2229	2234	2229	2151.2	b/g	1.035027	0.03	1.00	1.07
B	1910	1905	1904	1902	1985	1985	1985	1988	1828	1821	1822	1822	1838	1837	1837	1838	1893	1900	1893	1896	1889.45	g/r	0.848831	0.02	0.82	0.88
G	1822	1829	1821	1822	1900	1900	1900	1900	1759	1758	1759	1766	1772	1778	1779	1772	1868	1868	1868	1870	1825.55	r/b	1.1385209	0.06	1.07	1.21
W	627	625	619	620	652	648	646	657	599	607	603	598	600	600	600	600	631	628	629	629	620.9			58.57	562.33	679.47
Purple																						Ratios				
Sample->	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20						
R	2927	2927	2927	2926	3123	3123	3122	3123	2858	2858	2858	2860	3116	3116	3116	3116	3273	3266	3271	3271	3058.85	b/g	0.741364	0.00	0.73	0.75
B	2082	2082	2083	2083	2229	2228	2229	2228	2030	2030	2028	2030	2220	2220	2220	2220	2295	2296	2297	2290	2171	g/r	0.957494	0.02	0.93	0.98
G	2813	2812	2812	2812	2998	3000	3000	3000	2741	2741	2741	2741	2997	2997	2998	3000	3091	3090	3091	3090	2928.25	r/b	1.408803674	0.03	1.38	1.44
W	796	802	796	796	851	852	850	848	775	774	775	774	851	851	853	848	879	879	880	879	830.45			118.52	711.92	948.98
Orange																						Ratios				
Sample->	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20						
R	1702	1708	1711	1700	1628	1635	1628	1635	1585	1578	1578	1579	1511	1513	1518	1513	1820	1820	1820	1820	1650.1	b/g	0.826796	0.04	0.78	0.88
B	1817	1817	1816	1816	1728	1725	1730	1725	1918	1921	1922	1921	2057	2057	2057	2056	2099	2104	2104	2104	1924.7	g/r	1.414075	0.33	1.07	1.75
G	2248	2246	2240	2246	2118	2118	2118	2118	2311	2312	2312	2303	2421	2416	2421	2414	2540	2547	2540	2540	2326.45	r/b	0.861538197	0.24	0.62	1.11
W	596	596	598	596	568	566	569	565	598	606	599	598	615	621	613	615	669	669	669	675	610.05			105.31	504.74	715.36
Yellow																						Ratios				
Sample->	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20						
R	1485	1492	1486	1492	1501	1506	1501	1507	1150	1150	1150	1152	1320	1325	1325	1318	1318	1324	1325	1318	1357.25	b/g	1.175161	0.20	0.97	1.38
B	1820	1820	1825	1820	1845	1845	1845	1843	1759	1759	1758	1758	1757	1750	1750	1750	1757	1750	1750	1750	1785.55	g/r	1.126600	0.09	1.03	1.22
G	1628	1635	1628	1627	1646	1652	1646	1653	1352	1352	1352	1359	1499	1499	1492	1499	1498	1497	1492	1498	1525.2	r/b	0.759175116	0.18	0.57	0.95
W	516	519	519	514	518	526	518	526	431	439	439	439	477	477	484	477	477	478	484	477	486.75			95.44	391.31	582.19
Blue																						Ratios				
Sample->	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20						
R	2905	2910	2904	2902	2899	2891	2892	2899	2883	2883	2881	2881	2767	2767	2771	2773	2973	2971	2969	2969	2884.5	b/g	0.671096	0.05	0.62	0.72
B	1477	1484	1484	1484	1657	1663	1657	1662	1530	1528	1528	1530	1426	1432	1432	1431	1582	1576	1582	1576	1536.05	g/r	0.793000	0.05	0.74	0.84
G	2262	2262	2261	2254	2371	2377	2377	2376	2291	2291	2290	2290	2165	2164	2157	2164	2353	2346	2351	2351	2287.65	r/b	1.881392391	0.23	1.65	2.12
W	637	637	642	645	680	680	680	682	646	646	645	652	619	619	611	612	676	670	670	675	651.2			72.74	578.45	723.95
Pink																						Ratios				
Sample->	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20						
R	1740	1733	1733	1733	1778	1778	1778	1783	1854	1849	1849	1849	1771	1775	1771	1779	1904	1905	1904	1905	1808.55	b/g	0.743306	0.01	0.73	0.76
B	1736	1736	1741	1741	1775	1775	1774	1775	1713	1713	1720	1712	1719	1725	1719	1719	1688	1687	1684	1687	1726.95	g/r	1.286557	0.18	1.10	1.47
G	2324	2322	2326	2329	2391	2391	2389	2391	2304	2297	2304	2304	2329	2328	2328	2321	2274	2267	2274	2274	2323.35	r/b	1.047984809	0.15	0.89	1.21
W	606	606	612	612	625	619	625	619	615	610	610	615	606	611	611	604	611	606	606	611	612			18.34	593.66	630.34

It was necessary to find overlaps after these ranges were obtained. Table 3 shows how many ratios two colors share. Take red and orange for example; these two colors share three ratios. Since all three ratios are shared, individual values would be used to distinguish between each color.

Table 3 Overlap Regions of Color Calibration

Overlaps	Green	Purple	Orange	Yellow	Blue	Pink
Red	1	2	3	2	1	3
Green		0	2	2	2	2
Purple			1	0	1	1
Orange				3	1	3
Yellow					1	2
Blue						1

Wherever two colors didn't share three ratios, ratios for these colors could be opened up to allow more leniencies from the sensor. Once set, these ratios and raw values could be input into the code.

Ten different teams built an automatic Skittle sorter, all attempting to build the quickest and cheapest Skittle sorter. The 5S was the most successful implementation of a Skittle sorting machine. Cost was kept at a minimum and speed was found to be the fastest out of all machines built. With this affirmation of original design considerations, future buildings of this machine would also generate the greatest success. At time of presentation, the machine was found to be 80 percent accurate, with the ability to improve with the allotment of more time.

References

"CAR SEAT 3-12VDC GEAR MOTOR." *CAR SEAT 3-12VDC GEAR MOTOR*. N.p., n.d. Web. 24 Mar. 2014.
<http://www.sciplus.com/p/CAR-SEAT-312VDC-GEAR-MOTOR_49248>.

Digital Design Using Digilent FPGA Boards, by Richard E. Haskell and Darrin M. Hanna
Oakland University, LBE Books, Rochester, MI c. 2009

"HK15178 Analog Servo 10g / 1.4kg / 0.09s." *HobbyKing Store*. N.p., n.d. Web. 19 Mar. 2014.
<https://www.hobbyking.com/hobbyking/store/__16257__HK15178_Analog_Servo_10g_1_4kg_0_09s.html>.

IR Light Beam Break Circuits,
http://www.me.umn.edu/courses/me2011/arduino/technotes/irbeam/irbeam.html

Latcha, Michael, and Mohamed Zohdy. "SECS Senior Design Syllabus." *SECS Senior Design Syllabus*. N.p., n.d. Web. 19 Mar. 2014

"McMaster-Carr." *McMaster-Carr*. N.p., n.d. Web. 24 Mar. 2014.

"TCS3200 Color Sensor (SKU:SEN0101)." *Robot Wiki*. N.p., n.d. Web. 19 Mar. 2014.
<http://www.dfrobot.com/wiki/index.php/TCS3200_Color_Sensor_(SKU:SEN0101)>.

"Using the TCS3200 with Arduino or Parallax Propeller." *ReiBotorg*. N.p., n.d. Web. 14 Apr. 2014.
<http://reibot.org/2011/07/06/tcs3200/>.

Appendix

Data Sheets and Arduino Specifications

Taos color to light frequency convertor,
http://www.dfrobot.com/image/data/SEN0101/TCS3200%20TCS3210.pdf

Arduino spec sheet, http://arduino.cc/en/uploads/Main/arduino-mega2560_R3-sch.pdf

Arduino pin mapping, http://arduino.cc/en/Hacking/PinMapping2560

CMOS AND Gates data sheet, http://www.ti.com/lit/ds/schs057c/schs057c.pdf

CMOS decade counter/divider data sheet, http://www.ti.com/lit/ds/symlink/cd4026b.pdf

Comparator data sheet, http://www.ti.com/lit/ds/symlink/lm339-n.pdf

Summary of Microcontroller

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

http://arduino.cc/en/Main/arduinoBoardMega2560

Memory of Arduino

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

http://arduino.cc/en/Main/arduinoBoardMega2560

Input and Output of Mega

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 2 to 13 and 44 to 46.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **TWI: 20 (SDA) and 21 (SCL).** Support TWI communication using the Wire library. Note that these pins are not in the same location as the TWI pins on the Duemilanove or Diecimila.

http://arduino.cc/en/Main/arduinoBoardMega2560

Full Code for The 5S:

```
#include <QueueArray.h>
#include <Servo.h>

// The I/O pins for section one color sensor are initialized
// Each sensor has the same pins with the same functionality
// All of the output pins could be tied together.
// The only pins that could not be tied together are the input pins from the color sensor
// i.e. in this case pin 10

int S1_S0 = 8;
int S1_S1 = 9;
int S1_S2 = 12;
int S1_S3 = 11;
int S1_taosOutPin = 10;
int S1_LED = 13;

//------------------------------------------------------------------------------

// The I/O pins for section two color sensor are initialized
// Input pin in this case was pin 46

int S2_S0 = 8;
int S2_S1 = 9;
int S2_S2 = 12;
int S2_S3 = 11;
int S2_taosOutPin = 46;
int S2_LED = 13;

//------------------------------------------------------------------------------

// The I/O pins for section three color sensor are initialized
// Input pin in this case was pin 47

int S3_S0 = 8;
int S3_S1 = 9;
int S3_S2 = 12;
int S3_S3 = 11;
int S3_taosOutPin = 47;
int S3_LED = 13;

//------------------------------------------------------------------------------

// The I/O pins for section four color sensor are initialized
// Input pin in this case was pin 48

int S4_S0 = 8;
```

```
int S4_S1 = 9;
int S4_S2 = 12;
int S4_S3 = 11;
int S4_taosOutPin = 48;
int S4_LED = 13;

//--------------------------------------------------------------------------------

// The beam break was assigned to digital pin 2

int GBeam = 2;

//--------------------------------------------------------------------------------

// Initialize the servos for section 1
Servo S1_servoR;  // Red servo door section 1
Servo S1_servoG;  // Green servo door section 1
Servo S1_servoPu;  // Purple servo door section 1
Servo S1_servoY;  // Yellow servo door section 1
Servo S1_servoOr;  // Orange servo door section 1
Servo S1_servoPi;  // Pink servo door section 1

//--------------------------------------------------------------------------------

// Initialize the servos for section 2
Servo S2_servoR;  // Red servo door section 2
Servo S2_servoG;  // Green servo door section 2
Servo S2_servoPu;  // Purple servo door section 2
Servo S2_servoY;  // Yellow servo door section 2
Servo S2_servoOr;  // Orange servo door section 2
Servo S2_servoPi;  // Pink servo door section 2

//--------------------------------------------------------------------------------

// Initialize the servos for section 3
Servo S3_servoR;  // Red servo door section 3
Servo S3_servoG;  // Green servo door section 3
Servo S3_servoPu;  // Purple servo door section 3
Servo S3_servoY;  // Yellow servo door section 3
Servo S3_servoOr;  // Orange door section 3
Servo S3_servoPi;  // Pink door section 3

//--------------------------------------------------------------------------------

// Initialize the servos for section 4
Servo S4_servoR;  // Red servo door section 4
Servo S4_servoG;  // Green servo door section 4
```

```
Servo S4_servoPu;  // Purple servo door section 4
Servo S4_servoY;  // Yellow servo door section 4
Servo S4_servoOr;  // Orange servo door section 4
Servo S4_servoPi;  // Pink servo door section 4

//--------------------------------------------------------------------------------

int AABeam = 0;  // A temp variable that will act like a HIGH signal
int beamBreak = 0;  // Initialize the beam break
int myMotor = 7; // Pin to control the motor speed

//--------------------------------------------------------------------------------

// Queues to hold the values of the click count + pre numbers for section one
QueueArray <int> redDoorS1;
QueueArray <int> greenDoorS1;
QueueArray <int> purpleDoorS1;
QueueArray <int> yellowDoorS1;
QueueArray <int> orangeDoorS1;
QueueArray <int> pinkDoorS1;

//--------------------------------------------------------------------------------

// Queues to hold the values of the click count + pre numbers for section two
QueueArray <int> redDoorS2;
QueueArray <int> greenDoorS2;
QueueArray <int> purpleDoorS2;
QueueArray <int> yellowDoorS2;
QueueArray <int> orangeDoorS2;
QueueArray <int> pinkDoorS2;

//--------------------------------------------------------------------------------

// Queues to hold the values of the click count + pre numbers for section three
QueueArray <int> redDoorS3;
QueueArray <int> greenDoorS3;
QueueArray <int> purpleDoorS3;
QueueArray <int> yellowDoorS3;
QueueArray <int> orangeDoorS3;
QueueArray <int> pinkDoorS3;

//--------------------------------------------------------------------------------

// Queues to hold the values of the click count + pre numbers for section four
QueueArray <int> redDoorS4;
QueueArray <int> greenDoorS4;
QueueArray <int> purpleDoorS4;
```

```
QueueArray <int> yellowDoorS4;
QueueArray <int> orangeDoorS4;
QueueArray <int> pinkDoorS4;

//--------------------------------------------------------------------------------

int GbeamClickCount = 0;  // Click count that will increment only when a color is sensed
int startBreak = 0;  // Temp variable that reads the beam break that will start the machine
int startBeam = 4;  // This beam will start the machine
int externalTimer = 3;  // Send a signal to an external pin that will start a timer on a 7 segment display


// The following variables set the number of clicks that each Skittle has to travel before a trap door is open
int redPreNum = 1;
int greenPreNum = 6;
int purplePreNum = 5;
int yellowPreNum = 4;
int orangePreNum = 3;
int pinkPreNum = 2;
int servoDelay = 160;
int C = 0;
int D = 0;
//--------------------------------------------------------------------------------

void setup() {
  // Everything in the setup() function will only run one time, where all pins are initialized
  // Each of these method calls will initialize the inputs and outputs for each sensor
  TCS3200setupS1();
  TCS3200setupS2();
  TCS3200setupS3();
  TCS3200setupS4();

  //--------------------------------------------------------------------------------

  // Attach section 1 servos to pins and set their default positions
  S1_servoR.attach(40);
  S1_servoR.write(40);
  S1_servoG.attach(41);
  S1_servoG.write(40);
  S1_servoPu.attach(42);
  S1_servoPu.write(40);
  S1_servoY.attach(43);
  S1_servoY.write(40);
  S1_servoOr.attach(44);
  S1_servoOr.write(40);
  S1_servoPi.attach(45);
  S1_servoPi.write(40);
```

```
//--------------------------------------------------------------------------------

  // Attach section 2 servos to pins and set their default positions
S2_servoR.attach(22);
S2_servoR.write(40);
S2_servoG.attach(23);
S2_servoG.write(40);
S2_servoPu.attach(24);
S2_servoPu.write(40);
S2_servoY.attach(25);
S2_servoY.write(40);
S2_servoOr.attach(26);
S2_servoOr.write(40);
S2_servoPi.attach(27);
S2_servoPi.write(40);

//--------------------------------------------------------------------------------

  // Attach section 3 servos to pins and set their default positions
S3_servoR.attach(28);
S3_servoR.write(40);
S3_servoG.attach(29);
S3_servoG.write(40);
S3_servoPu.attach(30);
S3_servoPu.write(40);
S3_servoY.attach(31);
S3_servoY.write(40);
S3_servoOr.attach(32);
S3_servoOr.write(40);
S3_servoPi.attach(33);
S3_servoPi.write(40);

//--------------------------------------------------------------------------------

  // Attach section 4 servos to pins and set their default positions
S4_servoR.attach(34);
S4_servoR.write(40);
S4_servoG.attach(35);
S4_servoG.write(40);
S4_servoPu.attach(36);
S4_servoPu.write(40);
S4_servoY.attach(37);
S4_servoY.write(40);
S4_servoOr.attach(38);
S4_servoOr.write(40);
S4_servoPi.attach(39);
```

```
  S4_servoPi.write(40);

  //-------------------------------------------------------------------------------
  // The beam breaks are set as inputs, as it is needed to know when the value changes

  pinMode(GBeam,INPUT);
  pinMode(startBeam, INPUT);

  // Motor set as output to control the speed
  pinMode(myMotor, OUTPUT);
  analogWrite(myMotor,0);

  // Timer set as output to receive a signal to begin
  pinMode(externalTimer, OUTPUT);

}



void loop() {
  startBreak = digitalRead(startBeam);  // Checks to see what the current state of the start beam break is
and sets it to this variable

  // Starting condition for starting the machine
  // This nested conditional statement will check to see if the current clickcount is zero, which will always
be the case when the machine is first started
  // Click count will be > 0 as soon as the disk begins to spin therefore this statement becomes a NOP.

  if(GbeamClickCount == 0)
  {
    // Our beam breaks are defaulted to HIGH so if something passes by, the beam goes LOW
    if(startBreak == LOW){
      // Nothing here as the machine will not turn on with ambient light
    }
    else
    {
    analogWrite(myMotor, 133);  // Start the motor which turns the disk and the hopper
  }
  }

  //-----------------------------------------------------------------------------------------------
  // Check to see if the motor is spinning. If the motor is spinning then set
  // temp var C equal to D and set D equal to the millis() function divided by 1000, giving seconds.
  // Both C and D are integers so they can never have a decimal value, therefore if D minus C is equal
  // to 1, send a pulse to the timer. The externalTimer is set to LOW in the CheckClick() function so that
  // the pulse is at least 150ms, long enough for the 7Seg to read it.
  //-----------------------------------------------------------------------------------------------
```

```
 if((myMotor / 4) > 0)
 {
   C = D;
   D = millis() / 1000;
   if((D - C) == 1)
   {
     digitalWrite(externalTimer, HIGH);
   }

 }

 // AABeam will have the value that was last read from the beambreak
 AABeam = beamBreak;
 beamBreak = digitalRead(GBeam);

 // If the AABeam is high(Last value sensed by beambreak) and the beam break is sensed as LOW, a
falling edge is recognized
 // and the system know a Skittle is over a hole, ready to be sensed
 if (AABeam == 1 && beamBreak == 0)
 {
   CheckClick();  //  Call the function that will increment the click count, detect color, and open servo doors
 }

 if(startBreak == LOW && GbeamClickCount > 20){
   //Check if section 1 queues are empty
   if(redDoorS1.isEmpty() == true){
     if(greenDoorS1.isEmpty() == true){
       if(purpleDoorS1.isEmpty() == true){
         if(yellowDoorS1.isEmpty() == true){
           if(orangeDoorS1.isEmpty() == true){
             if(pinkDoorS1.isEmpty() == true){

    //Check if section 2 queues are empty
    if(redDoorS2.isEmpty() == true){
     if(greenDoorS2.isEmpty() == true){
       if(purpleDoorS2.isEmpty() == true){
         if(yellowDoorS2.isEmpty() == true){
           if(orangeDoorS2.isEmpty() == true){
             if(pinkDoorS2.isEmpty() == true){

    //Check if section 3 queues are empty
    if(redDoorS3.isEmpty() == true){
     if(greenDoorS3.isEmpty() == true){
       if(purpleDoorS3.isEmpty() == true){
         if(yellowDoorS3.isEmpty() == true){
           if(orangeDoorS3.isEmpty() == true){
             if(pinkDoorS3.isEmpty() == true){
```

```
    //Check if section 4 queues are empty
    if(redDoorS4.isEmpty() == true){
     if(greenDoorS4.isEmpty() == true){
       if(purpleDoorS4.isEmpty() == true){
         if(yellowDoorS4.isEmpty() == true){
           if(orangeDoorS4.isEmpty() == true){
             if(pinkDoorS4.isEmpty() == true){

               analogWrite(myMotor,0);  // Stop the motor
               digitalWrite(externalTimer, 0);  // Send a signal to the external timer to shut it off

             }
           }
         }
       }
     }
    }
          }
        }
      }
    }
   }
          }
        }
      }
    }
   }
          }
        }
      }
    }
   }

        }
      }
    }
   }
  }
 }

}
}

void CheckClick(){

   GbeamClickCount++;  // Increase the click count by one and get a color reading from each sensor

   // Get color reading from each of the 4 sections
   detectColorS1(S1_taosOutPin);
```

```
detectColorS2(S2_taosOutPin);
detectColorS3(S3_taosOutPin);
detectColorS4(S4_taosOutPin);

//--------------------------------------------------------------------------------
// Check to see if the red queue for section one has at least 1 index
// If it has at least one index, check if the number stored in the queue is equal to the
// current click count
if(redDoorS1.count() > 0){
  if (redDoorS1.peek() == GbeamClickCount){
    S1_servoR.write(0);  // open the servo door
    redDoorS1.pop();
  }
}
// Check to see if the green queue for section one has at least 1 index
// If it has at least one index, check if the number stored in the queue is equal to the
// current click count
if(greenDoorS1.count() > 0){
  if (greenDoorS1.peek() == GbeamClickCount){
    S1_servoG.write(0);  // open the servo door
    greenDoorS1.pop();
  }
}
// Check to see if the purple queue for section one has at least 1 index
// If it has at least one index, check if the number stored in the queue is equal to the
// current click count
if(purpleDoorS1.count() > 0){
  if (purpleDoorS1.peek() == GbeamClickCount){
    S1_servoPu.write(0);  // open the servo door
    purpleDoorS1.pop();
  }
}
// Check to see if the yellow queue for section one has at least 1 index
// If it has at least one index, check if the number stored in the queue is equal to the
// current click count
if(yellowDoorS1.count() > 0){
  if (yellowDoorS1.peek() == GbeamClickCount){
    S1_servoY.write(0);
    yellowDoorS1.pop();
  }
}
// Check to see if the orange queue for section one has at least 1 index
// If it has at least one index, check if the number stored in the queue is equal to the
// current click count
if (orangeDoorS1.count() > 0){
  if (orangeDoorS1.peek() == GbeamClickCount){
    S1_servoOr.write(0);  // open the servo door
    orangeDoorS1.pop();
```

```
    }
   }
   // Check to see if the pink queue for section one has at least 1 index
   // If it has at least one index, check if the number stored in the queue is equal to the
   // current click count
   if (pinkDoorS1.count() > 0){
    if (pinkDoorS1.peek() == GbeamClickCount){
     S1_servoPi.write(0);  // open the servo door
     pinkDoorS1.pop();
    }
   }
//--------------------------------------------------------------------------------

   // Check to see if the red queue for section two has at least 1 index
   // If it has at least one index, check if the number stored in the queue is equal to the
   // current click count
   if(redDoorS2.count() > 0){
    if (redDoorS2.peek() == GbeamClickCount){
     S2_servoR.write(0);  // open the servo door
     redDoorS2.pop();
    }
   }
   // Check to see if the green queue for section two has at least 1 index
   // If it has at least one index, check if the number stored in the queue is equal to the
   // current click count
   if(greenDoorS2.count() > 0){
    if (greenDoorS2.peek() == GbeamClickCount){
     S2_servoG.write(0);  // open the servo door
     greenDoorS2.pop();
    }
   }
   // Check to see if the purple queue for section two has at least 1 index
   // If it has at least one index, check if the number stored in the queue is equal to the
   // current click count
   if(purpleDoorS2.count() > 0){
    if (purpleDoorS2.peek() == GbeamClickCount){
     S2_servoPu.write(0);  // open the servo door
     purpleDoorS2.pop();
    }
   }
   // Check to see if the yellow queue for section two has at least 1 index
   // If it has at least one index, check if the number stored in the queue is equal to the
   // current click count
   if(yellowDoorS2.count() > 0){
    if (yellowDoorS2.peek() == GbeamClickCount){
     S2_servoY.write(0);  // open the servo door
     yellowDoorS2.pop();
    }
```

```
    }
    // Check to see if the orange queue for section two has at least 1 index
    // If it has at least one index, check if the number stored in the queue is equal to the
    // current click count
    if (orangeDoorS2.count() > 0){
      if (orangeDoorS2.peek() == GbeamClickCount){
        S2_servoOr.write(0);  // open the servo door
        orangeDoorS2.pop();
      }
    }
    // Check to see if the pink queue for section two has at least 1 index
    // If it has at least one index, check if the number stored in the queue is equal to the
    // current click count
    if (pinkDoorS2.count() > 0){
      if (pinkDoorS2.peek() == GbeamClickCount){
        S2_servoPi.write(0);  // open the servo door
        pinkDoorS2.pop();
      }
    }
//--------------------------------------------------------------------------------

    // Check to see if the red queue for section three has at least 1 index
    // If it has at least one index, check if the number stored in the queue is equal to the
    // current click count
    if(redDoorS3.count() > 0){
      if (redDoorS3.peek() == GbeamClickCount){
        S3_servoR.write(0);  // open the servo door
        redDoorS3.pop();
      }
    }
    // Check to see if the green queue for section three has at least 1 index
    // If it has at least one index, check if the number stored in the queue is equal to the
    // current click count
    if(greenDoorS3.count() > 0){
      if (greenDoorS3.peek() == GbeamClickCount){
        S3_servoG.write(0);  // open the servo door
        greenDoorS3.pop();
      }
    }
    // Check to see if the purple queue for section three has at least 1 index
    // If it has at least one index, check if the number stored in the queue is equal to the
    // current click count
    if(purpleDoorS3.count() > 0){
      if (purpleDoorS3.peek() == GbeamClickCount){
        S3_servoPu.write(0);  // open the servo door
        purpleDoorS3.pop();
      }
    }
```

```
    // Check to see if the yellow queue for section three has at least 1 index
    // If it has at least one index, check if the number stored in the queue is equal to the
    // current click count
    if(yellowDoorS3.count() > 0){
      if (yellowDoorS3.peek() == GbeamClickCount){
        S3_servoY.write(0);  // open the servo door
        yellowDoorS3.pop();
      }
    }
    // Check to see if the orange queue for section three has at least 1 index
    // If it has at least one index, check if the number stored in the queue is equal to the
    // current click count
    if (orangeDoorS3.count() > 0){
      if (orangeDoorS3.peek() == GbeamClickCount){
        S3_servoOr.write(0);  // open the servo door
        orangeDoorS3.pop();
      }
    }
    // Check to see if the pink queue for section three has at least 1 index
    // If it has at least one index, check if the number stored in the queue is equal to the
    // current click count
    if (pinkDoorS3.count() > 0){
      if (pinkDoorS3.peek() == GbeamClickCount){
        S3_servoPi.write(0);  // open the servo door
        pinkDoorS3.pop();
      }
    }
//--------------------------------------------------------------------------------


    // Check to see if the red queue for section four has at least 1 index
    // If it has at least one index, check if the number stored in the queue is equal to the
    // current click count
    if(redDoorS4.count() > 0){
      if (redDoorS4.peek() == GbeamClickCount){
        S4_servoR.write(0);  // open the servo door
        redDoorS4.pop();
      }
    }
    // Check to see if the green queue for section four has at least 1 index
    // If it has at least one index, check if the number stored in the queue is equal to the
    // current click count
    if(greenDoorS4.count() > 0){
      if (greenDoorS4.peek() == GbeamClickCount){
        S4_servoG.write(0);  // open the servo door
        greenDoorS4.pop();
      }
    }
    // Check to see if the purple queue for section four has at least 1 index
```

```
// If it has at least one index, check if the number stored in the queue is equal to the
// current click count
if(purpleDoorS4.count() > 0){
  if (purpleDoorS4.peek() == GbeamClickCount){
    S4_servoPu.write(0);  // open the servo door
    purpleDoorS4.pop();
  }
}
// Check to see if the yellow queue for section four has at least 1 index
// If it has at least one index, check if the number stored in the queue is equal to the
// current click count
if(yellowDoorS4.count() > 0){
  if (yellowDoorS4.peek() == GbeamClickCount){
    S4_servoY.write(0);  // open the servo door
    yellowDoorS4.pop();
  }
}
// Check to see if the orange queue for section four has at least 1 index
// If it has at least one index, check if the number stored in the queue is equal to the
// current click count
if (orangeDoorS4.count() > 0){
  if (orangeDoorS4.peek() == GbeamClickCount){
    S4_servoOr.write(0);  // open the servo door
    orangeDoorS4.pop();
  }
}
// Check to see if the pink queue for section four has at least 1 index
// If it has at least one index, check if the number stored in the queue is equal to the
// current click count
if (pinkDoorS4.count() > 0){
  if (pinkDoorS4.peek() == GbeamClickCount){
    S4_servoPi.write(0);  // open the servo door
    pinkDoorS4.pop();
  }
}
delay(servoDelay);  // Delay the speed of the disk / motor
digitalWrite(externalTimer,LOW);
// Reset all of the servos back to their original positions
S1_servoR.write(50);
S1_servoG.write(50);
S1_servoPu.write(50);
S1_servoY.write(50);
S1_servoOr.write(50);
S1_servoPi.write(50);
S2_servoR.write(50);
S2_servoG.write(50);
S2_servoPu.write(50);
S2_servoY.write(50);
```

```
    S2_servoOr.write(50);
    S2_servoPi.write(50);
    S3_servoR.write(50);
    S3_servoG.write(50);
    S3_servoPu.write(50);
    S3_servoY.write(50);
    S3_servoOr.write(50);
    S3_servoPi.write(50);
    S4_servoR.write(50);
    S4_servoG.write(50);
    S4_servoPu.write(50);
    S4_servoY.write(50);
    S4_servoOr.write(50);
    S4_servoPi.write(50);

    //loop();
  }

//--------------------------------------------------------------------------------
// This is sample code that has been revamped for the 5S
// Base code can be found at http://forums.parallax.com/showthread.php/136258-Code-for-using-a-TCS3200-with-Arduino-and-a-question

void TCS3200setupS1(){

    // Initialize the input and output pins
    pinMode(S1_LED,OUTPUT); // LED output pin

    // S2 and S3 control the 4 different color filters
    pinMode(S1_S2,OUTPUT); //Section 1 S2
    pinMode(S1_S3,OUTPUT); //Section 1 S3

    // The only input pin from the TAOS, controls the color value
    pinMode(S1_taosOutPin, INPUT); //Section 1 taosOutPin

    // S1 and S0 control the frequency or sensitivity of the sensor
    pinMode(S1_S0,OUTPUT); //Section 1 S0
    pinMode(S1_S1,OUTPUT); //Section 1 S1

    return;

}

//--------------------------------------------------------------------------------

void TCS3200setupS2(){
```

```
    // Initialize the input and output pins
    pinMode(S2_LED,OUTPUT); // LED output pin

    // S2 and S3 control the 4 different color filters
    pinMode(S2_S2,OUTPUT); // Section 2 S2
    pinMode(S2_S3,OUTPUT); // Section 2 S3

    // The only input pin from the TAOS, controls the color value
    pinMode(S2_taosOutPin, INPUT); // Section 2 taosOutPin

    // S1 and S0 control the frequency or sensitivity of the sensor
    pinMode(S2_S0,OUTPUT); // Section 2 S0
    pinMode(S2_S1,OUTPUT); // Section 2 S1

    return;

}

//-------------------------------------------------------------------------------

void TCS3200setupS3(){

    // Initialize the input and output pins
    pinMode(S3_LED,OUTPUT); //LED output pin

    // S2 and S3 control the 4 different color filters
    pinMode(S3_S2,OUTPUT); // Section 3 S2
    pinMode(S3_S3,OUTPUT); // Section 3 S3

    // The only input pin from the TAOS, controls the color value
    pinMode(S3_taosOutPin, INPUT); // Section 3 taosOutPin

    // S1 and S0 control the frequency or sensitivity of the sensor
    pinMode(S3_S0,OUTPUT); // Section 3 S0
    pinMode(S3_S1,OUTPUT); // Section 3 S1

    return;

}

//-------------------------------------------------------------------------------

void TCS3200setupS4(){

    // Initialize the input and output pins
    pinMode(S4_LED,OUTPUT); //LED ouput pin
```

```
    // S2 and S3 control the 4 different color filters
    pinMode(S4_S2,OUTPUT); // Section 4 S2
    pinMode(S4_S3,OUTPUT); // Section 4 S3

    // The only input pin from the TAOS, controls the color value
    pinMode(S4_taosOutPin, INPUT); // Section 4 taosOutPin

    // S1 and S0 control the frequency or sensitivity of the sensor
    pinMode(S4_S0,OUTPUT); // Section 4 S0
    pinMode(S4_S1,OUTPUT); // Section 4 S1

    return;

}

//--------------------------------------------------------------------------------------------
// The following method detects the color for the section one color sensor
// It calls the method colorReadS1 which reads the colors based on different color filters
// When colorReadS1 is finished it returns the color values and stores them in temp variables
// white,red,blue, and green.
// The variables RedR, BlueR, and GreenR hold the color ranges determined by taking a lot of
// color readings and normalizing them.
//--------------------------------------------------------------------------------------------
int detectColorS1(int taosOutPin){

double white = colorReadS1(taosOutPin,0,1);
double red = colorReadS1(taosOutPin,1,1);
double blue = colorReadS1(taosOutPin,2,1);
double green = colorReadS1(taosOutPin,3,1);
double RedR = (blue/green);
double BlueR = (green/red);
double GreenR = (red/blue);

// Set up the threshold values for normalizing the Skittle readings
double RedThreshold = (((GreenR >= 0.89) && (GreenR <= 1.28)) && ((BlueR >= 1.01) && (BlueR <=
1.49)) && ((RedR >= 0.71) && (RedR <= 0.78)));
double GreenThreshold = (((GreenR >= 1.08) && (GreenR <= 1.41)) && ((BlueR >= 0.76) && (BlueR <=
0.91)) && ((RedR >= 0.78) && (RedR <= 1.07)));
double PurpleThreshold = (((GreenR >= 1.66) && (GreenR <= 2.11)) && ((BlueR >= 0.75) && (BlueR <=
0.83)) && ((RedR >= 0.63) && (RedR <= 0.71)));
double OrangeThreshold = (((GreenR >= 0.63) && (GreenR <= 1.1)) && ((BlueR >= 1.08) && (BlueR <=
1.74)) && ((RedR >= 0.78) && (RedR <= 0.88)));
double YellowThreshold = (((GreenR >= 0.58) && (GreenR <= 1.04)) && ((BlueR >= 1) && (BlueR <=
1.22)) && ((RedR >= 0.89) && (RedR <= 1.37)));
double PinkThreshold = (((GreenR >= 0.89) && (GreenR <= 1.28)) && ((BlueR >= 1.01) && (BlueR <=
1.49)) && ((RedR >= 0.71) && (RedR <= 0.78)) && red <= 2100 && blue <= 1850);
```

```
// If one of the thresholds condition is met then push the current click count plus the predetermined color
distance for the servo door into each colors queue
if(PinkThreshold)
{
  pinkDoorS1.push(GbeamClickCount + pinkPreNum);
}
else if(OrangeThreshold)
{
  orangeDoorS1.push(GbeamClickCount + orangePreNum);
}
else if(RedThreshold)
{
  redDoorS1.push(GbeamClickCount + redPreNum);
}
else if(GreenThreshold)
{
  greenDoorS1.push(GbeamClickCount + greenPreNum);
}
else if(PurpleThreshold)
{
  purpleDoorS1.push(GbeamClickCount + purplePreNum);
}
else if(YellowThreshold)
{
 yellowDoorS1.push(GbeamClickCount + yellowPreNum);
}

}

//-------------------------------------------------------------------------------------------
// The following method detects the color for the section two color sensor
// It calls the method colorReadS2 which reads the colors based on different color filters
// When colorReadS1 is finished it returns the color values and stores them in temp variables
// white,red,blue, and green.
// The variables RedR, BlueR, and GreenR hold the color ranges determined by taking a lot of
// color readings and normalizing them.
//-------------------------------------------------------------------------------------------
int detectColorS2(int taosOutPin){

double white = colorReadS2(taosOutPin,0,1);
double red = colorReadS2(taosOutPin,1,1);
double blue = colorReadS2(taosOutPin,2,1);
double green = colorReadS2(taosOutPin,3,1);
double RedR = (blue/green);
double BlueR = (green/red);
double GreenR = (red/blue);
```

```
// Set up the threshold values for normalizing the Skittle readings
double RedThreshold = (((GreenR >= 1.11) && (GreenR <= 1.39)) && ((BlueR >= 1.07) && (BlueR <=
1.33)) && ((RedR >= 0.6) && (RedR <= 0.72)));
double GreenThreshold = (((GreenR >= 1.27) && (GreenR <= 1.56)) && ((BlueR >= 0.75) && (BlueR <=
0.95)) && ((RedR >= 0.74) && (RedR <= 0.94)));
double PurpleThreshold = (((GreenR >= 1.98) && (GreenR <= 2.4)) && ((BlueR >= 0.7) && (BlueR <=
0.83)) && ((RedR >= 0.54) && (RedR <= 0.66)));
double OrangeThreshold = (((GreenR >= 0.75) && (GreenR <= 1.26)) && ((BlueR >= 1.09) && (BlueR <=
1.65)) && ((RedR >= 0.67) && (RedR <= 0.79)));
double YellowThreshold = (((GreenR >= 0.73) && (GreenR <= 1.12)) && ((BlueR >= 1.03) && (BlueR <=
1.21)) && ((RedR >= 0.83) && (RedR <= 1.1)));
double PinkThreshold = (((GreenR >= 1.02) && (GreenR <= 1.39)) && ((BlueR >= 1.07) && (BlueR <=
1.45)) && ((RedR >= 0.6) && (RedR <= 0.72)) && (green <= 1900));

// If one of the thresholds condition is met then push the current click count plus the predetermined color
distance for the servo door into each colors queue
if(PinkThreshold)
{
  pinkDoorS2.push(GbeamClickCount + pinkPreNum);
}
else if(RedThreshold)
{
  redDoorS2.push(GbeamClickCount + redPreNum);
}
else if(GreenThreshold)
{
  greenDoorS2.push(GbeamClickCount + greenPreNum);
}
else if(OrangeThreshold)
{
  orangeDoorS2.push(GbeamClickCount + orangePreNum);
}
else if(PurpleThreshold)
{
  purpleDoorS2.push(GbeamClickCount + purplePreNum);
}
else if(YellowThreshold)
{
 yellowDoorS2.push(GbeamClickCount + yellowPreNum);
}

}

//-----------------------------------------------------------------------------------------
// The following method detects the color for the section three color sensor
// It calls the method colorReadS3 which reads the colors based on different color filters
// When colorReadS1 is finished it returns the color values and stores them in temp variables
// white,red,blue, and green.
```

```
// The variables RedR, BlueR, and GreenR hold the color ranges determined by taking a lot of
// color readings and normalizing them.
//--------------------------------------------------------------------------------------------
int detectColorS3(int taosOutPin){

double white = colorReadS3(taosOutPin,0,1);
double red = colorReadS3(taosOutPin,1,1);
double blue = colorReadS3(taosOutPin,2,1);
double green = colorReadS3(taosOutPin,3,1);
double RedR = (blue/green);
double BlueR = (green/red);
double GreenR = (red/blue);

// Set up the threshold values for normalizing the Skittle readings
double RedThreshold = (((GreenR >= 1.05) && (GreenR <= 1.44)) && ((BlueR >= 0.97) && (BlueR <=
1.38)) && ((RedR >= 0.6) && (RedR <= 0.78)) && red >= 1350);
double GreenThreshold = (((GreenR >= 1.13) && (GreenR <= 1.53)) && ((BlueR >= 0.74) && (BlueR <=
1.04)) && ((RedR >= 0.68) && (RedR <= 1)) && red >= 1250);
double PurpleThreshold = (((GreenR >= 1.91) && (GreenR <= 2.21)) && ((BlueR >= 0.7) && (BlueR <=
0.86)) && ((RedR >= 0.54) && (RedR <= 0.7)));
double OrangeThreshold = (((GreenR >= 0.76) && (GreenR <= 1.21)) && ((BlueR >= 1.13) && (BlueR <=
1.6)) && ((RedR >= 0.69) && (RedR <= 0.81)));
double YellowThreshold = (((GreenR >= 0.69) && (GreenR <= 1.07)) && ((BlueR >= 1) && (BlueR <=
1.25)) && ((RedR >= 0.86) && (RedR <= 1.18)));
double PinkThreshold = (((GreenR >= 1.08) && (GreenR <= 1.41)) && ((BlueR >= 1) && (BlueR <= 1.35))
&& ((RedR >= 0.63) && (RedR <= 0.75)) && red <= 1355 && blue <= 1000);

// If one of the thresholds condition is met then push the current click count plus the predetermined color
distance for the servo door into each colors queue
if(OrangeThreshold)
{
  orangeDoorS3.push(GbeamClickCount + orangePreNum);
}
else if(PinkThreshold)
{
  pinkDoorS3.push(GbeamClickCount + pinkPreNum);
}
else if(YellowThreshold)
{
 yellowDoorS3.push(GbeamClickCount + yellowPreNum);
}
else if(RedThreshold)
{
  redDoorS3.push(GbeamClickCount + redPreNum);
}
else if(GreenThreshold)
{
  greenDoorS3.push(GbeamClickCount + greenPreNum);
```

```
}
else if(PurpleThreshold)
{
  purpleDoorS3.push(GbeamClickCount + purplePreNum);
}

}

//--------------------------------------------------------------------------------------------
// The following method detects the color for the section four color sensor
// It calls the method colorReadS4 which reads the colors based on different color filters
// When colorReadS1 is finished it returns the color values and stores them in temp variables
// white,red,blue, and green.
// The variables RedR, BlueR, and GreenR hold the color ranges determined by taking a lot of
// color readings and normalizing them.
//--------------------------------------------------------------------------------------------
int detectColorS4(int taosOutPin){


double white = colorReadS4(taosOutPin,0,1);
double red = colorReadS4(taosOutPin,1,1);
double blue = colorReadS4(taosOutPin,2,1);
double green = colorReadS4(taosOutPin,3,1);
double RedR = (blue/green);
double BlueR = (green/red);
double GreenR = (red/blue);

// Set up the threshold values for normalizing the Skittle readings
double RedThreshold = (((GreenR >= 1.22) && (GreenR <= 1.45)) && ((BlueR >= 0.96) && (BlueR <=
1.15)) && ((RedR >= 0.67) && (RedR <= 0.75)));
double GreenThreshold = (((GreenR >= 1.17) && (GreenR <= 1.53)) && ((BlueR >= 0.81) && (BlueR <=
0.9)) && ((RedR >= 0.75) && (RedR <= 0.98)));
double PurpleThreshold = (((GreenR >= 1.74) && (GreenR <= 2.16)) && ((BlueR >= 0.72) && (BlueR <=
0.84)) && ((RedR >= 0.61) && (RedR <= 0.71)));
double OrangeThreshold = (((GreenR >= 0.85) && (GreenR <= 1.13)) && ((BlueR >= 1.15) && (BlueR <=
1.46)) && ((RedR >= 0.71) && (RedR <= 0.82)));
double YellowThreshold = (((GreenR >= 0.72) && (GreenR <= 1.12)) && ((BlueR >= 0.97) && (BlueR <=
1.2)) && ((RedR >= 0.85) && (RedR <= 1.18)));
double PinkThreshold = (((GreenR >= 1.14) && (GreenR <= 1.45)) && ((BlueR >= 0.96) && (BlueR <=
1.23)) && ((RedR >= 0.67) && (RedR <= 0.75)) && (green <= 2050) && (red <= 1900) && (blue <=
1450));

// If one of the thresholds condition is met then push the current click count plus the predetermined color
distance for the servo door into each colors queue
if(PinkThreshold)
{
  pinkDoorS4.push(GbeamClickCount + pinkPreNum);
```

```
}
else if(RedThreshold)
{
  redDoorS4.push(GbeamClickCount + redPreNum);
}
else if(GreenThreshold)
{
  greenDoorS4.push(GbeamClickCount + greenPreNum);
}
else if(OrangeThreshold)
{
  orangeDoorS4.push(GbeamClickCount + orangePreNum);
}
else if(PurpleThreshold)
{
  purpleDoorS4.push(GbeamClickCount + purplePreNum);
}
else if(YellowThreshold)
{
 yellowDoorS4.push(GbeamClickCount + yellowPreNum);
}

}

//--------------------------------------------------------------------------------

double colorReadS1(int taosOutPin, int color, boolean LEDstate){

// Turn on sensor and use a 1:50 ratio, allowing for more color variation

taosMode(3);

//set the S2 and S3 pins to select the color to be sensed
if(color == 0){//white
digitalWrite(S1_S3, LOW); //S3
digitalWrite(S1_S2, HIGH); //S2
// Serial.print(" w");
}

else if(color == 1){//red
digitalWrite(S1_S3, LOW); //S3
digitalWrite(S1_S2, LOW); //S2
// Serial.print(" r");
}

else if(color == 2){//blue
digitalWrite(S1_S3, HIGH); //S3
```

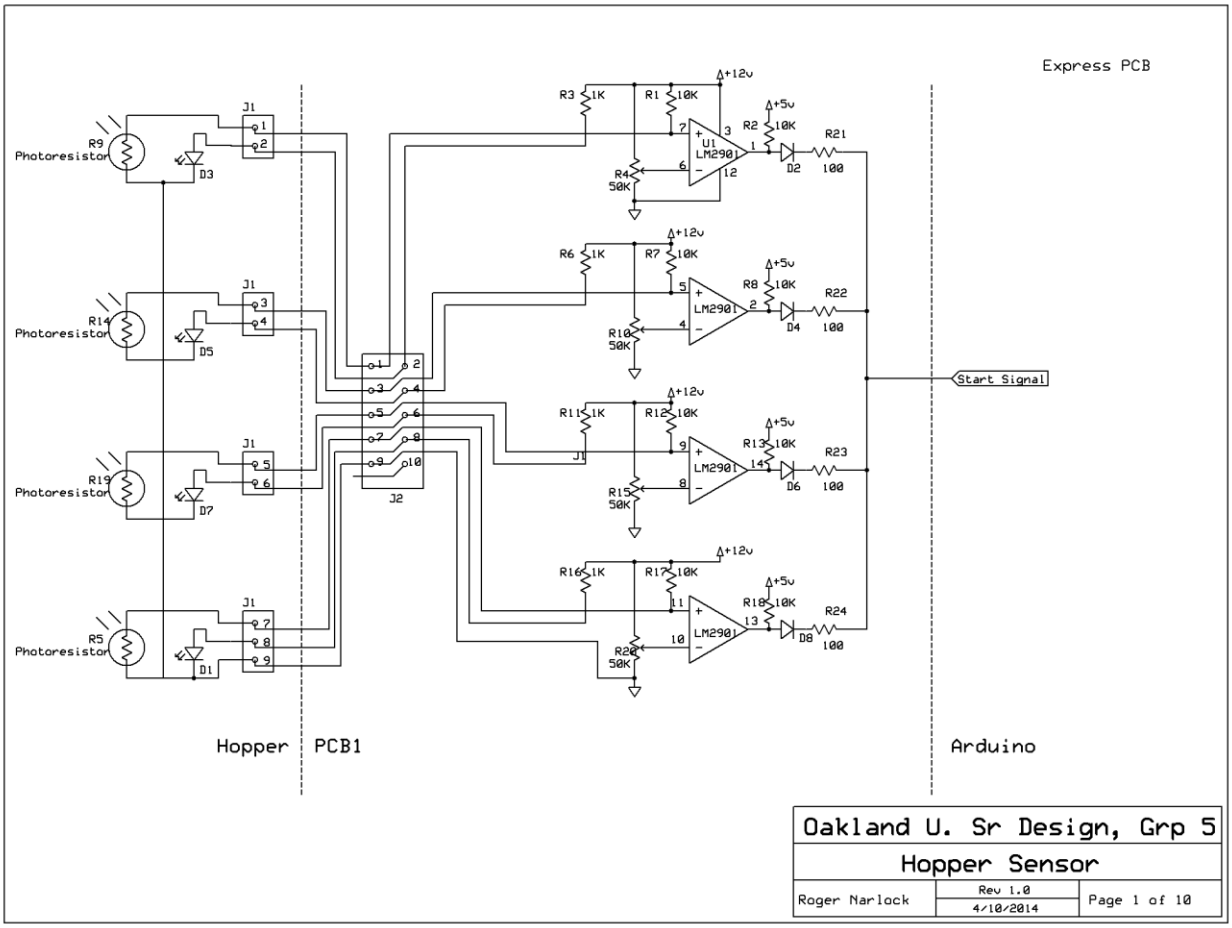
```
digitalWrite(S1_S2, LOW); //S2
// Serial.print(" b");
}

else if(color == 3){//green
digitalWrite(S1_S3, HIGH); //S3
digitalWrite(S1_S2, HIGH); //S2
// Serial.print(" g");
}

// create a var where the pulse reading from sensor will go
float readPulse;

//  turn LEDs on or off, as directed by the LEDstate var
if(LEDstate == 0){
   digitalWrite(S1_LED, HIGH);
}
if(LEDstate == 1){
   digitalWrite(S1_LED, HIGH);
}

// Reads a pulse from the sensor
readPulse = pulseIn(taosOutPin, LOW, 80000);
// If for some reason the pulse times out meaning no power or some other reason, then just set it to
80,000.
if(readPulse < .1){
readPulse = 80000;
}

return readPulse;

}

//------------------------------------------------------------------------------

double colorReadS2(int taosOutPin, int color, boolean LEDstate){

taosMode(3);

//set the S2 and S3 pins to select the color to be sensed
if(color == 0){//white
digitalWrite(S2_S3, LOW); //S3
digitalWrite(S2_S2, HIGH); //S2
// Serial.print(" w");
}

else if(color == 1){//red
```

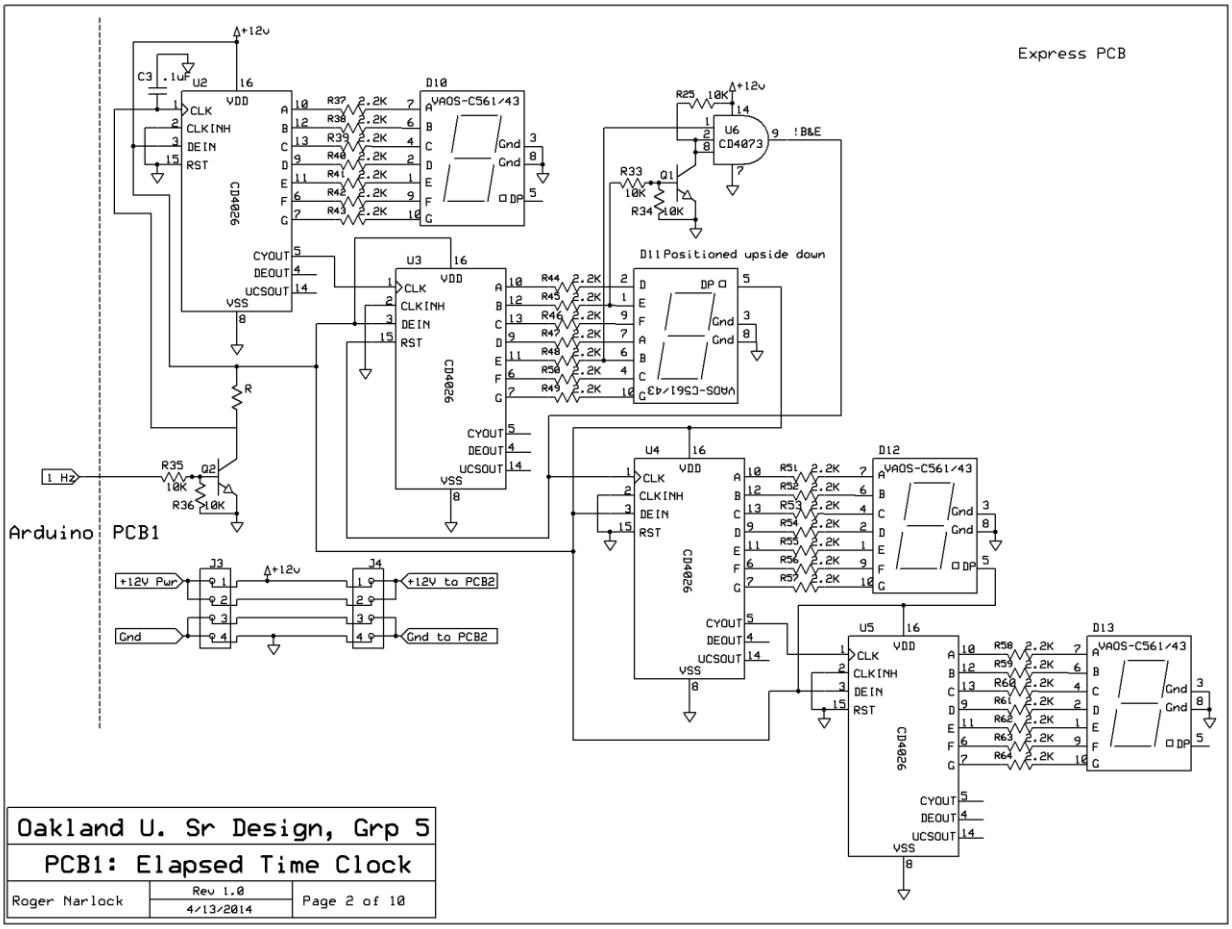
```
digitalWrite(S2_S3, LOW); //S3
digitalWrite(S2_S2, LOW); //S2
// Serial.print(" r");
}

else if(color == 2){//blue
digitalWrite(S2_S3, HIGH); //S3
digitalWrite(S2_S2, LOW); //S2
// Serial.print(" b");
}

else if(color == 3){//green
digitalWrite(S2_S3, HIGH); //S3
digitalWrite(S2_S2, HIGH); //S2
// Serial.print(" g");
}

// create a var where the pulse reading from sensor will go
float readPulse;

//  turn LEDs on or off, as directed by the LEDstate var
if(LEDstate == 0){
   digitalWrite(S2_LED, HIGH);
}
if(LEDstate == 1){
   digitalWrite(S2_LED, HIGH);
}

// Reads a pulse from the sensor
readPulse = pulseIn(taosOutPin, LOW, 80000);
// If for some reason the pulse times out meaning no power or some other reason, then just set it to
80,000.
if(readPulse < .1){
readPulse = 80000;
}

// return the pulse value back to whatever called for it...
return readPulse;

}

//---------------------------------------------------------------------------

double colorReadS3(int taosOutPin, int color, boolean LEDstate){

taosMode(3);
```

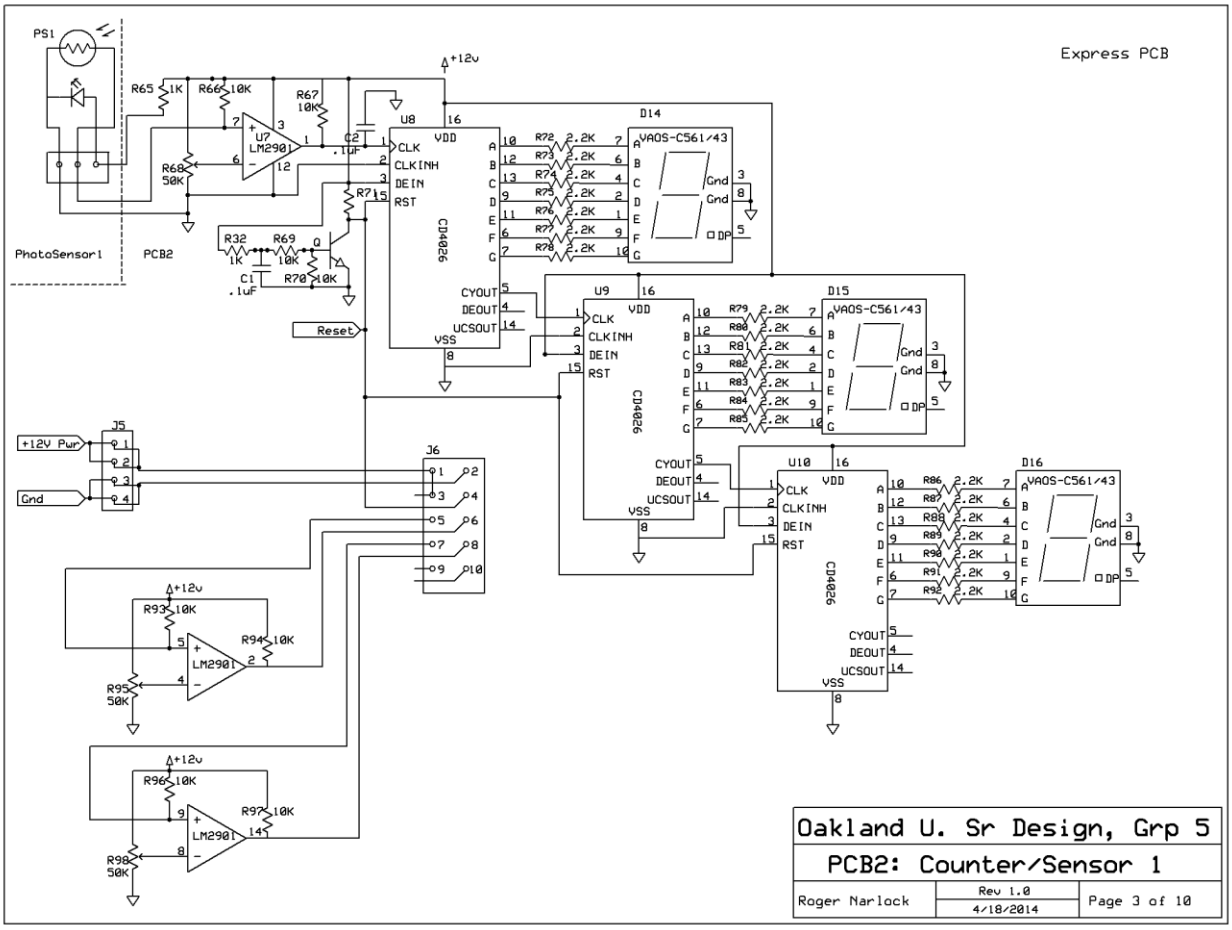
```
//set the S2 and S3 pins to select the color to be sensed
if(color == 0){//white
digitalWrite(S3_S3, LOW); //S3
digitalWrite(S3_S2, HIGH); //S2
// Serial.print(" w");
}

else if(color == 1){//red
digitalWrite(S3_S3, LOW); //S3
digitalWrite(S3_S2, LOW); //S2
// Serial.print(" r");
}

else if(color == 2){//blue
digitalWrite(S3_S3, HIGH); //S3
digitalWrite(S3_S2, LOW); //S2
// Serial.print(" b");
}

else if(color == 3){//green
digitalWrite(S3_S3, HIGH); //S3
digitalWrite(S3_S2, HIGH); //S2
// Serial.print(" g");
}

// create a var where the pulse reading from sensor will go
float readPulse;

//  turn LEDs on or off, as directed by the LEDstate var
if(LEDstate == 0){
   digitalWrite(S3_LED, HIGH);
}
if(LEDstate == 1){
   digitalWrite(S3_LED, HIGH);
}

// Reads a pulse from the sensor
readPulse = pulseIn(taosOutPin, LOW, 80000);
// If for some reason the pulse times out meaning no power or some other reason, then just set it to
80,000.
if(readPulse < .1){
readPulse = 80000;
}


// return the pulse value back to whatever called for it...
return readPulse;
```

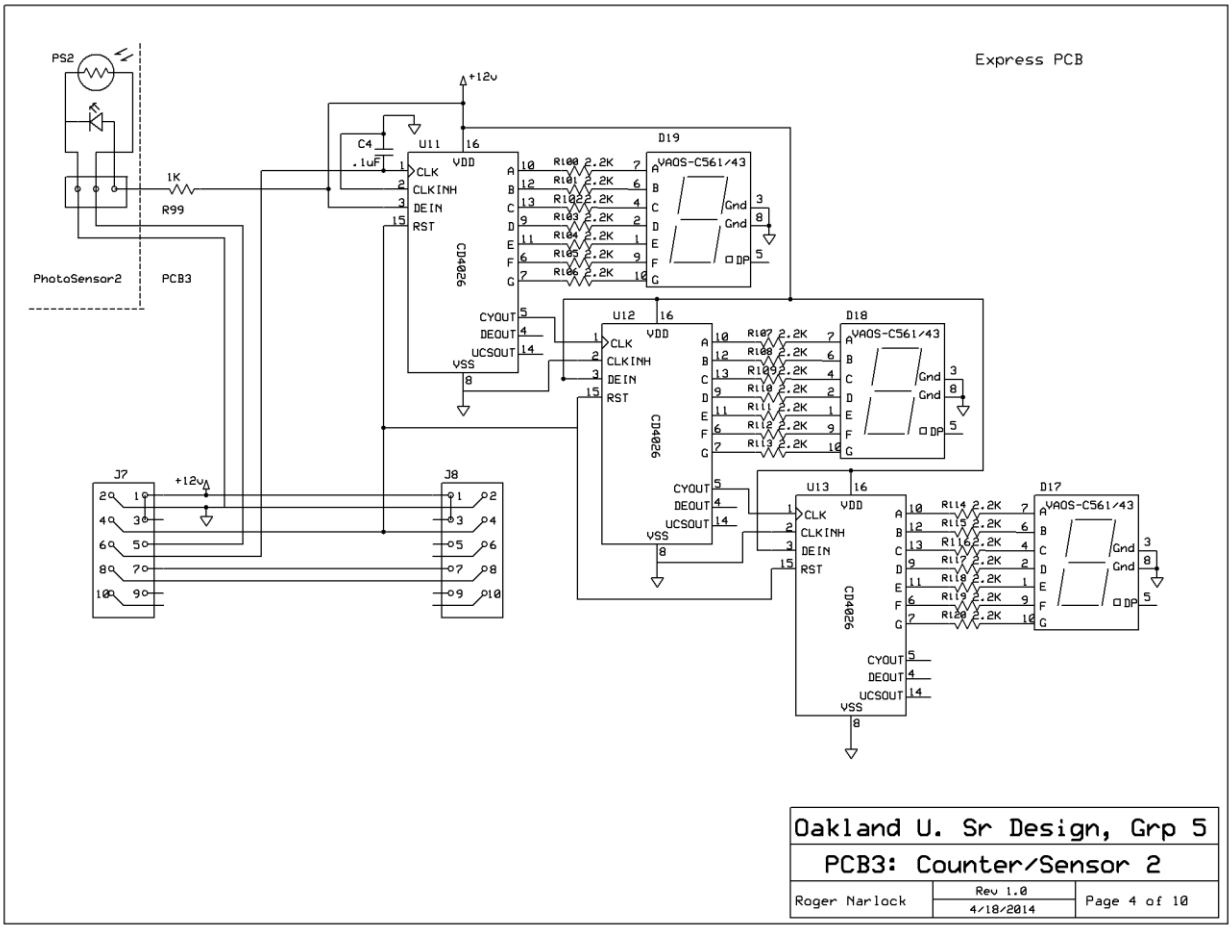
```
}

//--------------------------------------------------------------------------------

double colorReadS4(int taosOutPin, int color, boolean LEDstate){

taosMode(3);

//set the S2 and S3 pins to select the color to be sensed
if(color == 0){//white
digitalWrite(S4_S3, LOW); //S3
digitalWrite(S4_S2, HIGH); //S2
// Serial.print(" w");
}

else if(color == 1){//red
digitalWrite(S4_S3, LOW); //S3
digitalWrite(S4_S2, LOW); //S2
// Serial.print(" r");
}

else if(color == 2){//blue
digitalWrite(S4_S3, HIGH); //S3
digitalWrite(S4_S2, LOW); //S2
// Serial.print(" b");
}

else if(color == 3){//green
digitalWrite(S4_S3, HIGH); //S3
digitalWrite(S4_S2, HIGH); //S2
// Serial.print(" g");
}

// create a var where the pulse reading from sensor will go
float readPulse;

//  turn LEDs on or off, as directed by the LEDstate var
if(LEDstate == 0){
   digitalWrite(S4_LED, HIGH);
}
if(LEDstate == 1){
   digitalWrite(S4_LED, HIGH);
}

// Reads a pulse from the sensor
readPulse = pulseIn(taosOutPin, LOW, 80000);
```

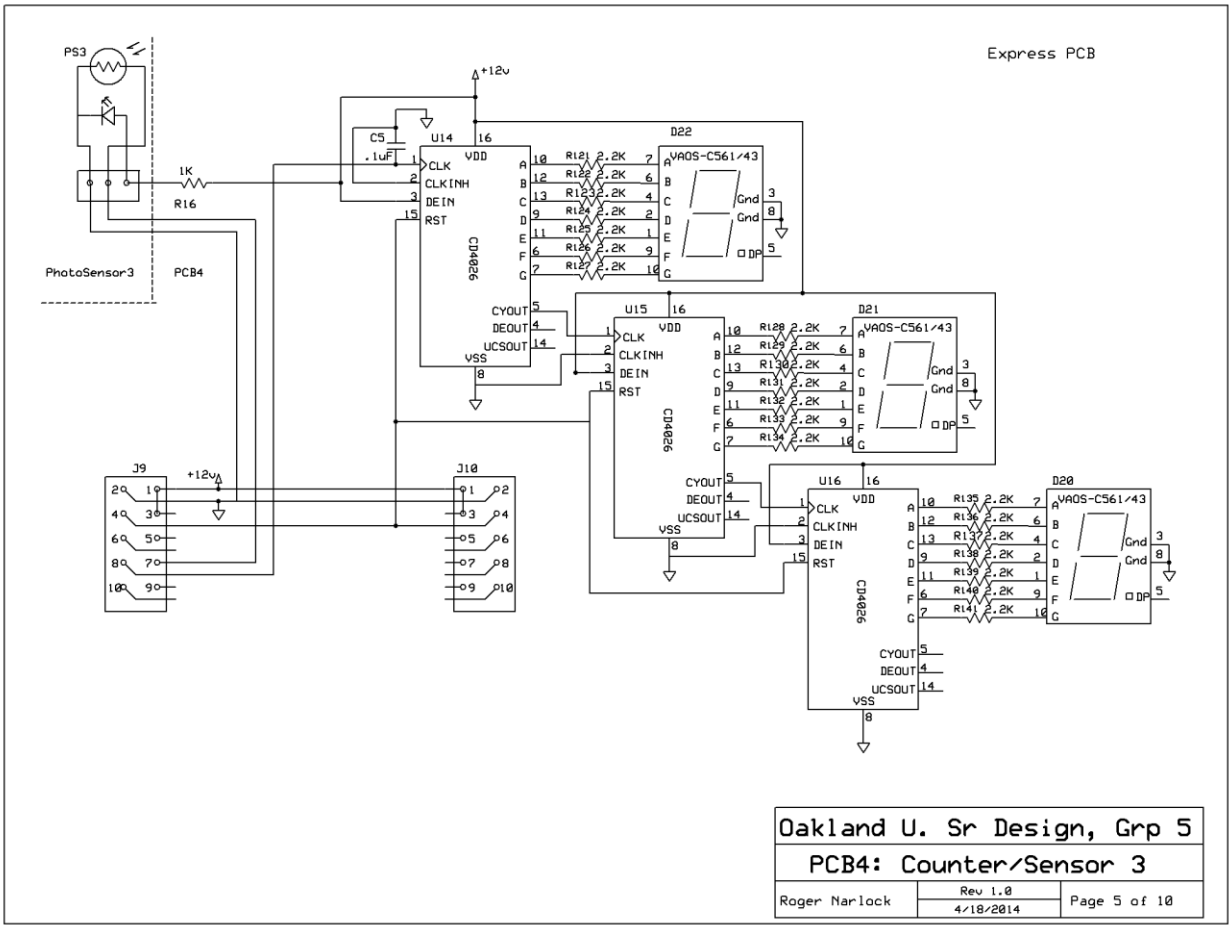
```
// If for some reason the pulse times out meaning no power or some other reason, then just set it to
80,000.
if(readPulse < .1){
readPulse = 80000;
}


return readPulse;

}




//--------------------------------------------------------------------------------

// Operation modes area, controlled by high/low settings on S0 and S1 pins.
// Setting mode to zero will put Taos into low power mode. taosMode(0);

void taosMode(int mode){

   // taosMode(3) is being used, i.e. 3
   if(mode == 3){
   //this will put in 1:50
   digitalWrite(S1_S0, LOW); //S0
   digitalWrite(S1_S1, HIGH); //S1

   digitalWrite(S2_S0, LOW); //S0
   digitalWrite(S2_S1, HIGH); //S1

   digitalWrite(S3_S0, LOW); //S0
   digitalWrite(S3_S1, HIGH); //S1

   digitalWrite(S4_S0, LOW); //S0
   digitalWrite(S4_S1, HIGH); //S1
   }

   return;

}
```

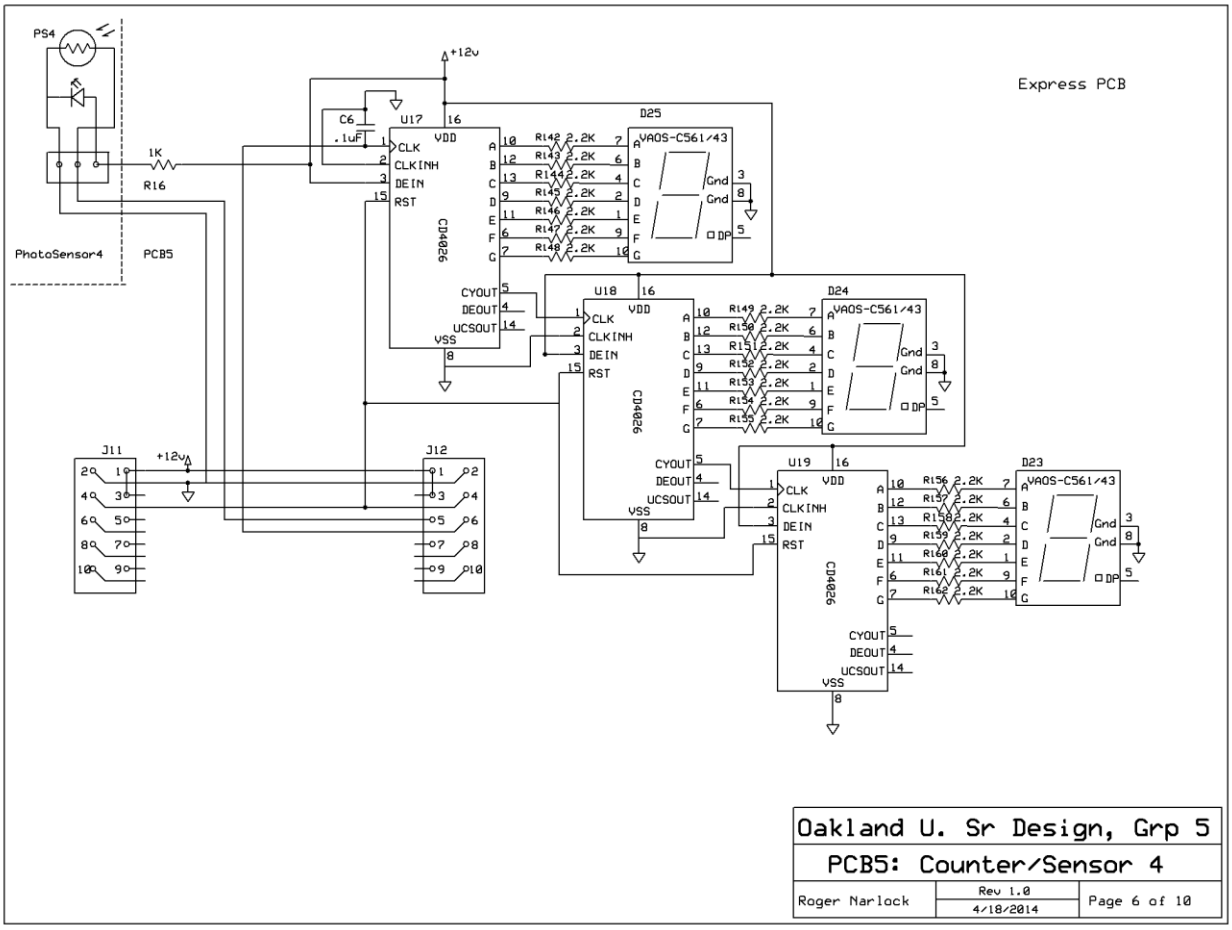

Express PCB
Photoresistor
R9
R14
R19
R5
D3
D5
D7
D1
J1
J2
R3 1K
R1 10K
R2 10K
R4 50K
U1 LM2901
R21
D2
100
+12v
+5v
R6 1K
R7 10K
R8 10K
R10 50K
R22
D4
R11 1K
R12 10K
R13 10K
R15 50K
R23
D6
R16 1K
R17 10K
R18 10K
R20 50K
R24
D8
Start Signal
Hopper
PCB1
Arduino
Oakland U. Sr Design, Grp 5
Hopper Sensor
Roger Narlock
Rev 1.0
4/10/2014
Page 1 of 10

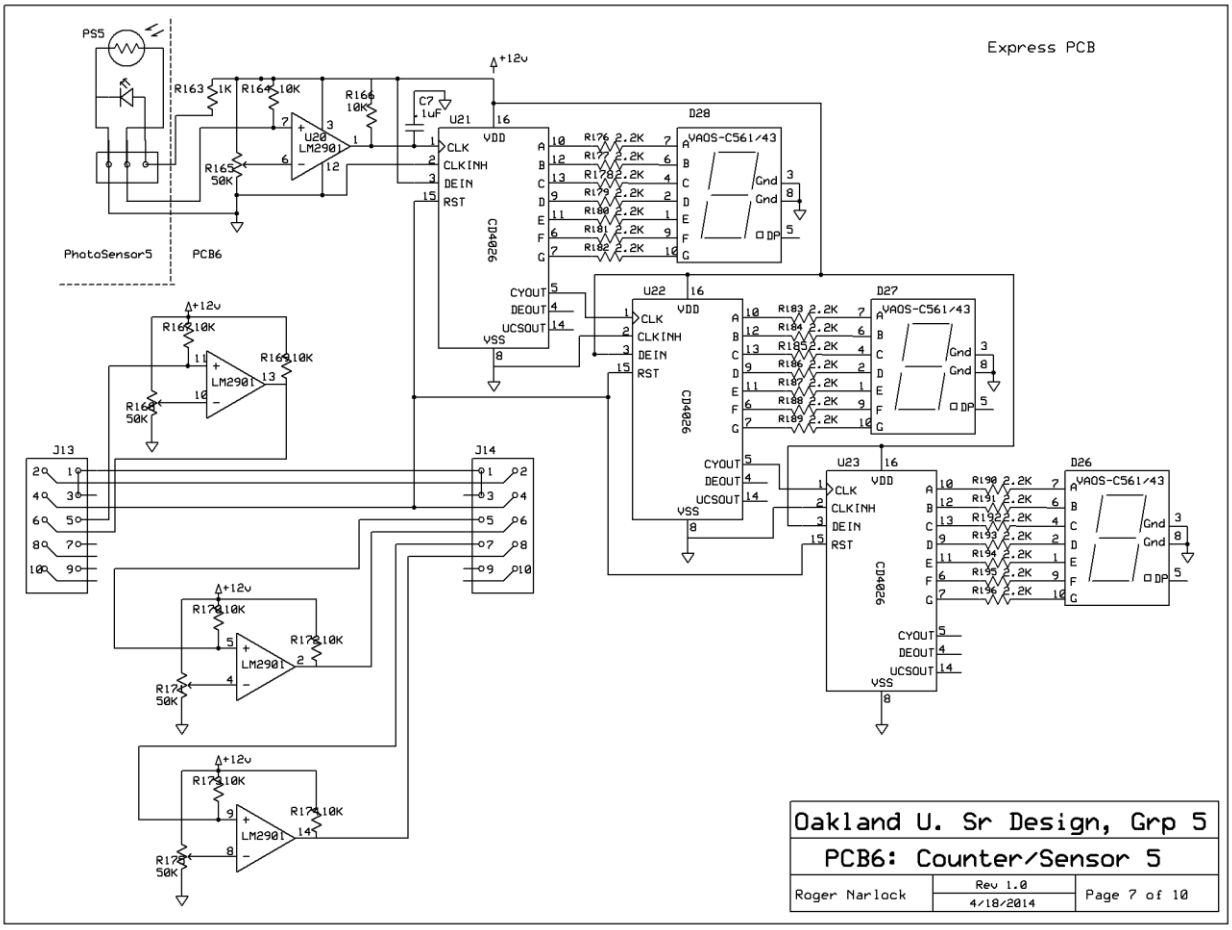

Express PCB
Arduino
PCB1
1 Hz
+12V Pwr
Gnd
+12V to PCB2
Gnd to PCB2
D11 Positioned upside down
Oakland U. Sr Design, Grp 5
PCB1: Elapsed Time Clock
Roger Narlock
Rev 1.0
4/13/2014
Page 2 of 10

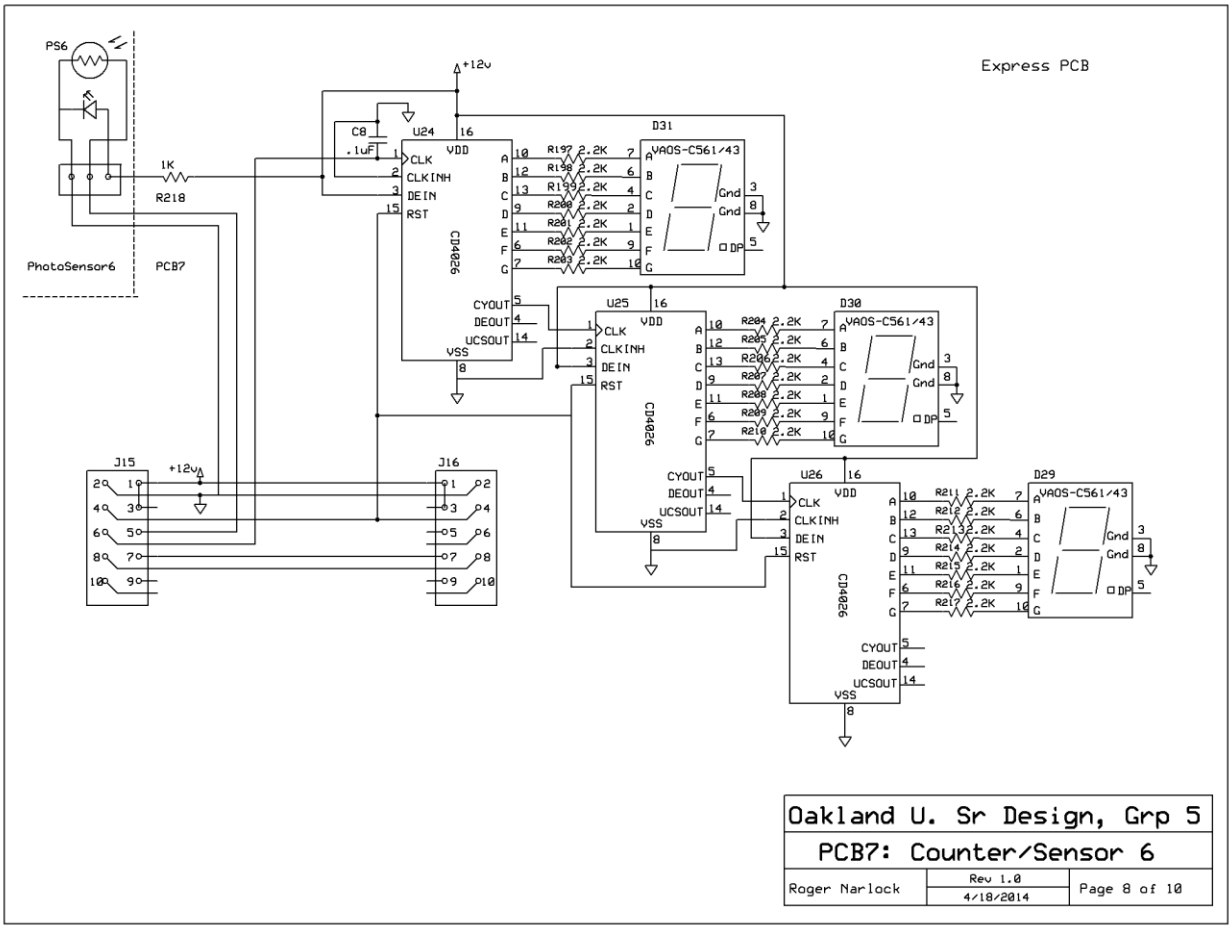

Express PCB
PS1
PhotaSensor1
PCB2
+12v
R65 1K
R66 10K
R67 10K
R68 50K
U7 LM2901
C2 .1uF
U8
CD4026
VDD
CLK
CLKINH
DEIN
RST
CYOUT
DEOUT
UCSOUT
VSS
R72 2.2K
R73 2.2K
R74 2.2K
R75 2.2K
R76 2.2K
R77 2.2K
R78 2.2K
D14
VAOS-C561/43
Gnd
DP
R32 1K
R69 10K
R70 10K
C1 .1uF
Q
R71 5
Reset
U9
D15
U10
D16
J5
+12v Pwr
Gnd
J6
R93 10K
R94 10K
R95 50K
R96 10K
R97 10K
R98 50K
Oakland U. Sr Design, Grp 5
PCB2: Counter/Sensor 1
Roger Narlock
Rev 1.0
4/18/2014
Page 3 of 10

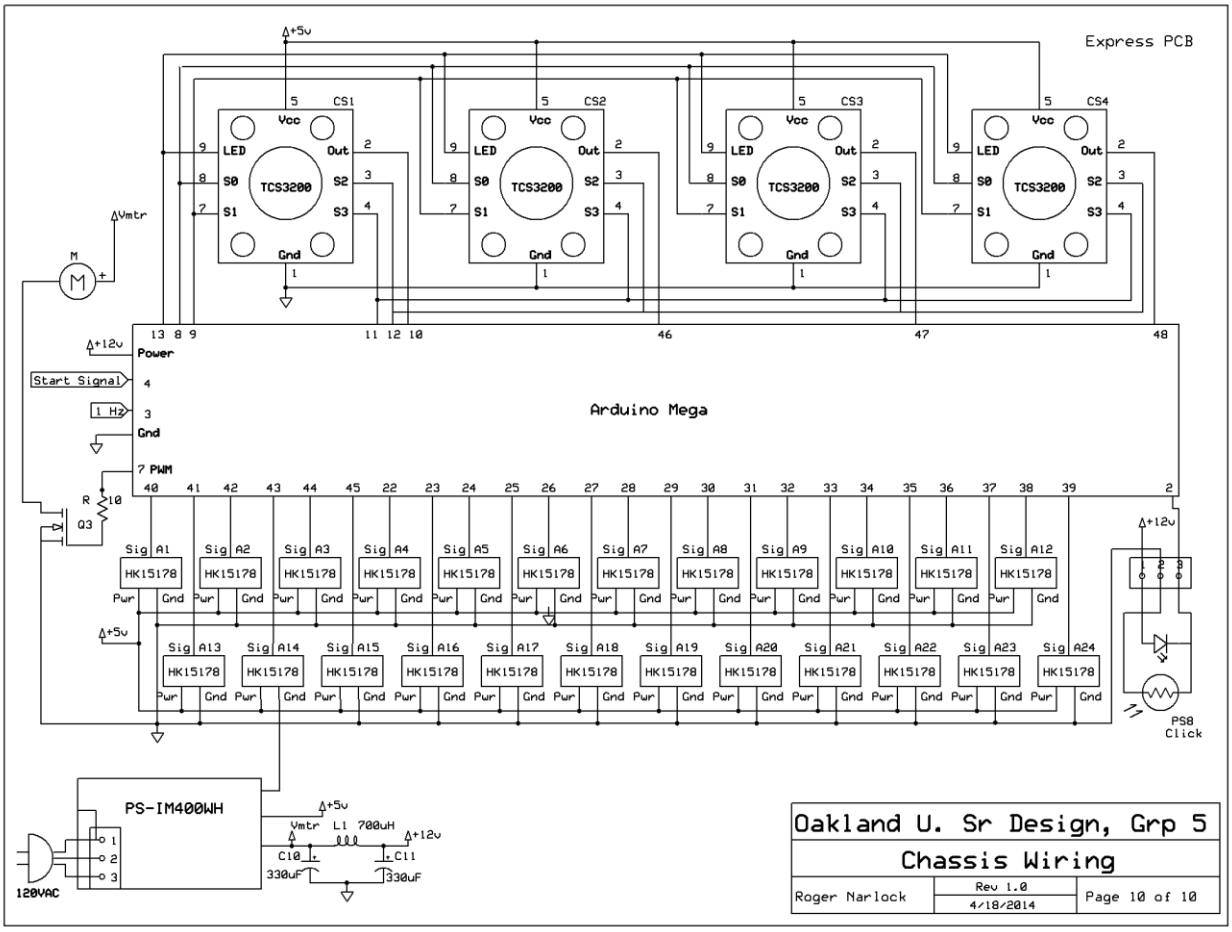

Express PCB
PS2
PhotoSensor2
PCB3
1K
R99
C4
.1uF
+12v
U11
CD4026
CLK
CLKINH
DEIN
RST
VDD
VSS
CYOUT
DEOUT
UCSOUT
D19
D18
D17
VAOS-C561/43
Gnd
DP
U12
U13
J7
J8
Oakland U. Sr Design, Grp 5
PCB3: Counter/Sensor 2
Roger Narlock
Rev 1.0
4/18/2014
Page 4 of 10


Express PCB
PS3
PhotoSensor3
PCB4
1K
R16
+12v
C5
.1uF
U14
CD4026
CLK
CLKINH
DEIN
RST
VDD
VSS
CYOUT
DEOUT
UCSOUT
R121 2.2K
R122 2.2K
R123 2.2K
R124 2.2K
R125 2.2K
R126 2.2K
R127 2.2K
D22
VAOS-C561/43
Gnd
DP
U15
R128 2.2K
R129 2.2K
R130 2.2K
R131 2.2K
R132 2.2K
R133 2.2K
R134 2.2K
D21
U16
R135 2.2K
R136 2.2K
R137 2.2K
R138 2.2K
R139 2.2K
R140 2.2K
R141 2.2K
D20
J9
J10
Oakland U. Sr Design, Grp 5
PCB4: Counter/Sensor 3
Roger Narlock
Rev 1.0
4/18/2014
Page 5 of 10


Express PCB
PS4
PhotaSensor4
PCB5
1K
R16
+12v
C6
.1uF
U17
U18
U19
CD4026
CLK
CLKINH
DEIN
RST
VDD
VSS
CYOUT
DEOUT
UCSOUT
R142 2.2K
R143 2.2K
R144 2.2K
R145 2.2K
R146 2.2K
R147 2.2K
R148 2.2K
R149 2.2K
R150 2.2K
R151 2.2K
R152 2.2K
R153 2.2K
R154 2.2K
R155 2.2K
R156 2.2K
R157 2.2K
R158 2.2K
R159 2.2K
R160 2.2K
R161 2.2K
R162 2.2K
D25
D24
D23
VAOS-C561/43
Gnd
DP
J11
J12
Oakland U. Sr Design, Grp 5
PCB5: Counter/Sensor 4
Roger Narlock
Rev 1.0
4/18/2014
Page 6 of 10


Express PCB
PS5
PhotoSensor5
PCB6
R163 1K
R164 10K
R165 50K
R166 10K
U20 LM2901
C7 1uF
+12v
U21
CD4026
CLK
CLKINH
DEIN
RST
CYOUT
DEOUT
UCSOUT
VSS
VDD
D28
D27
D26
VAOS-C561/43
Gnd
DP
U22
U23
R167 10K
R168 50K
R169 10K
R170 10K
R171 50K
R172 10K
R173 10K
R174 50K
R175 10K
J13
J14
Oakland U. Sr Design, Grp 5
PCB6: Counter/Sensor 5
Roger Narlock
Rev 1.0
4/18/2014
Page 7 of 10


Express PCB
PS6
PhotoSensor6
PCB7
1K
R218
+12v
C8
.1uF
U24
CD4026
CLK
CLKINH
DEIN
RST
VDD
VSS
CYOUT
DEOUT
UCSOUT
R197 2.2K
R198 2.2K
R199 2.2K
R200 2.2K
R201 2.2K
R202 2.2K
R203 2.2K
D31
VAOS-C561/43
Gnd
DP
U25
R204 2.2K
R205 2.2K
R206 2.2K
R207 2.2K
R208 2.2K
R209 2.2K
R210 2.2K
D30
U26
R211 2.2K
R212 2.2K
R213 2.2K
R214 2.2K
R215 2.2K
R216 2.2K
R217 2.2K
D29
J15
J16
Oakland U. Sr Design, Grp 5
PCB7: Counter/Sensor 6
Roger Narlock
Rev 1.0
4/18/2014
Page 8 of 10


Express PCB
+5v
CS1
CS2
CS3
CS4
Vcc
LED
Out
S0
S2
S1
S3
Gnd
TCS3200
Vmtr
M
+12v
Power
Start Signal
1 Hz
Gnd
PWM
R
10
Q3
Arduino Mega
Sig
HK15178
Pwr
PS8
Click
PS-IM400WH
120VAC
L1 700uH
C10
330uF
C11
Oakland U. Sr Design, Grp 5
Chassis Wiring
Roger Narlock
Rev 1.0
4/18/2014
Page 10 of 10